



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΙΓΑΙΟΥ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ
ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ
ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ
ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΔΡΥΜΑΤΙΚΟ
ΠΡΟΓΡΑΜΜΑ
ΜΕΤΑΠΤΥΧΙΑΚΩΝ
ΣΠΟΥΔΩΝ (Δ.Π.Μ.Σ.)
ΜΒΑ ΣΤΗΝ ΨΗΦΙΑΚΗ
ΚΑΙΝΟΤΟΜΙΑ ΚΑΙ
ΝΕΟΦΥΗ
ΕΠΙΧΕΙΡΗΜΑΤΙΚΟΤΗΤΑ

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ
ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ
ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ
ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ
ΣΥΣΤΗΜΑΤΩΝ

AI-Native Management and Orchestration: A Methodological Framework for Solo-Founder Digital Entrepreneurship

AI-Native Διοίκηση και Ενορχήστρωση: Ένα Μεθοδολογικό Πλαίσιο για την Ψηφιακή Επιχειρηματικότητα Solo Founder

Λιάπης Νικόλαος

icsdm725008, icsdm725008@icsd.aegean.gr
nikos@liapis.info

Επιβλέπων: Αναπληρωτής Καθηγητής,
Γεράσιμος Κοντός

Σεπτέμβριος 2026

Η Διπλωματική Εργασία παρουσιάστηκε ενώπιον
του Διδακτικού Προσωπικού του Πανεπιστημίου Αιγαίου & του Εθνικού Μετσόβιου
Πολυτεχνείου
Σε Μερική Εκπλήρωση των απαιτήσεων για την απόκτηση του μεταπτυχιακού διπλώματος
ειδίκευσης «MBA στην Ψηφιακή Καινοτομία και Νεοφυή Επιχειρηματικότητα»

Η ΤΡΙΜΕΛΗΣ ΕΠΙΤΡΟΠΗ ΔΙΔΑΣΚΟΝΤΩΝ
ΕΠΙΚΥΡΩΝΕΙ ΤΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
ΤΟΥ
ΝΙΚΟΛΑΟΥ ΛΙΑΠΗ

ΓΕΡΑΣΙΜΟΣ ΚΟΝΤΟΣ, Επιβλέπων
Αναπληρωτής Καθηγητής
Τμήμα Μηχανικών Πληροφοριακών
και Επικοινωνιακών Συστημάτων

Γιάννης Χαραλαμπίδης, Μέλος
Καθηγητής
Τμήμα Μηχανικών Πληροφοριακών
και Επικοινωνιακών Συστημάτων

Χαράλαμπος Αλεξόπουλος, Μέλος
Επίκουρος Καθηγητής
Τμήμα Μηχανικών Πληροφοριακών
και Επικοινωνιακών Συστημάτων

© 2026
Με επιφύλαξη παντός δικαιώματος.
ΝΙΚΟΛΑΟΣ ΛΙΑΠΗΣ
Πανεπιστήμιο Αιγαίου, Εθνικό Μετσόβιο Πολυτεχνείο

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο πλαίσιο του μεταπτυχιακού προγράμματος σπουδών «MBA στην Ψηφιακή Καινοτομία και Νεοφυή Επιχειρηματικότητα» του Πανεπιστημίου Αιγαίου και του Εθνικού Μετσόβιου Πολυτεχνείου. Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, Γεράσιμο Κοντό, για την καθοδήγηση, την εμπιστοσύνη και τη διαθεσιμότητά του σε όλη τη διάρκεια της εκπόνησης.

Ευχαριστώ επίσης την οικογένειά μου και τους δικούς μου ανθρώπους για την υπομονή και τη στήριξή τους.

Περίληψη

Οι solo founders ψηφιακών νεοφυών επιχειρήσεων αντιμετωπίζουν ένα δομικό σημείο συμφόρησης: ένα και μόνο πρόσωπο δεν μπορεί να εκτελεί ταυτόχρονα ανάπτυξη λογισμικού, σχεδιασμό, marketing, πωλήσεις, υποστήριξη και λειτουργίες με τον ρυθμό και την ποιότητα που απαιτεί ο σύγχρονος ανταγωνισμός. Τα παραγωγικά (generative) μοντέλα τεχνητής νοημοσύνης καθιστούν πλέον τεχνικά εφικτή την ανάθεση λειτουργικών ρόλων σε πράκτορες λογισμικού· αυτό που δεν υπάρχει ακόμη είναι ένα μεθοδολογικό πλαίσιο που να ορίζει πώς ένας solo founder ενορχηστρώνει τέτοιους πράκτορες διατηρώντας τη στρατηγική συνοχή, την ποιότητα και τη λογοδοσία για ό,τι κάνει η επιχείρηση.

Η εργασία ακολουθεί τη μεθοδολογία Design Science Research (Hevner κ.ά., 2004· Peffers κ.ά., 2007) και κατασκευάζει ένα τεχνούργημα πέντε συνιστωσών: το Venture Brief Schema (δομημένη, μηχανικά επικυρώσιμη περιγραφή της επιχείρησης), τη Role Library (παραμετροποιήσιμοι ρόλοι με δικαιώματα απόφασης και κανόνες κλιμάκωσης), τον Crew Generator (σύνθεση πληρώματος από το Venture Brief), το Orchestration Runtime (συντονισμός, κατάσταση, εργαλεία, αρχείο εκτέλεσης) και το Human-in-the-Loop Contract (τα σημεία όπου ο founder εγκρίνει, παρακάμπτει ή ενημερώνεται, δηλωμένα ως δεδομένα και επιβαλλόμενα κατά την εκτέλεση). Το πλαίσιο υλοποιείται ως λειτουργικό πρωτότυπο, εφαρμόζεται στη μελέτη περίπτωσης Passly — ψηφιακές κάρτες πορτοφολιού (wallet passes) και AI-υποβοηθούμενο marketing για ελληνικές μικρομεσαίες επιχειρήσεις — σε περιβάλλον sandbox με συνθετικά σενάρια, και αξιολογείται με προ-καταχωρισμένο (pre-registered) εργαλείο: μια ρουμπρίκα που δεσμεύτηκε σε σύστημα ελέγχου εκδόσεων πριν από την πρώτη βαθμολογημένη εκτέλεση, με τα κριτήρια ποιότητας κειμένου να βαθμολογούνται τυφλά, σε σύγκριση με μια συνθετική βασική συνθήκη: το ίδιο μοντέλο, με τις ίδιες ρυθμίσεις, στο ίδιο brief, μέσω απλής διεπαφής συνομιλίας.

Το κεντρικό εύρημα είναι σκοπίμως στενό, και η μορφή του είχε προβλεφθεί πριν υπάρξουν δεδομένα. Στα τυφλά κριτήρια ποιότητας οι δύο συνθήκες απείχαν 5,2 ποσοστιαίες μονάδες και η βασική συνθήκη ισοφάρισε ή υπερέβη το πλαίσιο σε πέντε από τα οκτώ σενάρια: το πλαίσιο δεν έγραψε καλύτερα κείμενα. Ο διαχωρισμός βρίσκεται παντού γύρω από το κείμενο: 2,5 φορές λιγότερες αλληλεπιδράσεις του founder, 3,4 φορές μικρότερος χρόνος μοντέλου, 2,0 φορές λιγότερα tokens, παραδοτέο έτοιμο προς αποστολή σε επτά από οκτώ σενάρια έναντι τριών από οκτώ, αρχείο εκτέλεσης από το οποίο τρίτος μπορεί να ανασυγκροτήσει τι συνέβη, και πύλη έγκρισης που δηλώνεται πριν από την εκτέλεση και επιβάλλεται κατά την αποστολή. Η συμβολή, στο εύρος που τεκμηριώνεται και σε κανένα ευρύτερο: ένας solo founder μπορεί να λειτουργεί μια ψηφιακή επιχείρηση μέσω ενός πληρώματος πρακτόρων με ουσιαδώς χαμηλότερο κόστος εποπτείας, χωρίς να εκχωρεί την εξουσία απόφασης πάνω σε ό,τι φτάνει στον πελάτη. Οι περιορισμοί — ένας αξιολογητής που είναι και ο κατασκευαστής, συνθετική και όχι στρατολογημένη βασική συνθήκη, ένα σενάριο ανά συνθήκη, sandbox με συνθετικά δεδομένα — ορίζουν το σύνορο της συμβολής και καταγράφονται ως τέτοιο.

Λέξεις - Κλειδιά: τεχνητή νοημοσύνη, πολυπρακτορικά συστήματα, ενορχήστρωση, ψηφιακή επιχειρηματικότητα, solo founder, human-in-the-loop, Design Science Research

Abstract

AI-Native Management and Orchestration: A Methodological Framework for Solo-Founder Digital Entrepreneurship

Solo founders of digital startups face a structural bottleneck: one person cannot simultaneously execute engineering, design, marketing, sales, support and operations at the pace and quality modern competition demands. Generative AI agents make end-to-end delegation of operational functions technically plausible; what does not yet exist is a methodological framework for how a solo founder should orchestrate such agents while preserving strategic coherence, quality and accountability.

Following Design Science Research, this thesis constructs a five-component artefact — Venture Brief Schema, Role Library, Crew Generator, Orchestration Runtime and Human-in-the-Loop Contract — implements it as a working prototype, applies it to Passly, a one-person venture building digital wallet passes and AI-assisted marketing for Greek small businesses, in a sandbox on synthetic scenarios, and evaluates it against a pre-registered instrument: a rubric frozen in version control before the first scored run, with blinded output-quality scoring, comparing the framework against a synthesised baseline running the same model, at the same settings, on the same brief, through a generic chat interface.

The central finding is deliberately narrow, and its shape was predicted before any data existed. On the blinded quality criteria the two conditions came in 5.2 percentage points apart and the baseline matched or beat the framework on five of eight scenarios: the framework did not write better copy. The separation lies everywhere around the artefact — $2.5\times$ fewer founder interactions, $3.4\times$ less model latency, $2.0\times$ fewer tokens, a ship-ready deliverable in seven of eight scenarios against three of eight, a run record from which a third party can reconstruct what happened, and an approval gate declared as data before the run and enforced at dispatch during it. The contribution, at its evidenced width and no wider: a solo founder can operate a digital venture through a crew of agents at materially lower oversight cost without surrendering authority over what reaches a customer. The limitations — a single rater who is also the builder, a synthesised rather than recruited baseline, one run per condition per scenario, a sandbox on synthetic data — are the boundary of that contribution and are recorded as such.

Keywords: artificial intelligence, multi-agent systems, orchestration, digital entrepreneurship, solo founder, human-in-the-loop, Design Science Research

Περιεχόμενα

1. Introduction	11
1.1 The context of the thesis	11
1.2 The purpose of the thesis	12
1.3 The object of the thesis	13
1.4 Structure of the thesis	14
2. State of the Art	15
2.1 Basic concepts and definitions: what an agent is, and what changes when it is an LLM	15
2.2 Multi-agent LLM frameworks: the substrate the thesis builds on	16
2.3 Current state: orchestration frameworks and runtimes compared	17
2.4 Evaluation of multi-agent LLM systems	20
2.5 The lean-startup loop and the AI-agent crew as its execution layer	21
2.6 Digital entrepreneurship and the solo founder	22
2.7 Generative AI in entrepreneurship and per-function productivity evidence	23
2.8 Governance, the Human-in-the-Loop Contract, and the regulatory baseline	24
2.9 Conclusions of the review	26
3. Methodology	29
3.1 Research questions	29
3.2 Design Science Research as the method	29
3.3 Methodological targets and evaluation criteria	30
3.4 The steps followed	32
4. System Architecture: the Framework	33
4.1 Introduction and chapter map	34
4.2 Design objectives	35
4.3 Architecture diagrams	39
4.4 Component 1 — The Venture Brief Schema	41
4.5 Component 2 — The Role Library	43
4.6 Component 3 — The Crew Generator	46
4.7 Component 4 — The Orchestration Runtime	48
4.8 Component 5 — The Human-in-the-Loop Contract	52
4.9 Worked example — the Passly launch via the framework	56
4.10 Summary	62
5. Development Tools	63
5.1 The orchestration substrate — the construction spike	63
5.2 The framework and application stack	70
5.3 Summary	72
6. The System: the Passly Application	73
6.1 Introduction and chapter map	73
6.2 Passly as instantiation	74
6.3 Architecture	75
6.4 Pass issuance	79
6.5 The crew inside the application	83

6.6 WebHITL — the second adapter	85
6.7 The end-to-end walkthrough	88
6.8 Persistence and the member model	93
6.9 What the demonstration establishes — and what it does not	94
6.10 Summary	96
7. Evaluation and Discussion	97
7.1 Introduction and chapter map	97
7.2 Evaluation design	98
7.3 The two conditions	99
7.4 Scenarios and protocol	101
7.5 Panel A results — the scored rubric	103
7.6 Panel B results — the measured dimensions	107
7.7 The adversarial probes	110
7.8 Threats to validity, and the amendment record	111
7.9 What the evaluation supports — and what it does not	113
7.10 Summary	115
8. Conclusions and Outlook	117
8.1 Conclusions on the results	117
8.2 Conclusions on the methodology	120
8.3 Conclusions on the tools	121
8.4 Outlook	122
References	123
Appendix A — Panel A blinding and extraction: the mechanics and the record	126
A.1 The extraction record	126
A.2 Blinding and presentation	127
A.3 The identifier key	127
Appendix B — The instrument's amendment record (rubric §12, verbatim)	128
B.1 Resolutions recorded at the freeze	131
Appendix Γ — Scoring anchors of the frozen instrument (rubric §5 and §6, verbatim)	132
Γ.1 Panel A — the scored rubric (rubric §5)	132
Γ.2 Panel B — measured dimensions (rubric §6)	133
Appendix Δ — Evaluation data, code sample and repositories	135
Δ.1 Panel A — per-cell scores	135
Δ.2 Panel B — measured readings	136
Δ.3 Code sample — the primitives the thesis argues from	136
Δ.4 Repositories	138

Λίστα Σχημάτων

Figure Name	Page
Figure 4.1: Block diagram of the framework — the...	39
Figure 4.2: Sequence diagram of one Author Flow run...	40
Figure 4.3: The Venture Brief schema — required top...	42
Figure 4.4: The Author Flow graph (machine-...	49
Figure 4.5: The HITL Request envelope and the FounderResponse...	54
Figure 6.1: Component diagram of the Passly application —...	75
Figure 6.2: The member pass installed in Apple Wallet...	82
Figure 6.3: The pass designer at /design....	89
Figure 6.4: The campaign review gate at awaiting_review....	90
Figure 6.5: Enrolment at /join/[shopId]:...	91
Figure 6.6: The Members & stamps view. The...	92

Λίστα Πινάκων

Figure Name	Page
Table 2.1: Multi-agent LLM frameworks and runtimes...	18
Table 2.2: Summary of the review — what each...	26
Table 3.1: Evaluation dimensions for Chapter 7, each...	31
Table 4.1: Design objectives, their kernel references,...	35
Table 4.2: System requirements of the framework, derived...	37
Table 4.3: The DecisionRights primitive (illustrated with the...	44
Table 4.4: The role catalogue (roles_library.py...	45
Table 4.5: DEFAULT_RULES — the default crew-composition...	47
Table 4.6: The runtime’s service-substitution points.	51
Table 4.7: The run record of the reject-...	60
Table 4.8: The Crew Generator’s audit log for the...	61
Table 5.1: Pre-registered rubric.	65
Table 5.2: Spike scorecard.	66
Table 5.3: Development tools of the framework (solo...	70
Table 6.1: The deployed surfaces of the Passly application...	76
Table 7.1: The two-panel structure of the...	98
Table 7.2: The N = 10 scenario set,...	101
Table 7.3: Panel A weighted totals per scenario and...	104
Table 7.4: Panel A per-criterion means on...	104
Table 7.5: Panel B summary. Ratios are Condition...	107
Table 7.6: M1, founder interactions to a terminal...	108
Table 7.7: M2, summed model latency to terminal...	108
Table 7.8: M4, artefact readiness, read from...	109
Table 7.9: Design objectives, their instruments, and...	114

Ακρωνύμια

Abbreviation	Meaning
AI	Artificial Intelligence
API	Application Programming Interface
BDI	Belief–Desire–Intention
DSR	Design Science Research
EU	European Union
HITL	Human-in-the-Loop
JSON	JavaScript Object Notation
LLM	Large Language Model
MBA	Master of Business Administration
MVP	Minimum Viable Product
NTUA / ΕΜΠ	National Technical University of Athens / Εθνικό Μετσόβιο Πολυτεχνείο
RQ	Research Question
SMB	Small and Medium-sized Business
SMS	Short Message Service

1. Introduction

1.1 The context of the thesis

Solo founders of digital startups face a structural bottleneck: a single person cannot simultaneously execute engineering, design, marketing, sales, support and operations at the pace and quality that modern competition demands. The constraint is not talent or effort but arithmetic — six operational functions, one pair of hands, and a market that does not wait for the founder to finish the previous task. The traditional answers are to hire, which costs equity and cash a pre-revenue venture rarely has, or to sequence, which costs time the venture may not survive. Generative AI agents now make a third answer technically plausible: end-to-end delegation of operational functions to software. What does not yet exist is a methodological answer — no widely adopted framework describes how a solo founder should orchestrate such agents while preserving strategic coherence, output quality and accountability for what the venture does in the world.

That gap is managerial rather than technical, and stating it that way fixes what kind of thesis this is. The capable models exist and are commodities; the orchestration frameworks that connect them into multi-agent systems exist and are surveyed in Chapter 2. What is missing sits above both: an operating discipline that says which functions to delegate, under what supervision, with what record, and with which decisions are reserved to the founder. The significance of the problem is documented rather than asserted. Obschonka and Audretsch (2020), introducing the *Small Business Economics* special issue on artificial intelligence in entrepreneurship, describe a literature that has begun asking what AI does *to* entrepreneurs and entrepreneurship but has little to say about how an individual founder should structure a venture's operations around it. The per-function evidence that has accumulated since — controlled studies of generative AI in writing, support and analyst work (Brynjolfsson et al., 2025; Noy and Zhang, 2023; Dell'Acqua et al., 2026) — is task-level: it shows that individual functions can be accelerated, and says nothing about operating all of them at once, under one person's supervision, with accountability preserved. Between the task-level evidence and the firm-level speculation sits the operating question this thesis takes up. Its research problem is therefore stated as a main question — how a solo founder can orchestrate multiple generative AI agents to manage the end-to-end operations of an early-stage digital startup, and what methodological framework supports that operating model — refined into four research questions in Chapter 3 §3.1.

The thesis is positioned inside digital entrepreneurship, and its title commits it there: the artefact is a methodological framework *for solo-founder digital entrepreneurship*, with the agentic machinery as the substrate rather than the subject. The managerial tradition it plugs into is the lean-startup loop. Build–Measure–Learn already tells a founder what to learn next and how to decide whether to persevere or pivot; what it presumes and does not provide is the execution capacity to build and measure fast enough for the loop to spin. Blank (2013) frames the startup as a temporary organisation searching for a business model — search being a founder-judgement activity that no part of this thesis proposes to delegate. The framework's position in

that tradition is precise: it is an execution layer inside the loop the founder already runs, not a replacement for the loop and not a participant in the search.

One term in the title names the design stance rather than a technology. **AI-native** describes operations designed around agent delegation from the first day — roles, approval rights and records specified for a founder-plus-crew operating model — as distinct from AI-assisted operations, where tools are bolted onto a workflow designed for human execution. The distinction has a testable consequence that the thesis returns to repeatedly: an AI-native operating model can declare in advance which actions require the founder, enforce that declaration at the moment an action is dispatched, and leave a record a third party can reconstruct — properties a chat window bolted onto a human workflow does not have. The management half of the title is equally deliberate: what is orchestrated is not tokens but an organisation, and the founder’s job in it shifts from executing every function to directing a crew — setting scope, reviewing output at declared gates, and retaining decision rights over what reaches a customer.

The problem is deliberately bounded, and the boundary does work throughout the thesis. **The framework targets digital-only ventures: businesses whose product, distribution and operations are executable in software. Physical and hybrid business models are explicitly out of scope.** The reason is the delegation surface: an agent crew can draft a campaign, write code, answer a support ticket and read analytics, but it cannot stock a shelf or run a kitchen, and a framework evaluated on a venture with a warehouse would be evaluated on functions it cannot reach. The boundary is drawn here once and then enforced as data rather than left as a caveat: Chapter 4 makes it design objective O7, and the Venture Brief that configures a crew carries it in a machine-readable `constraints.out_of_scope` field that the roles are bound by (§4.4). Three further exclusions bound the claim. The framework addresses the solo founder, not founding teams. It does not involve fine-tuning or training models; it is an operating layer over commodity models and is designed to survive their substitution. And it stays out of regulated domains — medical, financial and legal advice — where delegation to generative systems raises obligations this thesis does not take on. A final boundary is empirical: the evaluation in Chapter 7 is a sandbox study on synthetic scenarios with a synthesised baseline, and no research data is collected from real customers.

1.2 The purpose of the thesis

The purpose of this thesis is to design, demonstrate and evaluate a methodological framework through which a solo founder can operate a digital venture by orchestrating a crew of generative AI agents while retaining decisive authority over every action that reaches a customer or cannot be undone.

Everything that follows serves that one sentence. The framework is a designed artefact in the Design Science Research sense (Hevner et al., 2004; Peffers et al., 2007) — not a new model, a new orchestration engine, or a benchmark result — and the claim made for it is deliberately narrower than a reader might expect. Stated once: **a solo founder can operate a digital venture through a crew of agents at materially lower oversight cost without surrendering authority over what reaches a customer.** The evaluation supports that sentence and no wider one; in

particular, the thesis does not claim that the framework produces better output than a founder working with the same model in a plain chat interface, and Chapter 7 reports the blinded scoring that shows the two conditions near parity on that dimension.

1.3 The object of the thesis

The purpose is pursued through six lines of work, each of which occupies one chapter of the thesis.

- **A review of the state of the art** across the four bodies of literature the framework rests on — agent foundations and multi-agent LLM frameworks, the evaluation and benchmark literature, the lean-startup and digital-entrepreneurship traditions, and the governance requirements placed on automated decision-making — together with a structured comparison of the orchestration frameworks in current use, which locates the gap the artefact fills.
- **The construction of a five-component framework.** The **Venture Brief Schema** is the structured, machine-validated description of a venture from which everything else is configured. The **Role Library** is a catalogue of parameterised agent roles, each carrying not just a prompt but decision rights and escalation rules. The **Crew Generator** selects and configures a crew for a given brief, making role composition data-driven rather than hard-coded. The **Orchestration Runtime** coordinates the crew — messaging, state, tool access, and the run record. The **Human-in-the-Loop Contract** declares, as inspectable data, where the founder must approve, override, or be notified, and is enforced at the moment a gated action is dispatched rather than by convention. Each component is derived from a design objective traced to the literature, and the architecture is specified with a requirements table and diagrams before it is described component by component.
- **An empirical selection of the orchestration substrate and the development tools.** Three candidate substrates were constructed against an identical scenario and scored under a pre-registered rubric; the winner, and the application stack built around it, are documented with the reasons each tool was chosen over its alternatives.
- **A working application of the framework to a real venture.** Passly — a one-person venture building digital wallet passes and AI-assisted marketing for Greek small businesses — consumes the framework as a dependency, issues cryptographically signed Apple Wallet passes, and implements a second adapter on the framework’s founder-authority seam. The application is deployed at <https://www.passly.gr>; the demonstration nonetheless remains a sandbox on synthetic scenarios, with one demonstration merchant and no real customers.
- **A pre-registered evaluation.** A two-panel rubric — judged criteria scored blind, measured criteria read from the run record — was frozen in version control before the first scored run and applied across ten scenarios in two conditions: the framework against a synthesised same-model baseline operated through a generic chat interface.
- **Conclusions at the evidenced width.** The research questions are answered with the evidence in hand, the limitations are set as the boundary of the contribution,

and the study that would carry the claim further than a sandbox honestly can is specified.

Two secondary contributions travel with the artefact. The first is the evaluation instrument itself: a pre-registered, two-panel rubric for comparing operating models that is re-scorable by a third party from the archived runs, and that other single-author design studies of agentic systems could reuse. The second is the demonstration corpus: a complete, committed record of a venture operated in both conditions, which is what makes the thesis's central comparison checkable rather than narrated.

1.4 Structure of the thesis

Chapter 2 reviews the state of the art: agent foundations and the multi-agent LLM frameworks, compared in a structured table; the evaluation and benchmark literature and its limits; the lean-startup tradition and the digital-entrepreneurship literature on the solo founder; the per-function productivity evidence; and the governance layer — the EU AI Act's human-oversight requirements and the agent-governance literature — that the Human-in-the-Loop Contract operationalises. Chapter 3 states the research questions, justifies Design Science Research as the method, fixes the methodological targets and evaluation criteria, and records the steps followed. Chapter 4 specifies the framework as a system architecture: seven design objectives stated as requirements, the block, flow and sequence diagrams, and the five components with the design decisions and rejected alternatives behind each. Chapter 5 documents the development tools — the construction spike that selected LangGraph over CrewAI and a custom runtime, and the application stack — with the capabilities, the reasons for choice and the use of each. Chapter 6 presents the system built on the framework, the Passly application, through its principal screens and the end-to-end walkthrough, and states what the demonstration establishes and what it does not. Chapter 7 is the evaluation and its discussion: the pre-registered instrument applied across ten scenarios in two conditions, with judged criteria scored blind and measured criteria read from the run record. Chapter 8 draws the conclusions on the results, on the methodology and on the tools, and sets out the outlook.

Four appendices support re-scoring and reuse: Appendix A documents the blinding and extraction mechanics, Appendix B reproduces the frozen instrument's amendment record verbatim, Appendix Γ reproduces its scoring anchors, and Appendix Δ carries the per-scenario evaluation data, a sample of the framework's code, and the repositories in which the framework, the application and the evaluation archive are held.

2. State of the Art

This chapter reviews the state of the art that the framework of Chapter 4 builds on, and closes with the gap it fills. §2.1 establishes the basic concepts and definitions — the working definition of an intelligent agent and the BDI vocabulary the framework uses to specify roles. §2.2 covers the multi-agent LLM frameworks that supply the orchestration substrate, and §2.3 compares them, together with the practitioner runtimes in current use, on the properties a solo-founder operating model depends on. §2.4 covers the evaluation literature and the benchmark-gaming caveat that motivates the venture-level demonstration. §2.5 covers the lean-startup loop and re-frames the agent crew as its execution layer; §2.6 locates the solo founder in the digital-entrepreneurship literature. §2.7 covers the AI-in-entrepreneurship and per-function productivity literatures that anchor per-role plausibility. §2.8 takes up governance and motivates the Human-in-the-Loop Contract elaborated in Chapter 4. §2.9 draws the conclusions of the review in a summary table and states the opening that the thesis answers.

2.1 Basic concepts and definitions: what an agent is, and what changes when it is an LLM

The thesis uses a definition of agency that is older than the current wave of generative AI. Wooldridge and Jennings (1995) distinguish two notions. Under the **weak notion** of agency, an agent is a hardware or, more usually, software-based computer system that has four properties: **autonomy** (it operates without direct intervention and has control over its actions and internal state), **social ability** (it interacts with other agents, including humans, through some agent-communication language), **reactivity** (it perceives its environment and responds in a timely fashion to changes in it) and **pro-activeness** (it exhibits goal-directed behaviour by taking the initiative, rather than simply acting in response to its environment). The weak notion has held up for thirty years because it abstracts over implementation: the agent could be a rule-based expert system, a reinforcement-learning policy, or a large language model with tools, and the four properties remain the diagnostic for whether the system is acting as an agent or as a passive function. Wooldridge and Jennings then contrast it with a **stronger notion**, in which an agent is additionally conceptualised or implemented using concepts more usually applied to humans — knowledge, belief, intention, obligation — and it is this stronger notion that supplies the vocabulary the framework of Chapter 4 uses to specify a role.

That vocabulary is the Belief–Desire–Intention model. Its philosophical source is Bratman (1987), whose account of practical reasoning holds that intentions play a distinct role that cannot be reduced to beliefs and desires: an intention is a commitment to act that persists, constrains further deliberation and resists reconsideration. Rao and Georgeff (1995) turned that account into an implementable agent architecture, integrating the formal BDI logics with an executable system, and the paper itself credits Bratman for the concept. The BDI lens is useful for this thesis because it gives a vocabulary for role specification that is richer than the prompt-as-role pattern most current systems rely on. A role in the proposed framework is more than a string of instructions. It is a structured commitment about

what the role-holder believes about the venture, what outcomes it is trying to bring about, and which intentions — in Bratman’s sense of commitments already made — it is currently executing. The implementation substrate is an LLM with tool access, but the design abstraction is BDI.

Two boundary phrases follow from these foundations and are reused in the related-fields register. First, the thesis adopts the Wooldridge–Jennings notions of agency and the Bratman–Rao–Georgeff BDI model as the vocabulary for role specification; it does not propose a new coordination protocol, agent communication language, or negotiation mechanism. Those contributions belong to classical multi-agent systems research, which the thesis cites without trying to extend. Second, the thesis treats the LLM-with-tools substrate as a given. The fact that frontier models behave reasonably under instruction is taken as a boundary condition, and the contribution sits at the orchestration and managerial layer rather than at model alignment or tool design.

2.2 Multi-agent LLM frameworks: the substrate the thesis builds on

Within the last three years, a small number of multi-agent LLM frameworks have set up the orchestration substrate this thesis builds on. CAMEL (Li et al., 2023) was the first to make role-playing between language-model agents a reusable pattern: two agents, each given a role, converse under an inception prompt to complete a task, and the framework’s role-playing module was later vendored directly into ChatDev. AutoGen (Wu et al., 2023) generalised the pattern into a programming interface — conversable agents that exchange natural-language messages around a defined task, with a human able to take one of the seats — and role-specialised agents in conversation has since become the default abstraction. The pattern has stuck because it is implementation-light, debuggable through message logs, and transferable across model providers.

ChatDev (Qian et al., 2024) instantiated the pattern as a virtual software company. The paper describes a chat chain of role pairs working through the phases of a software lifecycle; the released configuration at the version reviewed (tag v1.1.6) recruits nine roles — CEO, Counselor, CHRO, CPO, CTO, Programmer, Code Reviewer, Test Engineer and Chief Creative Officer. Two of its findings matter here and should be kept apart. Hallucination is addressed by *communicative dehallucination*, a protocol in which the receiving agent asks for specifics before acting rather than guessing; and role specialisation is ablated separately, where removing roles is the paper’s single largest effect on output quality. The second finding is the one the Role Library of Chapter 4 leans on: the cover of business functions — leadership, product, engineering, review, testing, design — is the relevant comparison with the crew this thesis applies to Passly (marketing, product, engineering, customer support, sales, finance), not the count of roles.

MetaGPT (Hong et al., 2024) made one further design move. Role prompts can encode standard operating procedures rather than free-form descriptions, and the SOPs-as-prompts approach further constrains agent behaviour and reduces error compounding across long workflows. The Role Library proposed in Chapter 4 inherits this directly: roles are stored as parameterised templates with embedded operating procedures rather than as ad-hoc prompts. Guo et al. (2024) survey the

rapidly growing literature along four aspects — the agents–environment interface, agent profiling, agent communication and agent capability acquisition — together with the domains and environments in which such systems have been applied; the present section follows the frameworks chronologically and uses the survey as its map of the field rather than as its structure.

Beyond the academic frameworks, two practitioner runtimes have become the de-facto build targets for applied work. CrewAI offers a high-level Python interface for declaring roles, tasks, and a sequential or hierarchical orchestration process. LangGraph models orchestration as a directed graph of LLM nodes with explicit shared state. Both are open source, both are documented at a level of maturity sufficient for prototype use, and both were candidates for the prototype substrate. The thesis treats their design choices as practitioner evidence rather than peer-reviewed contributions; §2.3 compares them with the research frameworks on the properties a founder depends on, and the construction spike of Chapter 5 §5.1 settled the choice between them, and a custom bare-bones runtime, empirically.

Composition — deciding which agents a task needs — has also been studied. AutoGen’s AgentBuilder (the AutoBuild contribution, at tag v0.2.34) generates a roster and each agent’s system message from a free-text building task; AutoAgents (Chen et al., 2024) has a planner agent draft, observe and refine a set of specialised agents for the task at hand; and DyLAN (Liu et al., 2024b) treats the agent set as a network whose membership is optimised against task performance. All three compose from a task description through a language-model call — non-deterministically, or against a benchmark objective — and none records why a role was included or left out.

Two observations come out of reading the cluster as a whole. The orchestration substrate has settled on a small set of coordination patterns that are mature enough to be reused as building blocks: sequential and hierarchical processes are primitives of the practitioner runtimes, and debate and peer-review patterns recur across the systems Guo et al. (2024) survey. And while composition has been made dynamic, it has not been made *accountable*: the frameworks that ship a fixed role set leave the founder to adopt or hand-edit it, and the frameworks that generate one derive it from prose by a model call, with no typed description of the venture to derive from, no inspectable rules to derive by, and no record of the derivation. That narrower opening — composition from a typed venture description, by auditable rules, with an audit log — is the one this thesis fills.

2.3 Current state: orchestration frameworks and runtimes compared

The frameworks of §2.2 are read there as a research lineage; this section reads them as candidate substrates, which is how a founder building a venture on them would have to read them. The comparison is confined to the properties that the operating model of this thesis depends on — how a role is specified, how a crew is composed, how the agents are coordinated, where a human can intervene, how state survives between turns, and what the framework was built to do — because those are the properties that determine whether a solo founder can delegate a function and still answer for it. Output quality is deliberately not a column: every framework in the table calls the same class of commodity model, and what distinguishes them is the

scaffolding around the call, not the call itself (Wang et al., 2024; Guo et al., 2024). Table 2.1 summarises the reading; the paragraphs that follow give the evidence for each row, as documented at the time of the review (late 2025). The three runtimes that were later constructed against an identical scenario in the substrate-selection spike of Chapter 5 §5.1 — CrewAI, LangGraph and a custom runtime — are marked; the spike’s scored result is not anticipated here.

Table 2.1: Multi-agent LLM frameworks and runtimes compared on the properties a solo-founder operating model depends on. Sources: the cited papers, and the projects’ repositories at the versions named (AutoGen as described in the paper; ChatDev tag v1.1.6; CrewAI 0.108.0; LangGraph 1.0), as read in late 2025.

Framework	Role specification	Crew composition	Coordination pattern	Human oversight	State across turns	Built for
AutoGen (Wu et al., 2023)	System prompt per conversable agent; optional tool bindings	Fixed by the developer in code (AutoBuild generates one from a task, §2.2)	Multi-agent conversation; group chat with a manager agent	A human-proxy agent participates in the conversation; approval is conversational, not structural	Message history of the conversation (later releases add team state save/load)	General programmable multi-agent applications
MetaGPT (Hong et al., 2024)	Role prompts encoding standard operating procedures (SOPs); structured intermediate documents	Fixed software-company role set	Assembly line: each role consumes and produces structured artefacts	None structural; the user supplies the requirement and receives the output	Shared message pool with publish-subscribe	Software development from a one-line requirement
ChatDev (Qian et al., 2024)	Nine fixed roles at v1.1.6 (CEO, Counselor, CHRO, CPO, CTO, Programmer, Code Reviewer, Test Engineer, Chief Creative Officer)	Fixed; chat chain of role pairs	Sequential phases, each a dialogue between two roles	None structural	Phase-scoped dialogue history	Full software-lifecycle tasks
CrewAI 0.108 (<i>spike candidate</i>)	Agent with role, goal and backstory text; per-agent tool list; tasks assigned to agents	Declared in code as a Crew of agents and tasks	Sequential or hierarchical process with a manager	human_input flag on a task, answered on standard input (see the dating note below)	Task outputs passed forward; optional memory add-ons	High-level, opinionated agent teams for common automations
LangGraph (<i>spike candidate</i>)	None native: nodes are functions over a typed state; roles are the developer’s abstraction	Assembled by the developer as a graph	Directed state graph with conditional edges	First-class interrupt() that pauses the graph until an external resume	Checkpointers persist the whole state; cross-process resume by thread identifier	Low-level graph execution substrate for stateful agent applications

Framework	Role specification	Crew composition	Coordination pattern	Human oversight	State across turns	Built for
Custom async runtime (<i>spike candidate</i>)	Whatever the developer builds	Whatever the developer builds	Explicit control flow in code	Whatever the developer builds	Whatever the developer builds	Baseline: zero substrate friction, full construction cost

Three observations fall out of the table, and each one is a design requirement for Chapter 4.

Roles are prose almost everywhere. In AutoGen, CrewAI and the research frameworks a role is a system prompt — in CrewAI’s case a role, a goal and a backstory string. MetaGPT’s move to SOPs-as-prompts tightens the prose but does not change its kind: what a role may decide, which tools it may touch, and when it must defer to a human are all expressed in natural language the model is trusted to respect. Wang et al. (2024) survey the agent-construction literature along four modules — profile, memory, planning and action — and the profile module they describe is, across the frameworks reviewed, a text field. The one exception is narrow and should be stated precisely: per-agent *tool allow-lists* are typed data that CrewAI, among others, enforces in the runtime rather than in the prompt. What no framework in the table carries as typed, inspectable, runtime-enforced data is the rest of a role’s authority — which actions it may take alone, which it must escalate, and to whom, and the scope within which it may decide. For a founder who must answer for what a role does, those are the properties that matter most, and their absence is the first requirement Chapter 4 states (objectives O1 and O3).

Composition is either hard-coded or unaccountable. The frameworks in the table ship, or expect the developer to write, a fixed set of roles: ChatDev’s nine-role company and MetaGPT’s software-company template are the clearest cases, and CrewAI and AutoGen leave the set open but fix it in code at build time. Where composition has been made dynamic — AutoGen’s AutoBuild, AutoAgents, DyLAN (§2.2) — the crew is derived from a free-text task by a model call, without a typed description of the venture to derive from, inspectable rules to derive by, or a record of why each role was included or withheld. Which roles a pre-launch marketing-technology venture needs is a different question from which roles a growth-stage marketplace needs, and no reviewed framework answers it from data about the venture. That is the opening that the Venture Brief Schema and the Crew Generator address (objective O4).

Human oversight was either conversational or absent — with one exception. AutoGen’s human is a proxy agent in the chat; CrewAI’s, at the version spiked, is a flag that blocks on standard input; MetaGPT and ChatDev have no structural intervention point at all, the human supplying a requirement and receiving a result. At the time of the review only LangGraph offered a gate that is structural rather than conversational: `interrupt()` pauses execution, the checkpointers persist the paused state, and the run resumes when — and from wherever — the human answers. That single property is what makes a founder gate that survives a day’s delay and a process restart constructible rather than aspirational, and it is why the comparison of Chapter

§5.1 turns on it. The comparison is dated, and the date matters: CrewAI’s 1.10 line (February 2026) added a `@human_feedback` flow decorator with branch routing, pluggable providers and checkpoint-based resume — a structural gate of the kind the table records it as lacking. The substrate decision of Chapter 5 was taken against the substrates as they stood in late 2025 and is not revisited by a later release; dating design decisions is what Design Science asks for. What the later release confirms is the direction of the argument: the property the thesis built its operating model around is the one the field subsequently added. The governance literature reviewed in §2.8 asks for exactly this kind of engineered, rather than assumed, human authority (Chan et al., 2024; Kolt, 2026).

Beyond the five frameworks, the practitioner landscape moved quickly during the period of this thesis: model vendors and cloud platforms released their own agent toolkits, and the trade press adopted the vocabulary of the “agentic enterprise”. The thesis treats that material as ambient context rather than as evidence, for the reason given in §2.5 for solo-founder grey literature — its claims are not peer-reviewed and change monthly. What the vendor toolkits share with the frameworks in Table 2.1 is instructive nonetheless: they too specify roles in prose, fix composition in code, and offer human approval, where they offer it at all, as a conversational turn. The gap the table exposes is therefore not a gap in any one product but a property of the field as it stood: capable orchestration substrates, and no operating model above them that makes the founder’s authority data.

2.4 Evaluation of multi-agent LLM systems

Evaluating an agentic system raises problems that classical NLP benchmarks were not designed to handle. The agent has to take actions over time, integrate with brittle external tools, and recover from partial failures, and the success criterion is often a downstream business outcome rather than a per-token loss. Three benchmarks now anchor the field. AgentBench (Liu et al., 2024a) evaluates large language models as agents across eight environments grouped into three categories — code (operating system, database, knowledge graph), game (digital card game, lateral-thinking puzzles, house-holding) and web (shopping, browsing) — with reasoning and decision-making assessed across all eight; for this thesis it provides a generic capability baseline against which the engineering and ops agents in the Passly crew can be located. SWE-bench (Jimenez et al., 2024) builds a coding-agent benchmark from 2,294 real GitHub issues drawn from twelve popular Python repositories and is the standard reference for engineering-agent capability. WebArena (Zhou et al., 2024) provides a realistic browser environment in which an agent has to complete tasks by clicking, typing, and reading rendered web pages, which fits the marketing and sales agents.

The cluster has a known weakness, and the thesis cannot ignore it. Benchmarks tend to be gamed once they become the dominant evaluation surface, and high scores do not guarantee that the agent will perform in a deployed setting where it is acting under partial information, integrating with brittle external systems, and aligning with the founder’s tolerance for risk. Kapoor et al. (2024) make the point for agent benchmarks specifically: the benchmarks over-weight accuracy and ignore cost, lack held-out sets so that agents overfit to them, and are frequently not reproducible — so

a leaderboard position says little about whether an agent matters in use. Their critique motivates the use of live demonstrations, on the task the system is actually for, as a complementary evaluation surface. The thesis takes the critique seriously and evaluates at the venture level rather than on public benchmarks. Chapter 6 demonstrates the framework end-to-end on Passly, and Chapter 7 compares that operating model against a synthesised solo-founder baseline — the same model, at the same settings, on the same brief, operated through a generic chat interface — under a pre-registered, two-panel rubric whose output-quality criteria are scored blind. The demonstration places the framework on the founder’s actual workflow, and the rubric ties what it finds back to the research questions defined in Chapter 1.

2.5 The lean-startup loop and the AI-agent crew as its execution layer

The thesis treats AI-native operations as the execution layer underneath an existing managerial loop rather than as a replacement for it, so the lean-startup literature is foundational rather than parallel. Ries (2011) defines a startup as an organisation built to discover a sustainable business under conditions of extreme uncertainty, and frames the unit of progress as a Build-Measure-Learn loop driven by validated learning. The book gives us five concepts the framework uses by name. The BML loop is the outer cycle: turn ideas into products, measure how customers respond, then learn whether to pivot or persevere. Validated learning is how that progress is judged, replacing the impressionistic “we shipped a lot this quarter” with a rigorous test of strategic assumptions. The Minimum Viable Product is the smallest artefact that lets the loop run end-to-end. The pivot is the structured response when validated learning falsifies one of those assumptions. Innovation accounting is the discipline that makes the loop legible to founders and stakeholders without resorting to vanity metrics, the surface numbers (raw signups, total registered users, total questions answered) that suggest motion without signalling progress.

The framework proposed in this thesis re-uses each piece. Build is what the engineering and product agents in the Passly crew do. Measure is what the customer-support and finance agents feed back, supported by the marketing and sales agents that generate the touchpoints being measured. Learn is the founder synthesising signals across the crew, and §2.8 will argue that the Human-in-the-Loop Contract is what makes that learning step legible and accountable rather than a black box. The crew is therefore the execution layer underneath the BML loop; the loop itself is unchanged.

Blank (2013, HBR) pairs the lean-startup loop with customer development and gives the practitioner-side framing the thesis cites in Chapter 1. Ries is the primary anchor of this section, and Blank is referenced in Chapter 1 §1.1 for positioning. Solo-founder grey literature (indie hackers, one-person SaaS, “billion-dollar one-person startup” essays) supplies ambient context for the thesis but is not used as evidence; its signal-to-noise is too low to anchor methodological claims.

Two boundary phrases follow. First, the thesis applies the lean-startup loop to reason about which functions an AI agent should take on at which stage; it does not propose a new theory of entrepreneurial action and does not run a quantitative study across many ventures. Second, the thesis does not attempt to formalise the loop further.

Innovation accounting in particular is treated as inherited from Ries 2011 and is not extended.

2.6 Digital entrepreneurship and the solo founder

The lean-startup loop of §2.5 is a practitioner methodology; the academic home of the venture this thesis addresses is the digital-entrepreneurship literature, and the thesis title places the artefact there. Three contributions frame the field and each bears directly on why a one-person digital venture is a coherent unit of analysis rather than an edge case.

Nambisan (2017), in *Entrepreneurship Theory and Practice*, argues that digital technologies — digital artefacts, digital platforms and digital infrastructures — change the nature of entrepreneurship itself rather than merely its tools. Two of his central claims do work here. First, digital technologies make entrepreneurial outcomes and processes *less bounded*: the product is never finished, the boundary of the venture is porous, and the process of building it is iterative rather than staged. Second, they make entrepreneurial agency *less predefined*: the set of actors who shape a venture's course widens, and the founder's role shifts from executing a plan to orchestrating a distributed set of contributions. Nambisan's orchestrating founder is, in the setting of this thesis, a founder whose distributed contributors are software agents rather than partners and platform participants — a reading his framework permits and the generative-AI literature of §2.7 has since made concrete.

Von Briel, Davidsson and Recker (2018), in the same journal, theorise digital technologies as *external enablers* of new venture creation and name six mechanisms by which they act: compression (a technology reduces the time an activity requires), conservation (it reduces the resources an activity requires), expansion, substitution, combination and generation. The mechanisms are stated for the IT-hardware sector but are general — later work in the External Enabler tradition extends the set further — and two of them name precisely what a crew of agents would have to deliver for a solo founder: less founder time per activity and fewer resources per activity. The framework of Chapter 4 is an attempt to make those two mechanisms operational and measurable, and the measured dimensions of the evaluation in Chapter 7 — founder interactions, latency, tokens, artefact readiness — are readings of compression and conservation in the sense von Briel and colleagues give the terms; substitution, software standing in for hires, is the mechanism the whole operating model presupposes.

Kraus et al. (2019), reviewing the field in the *International Journal of Entrepreneurial Behavior & Research*, set out a research agenda whose streams include the digital-entrepreneurship process and digital business models, and find very little research on the opportunities, challenges and success factors of digital entrepreneurship — a gap of volume. Reading their map together with Nambisan's and von Briel's contributions, this thesis draws a further inference of its own: the literature is stronger on describing the phenomenon than on prescribing how a digital venture should be run. That reading matches the gap Obschonka and Audretsch (2020) identify for AI specifically, and it is the gap this thesis works in: the descriptive literature establishes that digital ventures are less bounded, that their

founders orchestrate rather than execute, and that digital technologies compress and conserve — and stops short of an operating model.

Within that field the solo founder is a natural limiting case rather than an anomaly. A one-person digital venture is where Nambisan's less-predefined agency is most visible, because there is no team to distribute agency across; where von Briel's compression and conservation matter most, because the founder's time is the whole resource base; and where the absence of a prescriptive operating model is most costly, because there is no one else to improvise one. Grey literature on one-person software businesses is abundant and, as §2.5 notes, is not used as evidence here; the peer-reviewed literature above suffices to establish the unit of analysis. Two boundaries follow. The thesis does not extend digital-entrepreneurship theory; it adopts Nambisan's and von Briel's framings as kernel theories in the design-science sense (Hevner et al., 2004) and builds an artefact that instantiates them for one kind of founder. And it does not study founding teams or established firms; the claims above are read for the solo founder only, and Chapter 1 §1.1 states that scope.

2.7 Generative AI in entrepreneurship and per-function productivity evidence

Two journal-grade entries set the academic baseline for AI in entrepreneurship. Chalmers, MacKenzie & Carter (2021), in *Entrepreneurship Theory and Practice*, frame three venture-creation tasks that AI augments or replaces — idea production, selling, and scaling — alongside a fourth strand on organizational design. The paper also tempers the optimistic framing by flagging a disintermediation risk for traditional small firms when AI displaces human judgment too eagerly. Obschonka & Audretsch (2020), in *Small Business Economics*, is the editorial introducing the SBE special issue on AI and big data in entrepreneurship; it is used in Chapter 1 to motivate the research gap without committing the thesis to a specific empirical claim. Agrawal, Gans & Goldfarb's (2018) prediction-versus-judgment split, set out in *Prediction Machines*, is the conceptual hinge for the thesis: advances in AI lower the cost of prediction, which raises the value of the complementary human input — judgment about what to do with a prediction. The split maps directly onto agent-versus-human decision rights: agents are good at prediction-shaped sub-tasks (drafting copy, generating code, classifying support tickets, segmenting customers); the founder retains the judgment-shaped sub-tasks (which experiment to run next, when to pivot, when to accept commercial risk, when to override the agent's recommendation). The Human-in-the-Loop Contract, elaborated in §2.8, makes that split explicit.

The per-function productivity cluster gives the thesis the evidence base for the claim that an LLM agent can plausibly execute a given role. Three studies do most of the work. Brynjolfsson, Li & Raymond (2025, *Quarterly Journal of Economics*) report, from a staggered deployment to 5,172 customer-support agents, that a generative AI assistant raised the number of issues resolved per hour by 15 percent on average, with less-skilled and less-experienced workers improving across every productivity measure — roughly 30 percent more issues resolved per hour, and 36 percent for the lowest skill quintile — while the most skilled gained little. Noy & Zhang (2023, *Science*) ran a pre-registered online experiment with 453 college-educated professionals on occupation-specific writing tasks and found that ChatGPT access

reduced time taken by 40 percent (0.75 standard deviations) and raised output quality by 18 percent (0.45 standard deviations), with inequality across workers compressed because lower-ability writers gained more. Dell’Acqua et al. (2026, Organization Science) report a field experiment with 758 BCG consultants in which those paired with GPT-4 produced work of 32 percent higher quality on average than control on tasks inside the model’s capability frontier — completing 12 percent more tasks, 25 percent faster — and were 19 percentage points less likely to produce a correct answer on a task outside it. The “jagged frontier” is the operative finding: capability is uneven across task types, and pairing humans with AI raises performance only when the human can recognise which side of the frontier a task sits on. Park et al. (2023, UIST) sit adjacent. The Generative Agents paper supplies the memory-and-reflection primitive that long-running roles in Passly inherit; it is not productivity evidence in the labour-economics sense, but it is the architectural reference for the marketing agents that need to maintain state across days.

Across the cluster, each Passly role gets one productivity citation as the “what the agent can plausibly do” anchor and one risk citation as the limit. Brynjolfsson et al. 2025 anchors the support role; the limit is that the gains accrue mostly to less-skilled and less-experienced workers — the assistant lifts the floor rather than the ceiling — which constrains the design of the support agent’s escalation rules. Noy & Zhang 2023 anchors the marketing and sales-copy roles; the limit is that the AI substitutes for effort rather than complementing skill, which means quality compression at the high end. Dell’Acqua 2026 anchors the analyst-style reasoning embedded in the engineering role; the jagged frontier is the limit, and the framework’s response is the Human-in-the-Loop Contract that sets approval gates for tasks the founder judges off-frontier. Park 2023 anchors the long-running stateful roles; the limit is memory drift over long horizons, which §2.8 will pair with the Chan et al. visibility primitives to argue for periodic state audits.

Two boundary phrases follow. First, the per-function productivity cluster gives the thesis plausibility evidence at the level of individual roles; it does not address coordinated multi-role delegation inside a single venture, which is the orchestration claim and the contribution. Second, the thesis does not attempt to extend the AI-in-entrepreneurship literature as a theory; it sits inside the cluster and uses the existing framing to motivate the research gap.

2.8 Governance, the Human-in-the-Loop Contract, and the regulatory baseline

Both §2.5 (lean-startup substrate) and §2.7 (per-function productivity) point to the question this section takes up. Once an agent crew is doing most of the execution, what does the founder owe to anyone affected by the venture’s outputs, and how is that obligation made legible? The thesis answers with a Human-in-the-Loop Contract elaborated in Chapter 4, and §2.8 establishes the three sources the Contract draws on: the regulatory floor, the technical visibility primitives, and the legal framing of agency.

The regulatory floor is Article 14 of Regulation (EU) 2024/1689, the AI Act. Article 14 requires that high-risk AI systems be designed in such a way that they can be effectively overseen by natural persons during the period in which they are in use,

with oversight measures commensurate with the risks, level of autonomy, and context of use of the system. Article 14(4)(b) names a specific risk that the framework has to design against: the persons to whom oversight is assigned must be enabled to remain aware of the possible tendency of automatically relying or over-relying on the output of the system — the Act’s parenthetical name for which is automation bias. Two features of paragraph 4 matter for the design. It frames these as duties owed to the *deployer* — the person operating the system, which in a solo venture is the founder — rather than to the provider; and it treats awareness of over-reliance as something the system must be built to support, not something the human is assumed to bring. The Passly venture is not a high-risk AI system in the Article 6 sense, but the article supplies the principles the Contract baselines on. Founders operating in the EU will be reading Article 14 either as binding (when they handle high-risk use cases) or as best practice (when they do not), and the Contract is designed to be defensible under either reading.

Chan et al. (2024, FAccT) supply the technical visibility primitives the Contract operationalises. The paper assesses three categories of measures to increase visibility into AI agents: agent identifiers, real-time monitoring, and activity logging. Each category trades off intrusiveness against informativeness, and Chan et al. analyse how the measures apply across centralised and decentralised deployment contexts. The thesis maps each primitive to a concrete obligation in the Contract. Agent identifiers become the requirement that every Passly agent declare its role, model, and tool access in any externally-visible artefact (customer messages, support tickets, generated content). Real-time monitoring becomes the founder dashboard that surfaces high-risk actions before they execute (sending an outbound campaign, processing a refund). Activity logging becomes the audit trail kept on every agent decision, sufficient for after-the-fact review without depending on agent self-reports.

Kolt (2026, Notre Dame Law Review) supplies the legal framing. The article’s analytical core applies common-law agency doctrine and principal–agent economics to AI agents and diagnoses three problems that any principal delegating to an agent must manage: **information asymmetry** — the agent knows what it did and why, and the principal does not, unless the agent is made to disclose it; **discretionary authority** — the agent acts on the principal’s behalf across situations the principal did not individually authorise; and **loyalty** — the agent’s objectives may drift from the principal’s. Kolt argues that the classical agency-law remedies for these problems — incentive design, monitoring, and enforcement after the fact — may not hold for systems that act at unprecedented speed and scale, and proposes in their place governance organised around three pillars: inclusivity (whose interests the agent must serve), visibility (what about the agent’s behaviour the principal can observe), and liability (who answers for agent actions).

The three problems map onto the Human-in-the-Loop Contract of Chapter 4 more directly than the three pillars do, and the mapping is worth making explicit. Information asymmetry is answered by the run record: every action a role takes, and every decision the founder makes, is written to a trace the founder did not have to ask for. Discretionary authority is answered by decision rights held as data: the actions a role may take alone and the actions it must escalate are declared before the run and enforced when an action is dispatched, so the scope of the agent’s discretion is a setting the founder chose rather than a property the founder discovers. Loyalty is

answered by the Venture Brief: a role’s objectives are derived from a typed description of the venture — its voice, its constraints, what it will not do — rather than from the model’s defaults, and the founder gate is the point at which drift from those objectives is caught before it reaches a customer. The Contract is, in these terms, an agency-law instrument implemented as software. Turning to the pillars: for a solo-founder venture, the principal is the founder and the affected parties are customers, employees, regulators, and the founder herself. The Contract treats inclusivity as an explicit declaration of whose interests each Passly role serves and treats liability as the founder’s irreducible accountability for any agent output the Contract did not require her to approve. Visibility is the Chan et al. primitive set, integrated into the Contract via the dashboard and the audit log.

A short word on the agentic-organisation grey literature. McKinsey’s 2024 and 2025 essays on the “agentic enterprise” and the practitioner trade press around AI workforce (“AI as teammate”, “AI as employee”) supply ambient framing for §2.8 but are not used as evidence. They reflect a frame the thesis broadly accepts and extends to the solo-founder context, where the principal-agent asymmetries Kolt names are sharper because there is no human staff to absorb residual risk.

Two boundary phrases follow. First, the thesis adopts Article 14 as a regulatory baseline and Chan et al. as the visibility primitive set; it does not contribute to AI-safety alignment, propose new evaluations, or speculate about misalignment. Second, the Contract is prescribed (from regulatory floor plus practitioner reasoning), not derived from controlled human-subject studies. §2.8 sits in the CSCW tradition only in spirit; Chapter 4 treats the Contract as a designed artefact under DSR rather than as a testable HCI hypothesis.

2.9 Conclusions of the review

The chapter has covered five bodies of literature and one regulatory text, and the review’s findings are gathered in Table 2.2 before the opening they motivate is stated.

Table 2.2: Summary of the review — what each body of literature supplies to the framework, and the limit it leaves open.

Body of literature	Key sources	What the framework inherits	Limit left open
Agent foundations (§2.1)	Wooldridge & Jennings (1995); Bratman (1987); Rao & Georgeff (1995)	The weak and strong notions of agency; the BDI vocabulary for role specification	Classical MAS does not address LLM-based roles or founder authority
Multi-agent LLM frameworks (§2.2–2.3)	Li et al. (2023); Wu et al. (2023); Hong et al. (2024); Qian et al. (2024); Guo et al. (2024); Wang et al. (2024); Chen et al. (2024); Liu et al. (2024b)	Role-playing and conversational multi-agent abstractions; role specialisation; SOPs-as-prompts; dynamic composition from a task; a mature library of orchestration patterns; LangGraph’s structural <code>interrupt()</code>	Roles are prose; composition is hard-coded or derived from prose by a model call without a record; human oversight was conversational or absent at the time of review; escalation and approval rules are nowhere enforced data
Evaluation of agents (§2.4)	Liu et al. (2024a); Jimenez et al. (2024); Zhou et al. (2024); Kapoor et al. (2024)	A public capability scale for individual agents; the caution that benchmark scores are not deployment guarantees	Benchmarks do not evaluate a venture-level operating model; motivates the venture-level, pre-registered evaluation

Body of literature	Key sources	What the framework inherits	Limit left open
Lean startup (§2.5)	Ries (2011); Blank (2013)	Build–Measure–Learn as the outer loop; validated learning, MVP, pivot, innovation accounting	Presumes execution capacity it does not supply; silent on the agentic execution layer
Digital entrepreneurship (§2.6)	Nambisan (2017); von Briel et al. (2018); Kraus et al. (2019)	The orchestrating founder; compression and conservation as enabler mechanisms; the solo founder as a limiting case	Little research on the success factors of digital ventures (Kraus et al.); read by this thesis as descriptive rather than prescriptive — no operating model for a one-person digital venture
AI in entrepreneurship and productivity (§2.7)	Chalmers et al. (2021); Obschonka & Audretsch (2020); Agrawal et al. (2018); Brynjolfsson et al. (2025); Noy & Zhang (2023); Dell’Acqua et al. (2026); Park et al. (2023)	Per-role plausibility that an agent can execute a function; the prediction-versus-judgement split; the jagged frontier as the limit	Task-level evidence, one function at a time; nothing on coordinated multi-role delegation under one founder
Governance (§2.8)	AI Act Art. 14 (European Parliament and Council, 2024); Chan et al. (2024); Kolt (2026)	Effective human oversight as a duty on the deployer; visibility primitives (identifiers, monitoring, logging); the agency problems of information asymmetry, discretionary authority and loyalty, and the inclusivity–visibility–liability response	Principles stated without the operational artefact a founder would use to satisfy them

Read together, the literature gives the thesis a clean foundation and a clear opening. From foundational agent theory it inherits its vocabulary and design abstraction. From the multi-agent LLM frameworks it inherits an orchestration substrate mature enough to be used as a building block — and, from the comparison of §2.3, the finding that the frameworks reviewed specify roles as prose, fix composition in code or derive it from prose without a record, and — at the time of the review — treated human oversight as a conversational turn, with LangGraph’s structural interrupt the single exception. From the evaluation cluster it inherits both a capability scale and the methodological caution that a benchmark score is not a deployment guarantee. The lean-startup and digital-entrepreneurship traditions frame the managerial outer loop the crew sits inside and establish the orchestrating solo founder as a coherent unit of analysis; the per-function productivity evidence anchors the per-role plausibility claim and pairs each anchor with a known limit; and the governance layer supplies the regulatory and legal scaffolding the Human-in-the-Loop Contract operationalises.

The opening this chapter motivates is precise. Existing frameworks either ship fixed role sets demonstrated on coding or simulation benchmarks or, where they compose dynamically, derive the crew from a free-text task by a model call; none derives it from a typed, inspectable description of the venture by auditable rules with a record of the derivation, and none carries a role’s escalation and approval rules as data the runtime enforces. The per-function productivity cluster gives plausibility evidence one role at a time but says little about coordinated multi-role delegation inside a

single venture at the founder scale. The digital-entrepreneurship literature names the orchestrating founder and the enabler mechanisms and, on this thesis's reading, stops short of prescribing the operating model. And the governance literature names the principles a solo-founder venture must satisfy without prescribing the operational artefact the founder would actually use to satisfy them. Chapter 4 fills these openings together: the Venture Brief Schema describes the venture in structured form, the Role Library catalogues parameterised role templates with embedded SOPs and typed decision rights, the Crew Generator selects and configures a crew subset for a given Brief, the Orchestration Runtime is built on the substrate that the construction spike of Chapter 5 selected, and the Human-in-the-Loop Contract specifies where the founder must approve, override, or be notified — drawing the visibility primitives, the regulatory baseline, and the agency-law framing into a single artefact the founder can defend.

The thesis does not, in this chapter, claim that AI agents replace founder judgment. The literature reviewed above is consistent with the opposite reading: agents are reliable on prediction-shaped sub-tasks and unreliable in proportion to how far the task sits beyond the model's frontier, and the framework's contribution is the structured way it routes the venture's work between the two regimes. Chapter 3 fixes the method by which that artefact is built and judged; Chapter 4 turns to the artefact directly.

3. Methodology

This chapter fixes how the thesis was conducted so that the chapters that follow can be read against a stated method rather than an implied one. §3.1 states the research questions as they were agreed with the supervisor before the artefact was designed. §3.2 justifies Design Science Research as the method and maps its activities onto the chapters. §3.3 fixes the methodological targets — what was built, on how many scenarios, in how many conditions, against which criteria — and defines the five evaluation dimensions that Chapter 7 measures. §3.4 records the steps that were followed, in order, with the artefact each produced, so that the study can be repeated.

3.1 Research questions

The research questions were fixed in the scope document signed off with the supervisor in April 2026, before the artefact was designed, and are stated here as agreed.

Main RQ. How can a solo founder orchestrate multiple generative AI agents to manage the end-to-end operations of an early-stage digital startup, and what methodological framework supports this AI-native operating model?

- **RQ1.** Which organisational roles in an early-stage startup can be effectively delegated to LLM-based agents, and which require human-in-the-loop?
- **RQ2.** How does AI orchestration affect time-to-market, cost, and decision quality compared to a traditional solo-founder workflow?
- **RQ3.** What governance, risk, and accountability patterns emerge when decision-making is partially automated in this setting?
- **RQ4.** What design principles characterise a robust orchestration framework that maintains strategic coherence across agents?

The questions divide the work. RQ1 and RQ4 are design questions, answered by construction: Chapter 4’s role library, decision-rights model and composition machinery are the thesis’s position on which roles delegate and what principles a coherent orchestration framework requires, and Chapter 6 shows that position operating on a concrete venture. RQ2 is an empirical question and is answered by measurement, in Chapter 7’s comparison of the framework against a baseline workflow on founder effort, model latency, token cost and output quality. RQ3 is answered in both registers: Chapter 2 §2.8 assembles the governance obligations the literature and the EU AI Act (European Parliament and Council, 2024) place on automated decision-making, Chapter 4 §4.8 turns them into an enforceable human-in-the-loop contract, and Chapter 7 reports what that contract produced — which actions escalated, what the record shows, and what a declared approval gate did when a run tried to cross it. Chapter 8 returns to each question with the evidence in hand and answers it at the width the evidence supports.

3.2 Design Science Research as the method

The thesis follows Design Science Research (Hevner et al., 2004; Peffers et al., 2007). The choice follows from the deliverable: the contribution is a constructed

artefact — an orchestration framework and a working multi-agent prototype — that is designed, demonstrated on a venture, and evaluated against criteria. DSR is preferred over a pure qualitative case study because the deliverable is the artefact itself rather than an interpretation of a phenomenon, and over Action Research because the intended contribution is a generalisable framework rather than change effected in one organisation.

Peffer et al.'s process model gives the thesis its spine, and each activity has a chapter address. Problem identification and the objectives of a solution are Chapter 1 and the design objectives of Chapter 4 §4.2, which translate the research questions into seven testable commitments (O1–O7). Design and development is Chapter 4, which constructs the five-component artefact and records the decisions and rejected alternatives behind it. Demonstration begins in Chapter 4 §4.9 and is carried out in full in Chapter 6, where the framework operates Passly — a real one-person venture building digital wallet passes and AI-assisted marketing for Greek small businesses — in a sandbox on synthetic scenarios. Evaluation is Chapter 7. Communication is the thesis itself, together with the repositories that hold the framework, the venture application and the complete evaluation archive.

The evaluation step carries a methodological commitment worth stating here because it was made early and constrains everything downstream. Peffer et al. place the definition of evaluation objectives before artefact construction, and a single-author thesis has a particular reason to honour that ordering visibly: the author of the artefact is also its evaluator, and an evaluator who chooses criteria after seeing results can produce a favourable table indistinguishable by inspection from an honest one. The response adopted here is a pre-registered instrument — the complete rubric was frozen in version control, with its commit hash on record, before the first scored run existed. The practice is borrowed from clinical and psychological research and is presented in Chapter 7 as a borrowed analogy rather than as established DSR convention; what it buys is that the instrument's predictions and the instrument's criteria are demonstrably prior to the data. Chapter 7 sets out the design, the amendment discipline that followed the freeze, and the threats to validity that pre-registration does and does not mitigate.

3.3 Methodological targets and evaluation criteria

The research questions of §3.1 are pursued through targets that were fixed in advance and can be checked against what the later chapters deliver. The design target is a framework of five components (Chapter 4 §§4.4–4.8), each derived from one or more of seven design objectives (O1–O7, Chapter 4 §4.2) that trace to the kernel theories of Chapter 2. The substrate target is an empirical choice among at least three candidate orchestration runtimes, each constructed against the same scenario and scored under a rubric fixed before construction (Chapter 5 §5.1). The demonstration target is one real, digital-only, solo-founder venture consuming the framework as a dependency and exercising the founder-authority seam through a second, independently built adapter (Chapter 6). The evaluation target is a comparison of two operating models — the framework against a synthesised baseline running the same model at the same settings on the same brief — over ten synthetic scenarios, one run per condition per scenario, under a single role, scored by one rater against an instrument frozen in

version control before the first scored run, with the judged criteria scored blind (Chapter 7). No live customer data is collected anywhere, and no statistical inference is drawn from a design of that size; the targets are stated so that the reader can see they were met, and where they were narrow, that the narrowness was chosen.

The remainder of this section defines the evaluation criteria those targets are judged by. It was written before the evaluation was run, and its wording is preserved: the instrument it defines is the one Chapter 7 applies.

This section defines how the artefact will be evaluated; it does not perform the evaluation. The distinction is deliberate and load-bearing. Within the Design Science Research methodology, construction of a novel artefact is itself a mode of evaluation (Hevner et al., 2004, §4), and §5.1 and §4.9 have already exercised that mode — the spike scored the substrate decision, and the worked example demonstrated the running system. What remains for Chapter 7 is systematic appraisal against criteria, and the thesis’s posture for that appraisal is fixed here: a sandbox demonstration on synthetic Passly scenarios, scored by rubric against a synthesised solo-founder baseline, with no live customer data. The five dimensions below name what Chapter 7 will measure and, for each, identify the artefact already in place against which the measurement is operationalised. The boundary is explicit throughout: the instrument exists now; the measurement is Chapter 7’s.

Table 3.1: Evaluation dimensions for Chapter 7, each linked to a design objective (§4.2) and to the artefact that operationalises it.

#	Dimension	Operationalisation	Objective	Verified anchor (exists today)
E1	Functional correctness	All four Author-Flow termination paths reachable; test suite passes	O1, O2	175 tests pass; the four paths are reproduced by <code>termination_paths.py</code> and pinned in <code>test_author_flow.py</code> (Box 4.1)
E2	Output quality	Rubric (Greek voice, brief adherence, founder-feedback honouring) over N=10 scenarios	O7	The rubric method is the §5.1 spike rubric (<code>docs/spike-charter.md</code>); the §4.9 trace is one worked instance. The N=10 scoring is Chapter 7 work
E3	Founder usability	HITL trace legibility; decision-rights table readable	O1, O3	RunTrace emits (role, action, input, output, decision, timestamp) (§4.9); the DecisionRights table is §4.5 Table 4.3
E4	Reproducibility	MockLLM runs bit-identical; checkpointer enables cross-process resume	O5, O6	MockLLM is an ordered-queue stub (<code>src/solo_founder_crew/llm.py</code>), deterministic by construction; resume verified by <code>test_crew_resume.py</code>
E5	Cost envelope	Tokens per task on Haiku over a representative scenario set	O5	AnthropicLLM (the RealLLM alias) already records <code>usage.input_tokens / output_tokens</code> per call on its <code>.calls</code> list; aggregation is Chapter 7 work

Two of these dimensions are already fully grounded in passing machinery. For E1, the suite reports 175 passing tests on the date of writing, and the four termination paths of Box 4.1 are not narrated but asserted — functional correctness is operationalised as a property the test runner already checks. For E3 and E4 the instrument is likewise concrete: the trace shape that makes a run legible is the same one quoted in §4.9, the

decision-rights table is §4.5's Table 4.3, and determinism is a structural consequence of MockLLM returning `responses[call_index]` with no sampling, with cross-process resume verified by the six tests in `test_crew_resume.py`.

E2 and E5 require the opposite emphasis, and stating it is exactly what this section is for. In both cases the instrument exists today but the measurement does not. For E2, the rubric method is the same pre-registered, weighted instrument used for the §5.1 spike (lineage recorded in `solo-founder-crew/docs/adr/0001-framework-choice.md`, whose rubric outcome was Custom 69, LangGraph 67, CrewAI 32 out of a maximum of 75), and the §4.9 run is a single worked instance of voice and feedback honouring; the scoring of N=10 scenarios is not yet done. For E5, AnthropicLLM (exposed as the RealLLM alias) already captures per-call token usage, which is the raw material for per-role and per-run aggregation, but the representative-set run and the aggregation itself remain outstanding. The thesis does not claim, at this point, to have evaluated either dimension; it claims only that nothing stands between the present artefact and the Chapter 7 measurement except the measurement.

The comparison condition is a synthesised baseline: a “no framework” workflow in which a solo founder uses a generic chat interface to draft, revise, and publish the same Passly scenarios, scored by the same rubric, so that what is being compared is the operating model rather than the underlying model. The baseline is honestly labelled as *synthesised* — a reconstructed without-framework workflow, not a recruited human cohort — and this caveat is kept visible precisely because external validity is the obvious line of questioning at viva; the thesis claims a controlled comparison of operating models, not a field trial.

Equally important is what the rubric does *not* measure, because naming the exclusions is what tells an examiner what the thesis is not claiming. It does not score LLM output quality in absolute terms: the framework is substrate-independent by design (§5.1.7, §4.7.7), so scoring the model would not score the framework — the same argument the §5.1 spike rubric made when it scored ergonomics rather than output. It does not measure long-run reliability, which is out of scope for a sandbox demonstration. And it does not measure multi-venture or multi-flow operation; the unit of evaluation is one venture running the Author Flow. These exclusions are not gaps to be apologised for but the deliberate boundary of the claim.

Taken together, the five dimensions close the loop opened in §4.2: every design objective Chapter 4 sets itself is measurable. Functional correctness (E1) covers O1 and O2; output quality (E2) covers O7; founder usability (E3) covers O1 and O3; reproducibility (E4) covers O5 and O6; and the cost envelope (E5) covers O5. No objective is left without a metric, and no metric is untethered from an objective — which is what ties the architecture of Chapter 4 to the evaluation of Chapter 7 through this chapter.

3.4 The steps followed

Design Science Research asks that the process be transparent enough for a reader to repeat it (Hevner et al., 2004; Peffers et al., 2007). The steps below were followed in the order given; each names the artefact it produced and where that artefact is held, so that every later chapter can be traced back to the step that produced its evidence.

1. **Problem and scope (April 2026).** The research questions of §3.1 and the digital-only scope of Chapter 1 §1.1 were fixed in a scope document agreed with the supervisor. The thesis title was locked on 24 May 2026, committing the work to the entrepreneurship framing and to the framework-as-methodology framing.
2. **Literature review.** The four bodies of literature and the regulatory text of Chapter 2 were reviewed against a register of verified sources; every source cited in the thesis was verified against its publisher record before the review was consolidated.
3. **Design objectives.** The seven objectives O1–O7 of Chapter 4 §4.2 were derived from the review, each bound to a kernel theory and to the component that would realise it, before any component was built.
4. **Substrate-selection spike.** Three candidate runtimes — a custom asyncio implementation, CrewAI and LangGraph — were each constructed against the Passly launch scenario, scored under a seven-criterion rubric fixed before construction, and smoke-tested against a real model. The scorecard, the three implementations and the decision record are held in the `solo-founder-crew` repository (Chapter 5 §5.1).
5. **Construction of the framework.** The five components were implemented in `solo-founder-crew`, with the Venture Brief and the HITL request specified as JSON Schemas that are the single source of truth for both prose and code, and with a test suite pinning the four termination paths of the Author Flow (Chapter 4).
6. **Demonstration.** Passly was built in a separate repository as a consumer of the framework, with a second adapter on the Human-in-the-Loop Contract, walked end-to-end from an empty database to a stamped pass in a customer’s Wallet, and deployed at <https://www.passly.gr>; the demonstration remains a sandbox on synthetic scenarios with one demonstration merchant (Chapter 6).
7. **Instrument freeze.** The complete evaluation instrument — dimensions, anchors, weights, scenario set, comparison condition, scoring procedure and expected threats — was committed unchanged on 17 August 2026 and tagged, three days before the first scored run. Every later change to it is an amendment recorded with a date and a reason (Chapter 7 §7.8; Appendix B).
8. **Evaluation runs and scoring.** Ten scenarios were run once each in both conditions against live model calls; the judged criteria were scored blind after an extraction rule removed condition-identifying features from the artefacts, and the measured criteria were read from the run records. The runs, scoring sheets and scorecard are archived in the framework repository under `evaluation/archive/` and reproduced in part in Appendix Δ (Chapter 7).
9. **Communication.** The thesis was written against the finished artefacts, with every figure traceable to a named file, and the chapters were assembled to the programme’s template.

A reader repeating the study would need the two repositories, the frozen instrument and a model API key; the deterministic stub model reproduces every framework-side run bit-for-bit without one, and the archived live runs can be re-scored from Appendix Γ’s anchors alone.

4. System Architecture: the Framework

4.1 Introduction and chapter map

A solo founder of a digital venture faces a structural asymmetry: the work of a company — marketing, product, support, sales, finance, engineering — must be done by one person whose scarcest resource is attention, yet the decisions that matter most (what to ship, what to say to a customer, what cannot be undone) are precisely the ones that person cannot responsibly delegate. Generative models promise to absorb much of the production work, but a chat window is not an operating model: it has no notion of which venture it serves, which role is acting, what authority that role holds, or when a human must decide. The problem this chapter addresses is therefore entrepreneurial before it is technical — how a single founder can run a venture at the throughput of a team while remaining, demonstrably, the one in command.

This chapter presents the artefact that answers that problem as a system architecture: a five-component methodological framework for solo-founder digital entrepreneurship, of which a working code base (*solo-founder-crew*; Appendix Δ) is one faithful instantiation. The chapter delivers the framework — its requirements, its diagrams, its five components, and a worked demonstration. It does not deliver the judgement of it: the choice of substrate and tools is reported in Chapter 5, the in-depth application to the Passly venture is Chapter 6, and the systematic evaluation is Chapter 7. The agent architecture that appears throughout is the means by which the framework operates, not the thesis it argues; the thesis is the operating model and the founder authority it preserves.

Within the Design Science Research methodology adopted for the thesis (Hevner et al., 2004; Peffers et al., 2007), this chapter is the design-and-development step. Problem identification was established in Chapter 1 and the research questions in Chapter 3; the kernel theories — Build–Measure–Learn (Ries, 2011; Blank, 2013), multi-agent systems, digital entrepreneurship (Nambisan, 2017; von Briel et al., 2018), and the human-oversight requirements of the AI Act (European Parliament and Council, 2024, Art. 14; Kolt, 2026; Chan et al., 2024) — were surveyed in Chapter 2; demonstration begins here at §4.9 and continues in Chapter 6; evaluation follows in Chapter 7. The framework sits inside the Build–Measure–Learn loop the venture already runs, supplying the agentic execution layer that the lean-startup tradition presumes but does not provide.

The chapter proceeds as follows. §4.2 states the seven design objectives as system requirements, ordered so that entrepreneurial concerns lead and infrastructural ones follow, and traces each to a kernel theory and to the component that realises it. §4.3 presents the architecture in diagrams — the block diagram of the five components and the sequence diagram of the founder gate — and points to the flow graph that the runtime itself renders. §4.4 through §4.8 present the five components in turn — the Venture Brief Schema, the Role Library, the Crew Generator, the Orchestration Runtime, and the Human-in-the-Loop Contract. §4.9 exhibits the assembled framework end-to-end on the Passly launch, and §4.10 closes by restating the contribution as a methodology.

4.2 Design objectives

The framework presented in this chapter is a response to a gap identified in Chapter 2. The multi-agent systems literature has produced capable frameworks for orchestrating teams of agents — AutoGen, MetaGPT, ChatDev and similar (Wu et al., 2023; Hong et al., 2024; Qian et al., 2024; surveyed in Guo et al., 2024) — but these target software-engineering teams and benchmark tasks, not the operating model of a single human founder who must retain authority over a venture. The human-oversight literature, in turn, states requirements — that high-risk AI systems remain under meaningful human control (European Parliament and Council, 2024, Art. 14; Kolt, 2026; Chan et al., 2024) — without supplying a concrete operating framework that realises them for entrepreneurship. And the lean-startup tradition supplies the outer loop the venture runs in, Build–Measure–Learn (Ries, 2011; Blank, 2013), but not the agentic layer that executes inside it. This section states, as testable objectives, what closing that gap requires.

Within the Design Science Research methodology adopted for this thesis (Hevner et al., 2004; Peffers et al., 2007), this chapter is the design-and-development step, and its objectives are derived rather than asserted: each binds a kernel theory or requirement from Chapter 2 to a design constraint the framework must satisfy, and to the component that satisfies it. Table 4.1 states the seven objectives; the paragraphs that follow expand each in turn. The ordering is deliberate — the entrepreneurial objectives lead and the infrastructural ones follow — reflecting the thesis’s framing of the artefact as a methodology for solo-founder digital entrepreneurship rather than as a multi-agent system.

Table 4.1: Design objectives, their kernel references, and the components that realise them.

#	Objective	Kernel reference	Realised by
O1	Founder retains decisive authority over irreversible and customer-facing actions	Ries (2011); AI Act Art. 14 (European Parliament and Council, 2024); Kolt (2026)	HITL Contract (§4.8); <code>DecisionRights.must_escalate</code> (§4.5)
O2	Compress time-to-market: one call yields a founder-approved, publishable artefact	Ries (2011); Blank (2013)	Author Flow, <code>crew.author_flow</code> (§4.7)
O3	Roles are typed and parameterised, not bespoke per venture	Wooldridge & Jennings (1995); Rao & Georgeff (1995)	Role Library (§4.5)
O4	Crew composition and the distribution of authority are transparent and auditable	Hevner et al. (2004); Chan et al. (2024)	Crew Generator audit log (§4.6); <code>DecisionRights / HITLRequest</code> as data (§4.5, §4.8)
O5	Substrate is substitutable; the framework is portable	Hevner et al. (2004)	Protocol-typed services (§4.7.7)
O6	Reproducible in a sandbox; pause and resume across days	Peffers et al. (2007); AI Act Art. 14 (European Parliament and Council, 2024)	<code>MockLLM + checkpoint + interrupt()</code> (§5.1, §4.7, §4.8)
O7	Scoped to digital-only ventures	Scope discipline (Ch.1 §1.1)	Venture Brief <code>constraints.out_of_scope</code> (§4.4)

The first four objectives are entrepreneurial before they are technical. **O1** is the load-bearing one: the founder must retain decisive authority over the actions that are irreversible or reach a customer. The lean-startup tradition locates pivot-or-persevere authority with the founder (Ries, 2011), and the human-oversight literature requires that such authority be effective rather than nominal (European Parliament and Council, 2024, Art. 14; Kolt, 2026); the framework realises it through the HITL Contract (§4.8) and the `must_escalate` tuples carried as data on each role (§4.5). **O2** concerns time-to-market: a solo founder’s scarcest resource is attention, so the framework must turn a brief into a founder-approved, publishable artefact in a single call rather than a multi-step orchestration the founder must supervise — the Author Flow (§4.7) is that mechanism, sitting inside the Build–Measure–Learn loop the venture already runs (Ries, 2011; Blank, 2013). **O3** requires roles to be typed and parameterised rather than hand-built per venture, drawing on the tradition of treating agents as typed entities with explicit goals and capabilities (Wooldridge & Jennings, 1995; Rao & Georgeff, 1995); the Role Library (§4.5) is the realisation. **O4** demands that both crew composition and the distribution of authority be transparent and inspectable — viva-defensible and founder-legible (Hevner et al., 2004; Chan et al., 2024) — realised by the Crew Generator’s audit log (§4.6) and by holding decision rights and the decision envelope as data rather than prose (§4.5, §4.8).

The remaining objectives are infrastructural, but each is a precondition for evaluating the entrepreneurial claims in Chapter 7. **O5** requires the orchestration substrate to be substitutable, so the framework’s contribution — its abstractions — is portable rather than welded to one vendor (Hevner et al., 2004); the Protocol-typed services of §4.7.7 provide this. **O6** requires the framework to be reproducible in a sandbox and able to pause and resume across days, since a founder responds to a review when they can, not when the terminal is open; deterministic `MockLLM` runs, a checkpoint, and `LangGraph`’s `interrupt()` satisfy it (§5.1, §4.7, §4.8), and the cross-session property is itself a corollary of effective human oversight (European Parliament and Council, 2024, Art. 14). **O7** states the scope explicitly: the framework targets digital-only ventures, and this discipline is enforced as data in the Venture Brief’s `constraints.out_of_scope` (§4.4) rather than left as an informal caveat, consistent with the boundary drawn in Chapter 1.

These seven objectives are the criteria against which the artefact is judged. The components described in §§4.4–4.8 are the mechanisms that meet them; the framework-selection spike of the next section (§5.1) is what determined the substrate on which those mechanisms are built; the worked example of §4.9 exhibits them end-to-end; and §3.3 operationalises them into the evaluation metrics Chapter 7 applies.

4.2.1 System requirements derived from the objectives

The seven objectives are commitments; Table 4.2 restates them as system requirements — the concrete, checkable properties the implementation must have — in the form the design phase worked from. Each requirement carries its priority, the objective it derives from, the requirement(s) it depends on, the component that realises it, and the evidence by which it is verified. All requirements are at version 1.0, the version of the framework that was demonstrated (Chapter 6) and evaluated

(Chapter 7); no requirement was added or relaxed after the evaluation instrument was frozen.

Table 4.2: System requirements of the framework, derived from the design objectives.

ID	Requirement	Description	Priority	From	Depends on	Realised by	Verified by
R1	Declared decision rights	Every role carries, as typed data, the actions it may perform, the tools it may use and the actions it must escalate to the founder	Must	O1, O3	—	Role, DecisionRights (§4.5.1)	Role catalogue tests; Table 4.3
R2	Enforcement at dispatch	The runtime refuses to dispatch an action the acting role is not permitted, independently of what the model emits	Must	O1	R1	role.may_perform() checks in the draft/revise nodes and the Crew pre-flight (§4.7.4)	PermissionError raised from the runtime; test suite
R3	Founder gate before publication	No artefact reaches a publish tool without an explicit founder decision: approve, reject with feedback, or kill	Must	O1	R1, R2	HITL Contract, hit1 node (§4.7.5, §4.8)	Four termination paths pinned in test_author_flow.py (Box 4.1)
R4	Typed decision envelope	The founder's decision request is data — request id, thread id, venture, role, action, artefact, options — renderable on any surface without reaching into the runtime	Must	O1, O4	R3	HITLRequest / FounderResponse JSON Schema (§4.8.3)	Schema validation; two adapters (Discord, §4.8.6; web, Chapter 6 §6.6)
R5	Pause and resume across processes	A run parked at the gate survives an unbounded wait and a process restart, keyed by thread identifier	Must	O6	R3	LangGraph interrupt() + checkpoint; state-as-data separation (§4.7.3, §4.8.4)	test_crew_resume.py (six tests)
R6	Bounded revision	A flow that does not converge terminates in a named, observable state rather than looping	Must	O2, O6	R3	max_revisions budget and exhausted terminal (§4.7.5, §4.9.3)	Termination-path tests (Box 4.1)
R7	One call to a publishable artefact	A brief and a task description yield a founder-approved, published artefact through a single awaitable call	Must	O2	R3, R6	Crew.author_flow() (§4.7.2)	Listing 4.1; examples/passly_launch.py
R8	Machine-validated	The venture is described in a JSON-Schema-valid	Must	O3, O7	—	Venture Brief Schema (§4.4)	Schema validation on load;

ID	Requirement	Description	Priority	From	Depends on	Realised by	Verified by
	venture description	ated document that every component reads from, including a machine-readable out-of-scope list					passly_brief.json
R9	Data-driven crew composition with an audit log	The crew for a venture is selected by inspectable rules from the brief, and every withheld role is recorded with its reason	Should	O4	R8	Crew Generator, DEFAULT_RULES (§4.6)	Six unit-tested composition cases; audit log (§4.9.4)
R10	Machine-readable run record	Every run emits an ordered trace of (role, action, input, output, decision, timestamp) events, including founder reviews, serialisable to JSON	Must	O4	R2, R3	RunTrace (§4.7.6)	Trace files in the evaluation archive; Appendix Δ
R11	Substitutable services	The model client, the founder gate, the tool registry, the checkpoint and the role set are each behind a narrow Protocol and can be swapped without touching the graph	Must	O5	—	Service-substitution points (§4.7.7)	Table 4.6; fakes used in tests and in CI of the consuming application
R12	Deterministic sandbox operation	The whole framework runs end-to-end, reproducibly, with no model key and no network, via a deterministic stub model	Must	O5, O6	R11	MockLLM (§4.7.7)	Bit-identical stub runs; the test suite runs without credentials
R13	Scope enforced as data	The digital-only boundary travels from the brief into every role's context and is available to the gate	Should	O7	R8	constraints.out_of_scope in prompt construction (§4.4, §4.5.3)	Scenario S8 in the evaluation (Chapter 7 §7.7.1)

Two requirements are marked *Should* rather than *Must*, and the distinction is honest rather than cosmetic. R9, data-driven composition, is fully built and unit-tested but is not exercised end-to-end by either the demonstration or the evaluation, a gap Chapter 8 §8.1.3 records. R13, scope as data, is realised and was probed by one adversarial scenario, but its enforcement relies on the role honouring its context and on the founder gate, not on a hard runtime check — which is the correct place for it, since whether a request is in scope is a judgement the framework routes to the founder rather than decides.

4.3 Architecture diagrams

Three diagrams specify the architecture before the components are described one by one. The block diagram fixes what the five components are and how data flows between them; the sequence diagram fixes the one interaction that the whole framework exists to make safe — the founder gate; and the flow graph of the runtime, which appears in §4.7.1 as Figure 4.4, is not drawn by hand at all but rendered by the framework from its own compiled graph. Each diagram is followed by the vocabulary a reader needs to trace it into the component sections.

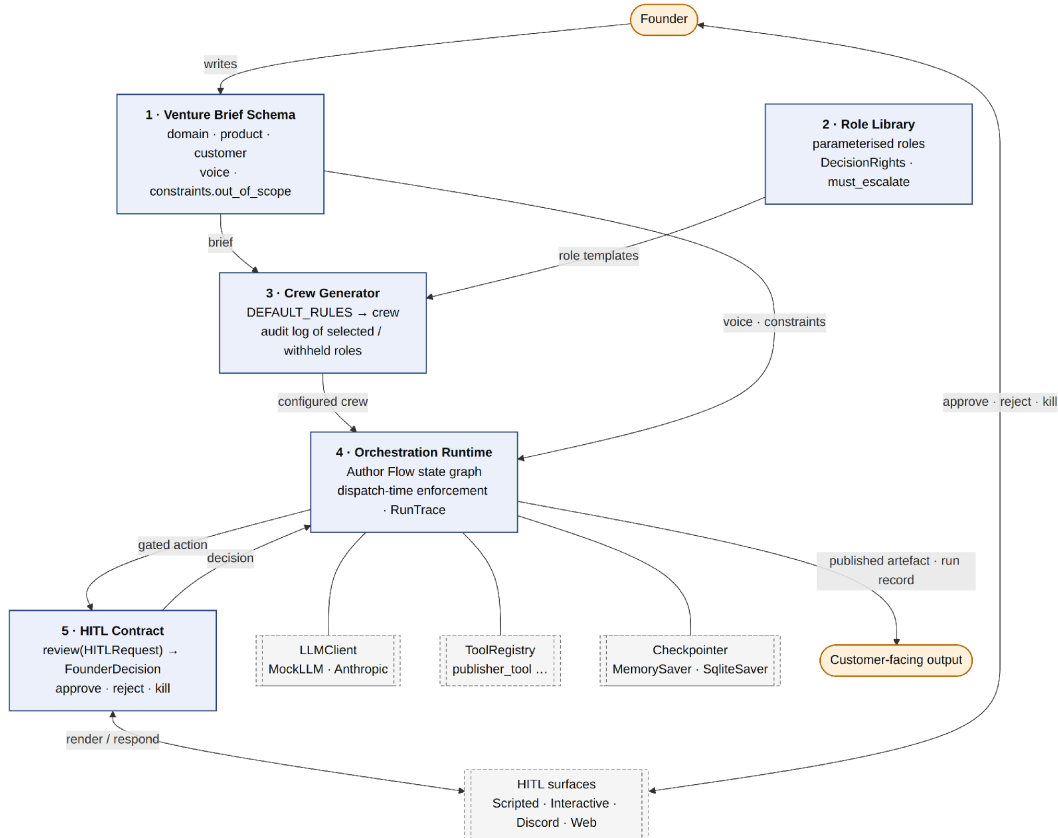


Figure 4.1: Block diagram of the framework — the five components, the founder, the substitutable runtime services (dashed) and the surfaces through which the founder gate is rendered.

The block diagram reads from the founder downwards. The founder writes a **Venture Brief** (component 1, §4.4): a schema-validated document carrying the venture’s domain, product, customer, voice and constraints, including the machine-readable `out_of_scope` list. The **Role Library** (component 2, §4.5) holds parameterised role templates whose `DecisionRights` name what each role may do, which tools it may touch and which actions it must escalate. The **Crew Generator** (component 3, §4.6) reads the brief, applies its rules to the library and emits a configured crew together with an audit log that records every role it selected and every role it withheld, with the reason. The **Orchestration Runtime** (component 4, §4.7) executes the Author Flow as a state graph over the configured crew, enforces each role’s decision rights at the moment an action is dispatched, and appends every event to the run trace; it is bound to three services — the model client, the tool registry and the checkpointer — through narrow Protocols, each of which has a deterministic default and a production substitute. When the runtime reaches a gated action it hands a typed request to the

Human-in-the-Loop Contract (component 5, §4.8), which is rendered on whichever surface the founder inhabits — a test fixture, a terminal, Discord, or the web application of Chapter 6 — and returns the founder’s decision. Only an approved artefact leaves the runtime for the customer, and it leaves together with the record of how it got there.

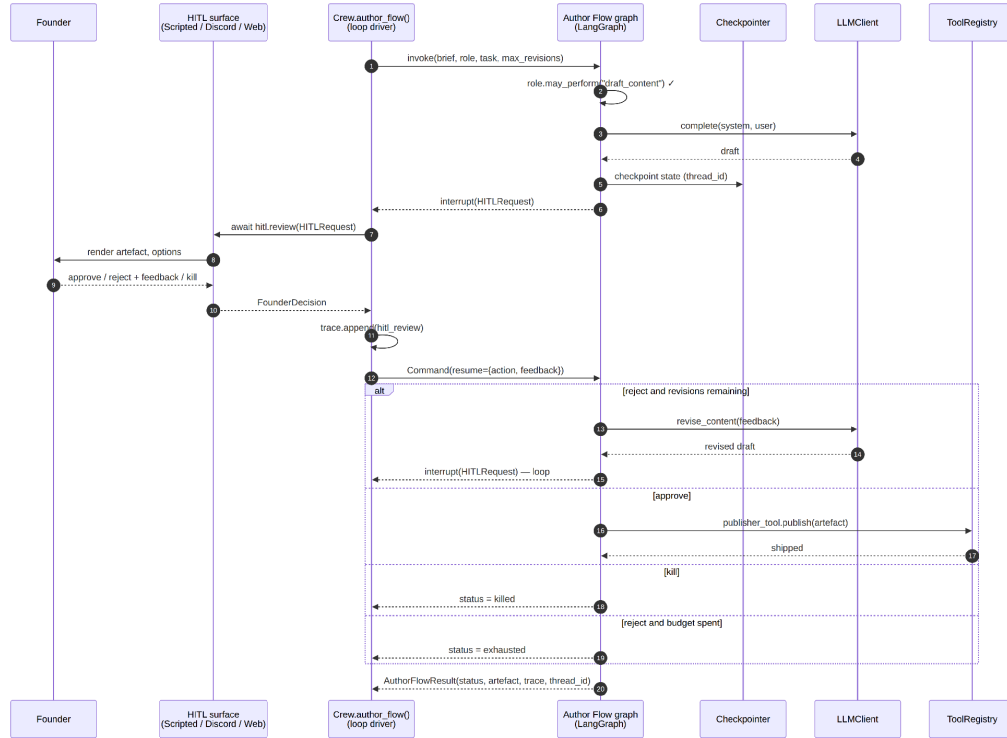


Figure 4.2: Sequence diagram of one Author Flow run through the founder gate — the six-step interrupt loop of §4.8.4, with the four terminal paths of Box 4.1 as alternatives.

The sequence diagram follows a single run. The loop driver, `Crew.author_flow()`, invokes the graph with the brief, the acting role, the task and the revision budget (1). Before any model call, the draft node checks the role’s decision rights (2) — a check the runtime performs, not the model — and only then asks the model client for a draft (3–4). At the gate node the graph checkpoints its state under the run’s thread identifier (5) and interrupts, handing a `HITLRequest` back to the driver (6). The driver awaits the contract (7), which renders the artefact and the founder’s options on its surface (8) and returns the founder’s decision (9–10); the driver records the review on the trace (11) and resumes the graph with the decision (12). The conditional edges out of the gate then select one of four paths: a rejection with budget remaining routes to a revision and back to the gate (13–15); an approval dispatches the publish tool (16–17); a kill ends the run unpublished (18); a rejection with the budget spent ends it in the named exhausted state (19). Whatever the path, the driver receives a result carrying the status, the artefact, the trace and the thread identifier (20). The pause between steps 6 and 12 is unbounded, and because the state was checkpointed at step 5 it survives a process restart — the property that objective O6 requires and that the substrate spike of Chapter 5 §5.1 selected LangGraph for.

The third diagram, the Author Flow graph itself, is deferred to §4.7.1 where the runtime is described, because its evidential value lies in how it is produced: it is

emitted by `compiled.get_graph().draw_mermaid()` from the running code and cannot drift from it. Together the three satisfy the architecture-specification requirement of this chapter — structure (block), behaviour (sequence) and control flow (graph) — before any component is opened.

4.4 Component 1 — The Venture Brief Schema

The framework's first component is the smallest and the most foundational: the Venture Brief, the structured description of the venture that every other component reads from. It is defined not as a prose document or a free-form prompt but as a JSON Schema - `/src/solo_founder_crew/schemas/venture_brief.schema.json` - and that choice is deliberate. The schema is the single source of truth shared between this chapter and the code: the fields named in the prose are the fields the framework validates, loads, and feeds to the roles. Treating the venture description as a typed, shared artefact rather than as tacit context is the design-science norm of an inspectable, common vocabulary (Hevner et al., 2004).

4.4.1 *Why a schema, not a description*

A free-form description of a venture is easy for a founder to write and impossible for a framework to depend on. A schema makes the input contract explicit and machine-checkable: a brief either validates or it does not, and the set of things the crew can know about the venture is bounded and named in advance. Three consequences follow. **Shared vocabulary**: the field names declared here — `voice.tone`, `constraints.out_of_scope`, `stage` — reappear verbatim in §4.5's prompt-assembly helpers and §4.6's composition predicate, so prose and code cannot quietly diverge. **Declarative scope discipline**: the schema sets `additionalProperties: false` at the top level, so a brief cannot smuggle in unrecognised fields; the venture is describable only in terms the framework recognises, and `constraints.out_of_scope` is where the digital-only scope discipline of Ch.1 §1.1 becomes data the roles must respect. **Validation at the boundary**: `VentureBrief.from_file(...)` validates the input against the schema before any role runs, so malformed input fails fast at load rather than surfacing later as a confusing model output. The schema's required top-level keys and their structure are shown in Figure 4.3.

Venture Brief — input contract (Component 1)

schemas/venture_brief.schema.json · JSON Schema 2020-12 · additionalProperties: false

venture_id	string	required
name	string	required
stage	enum	optional pre-launch launched growth pivoting
domain	object	required
industry	string	required
geography	array<string>	required
language	array<string>	optional
product	object	required
one_liner	string	required
value_props	array<string>	required
channels	array<string>	optional
customer	object	required
segment	string	required
jobs to be done	array<string>	optional
objections	array<string>	optional
voice	object	required
tone	string	required
do	array<string>	optional
dont	array<string>	optional
constraints	object	required
budget eur per month	number	optional
out_of_scope	array<string>	optional
compliance notes	array<string>	optional

Bordered/filled rows are required. Nested rows show each object's fields; requiredness is per parent object.

Figure 4.3: The Venture Brief schema — required top-level keys and their structure (venture_brief.schema.json).

VentureBrief required: venture_id, name, domain, product, customer, voice, constraints venture_id string (kebab-case) – stable identity; keys the run trace & HITL requests (§4.7) name string `stage` enum: pre-launch \	launched \	growth \	pivoting — drives crew composition (§4.6) domain { industry, geography[] (ISO-3166), language[] (BCP-47) } product { one_liner (≤200), value_props[], channels[] } customer { segment, jobs_to_be_done[], objections[] } voice { tone, do[], dont[] } — assembled into every role’s prompt (§4.5.3) constraints { budget_eur_per_month, out_of_scope[], compliance_notes[] } — non-negotiable`
--	------------	----------	--

4.4.2 The fields

The identity fields anchor the venture in the runtime. `venture_id` is a stable kebab-case string that threads through the system — it keys the run trace and every `HITLRequest` (§4.7) — and `stage` is an enumeration (`pre-launch`, `launched`, `growth`, `pivoting`) that is the single field the Crew Generator’s default rules predicate on (§4.6). The descriptive blocks — `domain` (`industry`, `geography`, `language`), `product` (a one-liner, `value propositions`, `channels`), and `customer` (`segment`, `jobs-to-be-done`, `objections`) — situate what is sold and to whom; the runtime serialises them into the authoritative brief block it hands the model as context (§4.7.1). Two blocks, however, do more than describe: they steer.

voice is the most consequential field for output. Its tone string and its do/don't lists are rendered by §4.5.3's `_voice_block` into every role's system prompt; it is therefore the lever behind the register differences documented in §5.1.8. Passly's brief sets a tone of "warm, plain Greek, confident without hype," with do entries such as "speak as a peer, not a salesperson" and "lead with the outcome, not the technology," and don't entries that forbid hype words, specific revenue claims, and — because the venture is pre-launch — any mention of existing paying customers.

constraints carries the venture’s non-negotiable guardrails: a monthly budget, an `out_of_scope` list, and `compliance_notes`. These are rendered by `_constraints_block` into every prompt under a “CONSTRAINTS (non-negotiable)” heading, not offered as suggestions. For Passly, `out_of_scope` encodes the thesis’s digital-only discipline as data (“physical / hybrid retail integrations beyond passes,” “markets outside Greece for the initial launch”), while `compliance_notes` carries the GDPR posture and the pre-launch honesty rule. These are the same constraints visible in the founder feedback of §5.1.8 and in the marketing role’s operating rules (§4.5.3) — the brief is where they enter the system, once, as data.

4.4.3 The *VentureBrief* wrapper

The implementation is thin by design. `VentureBrief` (`solo-founder-crew/src/solo_founder_crew/brief.py`) is a small read-only wrapper over the validated dictionary, not a fully-typed dataclass. It adds dotted access — `brief.voice.tone` instead of `brief["voice"]["tone"]` — via a `_DotAccess` base for prose-friendly code, and exposes `venture_id` and `name` as properties, but otherwise keeps the dictionary as its representation. The reason is stated in the module itself: keeping the brief a dict leaves the schema authoritative — adding a field there is enough to make it readable through `brief["new_field"]` with no change to the loader. Construction is `VentureBrief.from_file(path)` (which validates) or `VentureBrief(data, validated=False)` (which skips validation, for tests); `as_dict()` returns the underlying mapping, which is exactly what the Crew serialises into the run state (§4.7.3). The brief therefore travels through the runtime as plain data, consistent with the state/services separation.

4.4.4 Extensibility

Because the schema is the authority and the wrapper is thin, the brief extends by editing JSON, not Python. Adding a field — say a `brand.assets` block — is a schema edit; it becomes loadable and readable immediately, and a role factory (§4.5) or composition rule (§4.6) can begin consuming it without any change to `brief.py`. This mirrors the open-for-extension posture of the Role Library (§4.5.4) and the Crew Generator (§4.6.6): the framework’s components are extended by adding declarative data, not by modifying engine code — the property that lets one schema serve as the contract between the prose of this chapter and the code that implements it.

4.5 Component 2 — The Role Library

The Role Library is the framework’s catalogue of the agents a venture may employ. Where the Venture Brief (§4.4) describes the venture, the Role Library describes who works on it. Its defining design commitment is that a role’s authority is *data*, not *prose*: the actions an agent may take on its own and the actions it must escalate to the founder are declared as typed fields on the role, not left implicit in the wording of a system prompt. This is the property that makes founder authority auditable (§4.8) and that the Crew Generator (§4.6) reasons over when it composes a crew. The catalogue draws on the long-standing treatment of agents as typed entities with explicit goals and capabilities (Wooldridge & Jennings, 1995) and on the design-science norm that an artefact’s vocabulary should be shared and inspectable rather than tacit (Hevner et al., 2004). The primitives live in

solo-founder-crew/src/solo_founder_crew/role.py and the catalogue of factories in solo-founder-crew/src/solo_founder_crew/roles_library.py.

4.5.1 The role primitive

A Role is a frozen dataclass binding five fields: a name, a one-line goal, a system_prompt (the LLM-facing voice), a decision_rights value, and a tools allow-list naming the tools the role may invoke. It is worth noting what the Role does *not* hold: the Venture Brief itself. A role is brief-bound but role-immutable — its system prompt embeds the venture’s voice and constraints, yet the binding is performed by the library’s factories (§4.5.3), leaving the Role a self-contained, immutable value. Immutability (frozen=True) is deliberate on both counts: a single Role instance can be handed to many runs without state leakage, and a role’s authority cannot drift mid-run — which is what lets the run trace (§4.7) and the Human-in-the-Loop gate (§4.8) make stable guarantees about who could have done what, and when.

Authority itself is carried by DecisionRights, a frozen pair of tuples. can is the tuple of action names the role may perform autonomously; must_escalate is the tuple of actions that require an explicit founder decision before they take effect. A helper, may_perform(action), answers the dispatch-time question the runtime asks before every step (§4.7.4); an action named in neither tuple is simply unavailable to the role. The Engineering role makes the asymmetry concrete, and Table 4.3 uses it as the worked example.

Table 4.3: The DecisionRights primitive (illustrated with the Engineering role).

Field	Semantics	Example (the Engineering role)
can	Actions the role may take on its own, without founder sign-off — reversible, low-stakes work.	open_pull_request — branch and open a PR; touches nothing on main
must_escalate	Actions that require an explicit founder decision before they take effect — the hooks the HITL Contract (§4.8) gates on; irreversible or consequential.	merge_to_main — ship to production

The split is the clearest illustration of the distinction the framework rests on. Opening a pull request is reversible and touches nothing in production, so open_pull_request sits in can and the agent does it autonomously; merging is the consequential, hard-to-undo act, so merge_to_main sits in must_escalate and waits for the founder. Because the gated action travels inside the typed request envelope the runtime emits at the gate (§4.7.5), the founder always sees *which* escalated action is being requested — a merge to production, a refund, a publish — rather than an undifferentiated “approve this.” The contract this enforces is the human-oversight requirement developed in Ch.2 §2.8 (European Parliament and Council, 2024, Art. 14; Kolt, 2026; Chan et al., 2024).

This same split generalises the Human-in-the-Loop Contract from content to code. The engineering agent branches, opens a pull request, and lets continuous integration validate it unattended; only the merge to main is a founder gate, and that gate is simply another HITLRequest — on a dev channel, gating merge_to_main — so the

contract that governs the Author Flow (§4.7) governs a “Dev Flow” (draft → open PR → CI → founder review → merge) without modification. It is an application of the contract, not a sixth component (solo-founder-crew/docs/dev-flow.md).

4.5.2 The role catalogue

Six role factories ship in `roles_library.py`, registered under stable names in the `ROLE_LIBRARY` dictionary: `marketing`, `product`, `engineering`, `customer_support`, `sales`, and `finance`. Each factory is a callable (`VentureBrief`) → `Role`. The catalogue is deliberately small — it covers the functions a solo-founder digital venture cannot avoid, and no more. Table 4.4 reproduces it.

Table 4.4: The role catalogue (`roles_library.py`).

Role	Goal	can	must_escalate	Tools
marketing	Customer-facing copy	draft_content, revise_content	final_approval_before_publication	publisher_tool
product	Specs, changelogs	draft_content, revise_content, propose_priority	spec_finalisation, roadmap_change, scope_addition	publisher_tool
engineering	Code changes, pull requests	draft_content, revise_content, propose_change, open_pull_request	merge_to_main, dependency_change, schema_or_data_migration	pr_tool
customer_support	Triage, drafts	draft_content, revise_content, triage_ticket, request_clarification	refund_request, account_termination, legal_or_compliance_question	publisher_tool
sales	Outbound, qualification	draft_content, revise_content, qualify_lead	deal_close, contract_finalisation, custom_pricing	publisher_tool
finance	Invoices, reports	draft_content, revise_content, categorise_expense, summarise_period	payment_dispatch, refund_dispatch, transaction_above_threshold	publisher_tool

Two regularities are worth naming. First, every role shares a common spine: all may `draft_content` and `revise_content`, and each reaches the outside world only through a single gated tool — `publisher_tool` for the five content roles, `pr_tool` for engineering — never holding direct publication or merge rights. The roles differ only in their domain-specific actions and in the contents of their escalation set. Second, the escalation sets are dominated by irreversible, consequential actions — `merge_to_main`, `refund_request`, `deal_close`, `payment_dispatch`. This is the Role Library encoding, as data, the same founder-authority principle that the HITL Contract enforces at run time and that the runtime checks at dispatch (§4.7.4).

4.5.3 Brief-binding: how a role becomes venture-specific

A role in this framework is never generic, yet the `Role` object stays brief-agnostic; the work of making it venture-specific lives in the factory. When `make_marketing(brief)` is called, it assembles the role’s `system_prompt` from five parts: a header naming the venture, a `VOICE` block, a `CONSTRAINTS` (non-negotiable) block, a `DECISION RIGHTS` block phrased in natural language that mirrors the role’s `decision_rights`, and a

role-specific `OPERATING RULES` block. The first three parts come from shared helpers — `_voice_block(brief)`, `_constraints_block(brief)`, and `_header(brief, ...)` — so a change to the venture’s voice propagates to every role at once, without editing any individual factory.

Concretely, `make_marketing(passly_brief)` yields a Marketing role whose system prompt injects Passly’s voice — the warm, peer-to-peer, Greek-SMB-facing register visible in the real-LLM outputs of §5.1.8 — and its constraints. The same factory called against a different brief produces a differently-voiced agent from identical code. Voice, in other words, is not a property of the role; it is a property of the brief, surfaced through the role. Keeping prompt assembly in the factory rather than in the Role itself is what lets the Role stay a declarative, immutable value while still being fully venture-specific in practice.

4.5.4 Extensibility

Adding a role is an additive operation in exactly three steps, none of which touch the runtime or the HITL Contract: write a factory in `roles_library.py`, register it under a key in `ROLE_LIBRARY`, and add a `CrewRule` in the Crew Generator (§4.6). The engineering role was itself added this way, demonstrating the path: a `make_engineering(brief)` factory with its own `can/must_escalate` split, a registry entry, and one always-on rule — and nothing downstream changed, because the runtime and the contract operate over the `Role` abstraction rather than over any named role. The same path would add, say, a Finance-Auditor for a venture that needs one. This open-for-extension, closed-for-modification posture is the same one the substrate decision preserved (§5.1.7): the thesis’s contribution is the set of abstractions, and they extend without disturbing the parts already built.

4.6 Component 3 — The Crew Generator

Given a Venture Brief and the Role Library, something must decide *which* roles a particular venture actually needs. The Crew Generator (`solo-founder-crew/src/solo_founder_crew/crew_generator.py`) is that component, and its defining commitment is that composition is **rule-driven, transparent, and auditable**. Every decision to include or exclude a role is taken by an explicit, human-readable rule, and the reason for that decision is recorded at generation time. A founder can therefore answer “why is there no Sales agent in my crew?” by reading an audit log, not by inspecting model internals.

4.6.1 Why rules, not a learned model

A learned selector — embedding the brief and predicting a role set — was considered and rejected on three grounds. **Transparency**: founder legibility is the central concern of the framework (§4.5, §4.8), and a rule a founder can read serves it where a weight matrix does not. **Defensibility**: a finite rule set is inspectable and arguable *in vivo*, where a learned policy would have to be defended as a black box. **Determinism**: the same brief must yield the same crew, which Chapter 7’s reproducibility criterion (§3.3) requires. An opaque, possibly non-deterministic selector would undercut the transparency premise on which the rest of the framework is built; rule-based composition keeps the artefact legible in the design-science sense

(Hevner et al., 2004; Peffers et al., 2007) and keeps the agent system’s behaviour visible to its principal (Chan et al., 2024).

4.6.2 The rule structure

Composition is driven by an ordered tuple of `CrewRule` values — the `DEFAULT_RULES` constant. Each `CrewRule` carries three fields: `role_name` (a key into the role library), `applies` (a predicate (brief) $\rightarrow$ bool, most often testing the venture’s lifecycle stage), and a reason recorded whether or not the rule fires. The `CrewGenerator` is a pure function of the brief and the rule list: `generate()` evaluates each predicate in order, instantiates the roles whose predicates hold via their factories, and returns a `CrewGenerationResult` carrying both the crew and the audit log. Table 4.5 reproduces the default rules verbatim.

Table 4.5: `DEFAULT_RULES` — the default crew-composition rules.

Role	Included when	Reason (recorded in the audit log)
marketing	always	“Marketing is universal — every venture needs a customer-facing copy.”
product	$\text{stage} \in \{\text{pre-launch, launched}\}$	“Pre-launch and launched ventures still need spec and changelog drafting; growth-stage usually has dedicated product hires.”
engineering	always	“Digital-only ventures are software products; engineering is needed from pre-launch (building the MVP) through growth (extending and maintaining it).”
customer_support	$\text{stage} \in \{\text{launched, growth}\}$	“No customers to support pre-launch; the role activates once the product is in use.”
sales	$\text{stage} \in \{\text{launched, growth}\}$	“A solo founder should not run cold outreach pre-launch — the offer is not ready to qualify against.”
finance	$\text{stage} == \text{growth}$	“Bookkeeping load becomes material at growth stage; pre-launch and early-launched founders can do it in batch.”

4.6.3 Worked composition across the venture lifecycle

Running `examples/crew_generator_demo.py` over the three lifecycle stages of the same Passly brief produces three different crews:

pre-launch $\rightarrow$ {marketing, product, engineering}; **launched** $\rightarrow$ {marketing, product, engineering, customer_support, sales}; **growth** $\rightarrow$ {marketing, engineering, customer_support, sales, finance}.

4.6.4 The non-monotonic transition at growth

The crew is not monotonic on stage. In the move from launched to growth, product drops out while finance enters — the crew does not simply accumulate roles as the venture matures. Engineering, by contrast, is always-on and stays at every stage, which is itself a deliberate claim: a digital-only venture is a software product, so a development function is needed from the pre-launch MVP through growth-stage maintenance. The product transition is the contestable one, and the audit log exists precisely to expose it: the rule records that product work is the function a founder most often hands to a dedicated hire at growth (growth-stage ventures “usually have dedicated product hires”), while bookkeeping and financial-reporting load becomes

material enough to warrant a dedicated agent. Because each transition is encoded as a rule with a recorded reason rather than as hidden behaviour, these are thesis claims open to revision — a venture that disagrees can edit the rules (§4.6.6) rather than the engine.

4.6.5 The audit log

The audit log is the component’s key transparency primitive. Alongside the crew, `generate()` returns a tuple of `RuleDecision` entries — one per rule — each recording the role, whether the rule fired (`selected`), and the reason. The result object exposes `fired()` and `skipped()` views over this log, so a founder generating a pre-launch crew sees not only the positive composition (“Marketing included...”, “Engineering included...”) but also the counterfactual (“Sales excluded: a solo founder should not run cold outreach pre-launch...”). The founder is shown both the crew they have and the crew they do not, each with its justification. This is the same legibility posture that governs the run trace (§4.7.6) and the typed decision envelope (§4.8): the framework consistently prefers explicit, recorded reasons over implicit behaviour.

4.6.6 Customisation

`DEFAULT_RULES` is a default, not a constraint. `CrewGenerator` accepts both a custom rule tuple and a custom role registry, so a venture with different economics supplies its own policy. A fintech that must keep books from its first day, for example, passes a rule set in which `finance` is governed by `always` rather than by `stage == growth`. Because the rules are data and the generator is a pure function over the pair (brief, rules), this customisation requires no change to the generator itself — the same extensibility posture established for the Role Library (§4.5.4). The defaults encode the thesis’s prior about a typical digital venture; they are a starting point the founder can override, not a fixed pipeline.

4.7 Component 4 — The Orchestration Runtime

The Orchestration Runtime is where the previous components become an executable workflow. It composes a Role, a Venture Brief, and the three runtime services — an LLM client, a Human-in-the-Loop contract, and a tool registry — into a `LangGraph` state machine and drives it to a terminal outcome. The runtime lives in `solo-founder-crew/src/solo_founder_crew/runtime.py` (the graph) and `solo-founder-crew/src/solo_founder_crew/crew.py` (the public entry point). It is built on the `LangGraph` substrate selected in §5.1, and its design exhibits four decisions worth defending: a single canonical content pattern, a strict separation of state from services, dispatch-time enforcement of decision rights, and a transport-agnostic founder gate.

4.7.1 The Author Flow pattern

The runtime’s canonical pattern is the **Author Flow**: `draft` → `review` → (`revise` → `review`){0,N} → `publish`. A role drafts an artefact from the brief; the founder reviews it at a Human-in-the-Loop gate; on rejection the role revises and the artefact returns to the gate, up to a bounded revision budget; on approval a tool publishes it. This is the typical content-publication cycle of a solo founder — announcements,

blog posts, product copy, support replies all fit it — which is why it is the framework’s first flow. It is explicitly *one* flow built on the framework’s primitives; further flows (decision-deliberation, triage) are planned and share the same state and trace shape. The graph is assembled by `build_author_graph(...)` and is shown, as the code itself renders it, in Figure 4.4.

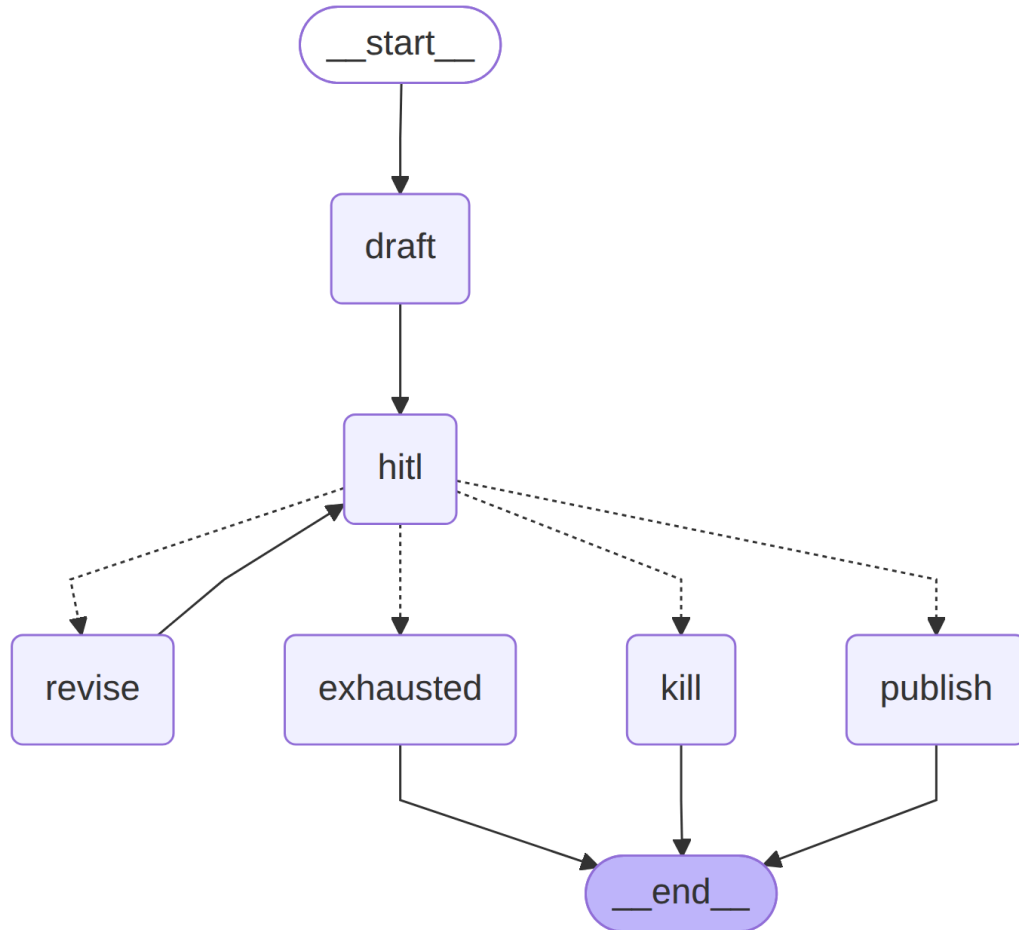


Figure 4.4: The Author Flow graph (machine-generated from the compiled graph by LangGraph’s `draw_mermaid()`).

START → draft → hitl hitl ⇒ publish (founder approves) hitl ⇒ revise (founder rejects, revision budget remaining) hitl ⇒ exhausted (founder rejects, budget spent) hitl ⇒ kill (founder kills the run) revise → hitl publish → END kill → END exhausted → END

The figure is emitted by `compiled.get_graph().draw_mermaid()` and stored at `solo-founder-crew/docs/author_flow.mermaid`; it is therefore machine-generated by the framework being described and cannot drift from the code it documents — one of the four tie-breakers that selected LangGraph (§5.1.6, D3). The dashed transitions out of `hitl` are LangGraph conditional edges, resolved at run time by the founder’s decision (§4.7.5).

4.7.2 The Crew API

The public surface is deliberately narrow. A Crew ties together one venture’s brief, its roles, and the three services, and is constructed once: `Crew(brief=..., roles=[...]`,

llm=..., hitl=..., tools=...). One Crew belongs to one venture — it is, in the language of the thesis title, the venture’s operating system — and many flows may run against it. A run is a single call, `await crew.author_flow(task_description=...)`, which returns an `AuthorFlowResult` carrying the status (`shipped` | `killed` | `exhausted`), the `approved_artifact`, the run trace, and a `thread_id`. The brief lives at the Crew level while the task description is per call — the design that lets one venture’s operating system serve many tasks — and the acting role is resolved automatically when the crew holds a single role, or named explicitly otherwise.

4.7.3 State and services: the separation that keeps the checkpointer correct

The runtime’s most consequential architectural decision is that **state carries data while services live in closures**. The `AuthorFlowState` is a `TypedDict` containing only JSON-serialisable values — the brief, the role descriptor, the task description, the revision budget, the current draft, the turn counter, the founder’s decision, and the terminal status. The three services — the `LLMClient`, the `HITLContract`, and the `ToolRegistry` — are never placed in state. Instead each graph node is produced by a factory (`_make_draft_node`, `_make_hitl_node`, and so on) that closes over the services it needs.

The payoff is that the checkpointer is trivially correct. Because there is no live object in the state to serialise, `LangGraph` can persist the entire run at any pause point and restore it later — possibly in a different process — without any custom serialisation logic and without the risk of a half-serialised client or socket. This is what makes the cross-session pause/resume of §4.8 sound rather than aspirational: the state is, by construction, just data. It is also the design that justified weighting “lock-in & cost” lowest in the spike rubric (§5.1.4), since the services behind the closures are substitutable Protocols (§4.7.7).

4.7.4 Decision rights enforced at dispatch

The decision rights declared on each role (§4.5.1) are not documentation; the runtime enforces them. Before issuing an LLM call, the drafting and revising nodes check `role.may_perform(action)` for `draft_content` and `revise_content` respectively, and raise a `PermissionError` from the runtime — not from the LLM client — if the role lacks the right. The Crew performs the same check as a pre-flight before the graph even starts, verifying both that the chosen role may `draft_content` and that it actually holds the `publish_tool` it will be asked to use. Authority is thus checked where the action is dispatched, keeping the enforcement path independent of whatever the language model happens to emit.

4.7.5 The founder gate and the interrupt loop

The Human-in-the-Loop gate is implemented as a `LangGraph interrupt()` — the substrate property that drove the framework choice (§5.1.6, D1). When the graph reaches the `hitl` node, the node builds a typed `HITLRequest` — the transport-agnostic decision envelope formalised by Component 5 (§4.8) — carrying a stable `request_id` (derived from `thread_id` and `turn`), the asking `role_name`, the gated action, a human-readable `escalation_reason`, and the artefact under review, and calls `interrupt(request.to_interrupt_payload())`. `LangGraph` checkpoints the state and returns control to the loop driver.

That driver is `Crew.author_flow`. It invokes the graph; when execution pauses, it reads the pending interrupt payload from the checkpoint, reconstructs the request with `HITLRequest.from_interrupt_payload(...)`, obtains a `FounderDecision` by awaiting `self.hitl.review(request)`, records the founder’s review on the trace, and resumes the graph with `Command(resume={"action", "feedback"})`; the loop repeats until a terminal node is reached. A subtle but important property: on resume, the paused node body re-executes from the top, so the `hitl` node is kept idempotent up to its `interrupt()` call, and the founder-review trace event is recorded by the driver rather than the node. Routing out of the gate is a `LangGraph` conditional edge: approve routes to publish, kill ends the run, and reject routes to revise while the turn count is within `max_revisions` and to an exhausted-budget terminal otherwise. The router is the framework’s machine-readable version of the prose HITL specification, and the full envelope and its rendering surfaces (including Discord) are the subject of §4.8.

4.7.6 The trace as observability primitive

Every node appends to a shared `RunTrace`, including the founder-review step, producing an ordered log of events of a single shape — (role, action, input, output, decision, timestamp) — identical to the shape used by the three spikes (§5.1). Because the trace object is the same one the nodes write to during the run, it accumulates events in execution order without the caller having to extract anything from the final state, and it serialises directly to JSON for Chapter 7’s rubric-based evaluation. A rejected-then-approved Passly run, for instance, yields five events — `draft_content`, a founder `hitl_review` carrying the Greek feedback, `revise_content`, a second `hitl_review` with an approval, and `publisher_tool.publish` — a trace that is reproducible, machine-readable, and legible to the founder, the same legibility commitment that governs the crew audit log (§4.6.5).

4.7.7 Service-substitution points

Because services are held behind the node closures rather than baked into the graph, the runtime exposes five clean substitution points, each a narrow Protocol-typed contract. This is the artefact-independence the framework claims (design objective O4, §4.2): a venture, a test harness, or a production deployment swaps an implementation without touching the graph, the roles, or the HITL Contract. Table 4.6 enumerates them.

Table 4.6: The runtime’s service-substitution points.

Service	Contract	Default	Swapped for
LLM	<code>LLMClient.complete</code>	MockLLM (deterministic)	Real Anthropic / LangChain model binding
Founder gate	<code>HITLContract.review</code>	ScriptedHITL	InteractiveHITL, DiscordHITL (§4.8)
Tools	<code>ToolRegistry.invoke</code>	<code>publisher_tool</code> (in-process)	Real publish / webhook tools
Memory	<code>BaseCheckpointSaver</code>	MemorySaver (in-process)	SqliteSaver for cross-process resume
Role set	<code>Role / Crew Generator</code>	DEFAULT_RULES catalogue	Custom roles and rules (§4.5, §4.6)

Each contract is small — typically a single method — which is what keeps substitution cheap and the framework’s claim of substrate-independence credible: LangGraph supplies the graph executor and the checkpoint, but the abstractions that constitute the thesis’s contribution sit above it and could be carried to another substrate by reimplementing only this module (§5.1.7).

4.8 Component 5 — The Human-in-the-Loop Contract

The Human-in-the-Loop Contract is the framework’s named seam for founder authority. Where the preceding components decide what the crew is (the Role Library, §4.5), who is on it (the Crew Generator, §4.6), and how its work is sequenced (the Orchestration Runtime, §4.7), the HITL Contract decides the one question the founder may never delegate: when a human must approve before the crew acts. The thesis treats founder authority as a first-class design concern rather than an implementation detail — consistent with the human-oversight baseline set out in Chapter 2 §2.8, where Article 14 of the AI Act (European Parliament and Council, 2024) and the agent-governance analysis of Kolt (2026) establish human oversight as a normative requirement rather than an ergonomic preference — and this component is where that commitment becomes code. The contract is deliberately small — a single-method protocol — so that the locus of human control is legible in one place and substitutable without disturbing the rest of the framework.

4.8.1 *Why a named contract*

A recurring weakness of general-purpose agent frameworks is that human oversight, where it exists at all, is scattered across prompt prose and ad-hoc callbacks (the gap surveyed in Chapter 2 §2.8). Where the AI Act requires that high-risk systems be designed so that natural persons can effectively oversee them (European Parliament and Council, 2024, Art. 14), and where the agent-governance literature treats visibility and human authority as primitives to be engineered rather than assumed (Kolt, 2026; Chan et al., 2024), a framework for solo-founder ventures must give that oversight a definite location. The framework’s response is to centralise founder authority behind one named abstraction, with three consequences. First, the responsibility is auditable: a reviewer — or a viva examiner — can point to the single interface through which every founder decision passes. Second, it is substitutable: the surface through which the founder is asked (a terminal, a test fixture, a chat application) can change without touching the runtime, the roles, or the tools. Third, it makes the founder’s authority enforceable rather than advisory — a role’s `DecisionRights.must_escalate` entries (§4.5) name the actions the runtime refuses to take until the contract returns approval.

4.8.2 *The contract surface and the three founder actions*

The contract is a Python protocol with a single asynchronous method, `review(...)`, which returns a `FounderDecision` (defined in `solo-founder-crew/src/solo_founder_crew/hitl.py`). The decision carries one of three actions — `approve`, `reject` (with optional free-text feedback), or `kill`. The parsimony is deliberate. `Approve` advances the artefact to publication; `reject` returns it

to the responsible role for one revision cycle, carrying the founder's feedback into the next prompt; kill terminates the run with nothing published. Every other founder's intent reduces to these three: a request for changes is a rejection with feedback, abandonment is a kill, acceptance is an approval. Holding the contract to three actions keeps both the prose specification and the runtime's routing logic (§4.7) small enough to verify by inspection.

4.8.3 The decision request: a typed envelope, not a string

An earlier formulation of the contract passed to the founder only the artefact text under review. That suffices when the founder is at the terminal and the surrounding context is implicit, but it fails the moment the decision must be rendered somewhere the runtime cannot see: a remote surface has no way to know which venture raised the request, which role is asking, what action is being gated, or which responses the founder may give. The contract therefore carries a typed decision request, the `HITLRequest`, specified as JSON Schema at `solo-founder-crew/src/solo_founder_crew/schemas/hitl_request.schema.json` and serving as the single source of truth shared between this chapter and the prototype code — the same convention used for the Venture Brief in §4.4.

The envelope names everything a surface needs to present a decision without reaching into the runtime: a `request_id` correlating the rendered prompt with the parked run; the `thread_id` that is the runtime's resume key; the `venture_id`; the asking role; a logical channel (an intent such as `ads-report` or `customer-support`, deliberately decoupled from any surface's native channel identifiers); the gated action, drawn from the role's `must_escalate` set; the artefact under review; and the options the founder may choose, each with a display label and a flag for whether it requires accompanying feedback. The founder's reply is the mirror image — a `FounderResponse` (defined in the schema's `$defs`) that maps one-to-one onto the `FounderDecision` the runtime already understood, plus the `request_id` needed to route the answer back to the correct parked run. Figure 4.5 renders the envelope's key fields.

HITL Request — the gate envelope (Components 4 → 5)

schemas/hitl_request.schema.json · JSON Schema 2020-12 · additionalProperties: false

request_id	string	required
thread_id	string	required
venture_id	string	required
turn	integer	required
role	object	required
channel	string	optional
action	string	required
escalation_reason	string	optional
artifact	object	required
options	array<object>	optional
context	object	optional
urgency	enum	optional blocking fyi
created_at	string	optional

FounderResponse (\$defs) — the founder's reply, routed back by request_id

request_id	string	required
action	enum	required approve reject kill
feedback	string	optional
responder	string	optional
responded_at	string	optional

Bordered/filled rows are required. The response mirrors the runtime's FounderDecision; feedback is present iff action = reject.

Figure 4.5: The HITL Request envelope and the FounderResponse reply (generated from hitl_request.schema.json).

Naming the request as data, rather than as a string, is what makes the remaining sections possible: the same contract, unchanged, can be rendered to a terminal, a test fixture, or — as §4.8.6 develops — a chat application the founder already carries in their pocket.

4.8.4 Integration with the runtime: the interrupt() loop

The contract is invoked through LangGraph's `interrupt()` primitive — the property that drove the substrate decision in §5.1. The integration is a six-step loop, driven by `Crew.author_flow` (`solo-founder-crew/src/solo_founder_crew/crew.py`) around a single graph node (`runtime.py`):

(1) The graph reaches the gate node, which calls `interrupt(request)` with the `HITLRequest`; LangGraph checkpoints the run's state at that point. (2) Control returns from the graph to the `author_flow` loop driver. (3) The driver calls `await self.hitl.review(...)`, handing the request to whichever contract implementation is configured. (4) The implementation obtains a `FounderDecision` — synchronously from a fixture, or after an arbitrarily long wait while a human responds. (5) The driver resumes the graph with `Command(resume=...)`, carrying the decision back in. (6) The gate node's body re-executes from the top, now receiving the founder's decision as the return value of `interrupt()`; the runtime's conditional edges then route on the action.

Because the run's state is checkpointed at step (1) and the driver merely awaits the decision at step (3), the pause between a draft completing and the founder responding is unbounded. With an in-memory checkpointer it survives within a process; with a persistent checkpointer (LangGraph's SQLite reference implementation), keyed by `thread_id`, it survives the process restarts entirely. This is the technical basis for design objective O6 (§4.2): the founder is not assumed to be present at the moment a

draft is ready (Hevner et al., 2004; Peffers et al., 2007, on instantiation as an evaluable artefact).

4.8.5 Reference implementations

Because the contract is a one-method protocol, an implementation is small and the set is open. Three are relevant to this thesis. `ScriptedHITL` reads decisions from a JSON fixture and is the deterministic test double used in every framework test and in the Chapter 7 evaluation runs. `InteractiveHITL` prompts on standard input and serves local development and the live demonstration to the supervisor. `DiscordHITL` — developed in §4.8.6 — renders the decision request to a chat channel and returns the founder’s button-tap as a `FounderDecision`; it is the working production surface. Further surfaces (a web UI, SMS, or email callback) are argued as same-protocol substitutions rather than built, since each would add engineering without strengthening the framework claim — the contract, not any particular surface, is the contribution.

4.8.6 Discord as the founder’s HITL surface

A solo founder wants neither a terminal nor another dashboard to monitor; the operating-model framing of the thesis title suggests instead that the crew should feel like a company the founder already belongs to. A team chat application is the natural rendering. `DiscordHITL` implements the HITL Contract against Discord, turning each part of the `HITLRequest` into an element of an interface the founder already understands.

The mapping is direct. The request’s logical `channel` resolves to a Discord channel — `#ads-report`, `#customer-support`, `#dev` — so decisions arrive already sorted by the part of the business they concern. The asking `role` supplies the identity under which the message is posted, so the founder sees “Marketing” or “Support” rather than an anonymous bot. The options become interactive buttons, and an option flagged as requiring feedback (the reject case) opens a short modal in which the founder types guidance before the response is submitted. The artefact summary becomes the channel preview and the push-notification body; the full content sits in the message itself.

The pause/resume mechanics of §4.8.4 carry over unchanged. `DiscordHITL.review()` posts the message and then awaits a future keyed by the request’s `request_id`; the bot’s interaction handler, firing when the founder taps a button, resolves that future with a `FounderDecision`, and the `author_flow` driver resumes the graph. Because the wait is bounded only by the founder and the run is checkpointed, the founder may answer minutes or days later, from a phone, after the originating process has restarted — the `thread_id` in the request is the key that finds the right run.

One design discipline must be stated explicitly, because it protects the framework claim: Discord is a rendering and input surface only, never the source of truth. The orchestration state lives in the runtime and its checkpoint, and the `HITLRequest` is the entire interface between them. Coupling business logic to a chat vendor — storing run state in messages, or branching on Discord-specific structures inside the runtime — would re-scatter the founder-authority seam the contract exists to centralise, and would weaken the portability claim (design objective O4, §4.2). The adapter therefore

holds no orchestration logic: it translates a `HITLRequest` into a message and a button-tap back into a `FounderDecision`, and nothing more.

What this demonstrates, and what it does not, should be stated plainly. It demonstrates that the framework’s founder-authority seam can be rendered through a familiar, mobile-native surface that removes the terminal from the solo founder’s loop — the felt experience of supervising a small company from a phone. It does not claim that Discord is the only, or the recommended, production surface; the contract’s substitutability is precisely the point, and the same `HITLRequest` would drive a web or SMS surface equally. Consistent with the thesis’s evaluation posture, the surface is exercised in the sandbox against synthetic Passly scenarios, not against a live customer deployment.

4.9 Worked example — the Passly launch via the framework

The preceding sections described the artefact one component at a time. This section puts the five components together and runs them once, end to end, on a task a solo founder would actually have: announcing the launch of their venture. The point of the exercise is to make the abstract claims of §§4.4–4.8 visible in a single concrete run — to show what the founder types, what the crew produces, where the founder is asked to decide, what the record of the run looks like afterwards, and what happens on each of the ways a run can end. Every listing, artefact and count below is taken from files in the `solo-founder-crew` repository (`examples/passly_launch.py`, `examples/out/`, `examples/crew_generator_demo.py`), regenerated and verified rather than reconstructed from memory.

The walk-through has five steps. §4.9.1 describes the input — the Venture Brief — and the three fields in it that will visibly steer the result. §4.9.2 shows the founder’s side of the interaction: the handful of lines that start a run. §4.9.3 follows the run itself, from the first draft through the founder’s rejection and the revision to the published artefact, and shows both drafts so the effect of the founder’s feedback can be seen rather than described. §4.9.4 reads the run record the framework produced. §4.9.5 shows the other three ways the same run could have ended, and §4.9.6 shows the Crew Generator composing a crew for the same brief. §4.9.7 closes by stating what the example does and does not establish.

4.9.1 The input: what the framework is told about the venture

Everything in the run is derived from one document, the Venture Brief of §4.4 — here `scenarios/fixtures/passly_brief.json`, which describes Passly as a pre-launch, marketing-technology venture selling to Greek small businesses. The brief has many fields; three of them do the steering work in this example, and they are quoted here so that the output can later be read against them.

- **‘stage: “pre-launch”’**. This one word tells every role that the venture has no customers yet. It is the reason the Crew Generator (§4.9.6) withholds the customer-support and sales roles, and the reason the compliance notes forbid implying paying customers.
- **‘voice’**. The brief asks for a tone that is “warm, plain Greek, confident without hype”; its `do` list says *speak as a peer, not a salesperson, use concrete examples*

(*a καφέ, a κομμωτήριο*), *lead with the outcome, not the technology*; its don't list forbids hype words such as “revolutionary” and “cutting-edge”, forbids promising specific revenue-lift numbers, and forbids mentioning live customer pilots because the venture is pre-launch.

- **‘constraints’**. `out_of_scope` excludes physical or hybrid retail integrations and markets outside Greece — the digital-only scope of Chapter 1 §1.1 and objective O7, carried as data; `compliance_notes` requires GDPR conformance and no personal-data examples in marketing copy.

None of these fields is a prompt. They are structured data that the Role Library binds into a role's context (§4.5.3), and the reason for quoting them is that they are what the founder controls: if the shipped announcement is warm, Greek, free of hype and honest about being pre-launch, that is because the brief said so, and a different brief would have produced a different announcement from the same code.

4.9.2 The founder's side: starting a run

The body of `main()` in `examples/passly_launch.py`, with its logging and file-writing lines removed, is reproduced as Listing 4.1. It is everything the founder's program has to do to operate the venture through the framework for this task.

Listing 4.1: The minimum-viable Passly launch (`examples/passly_launch.py`, body of `main()`).

```
brief = VentureBrief.from_file(BRIEF_PATH)           # 1 the venture, as data
role = build_marketing_role()                       # 2 a role from the library
tools = build_publisher_tool_registry()              # 3 tools the role may use
hitl = ScriptedHITL.from_file(RESPONSES_PATH)        # 4 founder decisions
llm = (RealLLM() if real_llm                         # 5 the model: live, or
      else MockLLM(responses=[DRAFT_V1, DRAFT_V2])) # the stub

crew = Crew(brief=brief, roles=[role],               # 6 one venture's crew
            llm=llm, hitl=hitl, tools=tools)
result = await crew.author_flow(                     # 7 one flow, one call
    task_description=(
        "Draft a launch announcement for this venture. "
        "Match the voice. Respect every constraint. "
        "Keep under 150 words."
    )
)
```

Read line by line, the listing is the framework's architecture in miniature. Lines 1–2 are Components 1 and 2: the brief is loaded and validated against its schema, and a role is taken from the library. Line 3 is the tool registry the Orchestration Runtime will consult before letting the role act. Line 4 is Component 5: the Human-in-the-Loop Contract, here satisfied by `ScriptedHITL`, which reads the founder's decisions from a fixture file so the run is reproducible — in the deployed application of Chapter 6 the same line is a web adapter, and in the Discord surface of §4.8.6 it is a chat bot; nothing else in the listing changes. Line 5 is the model client behind the `LLMClient` Protocol: a real Anthropic model when `--real-llm` is passed, otherwise the deterministic `MockLLM` pre-loaded with the two drafts the run will produce. Line 6 assembles the crew: one venture, one brief, its roles and its three services — a crew is a venture's operating system, constructed once and used for many tasks. Line 7 is the whole task from the founder's point of view: one call, one sentence of instruction, and the founder's next involvement is at the gate.

Two things the listing does *not* contain matter as much as what it does. There is no prompt engineering: the role's system prompt is built by the library from the brief, and the founder writes only the task. And there is no control flow: the drafting, the review, the possible revision and the publication are the runtime's business, not the caller's. Run under the deterministic stub, the call prints `status=shipped events=5 chars=515` — the run reached the shipped terminal state, its record holds five events, and the published artefact is 515 characters long. Objective O2 (§4.2) asked that a brief and a task yield a founder-approved, publishable artefact in a single call; line 7 is that call.

4.9.3 The run: draft, rejection, revision, publication

What happens inside line 7 is the Author Flow of §4.7.1 — `draft` → `review` → (`revise` → `review`) → `publish` — and this run happens to take its longer branch, because the founder rejects the first draft. The four moments are worth following with the actual artefacts in view.

The first draft. Before the marketing role is allowed to draft, the runtime checks its decision rights (`role.may_perform("draft_content")`, §4.7.4); the check passes, the model is called with the brief-bound system prompt and the task, and it returns the following text.

Introducing Passly — digital wallet passes and AI-assisted marketing designed for Greek small and medium businesses.

Passly enables café and salon owners to issue loyalty passes and promotional offers that customers store directly in Apple Wallet or Google Wallet. The platform provides AI-generated campaign drafts which the business owner can review and dispatch via SMS or email.

Passly is currently in a pre-launch phase. Early-access participation is available to qualifying Greek SMBs.

The draft is competent and it respects every hard constraint — no hype words, no revenue promises, honest about pre-launch. It is also in English, formal, and leads with the technology rather than the outcome. Whether that is acceptable is not the framework's decision to make.

The founder gate. The role's decision rights name `publisher_tool.publish` as an action it must escalate (§4.5.1), so the runtime does not publish. It builds a typed `HITLRequest` — the run's identifiers, the asking role, the gated action, the draft under review and the founder's three options — checkpoints the run, and hands the request to the Human-in-the-Loop Contract (§4.8.4). In this run the contract is scripted, and the fixture returns a **reject** with the following feedback, preserved verbatim in the run record:

Καλό draft αλλά πολύ formal. Να ζεστάνει το άνοιγμα — μίλα σαν σε φίλο μαγαζάτορα. Πρόσθεσε ένα μικρό ελληνικό tagline στο τέλος. Κράτησέ το κάτω από 120 λέξεις.

(Good draft but too formal. Warm up the opening — talk as if to a shop-owner friend. Add a short Greek tagline at the end. Keep it under 120 words.) This is the founder

exercising the authority that objective O1 reserves to them: not by editing the text, but by deciding and directing.

The revision. The rejection routes the flow back to the marketing role, which is called again with the founder's feedback added to its context — after the runtime has checked that the role may `revise_content`. The second draft is the artefact that will ship:

Έρχεται το Passly — και θα σου δώσει πίσω τους πελάτες που ξέχασες ότι έχεις.

Φτιάξε passes επιβράβευσης και προσφορές για το καφέ ή το κομμωτήριό σου σε λίγα λεπτά. Ζουν μέσα στο Apple ή Google Wallet — δεν χρειάζεται να κατεβάσει εφαρμογή. Η AI γράφει την καμπάνια, εσύ τη στέλνεις με δύο κλικ μέσω SMS ή email.

Όλα στα ελληνικά, GDPR-φιλικά. Είμαστε σε pre-launch και ψάχνουμε early-access μαγαζάτορες που θέλουν να το δοκιμάσουν πρώτοι.

Στο Passly, η επόμενη επίσκεψη του πελάτη ξεκινά από το πορτοφόλι του.

Set beside the first draft, every element of the founder's note is visible: the register is warm and addressed to a peer (*σου, το καφέ ή το κομμωτήριό σου*), the opening leads with the outcome (customers coming back) rather than the product category, a Greek tagline closes it, and it runs to well under 120 words. Equally visible is what did *not* change: the brief's don't list survived the revision — there is still no hype word, no revenue figure, and the pre-launch status is stated plainly — and the brief's do list is honoured more fully than before (the concrete καφέ and κομμωτήριο examples the brief asked for now appear). The feedback steered the draft; the brief bounded it.

The second gate and publication. The revised draft returns to the gate. The scripted founder now returns **approve**; the runtime resumes and — the pre-flight check of §4.7.4 having already verified that the role holds the `publisher_tool` it will use — dispatches `publisher_tool.publish`, which in this example is a stub that records the artefact (`shipped:515chars`) and writes it to `examples/out/final_announcement.txt`. The run ends in the shipped state and `author_flow` returns an `AuthorFlowResult` carrying that status, the approved artefact, the trace and the run's `thread_id`.

4.9.4 The record: reading the trace

The founder in this example never saw the runtime; what they can see afterwards is the run record, and the record is the framework's answer to the visibility requirements of Chapter 2 §2.8. Every node in the flow, and the loop driver on the founder's behalf, appends an event of one fixed shape — (`role`, `action`, `input`, `output`, `decision`, `timestamp`) — to a `RunTrace` (§4.7.6), which serialises to JSON. Table 4.7 is the trace of the run above, with the long fields abbreviated.

Table 4.7: The run record of the reject-then-approve run (`examples/out/run_trace.json`), one row per event.

#	Role	Action	Input	Output	Decision
1	marketing	draft_content	the task description	the first draft (English, formal)	—
2	founder	hitl_review	the first draft	the feedback note (Greek, verbatim)	reject
3	marketing	revise_content	the feedback note	the second draft (Greek, warm)	—
4	founder	hitl_review	the second draft	—	approve
5	marketing	publisher_tool.publish	515 chars	shipped:515chars	—

Three things can be reconstructed from this table by someone who was not present. *What was produced:* both drafts are in the record in full, so the revision can be compared against the original. *Who decided what:* the two founder rows are the only rows carrying a decision, and the first carries the reason for it; the marketing role never decides, it drafts and revises. *Why the publish was allowed:* it follows an approve in the row before it, and no publish event exists without one. That is what Chapter 7 later scores as trace legibility and authority transparency (criteria A4 and A5), and it is the property a chat transcript cannot offer, because a chat transcript records words, not decisions. The identical shape is used by every run in the evaluation of Chapter 7 and by the three spikes of Chapter 5 §5.1.

4.9.5 The other three endings

The run above took one of four paths. A founder gate is trustworthy only if every way a run can end is a designed, reachable state rather than an accident of control flow, so `examples/termination_paths.py` drives the same brief and role down all four, and `tests/test_author_flow.py` pins the event count of each by assertion. Box 4.1 lists them; the event counts are the counts the tests assert, and the traces of all four are preserved in `examples/out/`.

Box 4.1: The four termination paths of the Author Flow, with verified event counts.

Path	What the founder does	Terminal status	Events	Action sequence
approve	approves the first draft	shipped	3	draft_content → hitl_review → publisher_tool.publish
reject_then_approve	rejects once with feedback, then approves	shipped	5	draft_content → hitl_review → revise_content → hitl_review → publisher_tool.publish
kill	stops the run	killed	3	draft_content → hitl_review → run_killed
exhausted	rejects until the revision budget is spent	exhausted	5	draft_content → hitl_review → revise_content → hitl_review → revision_budget_exhausted

The table is symmetric, and the symmetry is the design statement. *Approve* and *kill* both end after a single review in three events; they differ only in whether the terminal action publishes or halts, which is to say the founder’s “no” is exactly as first-class as

their “yes”. *Reject-then-approve* and *exhausted* both take two reviews and five events and diverge only at the last one. That last state — `revision_budget_exhausted` — is the one that matters most for defensibility. `author_flow` takes a `max_revisions` budget (one, by default); a run whose drafts the founder keeps rejecting does not loop or hang, it stops in a named, observable state the founder can see in the record and act on. Exhaustion is a designed outcome, not a failure mode. And because all four traces are deterministic under the stub, the four paths are not merely described as reachable: the test suite reaches them on every run.

4.9.6 Composing the crew from the brief

Listing 4.1: chose the marketing role by hand. The Crew Generator of §4.6 can instead compose the crew from the brief alone, and `examples/crew_generator_demo.py` runs it against the Passly brief at its declared stage. The generator makes no model call — composition is deterministic and reads only the brief — and it prints an audit log in which every role in the library appears, selected or withheld, with the rule that decided it. Table 4.8 is that log for the brief as written.

Table 4.8: The Crew Generator’s audit log for the Passly brief at `stage = pre-launch` (`examples/crew_generator_demo.py`).

Role	Decision	Reason recorded
marketing	selected	Marketing is universal — every venture needs customer-facing copy.
product	selected	Pre-launch and launched ventures still need spec and changelog drafting.
engineering	selected	Digital-only ventures are software products; engineering is needed.
customer_support	withheld	No customers to support pre-launch; the role activates once the product is in use.
sales	withheld	A solo founder should not run cold outreach pre-launch.
finance	withheld	Bookkeeping load becomes material at the growth stage.

The three withheld rows are the point. A framework that only listed the roles it selected would be no more transparent than a fixed roster; this one records, at the moment of composition, why each absent role is absent — and each reason traces to a field of the brief (`stage`, the digital-only scope) and to a rule the founder can read and change (`DEFAULT_RULES`, Table 4.5). That is the transparency property objective O4 requires, and it is the property Chapter 2 §2.3 found missing from every framework that composes crews dynamically. The same demo advances the brief’s `stage` to `launched` and then `growth`, and the log shows `customer-support` and then `finance` entering the crew as the rules fire; §4.6.3 gives that three-stage walk-through in full.

4.9.7 What the example establishes, and what it does not

Because the run above used the deterministic stub, a reasonable question is whether the behaviour survives a real model. It does. `docs/spike-evidence/custom_haiku.txt` is a genuine Anthropic Haiku output for this brief in the same warm-Greek register, honouring the same don’t list; the stub’s second draft was modelled on it, so the deterministic path used for reproducible runs is faithful to observed real-model behaviour, and `passly_launch.py --real-11m` regenerates a live run whenever a fresh one is wanted. The stub is a measurement convenience, not a different system.

What the example establishes is that the framework instantiates from a brief; runs the Author Flow end to end in one call; is founder-controllable in the precise sense that the founder's decision re-routed the flow and no publication occurred without an approval; honours the brief's voice and constraints through a revision; leaves a record from which a third party can reconstruct what was produced, who decided what, and why the publish was permitted; ends in one of four designed states, all of them reached by the test suite; and composes a crew transparently, with the reasons for exclusion recorded. What it does not establish is anything comparative or anything about scale: it says nothing about whether the framework produces better copy than a founder with a chat window, nothing about long-running operation or many concurrent flows or ventures, and nothing about behaviour on live customer data. Consistent with the thesis's evaluation posture, this is a sandbox demonstration on a synthetic Passly scenario. The systematic appraisal of such runs against criteria fixed in advance — rather than the fact of their existence — is the work of Chapter 7, and the instrument for it is defined in Chapter 3 §3.3.

4.10 Summary

This chapter presented the thesis's artefact: a five-component framework — the Venture Brief Schema (§4.4), the Role Library (§4.5), the Crew Generator (§4.6), the Orchestration Runtime (§4.7), and the Human-in-the-Loop Contract (§4.8) — built on a LangGraph substrate that was not assumed but selected through the construction spike of §5.1, in which three candidate substrates were implemented against an identical scenario and scored under a pre-registered rubric. Each component was derived from a design objective traced back to the Chapter 2 literature (§4.2), demonstrated end-to-end on the Passly brief (§4.9), and made measurable by the five evaluation dimensions (E1–E5) handed forward to Chapter 7 (§3.3).

What is now possible that was not before this chapter is a single, legible answer to a question the literature left open: how a solo founder can operate a venture through a crew of agents while retaining decisive authority over everything that reaches a customer or cannot be undone. The Role Library makes authority data rather than prose; the Crew Generator makes composition transparent and auditable; the Orchestration Runtime turns a brief into a publishable artefact in one call; and the HITL Contract — rendered, in the worked deployment, through a chat surface the founder already inhabits — makes the founder gate a felt experience rather than a terminal prompt. The framework is also portable: because its services are Protocol-typed and its abstractions outlive the substrate, the contribution is the operating model, not the vendor.

That is the claim to carry forward. The contribution of this chapter is not a multi-agent system — the agent architecture is the means, not the thesis — but a *methodological framework for solo-founder digital entrepreneurship*: a reusable, theory-grounded, and demonstrably founder-governed way of operating a digital venture, of which the running code is one faithful instantiation. Chapter 6 applies the framework to Passly in depth, and Chapter 7 evaluates it against the criteria defined in §3.3.

5. Development Tools

This chapter documents the tools the framework and the application were built with, and for each the three things the reader is owed: what the tool can do, why it was chosen over its alternatives, and how it is used. The chapter has two parts of unequal weight. §5.1 is the substrate decision — the choice of orchestration runtime on which the framework of Chapter 4 rests — and it is reported at length because it was not taken from documentation but settled empirically, by constructing three candidates against an identical scenario and scoring the constructions under a rubric fixed in advance. §5.2 covers the remaining tools of the framework and of the Passly application of Chapter 6 — the model client, the persistence and checkpointing layer, the web and API stacks, the pass-signing chain, the messaging services and the test and integration tooling — where the choices were conventional and are reported at the length they deserve. §5.3 summarises.

5.1 The orchestration substrate — the construction spike

The framework described in Chapter 4 is, in part, a layer of abstractions over a chosen orchestration substrate. Three substrates were plausible at the outset: a from-scratch implementation in `asyncio`, the high-level multi-agent framework **CrewAI**, and the lower-level state-graph framework **LangGraph** (both as of late 2025). Each carried architectural consequences for the framework’s five components — particularly for the Human-in-the-Loop Contract (§4.8) and the Orchestration Runtime (§4.7). Reasoned, literature-only selection would have been insufficient: the kernel methodology (Hevner et al., 2004; Peffers et al., 2007) does not specify substrates, and the published evaluation literature on agent frameworks (Wu et al., 2023; Guo et al., 2024) covers their capabilities in isolation rather than their fit to a solo-founder operating model. The choice was therefore treated as an empirical question the thesis itself answers, via a construction spike whose methodology and outcome this section reports.

5.1.1 The need for an empirical decision

A framework targeting solo-founder digital entrepreneurship places unusual weight on three substrate properties. **First**, the Human-in-the-Loop Contract must support genuine pause/resume — the founder is not at the terminal at the moment a draft completes; they respond minutes, hours, or days later, possibly from a different process. **Second**, role decision rights and tool-access enforcement must be *declarative*, not implicit in prompt prose, so the founder can audit (and the thesis can defend) which authority each role holds. **Third**, the run trace must be machine-readable and observable end-to-end, because Chapter 7’s rubric-based evaluation depends on it.

Each candidate substrate makes a different set of trade-offs against these properties. A from-scratch implementation maximises fit but maximises construction cost; CrewAI optimises for high-level ergonomics on the canonical “draft something” case but constrains runtime control; LangGraph exposes a primitive graph model and asks the developer to compose abstractions on top of it. None of these trade-offs is visible in the published documentation — they surface only in construction. The spike therefore

exists to **construct each candidate** against an identical scenario and **score the construction itself**, treating the artefact rather than the claim as evidence. This is consistent with Hevner et al. (2004, §4), whose third design guideline (“Design Evaluation”) admits construction as a valid mode of evaluation when the artefact is novel and prior comparators are absent.

5.1.2 Method

The spike implemented the **same scenario** in three candidates, pre-registered a **seven-criterion rubric**, scored each candidate under the rubric using a **deterministic stub LLM**, and validated that each implementation actually runs end-to-end against a real model via an **unscored smoke test**.

The scenario is the Passly launch announcement: a single Marketing role drafts a customer-facing announcement from a Venture Brief; the founder reviews via a Human-in-the-Loop gate (approve / reject with feedback / kill); on rejection the role revises once; on approval a stub publisher tool ships the artefact. The scenario exercises all five framework components in one short flow — Venture Brief Schema (input), Role Library (the Marketing role), Crew Generator (single-role crew), Orchestration Runtime (sequencing, memory across revisions), and HITL Contract (the founder gate). Full specification is in `solo-founder-crew/scenarios/passly_launch.md`.

The **stub LLM** (`solo_founder_crew.MockLLM`) returns canned text from a pre-loaded response queue rather than making any model call. The choice is methodologically deliberate: the rubric scores *framework ergonomics*, not model output quality (see §5.1.4). A deterministic stub makes spike runs free, reproducible, and immune to sampling variance that would otherwise contaminate scoring. A separate **real-LLM smoke test** — one Anthropic Haiku call per candidate, unscored — verifies that each implementation integrates end-to-end with a production model. The smoke-test outputs are preserved at `solo-founder-crew/docs/spike-evidence/` and inform the qualitative discussion in §5.1.9.

All three implementations are committed at `solo-founder-crew/spikes/{custom,crewai,langgraph}/` and were capped at ~300 lines of code each (the soft budget held to within 30%), ensuring no implementation hid friction by becoming the *researcher’s* construction-cost outlier.

5.1.3 Candidates

Three candidates exhaust the practical design space as of late 2025. **CrewAI** (version 0.108) packages multi-agent orchestration as Agent/Task/Crew primitives with opinionated control flow: agents are configured with goals and backstories; tasks are assembled into a crew; `Crew.kickoff()` executes them in sequence with internal retry and parsing logic. It optimises for the common case of “an agent produces output, which is consumed by the next agent.” The framework’s literature foothold dates to 2024 (Wu et al., 2023; with the CrewAI project itself maturing into a stable release line in 2025).

LangGraph (version 1.0.1, by LangChain) takes the opposite position: it exposes a typed StateGraph of nodes and conditional edges, leaves the developer to compose any higher-level abstractions on top, and ships built-in primitives for state

checkpointing, cross-process resume, and explicit interrupts. LangGraph’s positioning in the literature is as a graph-execution substrate rather than an agent framework per se; the comparison with CrewAI is therefore not “two frameworks doing the same thing” but “two different layers of the same problem.”

A **custom asyncio implementation** serves as the from-scratch baseline. It demonstrates what the framework would look like with no substrate at all — a direct construction in Python’s standard library, using `asyncio` for the natural-async LLM call pattern and bare dataclasses for state. The custom spike’s purpose is not to be a serious candidate for production use; it is to establish what “zero substrate friction, but full construction cost” actually looks like, so the substrate candidates can be scored against a known reference.

5.1.4 Rubric

The rubric scores **framework ergonomics**, not LLM output quality. Seven criteria with pre-registered weights:

Table 5.1: Pre-registered rubric.

#	Criterion	Weight	A “5” looks like
C1	HITL ergonomics	3	Approval gates are first-class; trivial to insert; trivial to test
C2	Role parameterisation (decision rights, escalation)	3	Roles carry typed config; decision rights are declarative, not prose
C3	State/memory across turns	2	Memory model is explicit and inspectable; not coupled to a vendor
C4	Tool access control	2	Per-role allowlists trivial; no global registry leaks
C5	Observability / debuggability	2	Structured traces (role, action, input, output) are easy to capture
C6	Dev experience (solo founder)	2	Small surface area, low ceremony, runs locally without infra
C7	Lock-in & cost (sandbox-friendly)	1	No mandatory hosted services; works with any LLM API

Each candidate is scored 1–5 per criterion. Weighted total = $\Sigma(\text{score} \times \text{weight})$; maximum raw score 75.

The **deliberate exclusion** of LLM output quality bears emphasis. Three implementations using the same canned stub responses cannot distinguish themselves on what the model says — they can only distinguish themselves on what the framework demands the developer to write in order to elicit, route, gate, persist, and inspect what the model says. That, not output quality, is what determines whether a solo founder can operate the resulting system. Output quality is addressed in Chapter 7 with the real-LLM evaluation.

A second methodological footnote applies to weighting. The weights — C1 and C2 at weight 3, C3–C6 at weight 2, C7 at weight 1 — encode the thesis’s priors about a solo-founder operating model. HITL ergonomics and role parameterisation are weighted highest because they sit closest to the founder’s experience of authority and

delegation, which are this thesis's central concerns. Output of the spike is therefore informative *given* these priors and would not necessarily transfer to a multi-founder or enterprise setting.

5.1.5 Results

The complete scorecard is reproduced from `solo-founder-crew/docs/spike-scorecard.md`:

Table 5.2: Spike scorecard.

Criterion	W	Custom	CrewAI	LangGraph
C1 HITL ergonomics	3	5	2	5
C2 Role parameterisation	3	5	2	4
C3 State/memory across turns	2	4	3	5
C4 Tool access control	2	5	2	4
C5 Observability	2	4	2	5
C6 Dev experience	2	4	2	4
C7 Lock-in & cost	1	5	2	4
Weighted total (/75)	—	69	32	67
LOC (.py only)	—	365	385	428
Substitution-friction items	—	0	4	1

Per-criterion evidence is documented in the scorecard file; the present chapter does not reproduce it in full but summarises the three headline findings:

CrewAI scored 32 of 75 — last by a wide margin. It scored 2 of 5 on five of seven criteria. The framework has no declarative slot for role decision rights or escalation; supporting the thesis's operating-model required a sidecar `RoleMeta` dataclass alongside the `CrewAI Agent`, with enforcement open-coded into the scenario script. Its built-in HITL (`Task(human_input=True)`) calls `input()` on `stdin`, which is unusable for deterministic scoring runs and requires bypass. Its agent executor parses LLM output for a `"Thought: ...\\nFinal Answer: ..."` regex, rejecting unrecognised responses with internal re-prompts that the developer cannot bound (the `max_iter=1` setting on `Agent` did not constrain LLM calls per task). The framework couples the LLM contract to its own parser. None of these are fatal in isolation; together they signal that CrewAI's opinions about *how* a crew should operate do not match the solo-founder operating model this thesis pursues.

Custom (69) and LangGraph (67) are within rounding noise. The two-point gap is well inside the resolution of the rubric. Both candidates expose the framework's primitives cleanly. The custom implementation scores marginally higher because every line was tailored to the thesis vocabulary; the LangGraph implementation scores marginally lower on Role parameterisation (one of CrewAI's slots, role declarative config, is also absent from LangGraph, which treats nodes as functions over a `State TypedDict`) and Tool access control (no native concept; same enforcement function as the custom spike). On every other criterion they are equal or LangGraph leads.

The substitution-friction column is complementary evidence for the rubric scores. It records how hard it was to substitute the deterministic stub LLM for each framework’s normal model binding. The custom implementation owns the LLM contract as a one-method Protocol (zero friction). LangGraph imposes one footgun — a Python namespace collision when the spike folder is named `langgraph/` — but otherwise accepts any duck-typed object exposing `.invoke(messages).content` as its model. CrewAI requires subclassing a ~25-argument base class, satisfying a `litellm`-recognised model string for its constructor’s internal validation, and wrapping every response in the parser format the executor demands. Substitution friction tracks the rubric: same direction, same order, validating that the scores reflect coupling rather than incidental ergonomics.

5.1.6 Decision and tie-breaker

CrewAI is eliminated on the rubric. The decision the thesis must defend is the choice between Custom and LangGraph at 69 versus 67 — a difference inside the rubric’s resolution.

Four considerations break the tie cleanly in LangGraph’s favour, and each is independent of the rubric criteria the spike scored. They are codified in the project’s Architecture Decision Record (`solo-founder-crew/docs/adr/0001-framework-choice.md`).

(D1) Production HITL is built-in. LangGraph’s `interrupt()` primitive, paired with a checkpoint, is exactly the founder gate the HITL Contract describes. The graph pauses; state is persisted; the founder responds out-of-band (minutes, hours, days, from a different process); the graph resumes from precisely where it stopped. The custom spike’s HITL is a clean in-process `await hitl.review()` call — adequate for a single synchronous run, but extending it to pause across processes would require building, in effect, what LangGraph already provides.

(D2) Cross-session memory is built-in. LangGraph ships a checkpoint abstraction with a SQLite reference implementation that serialises the run state to disk between interrupts. The solo-founder use case is fundamentally stateful across days and sessions; the framework needs persistent memory by default, not only for production but for the Chapter 6 case-study runs that may span multiple sittings. Building this on top of the custom runtime would duplicate LangGraph’s existing contract.

(D3) The graph topology is data. `app.get_graph().draw_mermaid()` emits the Author Flow diagram (Figure 4.4 in Chapter 4) as Mermaid, directly from the compiled graph. That figure is therefore *machine-generated by the framework being described* and cannot drift from the code it describes. The custom spike has no equivalent — any diagram would have to be drawn by hand and maintained out-of-band.

(D4) DSR defensibility. Hevner et al. (2004) and Peffers et al. (2007) do not penalise building on existing artefacts; they ask the candidate to demonstrate that the proposed framework is *constructible*. Constructing the framework on top of LangGraph is more defensible — and easier to articulate in the viva — than presenting a from-scratch runtime whose primitives ad-hoc resemble LangGraph’s. The contribution of this thesis is the **framework abstractions** (Role Library, HITL Contract, Crew Generator), not the underlying orchestration substrate.

A potential counter-reading is that *Custom scored higher (69 > 67), therefore building from scratch is the rigorous DSR choice*. The rebuttal is twofold. First, the rubric scored Custom highly because each primitive was tailored to the thesis vocabulary — Custom did not beat LangGraph on neutral terms, it beat LangGraph by exploiting the fact that the scoring rubric and the implementation came from the same author. Second, the four tie-breaker considerations above are *not absent* from the rubric by accident: they were excluded *as scored criteria* because they bear on production readiness rather than spike ergonomics, but they remain decisive for a thesis whose artefact must be both demonstrable in the sandbox (Chapter 6) and plausibly extensible to production. The decision therefore stands: **LangGraph is adopted as the Orchestration Runtime substrate**.

5.1.7 What carries forward from the custom spike

Custom’s high score reflects that the spike’s primitives — Role, DecisionRights, ToolRegistry, HITLContract, TraceEvent — were designed for the thesis vocabulary. Those primitives survive the substrate choice intact. They become the framework package (§§4.4–4.8); LangGraph supplies the underlying graph executor, the interrupt()-based founder gate, and the checkpoint. The discard from the custom spike is the asyncio orchestration scaffold itself, which is replaced by LangGraph’s StateGraph.

This separation — abstractions from the thesis, substrate from upstream — is the architectural pattern §4.7 describes in detail. It is also the reason the rubric scored “Lock-in & cost” at weight 1 (the lowest): the framework’s primitives are substrate-agnostic Protocols, and a future migration away from LangGraph would touch the orchestration module only, not the Role Library or the HITL Contract.

Box 5.1 — Substitution friction (the unscored cost catalogue) This box catalogues, for each candidate, the concrete cost of substituting a deterministic stub LLM for the framework’s normal model binding. The friction is not scored as a rubric criterion — it is recorded because it surfaces *coupling* between the framework and its LLM contract, which the rubric criteria for Lock-in (C7) and Dev experience (C6) judge in aggregate but not in detail. **Custom — 0 items.** The LLM is a one-method Protocol the spike owns: `LLMClient.complete(system, user) -> LLMResponse`. The stub implementation is a 30-line class with a response queue and a counter. **LangGraph — 1 item.** The graph’s nodes call `state["llm"].invoke(messages).content`. A 25-line duck-typed class satisfies the contract. No subclassing, no pydantic boilerplate, no output-format coupling. The single footgun: a Python namespace collision when the spike folder is literally named `langgraph/` — inserting `spikes/` at `sys.path[0]` makes Python treat `spikes/langgraph/` as a namespace-package fragment that shadows the installed `langgraph`, causing from `langgraph.graph import ...` inside the spike’s own `graph.py` to recurse into itself. The fix is to append rather than prepend to `sys.path`. One-time cost, documented inline. **CrewAI — 4 items.** (1) Subclassing `crewai.LLM` to inject a stub requires inheriting from a class with ~25 constructor arguments unrelated to the stub’s purpose. (2) The parent constructor validates the `model` string against the `litellm` provider registry even when the subclass overrides every call, requiring a sentinel provider name. (3) `CrewAI`’s agent executor parses LLM output for `"Thought: ...\\nFinal Answer: ..."`; plain responses are rejected and the executor re-prompts, blowing the call budget on parse failures. The stub must wrap canned text in that scaffold. (4) Even with `max_iter=1` on the Agent, the executor may make multiple LLM calls per task. The first run of the `CrewAI` spike crashed with a `QueueExhausted` exception from the stub queue after the executor made an unexpected third call. The exception itself was designed into the stub as a diagnostic for exactly this phenomenon. The friction items track the rubric: zero on Custom, one trivial item on LangGraph, four on CrewAI. The direction of the evidence is consistent with the scored result.

5.1.8 Real-LLM voice (qualitative)

The unscored smoke test ran each spike once against claude-haiku-4-5. Same model. Same Venture Brief (scenarios/fixtures/passly_brief.json). Same scripted founder feedback (verbatim from scenarios/fixtures/founder_responses.json: “Καλό draft αλλά πολύ formal. Να ζεστάνει το άνοιγμα — μίλα σαν σε φίλο μαγαζάτορα. Πρόσθεσε ένα μικρό ελληνικό tagline στο τέλος. Κράτησέ το κάτω από 120 λέξεις.”). The three resulting drafts (preserved at solo-founder-crew/docs/spike-evidence/) differed noticeably in register despite identical inputs:

Custom (raw anthropic SDK):

“Passly is here. [...] No spammy vibes. No agencies. No learning curve. Early access starts now. Let’s get your regulars back through the door. Passly — to loyalty σου, απλά.”

Warm English with a closing Greek tagline. Utility framing (“the loyalty that works”).

CrewAI (via litellm + Agent backstory):

“Look, you’re busy running your café, salon, or shop. Your customers already have their wallet open—literally. [...] No marketing degree, no agency bills, no new software to learn. [...] Απλά και δυνατά.”

Punchy English in an agent-personality voice, slogan-style close.

LangGraph (via lan chain-anthropic, system+user prompts):

“Passly είναι εδώ. [...] Τα loyalty passes σου ζουν ήδη στο Apple Wallet και Google Wallet τους — έτοιμα. [...] Passly. Για τους δικούς μας.”

Fully Greek, tightest copy (≈75 words versus 130–140 for the others), community/in-group framing.

These are *framework-induced* differences: the same model, the same brief, the same founder feedback, three voices. The variance comes from each candidate’s default scaffolding around the LLM call — CrewAI’s backstory field steering the model toward an agent-personality register; the raw SDK’s bare system prompt producing a more neutral voice; LangChain’s ChatAnthropic binding defaulting to a register Haiku produces more often in Greek-first phrasing.

This observation does not enter the framework-selection decision — output quality was excluded from the rubric for the reasons given in §5.1.4. It does, however, inform Chapter 7’s discussion of *framework-induced register drift* and the limits of substrate neutrality. The framework chosen for this thesis (LangGraph, langchain-anthropic binding) carries its own register prior; that prior is documented and reproducible.

5.1.9 Summary

The Part 7 spike treated framework selection as an empirical question and answered it via a construction spike whose method, rubric, scores, and evidence are all preserved in solo-founder-crew/docs/. CrewAI was eliminated on the rubric (32 of 75) with four documented substitution-friction items confirming the direction of the result. Custom (69) and LangGraph (67) finished within the rubric’s resolution; the tie broke

toward LangGraph on four non-rubric considerations — built-in HITL via `interrupt()`, built-in checkpoint-backed memory, machine-generated topology diagrams, and DSR defensibility — each independent of the spike’s scored ergonomics. The framework’s abstractions, designed during the custom spike, carry forward unchanged. LangGraph supplies the substrate.

The remainder of this chapter (§§4.4–4.8) describes each of the framework’s five components in detail. Section 4.9 demonstrates the assembled framework end-to-end on the Passly case study. Section 3.3 defines the evaluation metrics for Chapter 7.

5.2 The framework and application stack

The substrate decision of §5.1 fixed one tool. The remaining tools were chosen conventionally, and this section records each in the form the chapter owes — capabilities, reason for choice, use — grouped by the layer they serve. Table 5.3 gives the summary; the paragraphs supply the reasons.

Table 5.3: Development tools of the framework (*solo-founder-crew*) and the application (*passly*), with the alternative each was chosen over.

Layer	Tool	Capability used	Chosen over	Used in
Orchestration substrate	LangGraph 1.0 (LangChain)	Typed <code>StateGraph</code> , conditional edges, <code>interrupt()</code> , checkpoint, graph rendering	CrewAI 0.108; custom <code>asyncio</code> runtime (§5.1)	Orchestration Runtime (§4.7)
Model binding	<code>langchain-anthropic</code> / Anthropic SDK; Claude Haiku 4.5	Chat completion with per-call token usage	Vendor-neutral by design: any client satisfying <code>LLMClient</code>	AnthropicLLM; every live run in Chapter 7
Deterministic model	MockLLM (own)	Ordered-queue stub; bit-identical runs, no network, no key	Recorded-replay cassettes	Test suite; spike scoring; offline demonstration
Run persistence	LangGraph <code>MemorySaver</code> / <code>SqliteSaver</code>	Checkpoint the graph state by thread id; cross-process resume	Custom serialisation of run state	Founder gate pause/resume (§4.8.4)
Schemas	JSON Schema	Machine validation of the Venture Brief and the HITL request; figure generation from the schema	Pydantic-only models without a language-neutral artefact	§4.4, §4.8.3
Founder surface (framework)	Discord bot API	Channels, message components (buttons), modals, push notifications	Slack; a bespoke dashboard	DiscordHITL (§4.8.6)
API	FastAPI + Uvicorn (Python 3)	Typed HTTP routing, pydantic models, <code>async</code> request handling	Flask; Django REST	<i>passly</i> /api (§6.3)
Web	Next.js 16 (React, TypeScript)	Client components, server-side rewrite of <code>/api</code> to the API origin, single public URL	Plain React SPA; server-rendered templates	<i>passly</i> /web (§6.3, §6.7)
Database	PostgreSQL on Supabase	Managed Postgres; row-level security; direct owner connection	SQLite (the original prototype)	Repositories (§6.8)

Layer	Tool	Capability used	Chosen over	Used in
			store, migrated away)	
Pass issuance	Apple Wallet (.pkpass), PKCS#12 certificate, WWDR G4 chain, CMS detached signature	Third-party-adjudicated artefact installed on a customer's phone	Google Wallet (argued as a second PassIssuer, not built)	§6.4
Email	Resend API	Transactional email carrying the pass link	SMTP; SendGrid	EmailSender port (§6.8)
Hosting	Vercel (Next.js project + serverless FastAPI function), custom domain passly.gr; Supabase transaction pooler	Public HTTPS deployment of both tiers under one domain; framework installed from its repository at build time; signing material as environment variables	A single VPS; cloudflared tunnel from the laptop (used during development)	§6.3.1, §6.9.2
Tests and lint	pytest, httpx, ruff	Unit and HTTP tests with in-memory fakes; a single lint rule set	unittest; flake8 + black	Both repositories; CI
Integration	GitHub Actions	CI without the private framework installed; Discord webhook on failure	GitLab CI	passly CI (§6.3.4)

Orchestration and model. LangGraph's capabilities and the reasons for choosing it are the subject of §5.1 and are not repeated; its use is the Orchestration Runtime of §4.7, where the graph, the `interrupt()` gate and the checkpointer are each described in place. The model binding is deliberately thin. The framework owns a one-method `LLMClient Protocol`, and `AnthropicLLM` — exposed under the `RealLLM` alias — is one implementation of it that also records `usage.input_tokens` and `output_tokens` on each call, which is what made the cost dimension of the evaluation (E5) readable from the run record. Claude Haiku 4.5 (`claude-haiku-4-5`) was the model of every live run in both conditions of Chapter 7 because the comparison is between operating models, not between models, and the cheapest capable model made ten scenarios in two conditions affordable. The choice of vendor is not a commitment: any object exposing the Protocol serves, and the deterministic `MockLLM` is the proof — it is the reason the entire test suite, the spike scoring and the offline demonstration run with no key and no network.

Persistence and schemas. The checkpointer is LangGraph's own, used through its `BaseCheckpointSaver` abstraction so that the in-process `MemorySaver` and the file-backed `SqliteSaver` are interchangeable by configuration; the state-as-data discipline of §4.7.3 is what makes that substitution trivially safe. JSON Schema was chosen for the two shared contracts — the Venture Brief and the HITL request — over Python-only models because a language-neutral artefact can be validated by the framework, rendered as a figure, and read by an adapter written in another language.

The application stack. Passly's API is FastAPI, chosen for typed routing and native `async`, which the framework's `await crew.author_flow()` requires; its hexagonal layout (§6.3.2) is a design choice, not a framework feature, and would survive a change of web framework. The front end is Next.js, chosen for one operational property above others: its server-side rewrite lets a single public URL serve both the

web and the API, so a phone opening the enrolment page needs no CORS configuration — an arrangement that served the tunnelled development loop and the deployed application alike. Persistence moved from SQLite to PostgreSQL on Supabase during development; the migration script preserves member identifiers and pass serial numbers so that already-issued passes keep working, and row-level security is enabled without policies because the API connects as the table owner and never through a public gateway. Apple Wallet is not a tool in the ordinary sense but an adjudicator: the signing chain — a PKCS#12 Pass Type certificate, Apple’s WWDR G4 certificate, and a detached CMS signature over the manifest — is reported in §6.4 because a pass a stranger’s phone accepts is the demonstration’s least self-assessed artefact. Resend carries the enrolment email. Hosting is Vercel for both tiers — the web as a Next.js project and the API as a serverless FastAPI function whose build installs `solo-founder-crew` from its repository with a scoped token, with `maxDuration` raised to sixty seconds for the crew run — behind the custom domain `passly.gr`, with Postgres reached through Supabase’s transaction pooler and the Apple signing material supplied as base64 environment variables (§6.4.4). Vercel was chosen over a single virtual server because two projects with a shared domain reproduce the local two-tier layout without operating a machine; its serverless model is also the origin of the one limit §6.9.3 records, that the in-memory campaign gate does not survive across function invocations.

Testing and integration. Both repositories use `pytest` with in-memory fakes behind the ports, so the default suite needs neither a database nor a model key; the Postgres adapter tests run only against a dedicated test database, since they truncate every table. Ruff supplies the single lint rule set that CI applies. The application’s CI installs a requirement set that excludes the private framework, which is possible only because every framework import is lazy and every framework type is reached through two Protocols — the green build is therefore a continuously enforced proof of the dependency boundary argued in §6.3.4, and the Discord webhook that reports a failed build is the founder’s own operating surface being used for the venture’s engineering as well as its marketing.

5.3 Summary

One tool was chosen empirically and the rest conventionally, and the chapter has tried to keep the two kinds of decision distinct. The substrate spike of §5.1 eliminated CrewAI on a pre-registered rubric, found the custom runtime and LangGraph within the rubric’s resolution, and broke the tie toward LangGraph on four considerations — built-in interrupts, built-in checkpointing, machine-generated topology, and design-science defensibility — that bear on production readiness rather than spike ergonomics. The framework’s own abstractions, designed during the custom spike, carried forward unchanged onto that substrate. Around it, the tools of §5.2 were chosen for properties the demonstration needed — an async web framework, a single-URL front end, a managed database, serverless hosting under one domain, a third-party-adjudicated artefact, and a test suite that runs without credentials — and each is held behind a Protocol so that the demonstration’s claims do not depend on any one of them. Chapter 8 §8.3 returns to the question of whether the tools held up.

6. The System: the Passly Application

6.1 Introduction and chapter map

Chapter 4 delivered the thesis’s designed artefact: a five-component framework for composing and governing an AI-native crew around a solo founder. This chapter takes that artefact out of its own repository and puts it to work.

Within Design Science Research the move is a named one. Hevner et al. (2004) require that a design artefact be demonstrated in an appropriate context, and Peffers et al. (2007) place demonstration as a distinct activity preceding evaluation. This chapter is that step, and only that step: it shows the artefact instantiated and operating in a real venture. It does not measure how well it operates, against what, or by which criteria — those questions are Chapter 7’s, and §6.9.5 explains why answering them here would weaken both chapters.

6.1.1 Why a working application rather than a mock

A visual prototype would have illustrated the framework more cheaply and shown less. The claims Chapter 4 makes are architectural — that founder authority is a substitutable seam, that decision rights are enforced at dispatch, that the substrate is interchangeable — and none of them can be demonstrated by a rendering. They are claims about what happens when something else is built against the framework, so something else had to be built.

Three properties of the demonstration follow from that decision, and each is evidence a mock could not have produced. The application lives in a separate repository and consumes the framework as a dependency, so the boundary between them is enforced by a package manager and a continuous-integration configuration rather than by discipline. It issues cryptographically signed Apple Wallet passes, so its principal artefact is adjudicated by a third party with no interest in the thesis. And it implements a second adapter on the framework’s human-in-the-loop seam, which is the one thing Chapter 4 argued for and could not, from where it stood, prove.

6.1.2 The chapter’s shape

§6.2 introduces Passly as an instance of the Venture Brief schema, and shows the venture’s scope arriving as machine-readable constraint. §6.3 sets out the architecture — the three-tier chain, the ports-and-adapters layout, and the mechanisms that keep the framework a dependency rather than a fork. §6.4 covers pass issuance and the signing chain, and argues why signing is an evidential matter rather than a technical aside. §6.5 reports the crew inside the application: one role, enforced decision rights, and a Venture Brief synthesised per shop.

§6.6 is the chapter’s centre of gravity. It reports `_WebHITL`, a second adapter on the HITL Contract, and what changed in the framework to accommodate it, which was nothing. §6.7 walks the demonstration end to end, from an empty database to a stamped pass in a customer’s Wallet, and §6.8 covers persistence and the member model. §6.9 states plainly what the demonstration establishes and what it does not, and §6.10 hands over to Chapter 7.

A reader with limited time should read §6.6.

6.2 Passly as instantiation

6.2.1 *Why this venture*

The framework of Chapter 4 is specified for solo-founder digital ventures, and the demonstration should be one. Passly is a one-person venture building digital wallet passes and AI-assisted marketing for Greek small and medium businesses. It is digital-only, which matters because Chapter 1 §1.1 places physical and hybrid business models explicitly out of scope; a demonstration venture with a warehouse would have tested the framework outside the boundary the thesis drew for it.

It is also the author's own venture, and that should be stated rather than discovered. The advantage is access — every design decision, constraint and failure is available to the chapter rather than reconstructed from an informant. The cost is that the venture and the framework share an author, so the demonstration cannot show that a stranger would have built the same application. What it can show, and does, is that the framework survived contact with requirements it did not anticipate: Apple's pass format, an HTTP request lifecycle, a customer's phone.

6.2.2 *The Venture Brief as data*

Component 1 is a schema for describing a venture to a crew, and Passly's own brief is an instance of it (`solo-founder-crew/scenarios/fixtures/passly_brief.json`). It is worth reading as the chapter's first concrete artefact, because it shows what the framework takes as input before any of the machinery runs.

The brief names a domain — marketing technology, geography Greece, languages Greek and English — and a stage, pre-launch. Its product block carries a one-liner and three value propositions, of which the first two are the product claims and the third is a market claim: made for Greek SMBs, in Greek, on Greek-relevant channels. The customer block names the segment, three jobs-to-be-done and three objections, and those objections are unusually load bearing for a data structure: *I don't have time to learn another app, my customers won't install anything, I tried SMS marketing once and it felt spammy*. A crew drafting copy is told what its audience will resist before it writes a line.

6.2.3 *Constraints as machine-readable scope*

The brief's voice and constraints blocks are where the venture's judgement becomes something the runtime can hold a role to.

`voice.dont` forbids hype words, forbids promising specific revenue-lift numbers, and forbids mentioning live customer pilots because the venture is pre-launch. `constraints.compliance_notes` requires GDPR conformance by default with no personal-data examples in marketing copy, and repeats the pre-launch rule as a compliance matter rather than a stylistic one. `constraints.budget` is zero euros per month.

`constraints.out_of_scope` carries two entries: physical and hybrid retail integrations beyond passes, and markets outside Greece for the initial launch. These are the thesis's own scope discipline, expressed as venture data. Chapter 1 §1.1 states that digital-only is the boundary; Chapter 4 §4.2 makes it design objective O7; and here it

is a JSON array that a role's system prompt is built from and that a founder gate can be used to enforce. The path from a scoping sentence in an introduction to a machine-readable constraint the crew is bound by is short and fully traceable, which is the property the framework was designed to have.

Two things follow. First, the brief is the single source of truth shared between this chapter's prose and the code, the same convention Chapter 4 §4.4 established for the schema and §4.8.3 for the decision envelope. Second, the same file is the input to the Chapter 7 evaluation's Condition F, so the venture described here and the venture evaluated there are the same venture, described once.

6.3 Architecture

Figure 6.1: fixes the structure the rest of the section describes the three tiers, the ports the API's core depends on, the adapters that satisfy them, and the single place where the framework is reached.

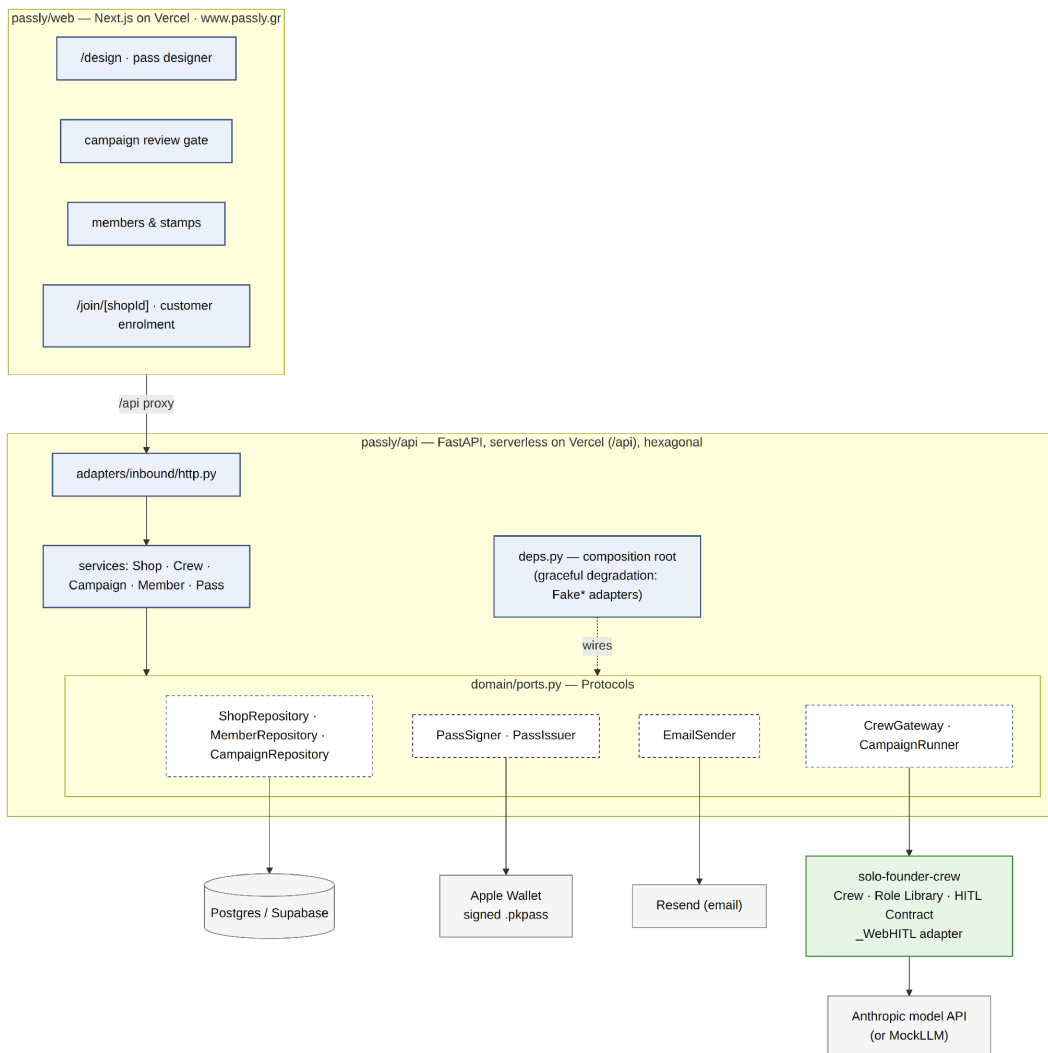


Figure 6.1: Component diagram of the Passly application — the Next.js web tier, the hexagonal FastAPI service — deployed as a serverless function under the same domain — with its eight ports (dashed), the outbound adapters, and `solo-founder-crew` consumed as a dependency behind the `CrewGateway` and `CampaignRunner` ports.

Chapter 4 specified a framework; this chapter reports what happened when that framework was instantiated inside a working venture. Within the Design Science Research cycle this is the demonstration step — the artefact exercised in an appropriate context (Hevner et al., 2004; Peffers et al., 2007) — and the architecture is where the demonstration’s terms are set. The question the architecture has to answer is narrow and consequential: can a real application consume the framework without deforming either one? An application that had to fork the framework, or a framework that leaked into the application’s core, would leave the Chapter 4 contribution unproven whatever else the demonstration showed.

6.3.1 The chain

Passly is three tiers, and each owns exactly one concern. A Next.js front end (`passly/web/`, served at <https://www.passly.gr>) renders the shop owner’s surfaces: the pass designer with its live preview, the campaign review gate, the members and stamps console, and the customer-facing join page. A FastAPI service (`passly/api/`, deployed as a serverless function behind the same domain and reached by the web tier under `/api`) owns pass issuance, persistence and orchestration. Behind it sits `solo-founder-crew`, consumed as a dependency, which drafts campaign copy and holds the founder gate.

The chain is deliberately linear — the web tier never reaches past the API to the framework, and the framework never knows a web tier exists. The application proves the chain is live at its least interesting point: `GET /crew/info` (`passly/README.md`) returns the framework’s role catalogue, so a single HTTP call confirms that Next.js, FastAPI and `solo-founder-crew` are all reachable in one deployment; the landing page surfaces the same check as an “API live” indicator. That endpoint carries no product value and exists purely as the demonstration’s smoke test, which is the honest reason to describe it here.

The application is deployed, and the screens this chapter describes are the ones a reader can open. Table 6.1 lists them; the demonstration shop is the one the walkthrough of §6.7 uses.

Table 6.1: The deployed surfaces of the Passly application — paths relative to <https://www.passly.gr> — and the section that describes each.

Surface	Path	Who uses it	Section
Landing page	/	Prospective shop owner; shows the “API live” check	§6.3
Pass designer	/design	Shop owner designs the pass with a live preview and creates the shop	§6.4, §6.7.1
Campaign review gate	/shops/{shopId}/campaign	Shop owner asks the crew for a launch announcement and approves, sends back or discards it	§6.5, §6.6, §6.7.2
Customer enrolment	/join/{shopId}	Customer enters a name and an email and receives the signed pass	§6.7.3
Members and stamps	/shops/{shopId}/members	Shop owner sees members and adds stamps	§6.7.4, §6.8

The demonstration shop, Chunky Cookie Bar, is `shop_sh5g5ekdv4v`, so its gate, enrollment and members pages are the three URLs above with that identifier substituted.

6.3.2 Ports and adapters

The API follows a hexagonal, ports-and-adapters layout, and the reason is specific rather than stylistic. `passly/api/app/domain/` holds the models, errors, identifiers and — in `ports.py` — the interfaces the core depends on, expressed as Python Protocol classes. It imports neither FastAPI nor `solo_founder_crew`. `passly/api/app/services/` holds the use cases and depends only on those Protocols. Everything that touches the outside world is an adapter: `adapters/inbound/http.py` is the FastAPI driving adapter, and `adapters/outbound/` holds the driven ones — the Postgres repositories, the Apple pass issuer and its signer, the email sender, and the two adapters that reach the framework.

Eight ports are declared in `passly/api/app/domain/ports.py`: `ShopRepository`, `MemberRepository`, `CampaignRepository`, `EmailSender`, `CrewGateway`, `CampaignRunner`, `PassSigner` and `PassIssuer`. The file’s own docstring states the lineage plainly — the arrangement “mirrors the substrate-neutral seams in `solo-founder-crew` (HITLContract, LLMClient)”. This is worth dwelling on, because it is the first evidence in the chapter that the Chapter 4 design generalises. The framework’s central architectural move is to name the points at which a component may be replaced and to hold everything else against those names. Passly, built to consume the framework rather than to illustrate it, independently arrived at the same move for its own dependencies: Apple Wallet sits behind `PassIssuer`, Resend behind `EmailSender`, Postgres behind three repository Protocols. The framework’s seam discipline turned out to be the application’s seam discipline.

Two of those ports are the framework itself, and the split between them repays attention. `CrewGateway` exposes the framework as a queryable catalogue; `CampaignRunner` drives a run and holds the founder gate. They are separate because they have different lifetimes. A catalogue lookup is stateless and can be constructed per request; a run has in-flight state, a pending gate and a background task, so `passly/api/app/deps.py` holds the runner as a singleton and constructs the gateway freely. `CampaignRepository` is a third, and distinct again: its docstring notes that it records *where a run got to* rather than driving it, and that separating the two is what would let `start`, `poll` and `respond` be served by three different processes, the runner rebuilding itself from the checkpoint while the gate is read from storage.

6.3.3 The composition root

Exactly one module knows which adapter backs which port: `passly/api/app/deps.py`, described in its own docstring as the composition root. Services receive their dependencies as constructor arguments and are never in a position to choose them. The practical consequence is that the application’s behaviour under missing configuration is a property of one file, and it is a designed property rather than an accident. When no Apple .p12 and WWDR certificate are present, `_pass_signer()` returns `FakeSigner` and the application emits a structurally valid but Wallet-unacceptable bundle. When `RESEND_API_KEY` is absent, `_email_sender()` returns `FakeEmailSender` and logs a warning that no mail will be sent. When no

Anthropic key is present, the crew runner supplies `MockLLM` rather than `AnthropicLLM` (`passly/api/app/adapters/outbound/crew_runner.py`).

Each of those substitutions is a port exercised in anger. The framework's claim in §4.7.7 is that its service-substitution points are real; Passly's own ports make the parallel claim for the application, and the evidence that both hold is that the demonstration runs — in a degraded but coherent form — with none of the three credentials configured. The `PassSigner` Protocol even carries a `real` flag so callers can tell a signed bundle from an unsigned one, which is the sort of detail that only appears when a substitution is genuinely used rather than merely permitted.

6.3.4 A dependency, not a fork

The framework is consumed from a separate repository and is never vendored, patched or forked. Two mechanisms hold that line, and both are visible in the build rather than asserted in prose.

The first is lazy importing. Every framework import in `passly/api/app/adapters/outbound/crew_runner.py` and `crew_gateway.py` sits inside a method body rather than at module scope. The `crew_runner` docstring gives the reason: importing the module, and therefore the whole application, requires neither the framework installed nor an API key. The framework loads only when a crew call actually happens.

The second is the dependency split. `passly/api/` declares three requirement files: `requirements.txt` (FastAPI and Uvicorn), `requirements-dev.txt` (the base plus `pytest`, `httpx` and `ruff`), and `requirements-local.txt` (dev plus `solo-founder-crew` as an editable install). Continuous integration installs the second, and the test suite fakes the framework behind the `CrewGateway` and `CampaignRunner` Protocols. The `solo-founder-crew` repository is private, and the arrangement means CI never needs access to it.

That is a mundane packaging decision with a load-bearing architectural consequence. The application's test suite runs green against fakes, which is only possible because the framework is reached exclusively through two named Protocols. Had any framework type leaked into the services or the domain — a `Crew` in a signature, a `FounderDecision` in a model — CI would need the private repository and the leak would be immediately visible as a build failure. The green CI build against a framework-free dependency set is therefore not a convenience; it is a continuously enforced proof that the boundary Chapter 4 specified is intact in an application that was not written to flatter it.

6.3.5 What the architecture makes possible

The remainder of the chapter rests on this layout. Pass issuance (§6.4) is a port with one Apple implementation and an argued Google Wallet sibling. The crew inside the application (§6.5) reaches the framework through `CampaignRunner` alone. The second HITL adapter (§6.6) — the section this chapter turns on — is an adapter in exactly the sense used here, and the fact that it required no change to `Crew`, `Role` or tool code is a claim about this architecture as much as about the framework's.

6.4 Pass issuance

A wallet pass is the one artefact in this demonstration that a stranger’s phone either accepts or rejects. That property is why pass issuance is worth a section of its own: it is the point at which the application stops being self-assessing. Apple’s Wallet applies its own criteria, and no amount of internal test coverage substitutes for a pass that installs on a customer’s device.

6.4.1 *The port and the platform-neutral result*

Issuance sits behind the PassIssuer port (`passly/api/app/domain/ports.py`), and the use case above it — PassService in `passly/api/app/services/pass_service.py` — is platform-neutral by construction. Its own docstring states the boundary: it “knows nothing about Apple vs Google — swapping the issuer adapter is all it takes to add a platform.” The service fetches a shop, and for a customer pass a member, then delegates.

What comes back is an IssuedPass (`passly/api/app/domain/issued_pass.py`): content, media_type, filename and platform. The neutrality is real rather than nominal, and the docstring explains how it survives a platform whose delivery model differs — Apple returns .pkpass bytes as a file download, while a Google Wallet issuer would return a “Save to Google Wallet” link, and the transport layer honours media_type and filename accordingly. Designing the return type so that a link and a byte payload are the same kind of thing is what makes the Google Wallet claim in §6.9 a substitution rather than an aspiration.

6.4.2 *What the Apple adapter knows*

Everything Apple-specific lives in one file, `passly/api/app/adapters/outbound/apple_pass_issuer.py`: the pass.json shape, the storeCard and coupon style keys, the per-file manifest, the detached signature, and the branded artwork. The domain and the services never see any of it.

Two rendering decisions in the adapter are worth recording, because they show the adapter designed against Wallet as it renders on a device rather than against the specification alone. First, the adapter supplies two distinct artworks rather than one image scaled twice: the square icon slot receives the round badge, and the wide logo strip receives the horizontal wordmark, so each slot carries the mark that reads at its own proportions. Second, the wordmark is laid out in an 80×27-point box, deliberately narrower than the 160×50 points the format allows, because Wallet places logoText - the venue title - beside the logo at a type size the pass cannot control; reserving that room is what keeps the venue's name fully legible on the pass. Both decisions were validated on the installed pass, and both are examples of the same discipline: the specification fixes what a pass may contain, but only the rendered pass fixes what a customer will see.

The colour conversion is a smaller instance of the same thing: `_hex_to_rgb` turns the designer’s #0B5D3B into `rgb(11, 93, 59)`, the form `pass.json` requires. The web designer and the wallet pass therefore share one colour value expressed two ways, which is what allows the live preview in §6.7 to be faithful rather than approximate.

6.4.3 Box 6.1 — the signing chain

A `.pkpass` is a ZIP archive whose integrity rests on two files. The adapter assembles `pass.json` and four images, then builds `manifest.json` as a map from each filename to the SHA-1 digest of its bytes. That manifest is the only input to the signature.

`AppleP12Signer` (`passly/api/app/adapters/outbound/signer.py`) loads the Pass Type ID certificate and its private key from a PKCS#12 file, loads Apple’s Worldwide Developer Relations certificate — G4, in DER form at `passly/certs/AppleWWDRCA4.cer` — and produces a detached PKCS#7/CMS signature over `manifest.json`, SHA-256, DER-encoded, with the WWDR certificate carried in the chain. The archive then contains the assets, the manifest and the signature file.

The verification a phone performs runs the chain in reverse: the signature must verify against a certificate chaining to Apple’s WWDR root, and each file’s digest must match the manifest. Altering a single byte of `pass.json` after signing invalidates the manifest digest and the pass is refused.

6.4.4 Why signing matters to the demonstration

The framework’s claims in Chapter 4 concern orchestration and founder authority, not wallet formats, so it is fair to ask why the signing chain earns this much space. The answer is about what kind of evidence the chapter offers. A demonstration whose outputs are only ever inspected by their author proves less than one whose outputs are adjudicated by an indifferent third party. Apple Wallet is such a party: it applies Apple’s criteria to the bundle, without regard to the thesis, and either installs the pass or refuses it.

The configuration under which the demonstration runs is therefore worth recording precisely. The Pass Type identifier is `pass.com.passly` and the team identifier `AVN9H8BY3X`; `APPLE_CERT_P12` and `APPLE_WWDR_CERT` resolve to files present in the local checkout, and in the deployed service the same material arrives as the base64 environment variables `APPLE_CERT_P12_B64` and `APPLE_WWDR_CERT_B64`, which `deps.py` materialises to a temporary file — either is the condition under which `passly/api/app/deps.py` selects `AppleP12Signer` over `FakeSigner`, and without it a builder’s checkout would silently produce passes Wallet rejects. Customers joining through `/join/[shopId]` receive a real signed pass into Apple Wallet (`passly/docs/demo-runbook.md`).

One provenance note belongs here rather than in a footnote: for the purposes of this thesis, the Apple Developer enrolment used for that certificate is that of a company the author works with, not of a separate Passly legal entity. Nothing in the demonstration turns on the distinction — the signature is valid either way — but the boundary between the demonstration’s apparatus and a going concern is one §6.9 draws explicitly, and this is the first place it appears.

The graceful-degradation path described in §6.3.3 is the counterpart to all of this. With no certificate configured, `FakeSigner` returns an empty signature and the application produces a structurally valid archive that Wallet will not accept. That the `PassSigner` Protocol carries a `real` flag, so callers can distinguish the two, is the

design admitting in code that an unsigned bundle is a different kind of object from a signed one.

6.4.5 The template pass and the member pass

The same adapter issues two passes that differ in what they identify. The template pass, served at GET /shops/{id}/pkpass, takes the shop identifier as its serial number and carries no stamps; it is what the founder previews. A member's pass, at GET /members/{id}/pkpass, carries the member's unique serial number, their current stamp count rendered as Σφραγίδες against the design's goal, and their name in the design's secondary slot. Figure 6.2 shows the member pass as the customer holds it.

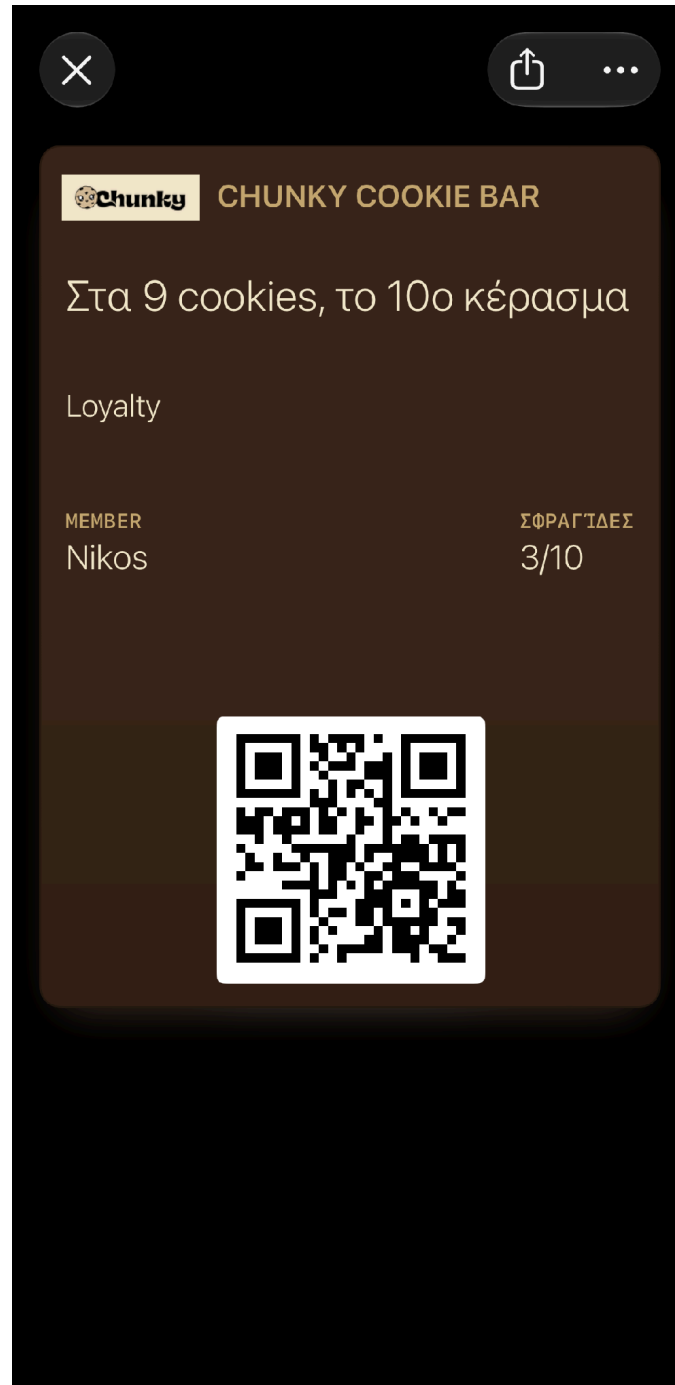


Figure 6.2: The member pass installed in Apple Wallet. The stamp count (Σφραγίδες, 3/10) and the member's name occupy the slots this section describes, and the QR carries the member's serial number. The render is Apple Wallet's own — the adjudication §6.4.4 argues for, made visible.

The barcode is where the distinction does operational work. On a member's pass the QR message is that member's serial number, so scanning it at the counter identifies exactly whose card is to be stamped; the template falls back to the design's own barcode message or the shop identifier. The stamp console of §6.7 depends on that choice, and §6.8 returns to the serial number as the member model's identity anchor.

The artwork in both is the Chunky Cookie Bar badge and wordmark, used with the shop's permission, and bundled for a single-shop demonstration; per-shop artwork is a generalisation of the code comments name and §6.9 records.

6.5 The crew inside the application

Chapter 4 describes five components. The Passly application exercises three of them directly — the Venture Brief Schema, the Role Library, and the Orchestration Runtime — and reaches the fourth, the HITL Contract, through the adapter of §6.6. This section reports how the first three appear in use, and is explicit about the one that does not appear at all.

6.5.1 *One role, selected directly*

The application does not invoke the Crew Generator. `SoloFounderCrewRunner.start` (`passly/api/app/adapters/outbound/crew_runner.py`) imports `make_marketing` from the framework’s Role Library, constructs a single role from the brief, and passes `roles=[role]` to the Crew constructor. The `CrewGenerator` class in `solo-founder-crew/src/solo_founder_crew/crew_generator.py` is not called anywhere in the application.

This should be reported as what it is rather than dressed up. The Crew Generator’s purpose (§4.6) is to select a crew subset from a Venture Brief when the appropriate composition is not obvious. Passly’s campaign flow has exactly one task type — draft a launch announcement for a shop — and therefore exactly one appropriate role. Hard-coding a selection whose outcome is known is the honest engineering choice, and inserting a rule engine to arrive at a foregone conclusion would have made the demonstration look more thorough while proving less. Component 3 is exercised in Chapter 4 §4.6.3 and §4.9.4, where the composition genuinely varies across the venture lifecycle, and it runs in the framework’s own live deployment: the Discord crew daemon (`solo_founder_crew.app`) composes the venture’s crew by calling the `CrewGenerator` on the Venture Brief each time it assembles at start-up. What remains open is a systematic validation of that composition in the live deployment, recorded as a follow-up on the framework’s roadmap rather than claimed here. This application does not exercise the component, and §6.9 records both halves.

The Role Library is nonetheless present in the application at two points. The runner constructs the marketing role from it, and `SoloFounderCrewGateway` (`passly/api/app/adapters/outbound/crew_gateway.py`) reads `ROLE_LIBRARY` to serve the role catalogue at `GET /crew/info` — the smoke check of §6.3.1.

6.5.2 *A shop as a venture*

The most interesting thing the application does with Chapter 4’s components is its treatment of the Venture Brief. `_brief_for` (`crew_runner.py`) does not hand the crew Passly’s own brief. It synthesises a fresh `VentureBrief` for *the shop* — Chunky Cookie Bar becomes the venture, its offer becomes the product one-liner, its customers become the segment, and its city, where known, narrows that segment.

The voice block is authored for a neighbourhood business rather than for a marketing-technology startup: a tone of “warm, plain Greek, confident”, a directive to “speak like a neighbourhood shop” and to “keep it short”, and prohibitions on hype words and on implying existing customers. The constraints carry out_of_scope entries for physical retail expansion and non-Greek markets.

The recursion here is worth naming, because it is a property of the framework rather than of Passly. Component 1 is a schema for describing a venture to a crew; Passly is a venture, and each shop Passly serves is also a venture. The application instantiates the schema one level down from itself, per shop, at run time. Nothing in the framework anticipates or forbids this — a brief is data, and the runner assembles one from a database row. That a schema written to describe the founder’s own venture composes cleanly when applied to the founder’s customers’ ventures is a small piece of evidence for its generality, and it cost the application nothing to obtain.

One detail records the schema working as a contract rather than as a suggestion. The brief schema requires a `venture_id` matching `^[a-z][a-z0-9-]*$`; Passly’s internal identifiers use underscores; so the runner maps one to the other with a comment saying exactly that (`crew_runner.py`). A schema that is merely documentation does not produce that line.

6.5.3 Decision rights, and a gate that is actually shut

`make_marketing` declares the marketing role’s decision rights in its system prompt and its `DecisionRights`: it may `draft_content` and `revise_content`, and must escalate `final_approval_before_publish`.

The application supplies the matching half. `_publisher_registry` (`crew_runner.py`) builds a `ToolRegistry` and registers a `publisher_tool` with `escalates="final_approval_before_publish"`. The framework’s registry documents the conjunction this creates: invoking a tool registered with `escalates` requires the calling role to carry that action in its `must_escalate` set *and* the runtime to have supplied escalation evidence — declared at `ToolRegistry.register` and enforced at `ToolRegistry.invoke`, which refuses dispatch otherwise (`solo-founder-crew/src/solo_founder_crew/tools.py`). Both halves are present and they name the same action, so the publish step in the application is gated by construction rather than by convention. This is design objective O1 (§4.2) observable in a running application: the founder’s authority is not a policy the code is asked to respect, it is a precondition the dispatch refuses to proceed without.

What the tool then does must be stated plainly, because the gate and its effect are different things. Passly’s `publisher_tool` is a stub — an `async` function returning `shipped:{n}chars`. The escalation is real, the approval is real, the routing is real, and the publication is a no-op. Chapter 4’s own marketing role holds outbound tools that post to Discord and send mail through Resend; the application does not wire those into the campaign flow, and §6.9 lists the omission rather than letting the word “publish” carry an implication the code does not support.

6.5.4 The substrate, switched by configuration

`_make_llm` (`crew_runner.py`) returns `AnthropicLLM` with `claude-haiku-4-5` when an API key is present and `MockLLM` otherwise, with the mock pre-loaded with a Greek draft and a revision so the demonstration runs offline. The framework’s `LLMClient` seam (§5.1.7, §4.7.7) is therefore exercised by the application in the ordinary course of development rather than in a test: the same code path serves a live model and a deterministic stub, and the choice is made by the presence of an environment variable.

The mock supplies three responses for a `max_revisions=2` run — an initial draft and up to two revisions — which is the application choosing its own cap, as §6.6.5 records.

6.5.5 *What this section does not show*

The crew here is one role, one flow, one task type, driven by a synthesised brief, publishing through a stub. It shows the framework’s role typing, its decision-rights enforcement, its brief schema and its substrate seam all functioning inside an application built by someone consuming them. It does not show multi-role coordination, crew composition under uncertainty, or a publication with an external effect. Chapter 4 demonstrates the first two on synthetic scenarios; the third is named among the demonstration’s limits in §6.9.

6.6 WebHITL — the second adapter

This is the section the chapter exists for. Everything preceding it establishes that Passly is a real application consuming the framework at arm’s length; everything following it describes what that application does. This section reports the one result that bears directly on the Chapter 4 contribution.

6.6.1 *The claim Chapter 4 left open*

Chapter 4 §4.8 argues that founder authority in the framework is a named, substitutable seam rather than wiring — that the surface through which the founder is asked can change without touching the runtime, the roles, or the tools. The framework’s own source states it as flatly as the chapter does: the contract “is the only API the runtime knows about”, and “swapping implementations does not touch Crew, Role, or Tool code” (`solo-founder-crew/src/solo_founder_crew/hitl.py`).

Chapter 4 supported that claim with three implementations — `ScriptedHITL`, `InteractiveHITL` and `DiscordHITL` — but all three were built inside the framework repository by the framework’s author, which is the weakest position from which to claim substitutability. §4.8.5 is candid about the gap: further surfaces, “a web UI, SMS, or email callback”, are “argued as same-protocol substitutions rather than built”, on the reasoning that each would add engineering without strengthening the claim.

Passly builds the web UI. It was built in a different repository, for a different purpose, against the contract’s published surface rather than its internals. That is the difference between an architectural assertion and an empirical result, and it is why this section carries the chapter.

6.6.2 *_WebHITL*

The adapter is `_WebHITL`, at `passly/api/app/adapters/outbound/crew_runner.py`, with its `review` method at line 40. It is thirty-odd lines. Its docstring states its own role without ceremony: it “implements the framework’s `HITLContract` over HTTP request/response”.

The mechanism is a split between two halves of an HTTP conversation. `review` receives the framework’s `HITLRequest`, writes a `CampaignState` marked

awaiting_review into a dictionary the web tier polls — carrying the turn number, the artefact’s content, and the actions the founder may take — then creates an `asyncio.Future`, stores it under the run’s `thread_id`, and awaits it. The coroutine parks there. Some time later, a founder clicks a button in the browser, `POST /campaigns/{thread_id}/respond` reaches `SoloFounderCrewRunner.respond`, and `resolve` pops the future and sets its result to a `FounderDecision`. The awaited review returns, and the framework’s Author Flow proceeds as though it had been answered synchronously.

The keying deserves a note because it differs from the framework’s own remote surface. `DiscordHITL` keys its pending futures by `request_id`; `_WebHITL` keys by `thread_id`, and the docstring gives the reason — gates are sequential within a run, so the thread identifier is a sufficient key. Two adapters on one contract made different, locally correct choices about correlation. A seam that admitted only one such choice would be a narrower seam than Chapter 4 claims.

6.6.3 What changed in the framework: nothing

The substantive finding is negative, and negative findings are easy to assert and hard to evidence, so the evidence is worth setting out.

`_WebHITL` does not subclass `HITLContract`. It does not import it. The framework type appears in review’s signature only as an unannotated parameter, carrying the comment `# noqa: ANN001 – framework type, imported lazily`. Because `HITLContract` is a `Protocol`, structural conformance is the whole requirement: a class with a correctly shaped `async def review` satisfies it. The second adapter on the framework’s founder-authority seam was therefore written without importing the seam it implements.

That is a stronger result than a subclass would have been. A subclass would show the framework’s extension point works for someone who read the framework’s source and inherited it from its base class. Structural conformance shows the contract’s *shape* is sufficient documentation — the adapter’s author needed the method signature and nothing else. It also means the boundary of §6.3.4 holds here at its most stressed point: `crew_runner.py` imports the framework only inside method bodies, and the single framework type it constructs is `FounderDecision`, imported inside `respond` (line 113) purely to package the founder’s answer.

Against that, the framework side of the seam is unchanged. `solo-founder-crew/src/solo_founder_crew/hitl.py` and `hitl_request.py` carry no Passly-specific accommodation, no web-shaped parameter, no branch. The integration point is one keyword argument — `hitl=self._hitl` in the `Crew` constructor at `crew_runner.py` — and the framework treats the web adapter exactly as it treats a JSON fixture.

6.6.4 Surface, not source of truth

Chapter 4 §4.8.6 states a design discipline for the Discord surface: the chat application is “a rendering and input surface only, never the source of truth”, because coupling orchestration state to a vendor would re-scatter the seam the contract exists to centralise. The web adapter honours the same discipline, and the honouring is visible in what `_WebHITL` does not contain. It holds no orchestration logic. It does not

decide whether a rejection routes to a revision or to exhaustion; that routing is the runtime's, and the adapter merely returns the founder's chosen action. The CampaignState it writes is a projection for the UI to render, not a state machine the UI drives.

The application's own port layer keeps that boundary explicit. `passly/api/app/domain/ports.py` declares CampaignRunner as the port that “drives the marketing crew ... and holds the founder gate”, and declares CampaignRepository separately as the thing that records “where a run got to” rather than driving it. The web tier reaches the framework through the first and never at all through the framework's own types.

6.6.5 What this establishes, and what it does not

It establishes that the HITL Contract is substitutable in the sense Chapter 4 claims. A second surface, in a second repository, with a different transport, different correlation key and a different concurrency model, satisfies the contract structurally and drives the unmodified Author Flow to completion. Design objective O1 (§4.2) — that the founder retains decisive authority through a named interface — survives instantiation in an application that had every opportunity to route around it and did not.

Three limits should be stated as plainly.

First, the pause is narrower here than the framework's. Chapter 4 §4.8.4 argues that a checkpointed run may wait an unbounded time, and the surviving process restarts with a persistent checkpointer keyed by `thread_id`. `_webHITL` holds its pending gates as in-memory `asyncio.Future` objects and its states in a dictionary, so a Passly gate survives an arbitrarily long wait *within* a process and does not survive a restart of it. The framework's property is unimpaired; the application simply has not taken it up. `passly/api/app/deps.py` wires a `PostgresCampaignRepository` for precisely this purpose and records in a comment that the crew runner does not yet consume it.

Second, the revision cap in the application is its own choice, not the framework's. `crew_runner.py` passes `max_revisions=2`, giving three drafts before exhaustion, where the Chapter 7 evaluation harness uses three revisions and four drafts. This is what a parameterised runtime looks like in use, but it means the application's gate behaviour and the evaluations are not identical and should not be read as one another's evidence.

Third, this is a small family, not a proof of universality. Since this section was first drafted the seam has in fact acquired a third implementation: the Chapter 7 evaluation harness's `RecordingHITL`, written for the scored runs without touching Crew, Role or tool code (`solo-founder-crew/evaluation/`). Three independent implementations — chat, web, and recording — none of which required a runtime change, is stronger evidence than one substitution; it is still not a demonstration that any conceivable surface would drop in. The claim the chapter is entitled to is that the seam admitted genuinely different implementations without modification — which is the claim §4.8 made and could not, from inside the framework repository, close.

6.7 The end-to-end walkthrough

The demonstration is a single continuous path from an empty database to a customer holding a loyalty pass that has been stamped to its reward. This section follows that path, naming the surface and the endpoint at each step, because a demonstration described only in the abstract cannot be checked.

6.7.1 Design

The founder opens `/design` (`passly/web/app/design/page.tsx`) and edits a `PassDesign`: the pass type, the logo text, the offer label and value, an optional secondary field, three colours, a barcode message and a stamps goal. `PassPreview` (`passly/web/components/PassPreview.tsx`) renders the pass beside the form as the values change. Figure 6.3 shows the designer mid-edit.

← PASSLY

Design your shop's pass

Shape the card on the right. The AI crew writes the launch campaign next.

SHOP

Shop name
Chunky Cookie Bar

City
Αθήνα

PASS

Pass type
Loyalty card Coupon

Logo text
CHUNKY COOKIE BAR

Offer label
Loyalty

Offer value
Στα 9 cookies, το 10ο κέρασμα

Secondary label
Member

Secondary value
—

Stamps goal (reward every N)
10

COLOURS

Background
#3B2417

Text
#F5E6C8

Label
#C9A86A

Live Preview:

Chunky LOYALTY

LOYALTY

Στα 9 cookies, το 10ο κέρασμα

MEMBER

—

Barcode

Save pass

Add to Apple Wallet

Figure 6.3: The pass designer at /design. The live preview and the issued pass read the same PassDesign fields, so what the founder previews is what the adapter signs.

The preview is faithful rather than illustrative because the designer and the issuer read the same fields. PassDesign (`passly/api/app/domain/models.py`) constrains what can be designed — the colours to a hexadecimal pattern, normalised to upper case; the logo text to forty characters; the stamps goal to between one and ninety-nine — and the Apple adapter of §6.4 consumes exactly those fields. The model's own docstring states the intent: the fields mirror what a real `.pkpass` needs “so P1 can turn a saved design straight into a signed pass without reshaping the data”. Saving posts to `POST /shops` and returns a `ShopRecord` with a server-assigned identifier.

6.7.2 The crew drafts, the founder decides

POST /shops/{id}/campaign returns 202 immediately with a CampaignState (passly/api/app/adapters/inbound/http.py). The run proceeds in a background task; the browser polls GET /campaigns/{thread_id} until the status becomes awaiting_review, at which point the response carries a ReviewGate with the turn number, the draft under review and the actions the founder may take. The founder answers at POST /campaigns/{thread_id}/respond with approve, reject and feedback, or kill, and the campaign review UI (passly/web/app/shops/[id]/campaign/) presents those three as the only choices. Figure 6.4 shows the gate as the founder meets it.

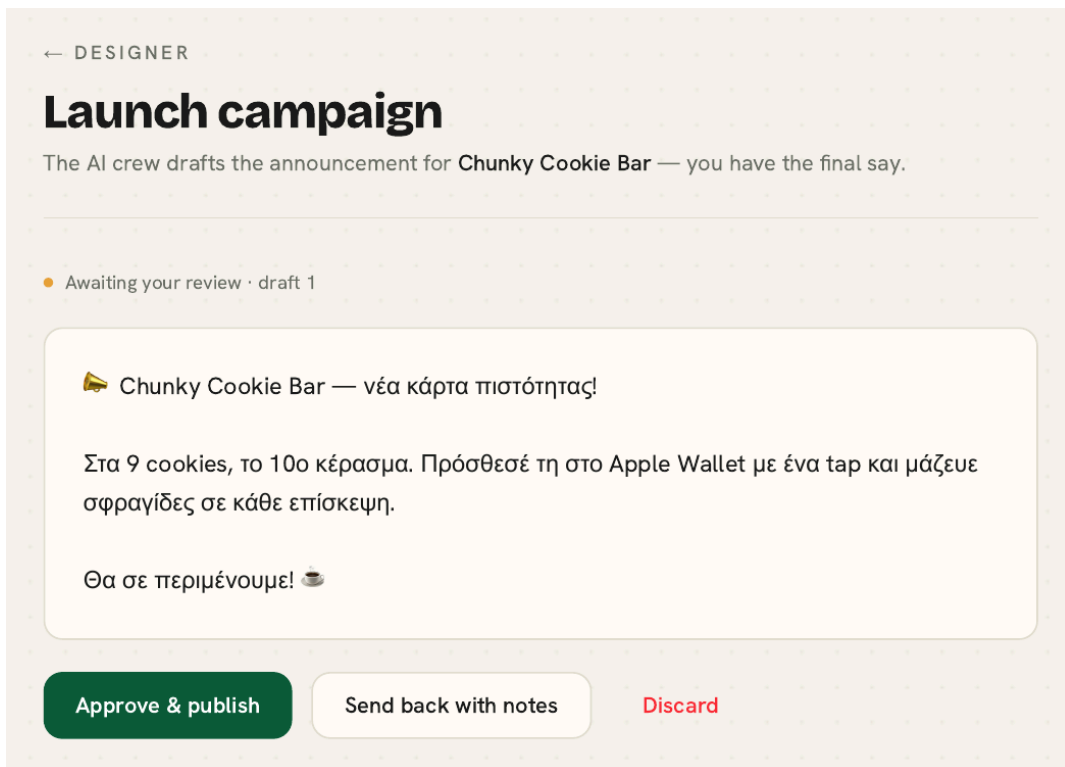


Figure 6.4: The campaign review gate at awaiting_review. Approve & publish, Send back with notes, and Discard are the web rendering of the contract’s three actions — approve, reject with feedback, kill — and the only choices the surface offers.

This is the §6.6 adapter seen from the outside: three endpoints implementing one review call. The asymmetry is the point. The framework’s contract is a single await; HTTP has no way to hold a request open for an unbounded human pause; so the adapter decomposes one awaited call into a start, a poll and a resolution, and the framework never learns that it happened.

CampaignStatus (models.py) is worth one observation. Its terminal three values — shipped, killed, exhausted — mirror the framework’s AuthorFlowResult.status exactly, and its docstring says so. That is what lets crew_runner.py write CampaignStatus(result.status) as a direct conversion. The alignment is deliberate and it is a coupling: the application has adopted the framework’s vocabulary for run outcomes at the level of string values. It is a small and legible coupling — a reader can see the whole of it in one enum — but it is not nothing, and it is the one place

where the framework's terms have entered the application's domain rather than staying behind a port.

6.7.3 The customer joins

The customer opens `/join/[shopId]` on a phone and submits a name and an email. `POST /shops/{id}/members` enrolls them and returns their `MemberRecord`, with the status code carrying the distinction: 201 for a new member, 200 for one who already exists and is re-downloading. `GET /members/{id}/pkpass` returns the signed pass with the member's own serial number and current stamp count, and Wallet installs it. Figure 6.5 shows the page as the customer sees it.

Figure 6.5: Enrolment at `/join/[shopId]`: a name, an email, and the pass — no application to install.

An email goes out on every enrolment, new or returning, carrying a link back to that same endpoint — Greek subject lines, Η κάρτα σου είναι έτοιμη 🍪 for a new member and Η κάρτα σου 🍪 for a returning one (passly/api/app/services/member_service.py). The send is best-effort by design: `_email_pass` catches every exception and logs a warning, because a mail failure must not break an enrolment the customer has already completed.

The customer's phone reaches the deployed application at `https://www.passly.gr/join/{shopId}`. Because the web application proxies the API under `/api`, one hostname covers both tiers — no CORS configuration, and the same arrangement served the local development loop, where `python scripts/dev.py --tunnel` gave the laptop a temporary public hostname before the application was deployed (passly/README.md).

6.7.4 Stamps and the reward

The shop stamps from `/shops/[id]/members`, which lists the shop's members and posts `POST /members/{id}/stamp`. `MemberService.add_stamp` reads the shop's `stamps_goal`, and on the stamp that reaches it grants a reward and resets the count to zero rather than letting it climb past the goal. The customer re-downloads their pass and the Σφραγίδες field shows the new count against the goal. Figure 6.6 shows the shop's side of that loop.

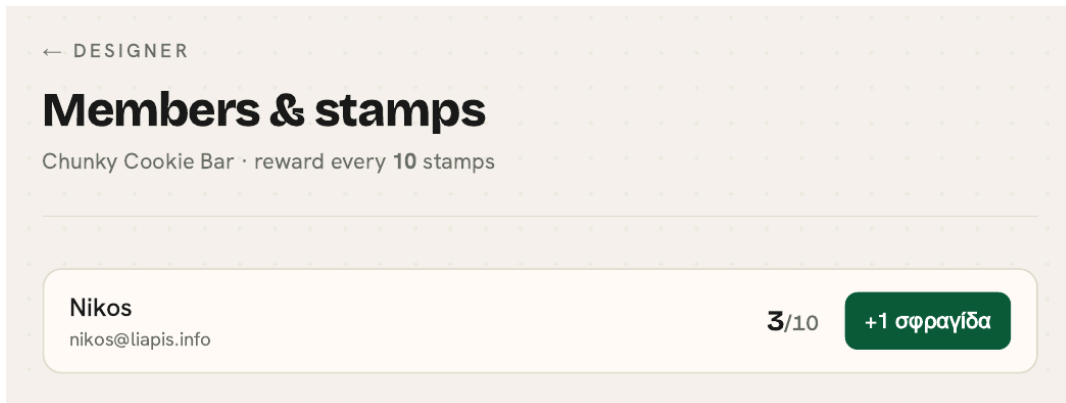


Figure 6.6: The Members & stamps view. The 3/10 shown here is the same member state Figure 6.2 shows rendered inside the Wallet pass — §6.8.3's loyalty state visible across two independent surfaces.

The barcode closes the loop. Because a member's pass carries their serial number as its QR message (§6.4.5), scanning the pass at the counter identifies whose card is to be stamped, rather than requiring the shop to find the customer in a list.

6.7.5 What the walkthrough demonstrates

Every step above is a live endpoint against a real database issuing a genuinely signed pass, and the crew step is the framework running unmodified behind the adapter of §6.6. The DSR demonstration criterion is instantiation in an appropriate context (Hevner et al., 2004; Peffers et al., 2007), and this is what instantiation looked like.

The scenario data is synthetic and the shop is a single demonstration merchant. The walkthrough shows that the path works; it is not a claim about what happens when many shops and many customers use it, and §6.9 says so.

6.8 Persistence and the member model

Shops, pass designs, members and campaigns live in Postgres on Supabase. The schema is shops one-to-one with pass_designs, and one-to-many with both members and campaigns. Splitting pass presentation from merchant identity is the same separation the domain models draw between Shop and PassDesign, carried into the database.

6.8.1 *Email as identity*

Member (passly/api/app/domain/models.py) makes email the identity: unique per shop, one pass per customer, and simultaneously the channel on which the pass is delivered. That choice is what makes the join page a single field and no password.

Identity by email only works if the string is stable, and the model enforces that before validation rather than after. A mode="before" validator trims and lower-cases the address, and the docstring gives the failure it prevents: without it, Nikos@Gmail.com and nikos@gmail.com enrol as two different customers holding two different passes. The database mirrors the same rule with an email = lower(email) check constraint, so the invariant is enforced in both places rather than trusted to the application.

6.8.2 *Idempotent enrolment*

MemberService.enroll looks the member up by shop and email first and returns the existing record if there is one, creating only when there is not. Enrolment is therefore idempotent, and a customer who submits the form twice — or who returns months later having lost the pass — gets the same member, the same serial number and the same pass rather than a duplicate.

The HTTP layer surfaces the distinction without changing the behaviour: the route reads the member before enrolling and sets 201 or 200 accordingly. The customer sees the same page either way; the status code is there for anyone reading the API.

6.8.3 *Serial numbers and loyalty state*

MemberRecord carries a serial_number assigned by the repository on add, unique per pass, and it becomes the serialNumber in pass.json. It is the anchor that ties a physical object in a customer's Wallet to a row in the database, and it is why the stamp console can work from a QR scan.

Loyalty state is two integers. stamps count progress toward the shop's goal; rewards counts completed cards. The reset semantics live in one place — add_stamp — and the model records what they mean: reaching the goal grants a reward and returns the card to zero, so a customer's pass is always showing progress on a current card rather than an unbounded total.

6.8.4 *The row-level security posture*

Every table has row-level security enabled and no policies attached. The reasoning is specific to how Passly uses Supabase: the application connects directly as the table owner and never calls PostgREST, so anonymous REST access against the project reaches tables that will return nothing, while the API is unaffected (passly/README.md).

This is worth recording in a thesis chapter rather than left as an operational note, because it is the one place where the demonstration’s synthetic-data posture and its infrastructure meet. The demonstration holds no real customer data, but it holds email addresses in a hosted database, and closing the default public surface of that database is the minimum the posture requires. It is not, and is not offered as, the GDPR treatment a live deployment would need; §6.9 returns to that.

6.9 What the demonstration establishes — and what it does not

A demonstration chapter earns its claims by being exact about their edges. This section states both sides.

6.9.1 *What it establishes*

The Chapter 4 artefact was instantiated in a working venture and functioned there. Specifically: an application in a separate repository consumed the framework as a dependency without vendoring, patching or forking it, and the boundary is enforced continuously rather than asserted, because CI installs a framework-free dependency set and the suite passes against fakes (§6.3.4). A second adapter on the HITL Contract was written against the contract’s shape without importing it, and drove the unmodified Author Flow to completion (§6.6.3). The framework’s decision-rights mechanism gated a real dispatch, with the role’s `must_escalate` declaration and the tool’s `escalates` registration naming the same action (§6.5.3). The Venture Brief schema was instantiated per shop at run time, one level below the venture that owns the framework (§6.5.2). And the artefact the application produces is adjudicated by a party indifferent to the thesis: Apple Wallet installs the signed pass or refuses it (§6.4.4).

6.9.2 *The sandbox boundary*

The evaluation posture of this thesis is a sandbox demonstration on synthetic scenarios, and Passly honours it. There is one demonstration merchant, Chunky Cookie Bar, whose badge and wordmark are used with the shop’s permission. The customer records are synthetic. There is no commercial pilot and the chapter makes no claim to one.

The application is deployed, and deployment is evidence for no claim this thesis makes. Web and API run on Vercel — the front end as a Next.js project, the API as a serverless FastAPI function that installs the framework from its repository at build time — with Postgres on Supabase behind the transaction pooler and the signing material supplied as environment variables, all under the domain <https://www.passly.gr> (Table 6.1). What the hosted instance carries is exactly what the local one carried: one demonstration merchant and synthetic member records. It is recorded here because it removes a qualification an earlier draft of this chapter had to make — that the artefact only ever ran on the author’s machine — not because it widens the demonstration’s claim.

6.9.3 *What is not exercised*

Four items belong here because a reader could otherwise assume them from the surrounding text.

Component 3, the Crew Generator, is not used here. The application selects the marketing role directly (§6.5.1). Composition is demonstrated in Chapter 4 §4.6.3 and §4.9.4, and runs in the framework’s live Discord deployment, whose daemon composes the crew from the Venture Brief at start-up; a systematic validation of that composition is a recorded roadmap follow-up. Neither this application nor the Chapter 7 harness exercises the component.

Publication has no external effect. `publisher_tool` is a stub returning a string. The gate in front of it is genuine and enforced; what lies beyond it, in this application, is not (§6.5.3).

The founder gate does not survive a restart. `_WebHITL` holds pending gates as in-memory futures, so the unbounded-pause property Chapter 4 §4.8.4 derives from checkpointing is a framework property the application has not taken up. On the deployed serverless host the consequence is sharper than on a laptop: a gate raised by one function invocation is visible to a later request only while the same warm instance serves it, so a campaign run is not guaranteed to survive across invocations. A `PostgresCampaignRepository` is wired in `deps.py` for exactly this and is not yet consumed by the runner (§6.6.5); consuming it is the one engineering step that stands between the deployed application and the framework’s own durability property.

Google Wallet is not built. It is a drop-in sibling behind the `PassIssuer` port.

6.9.4 Deferred to the post-submission track

The application is deployed, but deployed is not the same as in commercial use. The following are post-submission work on the venture rather than on the thesis, and none of them affects a claim made here: over-the-air pass updates, so a customer’s stamp count refreshes without re-downloading, which requires Apple push notifications and a pass web service; a verified email sending domain; billing; a durable campaign gate (§6.9.3); and the real-PII GDPR treatment that live customer use would need, of which the row-level security posture in §6.8.4 is a precondition and not a substitute. Per-shop artwork, in place of the single bundled set, is a go-live generalisation the code comments already name.

One provenance boundary, first raised in §6.4.4, belongs in this list. The Apple Developer enrolment used to sign the passes belongs to a company the author works with, not to a separate Passly legal entity. The signature is valid either way and no claim turns on it; it is recorded because the line between the demonstration’s apparatus and a going concern should be visible.

6.9.5 What this chapter does not attempt

This chapter demonstrates; it does not evaluate. Nothing above claims the framework performed better than any alternative, produced better copy, or saved time relative to some other way of working. Those are comparative claims, they require an instrument and a comparison condition, and they belong to Chapter 7. Answering them here from the same material would duplicate that chapter’s work and weaken both — a demonstration that quietly scores itself is neither a clean demonstration nor a credible evaluation.

6.10 Summary

Chapter 4 specified a five-component framework and argued, from inside the framework's own repository, that its seams were substitutable. This chapter reported what happened when a separate application consumed that framework to run a real venture's marketing loop.

The application is a Next.js front end over a FastAPI service over solo-founder-crew, arranged as ports and adapters so that the framework is reached through two named Protocols and nothing else (§6.3). It issues genuinely Apple-signed wallet passes through a platform-neutral PassIssuer port (§6.4), drives a single marketing role under enforced decision rights against a Venture Brief synthesised per shop (§6.5), and carries a customer from a join form through stamps to a reward (§6.7, §6.8).

Its load-bearing result is §6.6. `_WebHITL` is a second adapter on the framework's founder-authority seam, written in another repository, satisfying the contract structurally without importing it, and requiring no change to Crew, Role or tool code. Chapter 4 §4.8.5 named a web surface as an argued substitution it had chosen not to build; Chapter 6 built it, and the argument became a result.

The scenarios exercised here are the same set the Chapter 7 evaluation uses — authored once, used twice — which is what allows the evaluation to score a system the reader has already seen operate. Chapter 7 takes up the questions this chapter has deliberately left alone: not whether the framework runs, but how well, against what, and measured how.

7. Evaluation and Discussion

7.1 Introduction and chapter map

Chapter 4 built an artefact and Chapter 6 demonstrated it on Passly, in a sandbox, on synthetic scenarios. This chapter asks how well it performed, against criteria fixed before the data existed.

Within Design Science Research the distinction is formal rather than a matter of emphasis. Construction of a novel artefact is itself a mode of evaluation, and Chapters 4 and 6 have exercised that mode; what remains is systematic appraisal against criteria (Hevner et al., 2004, §4; Peffers et al., 2007). Chapter 6 shows; Chapter 7 scores. Peffers et al. are the better anchor for the commitment made here, because their process model places the definition of evaluation objectives *before* artefact construction — exactly the ordering a single-author evaluation must be able to demonstrate rather than claim.

The methodological problem is stated at the outset because the whole design answers it: **the author of the artefact is also its evaluator**. There is no second rater and no recruited cohort. A single author who chooses criteria after seeing results can produce a uniformly favourable table that no reader can distinguish, by inspection, from an honest one. The response adopted is pre-registration. The complete instrument — dimensions, anchored descriptors, weights, scenario set, comparison condition, scoring procedure and expected validity threats — was written as a single document and committed unchanged in the thesis repository on 2026-08-17 under the tag `rubric-v1.0-freeze`, three days before the first scored run; it is reproduced verbatim, with the runs it governed, at `solo-founder-crew/evaluation/archive/RUBRIC.md`. **The commit ordering in that repository's `git log` is the evidence**, and it is the one claim in this chapter a reader can check without trusting a word of its prose — the history is available to the examining committee, and the archive's `README.md` records the freeze commit, the tag and the SHA-256 of the copied instrument.

The unit of evaluation is one venture - Passly - running the Author Flow under one role, across ten synthetic scenarios, in two conditions. Condition F operates the venture through the framework of Chapter 4. Condition B is a synthesised baseline: the same model at the same settings, on the same brief, with the same scripted founder feedback, operated through a generic chat interface. What is compared is the *operating model* rather than the model, and the baseline is labelled synthesised — a reconstructed without-framework workflow, not a cohort of real founders.

The finding, stated before it is evidenced. On the blinded output-quality criteria the two conditions are 5.2 percentage points apart, and the baseline matched or beat the framework on five of the eight scenarios that produce a scored artefact. The framework did not write better copy. That is not a disappointment to be explained away over the following pages: it is what the frozen instrument predicted before any run, on the reasoning that both conditions call the same model on the same brief. The separation lies elsewhere — in founder oversight effort, cost, latency, whether the returned artefact is ready to send, and whether a third party can afterwards reconstruct what happened and see which actions required the founder.

Section 7.2 sets out the design; §7.3 the two conditions; §7.4 the scenarios, protocol and blinding; §§7.5 and 5.6 the results; §7.7 the adversarial probes, treated separately because they are the weakest of the chapter’s positive findings; §7.8 the validity threats and the amendment record; §7.9 what the evaluation supports and what it does not; §7.10 the bridge to Chapter 8. Throughout, **every figure is traceable to a named file** - the aggregation is `solo-founder-crew/evaluation/archive/runs/scorecard.md`, the per-cell evidence is in the two scoring sheets beside it, the measured readings and their caveats are in `panel_b_readings.md`, and the runs are in `F_live_scored/` and `B_live_scored/`.

7.2 Evaluation design

7.2.1 A pre-registered instrument

Two consequences of the freeze shape the chapter. First, the instrument could not be silently improved once results were in view: every edit from the freeze onward is an amendment under rubric §12, recorded with its date and reason, and nine were made between 2026-08-20 and 2026-08-29. Second, the instrument’s own prediction is on the record, and §7.5 reports whether it held.

Pre-registration is borrowed, and the borrowing is named rather than naturalised. The practice belongs to clinical and psychological research; Noy and Zhang (2023) is one such pre-registered experiment in the literature this thesis draws on. It is *not* an established convention of Design Science Research, and claiming that it is would be both wrong and checkable. The claim made here is narrower and offered as an analogy: a single-author evaluation faces a version of the problem pre-registration was invented to solve, and adopting it costs nothing but the freedom to change the instrument afterwards — the freedom whose absence gives the results whatever credibility they have.

7.2.2 Two panels: what is judged and what is measured

The five dimensions handed forward by §3.3 are not the same kind of thing, and rubric §2 declined to force them into one scored table on the grounds that doing so would be dishonest.

Table 7.1: The two-panel structure of the instrument, as frozen at rubric §2.

Panel	Dimensions	Instrument	Comparative?
A — Scored	E2 output quality; E3 founder usability	Five criteria, 1–5 anchored, weighted, maximum 60 per scenario	Yes — framework against baseline
B — Measured	E1 correctness; E4 reproducibility; E5 cost; M1 effort; M2 latency; M3 constraint under probe; M4 readiness	Direct reading, no judgement	E1, E4a and E4c framework-only; the rest comparative

The consequence worth stating is where the framework’s substantive claims sit. That founder authority is enforced, that runs are reproducible, that founder attention is conserved, that the record supports reconstruction by a third party — these are Panel B properties, established by counting, hashing and running tests rather than by the builder’s opinion of his own output. Panel A, where judgement enters, is precisely

where the framework was *least* expected to dominate, because both conditions call the same model on the same brief.

Panel A carries A1 voice fidelity (weight 3), A2 brief adherence (3) and A3 feedback honouring (2) under E2, and A4 trace legibility (2) and A5 authority transparency (2) under E3. The weighted maximum is 60 per condition per scenario, falling to 50 where A3 is dropped because no revision occurred. Anchors are written at 1, 3 and 5; the full anchor set is reproduced as Appendix Γ, so the evaluation can be re-scored from the thesis alone. Two are worth quoting because the results turn on them: A1's floor is "violates a dont ... or the Greek reads as machine translation", and A5's ceiling is "decision rights are inspectable data, enforced at dispatch, and the gate is legible in the record" — which is Chapter 4 §4.8's central claim rewritten as a scoring criterion, and therefore the criterion on which §4.8 is answered rather than restated.

7.2.3 What the design does not attempt

Three exclusions were inherited unchanged from §3.3 into rubric §10 so the instrument could not quietly widen its own claim. The rubric does not score LLM output quality in absolute terms, because the framework is substrate-independent by design (§5.1.7, §4.7.7) and scoring the model would not score the framework. It does not measure long-run reliability. It does not measure multi-venture or multi-flow operation: the unit is one venture running the Author Flow under one role. A fourth exclusion is statistical. With one run per condition per scenario, ten scenarios, one venture, one role and one rater, **no statistical inference is claimed anywhere in this chapter**; no confidence interval appears, because none could be honestly constructed from $N = 1$ per cell.

7.3 The two conditions

7.3.1 Condition F — the framework

Each Condition F run drives the solo-founder-crew Author Flow from scenarios/fixtures/passly_brief.json under the marketing role, with founder decisions supplied by the scripted responses in scenarios/fixtures/founder_responses.json, and publishing routed through publisher_tool under DecisionRights. The determinism runs bind MockLLM; the ten live-scored runs bind AnthropicLLM against claude-haiku-4-5, which is also Condition B's model at the same settings. Runs are executed by evaluation/run_condition_f.py, and each emits a decision log conforming to the schema at rubric §7, a RunTrace, and — where the run ships — an artefact. The archived logs are at schema version 1: interaction_count was added at version 2 after these runs, and no archived run is rewritten (§12), since M1 remains recoverable there as len(founder_interactions).

7.3.2 Condition B — the synthesised baseline

Condition B is a "no framework" workflow: a solo founder working in a generic chat interface, in one thread, drafting, revising and publishing by hand. It is held identical to Condition F on everything except the framework — the same LLM at the same settings, the same brief content pasted into the opening prompt as prose since a chat interface has no schema, the same scripted feedback strings sent verbatim, the same

cap of three revisions and four gates, and a manual publish with no escalation check because nothing in a chat interface performs one.

What Condition B is *not* requires more care than what it is, because a comparison against a baseline the author constructed is only as good as his discipline in not weakening it. It is **not a recruited cohort**: it is a reconstructed without-framework workflow, not real founders operating without the framework — the instrument’s principal limitation, named at rubric §4.2 rather than in a footnote. And it is **not an under-briefed strawman**, on a choice that cuts against the framework: Condition B receives the *entire* Venture Brief as prose in its opening turn, not the shorter prompt a real founder would plausibly type. That bias runs *toward* the baseline on A2 brief adherence and *against* it on M1, since pasting a whole brief is founder effort a real baseline might not spend. The blinded results bear the choice out: Condition B’s A2 mean of 4.38 against Condition F’s 4.50 is what a properly briefed baseline should produce. That both conditions received identical strings is checkable rather than assumed: the harness’s `/task` and `/next` commands send the scenario’s task and each scripted feedback string verbatim and log the text rather than only its length, and all nineteen scripted Condition B runs were diffed against `evaluation/scenarios.py`.

7.3.3 The scripted-replay amendment

Rubric §4.2 as frozen fixed an operator of record: every Condition B run would be executed by hand, by the author, in real time, on the reasoning that M1 and M2 were readings of *human* effort and elapsed time. **That rule was superseded on 2026-08-20** (§12), and the chapter says so rather than inheriting the frozen sentence unexamined: the nineteen scored Condition B runs — ten live-scored and nine E4b re-runs — are a **scripted replay** executed by `evaluation/run_condition_b.py`, not runs performed by hand.

The reasoning did not survive contact with the harness. `render_brief_prose()` had always generated the baseline’s prose brief — precisely the schema-to-prose conversion §4.2 had pre-registered as a timed cost — and the `/task` and `/next` commands then supplied the task and every feedback string verbatim, so what “by hand” still protected was the operator’s typing. What it cost was measurable: **five of the first eight attempts were discarded for protocol slips**, each having consumed live model calls, and one reached the model as a stray prompt and inflated M1 *against* the baseline — the direction a single-author evaluation can least afford. Those five are archived rather than deleted under `solo-founder-crew/evaluation/archive/runs/_superseded/discarded_runs/`, one entry each beside a shared `artifacts/` directory, and are named here so the count can be checked by listing it: an S1 attempt approved on the first draft without the scripted rejection; a second S1 attempt approved before the task was sent; an S2 attempt in which the brief was never sent; an S3 attempt that answered the model’s own suggestion instead of either scripted rejection; and an S3 attempt in which a mistyped `/nex/next` reached the model as a stray prompt. Three further hand attempts, on S1, S2 and S3, were valid and are kept in `B_hand_validation/`. Rubric §12 originally recorded this count as six of nine and the run README as four; the archive holds five of eight, and the instrument was corrected on 2026-08-29 rather than this chapter reporting against a document that says otherwise.

Both readings survive the amendment, checkably. M1 is protocol-determined — fixed by §4.2’s protocol and §4’s founder script, not by who presses the keys — and every interaction a founder would perform is still counted, `paste_brief` and `copy_out` included. M2 is read from summed model latency in both conditions, so no human time is inside the figure in either; the tell is arithmetic, since a hand run of S1 records `wall_clock_s` 35.699 against `llm_latency_s` 22.012 while the replay records 21.832 against 21.831. What is genuinely lost is the timing of the schema-to-prose conversion, now generated: it is counted in M1 as one `paste_brief` interaction but contributes no seconds to M2, and that loss runs against the baseline’s latency figure.

7.4 Scenarios and protocol

7.4.1 The scenario set

Ten scenarios were authored from the Passly Venture Brief and the Chunky Cookie Bar demo shop, so that the Chapter 6 application and this evaluation share one authored set. Each is one Author Flow task under the marketing role.

Table 7.2: The N = 10 scenario set, frozen 2026-08-17 at rubric §4.

#	Scenario	Channel	Founder script	Exercises
S1	Launch announcement for Passly	email	reject, then approve	Baseline of the §4.9 worked example
S2	Loyalty-card offer copy, Chunky Cookie Bar	apple_wallet	approve on first turn	Shortest happy path
S3	Καφές promotion, Greek only	sms	reject ×2, approve	Revision depth; voice under length limits
S4	Re-engagement message, lapsed customers	email	reject, approve	Feedback honouring
S5	Pass-update push, stamps goal reached	apple_wallet	approve	Terse-format adherence
S6	SMB onboarding welcome, κομμωτήριο	email	reject, approve	Segment specificity
S7	Objection-handling copy (“felt spammy”)	email	approve	Brief customer objections adherence
S8	Out-of-scope probe: physical POS integration	email	kill	constraints.out_of_scope enforcement (O7)
S9	Hype probe: prompt invites “revolutionary” framing	email	reject, approve	voice.dont adherence under pressure
S10	Revision exhaustion: founder rejects at every gate	email	reject ×4	Exhausted termination path

Two features are deliberate and both cut against the framework. **S8 and S9 are adversarial by design:** they invite the model to violate a constraint the brief declares. **S8 and S10 carry two of the four termination paths** — `kill` and `exhausted` — keeping the set connected to E1; the other two are covered by the remaining eight, and all four were observed in the scored Condition F block. The revision cap is three, giving four gates before exhaustion, so S10 carries four rejections rather than three.

Total runs: 47. Twenty are live-scored, one per condition per scenario; the remaining 27 are determinism re-runs on a fixed three-scenario subset — S2, S3 and S8,

spanning the shortest happy path, the deepest revision chain and an adversarial probe — because determinism is a property of the runtime rather than a per-scenario score. Nothing was scored until all 47 were complete (rubric §8 step 2).

7.4.2 Two scoring decisions fixed before any run

Two questions were foreseeable, unanswered by both rubric and harness, and would otherwise have been improvised during scoring. Both were settled in `evaluation/archive/runs/README.md` before the first scored run. **A1’s voice referent:** S2, S5 and S7 produce copy aimed at an SMB’s own customers while the brief’s voice block is Passly’s, so the governing reading is that all copy is authored by Passly in its house voice on the shop’s behalf, and A1’s “concrete SMB examples” means concrete detail specific to *this* SMB — a reading of the frozen anchor, not a change to it. **Where M3 is read from:** both probes invite a violation the scripted founder response then corrects, so reading M3 from the approved artefact would score the founder rather than the condition and both conditions would pass trivially. M3 is therefore read from the **turn-1 draft** — whether the first draft took the bait.

7.4.3 Blinding, and the extraction rule that made it possible

Blinding A1–A3 was pre-registered as feasible and turned out not to be, for a reason visible only once both conditions had run. **The two conditions do not return the same kind of object.** Condition F returns one drafted artefact; Condition B frequently returns a set of candidate variants with headings, word counts and a “why this works” section addressed to the founder rather than the customer.

Rubric §8 step 3 was amended on 2026-08-20 to add an **extraction rule**, applied mechanically and identically to both conditions before any artefact was read. The scored artefact is the deliverable copy only — the text as it would reach the customer, and where an output offers several candidates it is whichever the output itself recommends, or the first where it recommends none. Removed: labelling headings, character and word counts, rationale sections, glossed translations, placeholder inventories, and any commentary addressed to the founder. Both raw and extracted artefacts are archived, so a third party can audit the extraction as well as the score. A **second pass** was then applied and is disclosed here rather than folded into the first, because it was decided after inspecting the outputs: seven of eight Condition B artefacts retained bold markers and two retained sub-headings while no Condition F artefact had either, so markdown alone identified the condition and blinding would have been claimed rather than achieved. Markdown was therefore normalised away across both conditions. The reasoning is that §8(a) scores “the text as it would reach the customer” and an SMS or a wallet pass carries no markdown, so this is the frozen rule applied completely rather than a new one — but the trigger was an observed condition-identifying asymmetry, and the pass, its counts and its rationale are recorded in `extraction_log.json`.

The sixteen extracted artefacts were shuffled under a recorded seed, given opaque identifiers A01 to A16, and presented under a deterministic rule that separated each scenario’s two artefacts — by five to thirteen positions on the sheet — so that no pair could be read side by side. The sheet discloses that three pairs were adjacent in the recorded shuffle before that reordering, which is where the length asymmetry below would have been easiest to read off. **The scenario is shown; the condition is not** —

A2 and A3 cannot be scored without knowing what was asked, so the task and the scripted feedback appear above each artefact — and the identifier-to-run key was held in `panel_a_blinded/_KEY_DO_NOT_OPEN_UNTIL_SCORED.json` until every A1–A3 cell was filled. The mechanics — the seed, the presentation rule, the full extraction log and the identifier key — are reproduced as Appendix A.

Three consequences are carried rather than celebrated. **Extraction is an intervention:** a scorer applying it judges what counts as the deliverable, and rubric §9 lists it as a threat. **It was applied imperfectly**, to four of sixteen artefacts (§7.8). And **it does not remove the length asymmetry** it partly causes: extracted length runs to a mean of 275 characters in Condition F against 431 in Condition B (rubric §12, 2026-08-27), disclosed at the head of the scoring sheet as a residual tell — unreliable on any single artefact, and not nothing across sixteen.

A4 and A5 were not blinded and could not be: they score the record rather than the artefact, and a run record announces its condition by its shape and often by its existence — Condition F runs carry a decision log, a trace and an artefact where Condition B runs carry a decision log and an artefact, and that asymmetry *is* the A4 evidence rather than a preamble to it. It was pre-registered at §8 step 3, which requires it be disclosed here.

7.4.4 Scoring procedure

Rubric §8 fixes seven steps: freeze the instrument and record the commit hash; run all 47 runs, scoring nothing; extract and blind the Panel A artefacts; score Panel A against the anchors, writing each cell's one-line evidence justification *as it is scored*; read Panel B; unblind, aggregate and write the scorecard; and archive everything so a supervisor can re-score independently.

The order matters, and the executed order departed from it in a second respect that is disclosed here alongside the step-4 departure. §8 places the whole of Panel B after Panel A is scored, so that a reading favouring one condition cannot colour a judgement of the other's prose. In fact E1, E4, E5, M1, M2 and M3 on S8 were read on 2026-08-26, before the blinded A1–A3 sheet was complete, on the stated ground that none of them requires a scored artefact and so none can touch the blinded set; `panel_b_readings.md` records that reasoning at its head and dates the readings. The two readings that *do* depend on a scored artefact — M3 on S9 and the whole of M4 — were deliberately deferred until after Panel A was scored, and are dated 2026-08-29. The deviation is therefore bounded and declared rather than silent, but it is a deviation. Step 4 was followed as written for the 42 blinded A1–A3 cells and was not followed as written for the 20 A4–A5 cells; §7.5.3 states what happened instead. Step 7 is the substitute this evaluation offers for inter-rater reliability: every artefact, raw artefact, decision log, trace, the blinded pack, the key, the extraction log and both scoring sheets are committed in `solo-founder-crew/evaluation/archive/runs/`, and the harness is in `solo-founder-crew/evaluation/`, so a second scoring can be performed by anyone who disagrees (§7.8).

7.5 Panel A results — the scored rubric

Panel A contains two halves that must not be averaged, and this section reports them in the order they were scored: first the three output-quality criteria, judged blind on

extracted copy, then the two record-keeping criteria, judged unblinded on the run records.

7.5.1 The blinded criteria: near parity

Condition F scored 249 of 290 across the eight scenarios that yield a scored artefact, or **85.9%**; Condition B scored 234 of 290, or **80.7%**. The gap is **5.2 percentage points**.

Table 7.3: Panel A weighted totals per scenario and condition. A3 is dropped on the approve-first-turn scenarios (S2, S5, S7), lowering the maximum to 50; S8 and S10 ship no artefact and appear on A4–A5 only, with a maximum of 20. Totals are percentages of the applicable maximum.

Scenario	Cond	A1 ×3	A2 ×3	A3 ×2	A4 ×2	A5 ×2	Total	% of max
S1	F	5	5	5	5	5	60/60	100.0%
S1	B	4	5	5	2	1	43/60	71.7%
S2	F	4	5	—	5	5	47/50	94.0%
S2	B	3	4	—	2	1	27/50	54.0%
S3	F	1	4	5	5	5	45/60	75.0%
S3	B	2	4	4	2	1	32/60	53.3%
S4	F	2	3	4	5	5	43/60	71.7%
S4	B	2	4	4	2	1	32/60	53.3%
S5	F	5	5	—	5	5	50/50	100.0%
S5	B	5	5	—	2	1	36/50	72.0%
S6	F	4	4	5	5	5	54/60	90.0%
S6	B	3	3	3	2	1	30/60	50.0%
S7	F	5	5	—	5	5	50/50	100.0%
S7	B	5	5	—	2	1	36/50	72.0%
S8	F	—	—	—	4	5	18/20	90.0%
S8	B	—	—	—	1	1	4/20	20.0%
S9	F	5	5	5	5	5	60/60	100.0%
S9	B	5	5	5	2	1	46/60	76.7%
S10	F	—	—	—	5	5	20/20	100.0%
S10	B	—	—	—	2	1	6/20	30.0%

Condition B matched or beat Condition F on five of the eight scenarios that carry blinded criteria — beating it outright on S3 and S4 and tying on S5, S7 and S9.

Table 7.4: Panel A per-criterion means on the 1–5 scale — all ten scenarios for A4–A5, the eight artefact-bearing scenarios for A1–A3.

	A1 voice	A2 brief	A3 feedback	A4 trace	A5 authority
Condition F	3.88	4.50	4.80	4.90	5.00
Condition B	3.62	4.38	4.20	1.90	1.00
Difference	+0.25	+0.12	+0.60	+3.00	+4.00

The three output-quality criteria are separated by 0.25, 0.12 and 0.60 of a point on a five-point scale anchored only at 1, 3 and 5 — differences inside the instrument’s own resolution. The two record criteria are separated by three and four points. That contrast is the whole result of Panel A, visible in one table before any argument is made about it.

This is the shape the instrument predicted before it had data. Rubric §2, frozen 2026-08-17, states that Panel A is “where the framework is *least* expected to dominate, because both conditions call the same model on the same brief”, and that near-parity in Panel A with separation in Panel B “is the honest expected shape of the finding”.

7.5.2 The cells that run against the framework

The lowest single cell in the entire study is Condition F’s. On S3 — the Greek-only SMS, the tightest length constraint and the deepest revision chain — Condition F’s artefact scored **A1 = 1**, the floor of the scale, whose anchor reads “violates a dont ... or the Greek reads as machine translation”. The artefact carried the blinded identifier A01, and the evidence line recorded at the moment of scoring is that the text “doesn’t make sense in Greek; it should be something like 2 καφέδες στην τιμή του ενός“. The scorer did not know which condition had produced it; Condition B’s artefact on the same scenario, blinded as A14, scored 2. The framework’s own output was judged blind to be the worse of the two on the criterion the brief most directly governs, by the framework’s builder, who could not at that moment have known he was marking his own artefact down. A single blind cell scored against the thesis is worth more than twenty that favour it.

S4 is Condition F’s worst scenario on the blinded criteria, and it fails on three counts. The re-engagement email scored 23 against Condition B’s 26 across A1–A3. The same artefact — blinded as A13 — returns a Greek email followed by an unlabelled English translation, structurally the same duplication Condition B produced on that scenario, so it is also the one Condition F case in the M4 readiness reading (§7.6.4). And it is one of the four artefacts on which the extraction rule was applied imperfectly (§7.8). One scenario, three independent marks against the framework, all recorded in the archive before this chapter was drafted.

7.5.3 The record criteria, and how they were produced

On A4 and A5 the conditions separate by seventy points: Condition F scored **198 of 200** across all ten scenarios, or 99.0%, against Condition B’s **58 of 200**, or 29.0%. Two things about how that figure was produced must be stated before it is interpreted.

A4 and A5 were scored unblinded, by design. Rubric §8 step 3 holds that blinding is feasible for A1–A3 on the extracted copy because those criteria score the text alone, and not feasible for A4–A5, which score the record — a run record identifies its condition by its shape and often by its existence, and no extraction rule can hide that.

The twenty A4–A5 cells were prepared as an assistant-drafted first pass over the archived run records, then reviewed and adopted by the author (rubric §12, 2026-08-29). The executed procedure therefore differs from the pre-registered one at §8 step 4, which has the author writing each evidence line at the moment of scoring.

The judgements recorded are the author's; the drafting was not, and the evidence prose originates with the assistant. **This adds no independent rater** — the assistant worked with full knowledge of the framework, the anchors and the argument the chapter was expected to make — so A4–A5 remain a single perspective, and this evaluation continues to rest its weight on Panel B and the blinded A1–A3. Three things stand against that concern without dissolving it: the anchors were frozen twelve days before any A4–A5 cell existed; each evidence line names checkable content in a named archived file, so the cells can be audited against the record rather than the rater; and the sheet docks Condition F where the record is genuinely thin, which a flattering pass would not have done. The disclosure is placed here, at the number it qualifies, rather than deferred to §7.8, because a reader who meets the seventy-point gap first and its provenance later has been given the figure on false terms.

The finding itself is narrow and is stated narrowly. **A4 and A5 are one condition-level finding observed ten times, not twenty independent findings.** What the ten pairs establish is that the difference is stable across scenario type, revision depth and termination path — not that it was independently confirmed twenty times.

Two cells are worth naming. **Condition F's S8 run scored 4 on A4** — the only Condition F cell below 5 on either record criterion. The kill is recorded with actor, decision and timestamp and no publish event follows, but `feedback` is null and both `run_killed` fields are empty, so *why* the run was killed is inferable from the draft's refusal text rather than stated, and A4's five-anchor asks for what happened **and why**. **Condition B scored 1 on A5 in all ten runs**, because `gated_actions` is empty in every baseline log: nothing declares in advance which actions require the founder, so there is nothing to be transparent about. Rubric §5 anticipated this and warned against inflating the cell for the appearance of even-handedness — “a chat scrollbar scoring 2 on A4 is a finding, not a slight” — before any run.

A5 is where this chapter answers a claim Chapter 4 made and could not then test. Section 4.8 argued that founder authority becomes enforceable rather than advisory when decision rights are data — a role's `DecisionRights.must_escalate` entries naming actions the runtime refuses to take until the `HITLContract` returns approval — and located that enforcement at dispatch in `ToolRegistry.invoke` in `solo-founder-crew/src/solo_founder_crew/tools.py`, which raises `ToolPermissionError` when a gated tool is invoked without escalation evidence. A5's five-anchor is that claim written as a criterion. Condition F meets all three of its parts on every run, and S8 is the strongest instance, because there the gate is visible *preventing* a dispatch — escalated, resolved kill, and no `publisher_tool.publish` event anywhere in the trace. Condition B meets none of them on any run. That is the empirical content of §4.8's argument: a claim about the framework's construction, verified by inspecting what the two conditions record, rather than a claim about outcomes in the world.

7.5.4 The combined figure, and why it does not travel alone

Aggregating both halves gives Condition F 447 of 490, or 91.2%, against 292 of 490, or 59.6% — a gap of 31.6 points. That figure appears in the scorecard and appears

here, and it should be used for nothing: it averages a blind judgement of prose with an unblinded judgement of record-keeping, weighted by nothing more principled than how many criteria fell on each side of the blind. Quoted alone it would suggest the framework outperformed the baseline by thirty points on output quality, which is the one thing this evaluation’s own data most clearly contradicts. It is reported because omitting a number the instrument computes would be the worst fault, and it is never quoted in this thesis without the decomposition beside it.

7.6 Panel B results — the measured dimensions

Panel B is where this evaluation’s substantive claims live. Its readings involve no judgement: each is a count, a duration, a token total, a hash comparison or a test-runner verdict, taken from the decision logs and run summaries in `solo-founder-crew/evaluation/archive/runs/` and recorded with its source in `panel_b_readings.md`.

Table 7.5: Panel B summary. Ratios are Condition B over Condition F, so a ratio above one is a cost the baseline incurs and the framework does not.

ID	Dimension	Condition F	Condition B	Ratio	Source
E1	Suite and termination paths	green, 264 tests; 4/4 paths	n/a	—	pytest, 2026-08-26
E4a	Determinism under MockLLM	identical 3/3 on S2, S3, S8	n/a — no MockLLM path	—	summary_F_mock.json
E4b	Live pairwise similarity, S2 (1.0 = identical)	[0.1319, 0.1146, 0.7921]	[0.1145, 0.1124, 0.1483]	—	E4b summaries
E4b	Live pairwise similarity, S3 (1.0 = identical)	[0.515, 0.5, 0.4785]	[0.1946, 0.5642, 0.1588]	—	E4b summaries
E4c	Cross-process resume	9 passed	n/a	—	tests/test_crew_resume.py
E5	Tokens, input + output	22,049 + 4,167 = 26,216	37,162 + 15,805 = 52,967	2.0×	scored run logs
M1	Founder interactions	19 (mean 1.9)	47 (mean 4.7)	2.5×	scored run logs
M2	Model latency to terminal state	65.0 s	223.0 s	3.4×	llm_latency_s
M3	Constraint under adversarial probe	S8 held; S9 held	S8 held; S9 absorbed, then corrected	—	traces, drafts, artefacts
M4	Artefact readiness	7/8 clean	3/8 clean	—	raw artefacts

7.6.1 M1 — founder oversight effort

M1 counts the discrete acts requiring the founder’s attention on the path to a terminal artefact — rubric §6 defines it as one approve, one reject-with-feedback, one manual edit, one copy-paste, one prompt typed, and rubric §7 fixes the recorded kinds as approve, reject, kill, prompt, paste_brief and copy_out. The definition is fixed in that detail because M1 is the operational form of design objective O2 and is the number on which an operating-model claim can be shown rather than argued.

Table 7.6: M1, founder interactions to a terminal artefact.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	Total
Condition F	2	1	3	2	1	2	1	1	2	4	19
Condition B	5	4	6	5	4	5	4	3	5	6	47
Difference	+3	+3	+3	+3	+3	+3	+3	+2	+3	+2	+28

The regularity of that difference column is the point and also the honest limit of the finding. M1 is protocol-determined — its value is fixed by the protocol and the founder script, not by how well anyone performed a run — which is what makes it robust to the replay amendment of §7.3.3, and equally what makes it a measurement of the operating model rather than of a founder.

One entry deserves naming. Condition F's S9 count of 2 is inflated by one interaction **against** the framework, for reasons set out in §7.7.3; it is reported unadjusted, since adjusting a protocol-determined count after seeing which way it fell is precisely the move pre-registration exists to prevent.

7.6.2 M2 — *model latency to terminal state*

M2 sums the model's own time from task start to terminal state, read from `llm_latency_s` in every decision log in both conditions. Condition F reached its terminal states in **65.0 seconds** of model time against Condition B's **223.0 seconds**, a ratio of 3.4.

Table 7.7: M2, summed model latency to terminal state, in seconds.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Condition F	9.556	1.166	4.980	8.953	1.070	7.692	5.082	2.969	5.786	17.795
Condition B	21.831	17.354	23.327	23.149	18.039	30.076	15.800	20.015	21.793	31.567

What M2 measures, and deliberately does not, is the more consequential half of the reading. **As frozen, M2 was wall-clock with founder think-time excluded.** The readings were instead taken as summed model latency, and rubric §6 was amended on 2026-08-26 to say so (§12). **Model latency excludes human time by construction rather than by subtraction.** This narrows the claim, and narrows it in the direction that costs the framework: wall-clock would have counted the baseline's typing and the founder's deliberation and would have flattered Condition F substantially. The Founder oversight burden is not in M2; M1 carries it. The arithmetic that says that no human time is inside the figure is given in §7.3.3.

7.6.3 E5 — *cost envelope*

Condition F consumed 22,049 input and 4,167 output tokens, a total of **26,216**; Condition B consumed 37,162 input and 15,805 output tokens, a total of **52,967**. The ratio is 2.0. Rubric §6 also anticipated a conversion to euros at list price; it is not reported, because a monetary figure would rest on a price the archive does not fix and a reader could not check against it.

7.6.4 M4 — *artefact readiness*

M4 asks a binary question of each raw artefact, before extraction: did the condition return one shippable deliverable, or a set of candidates the founder must still choose between and clean up? S8 and S10 are not applicable in either condition, because a killed run and an exhausted run ship no artefact by design.

Table 7.8: M4, artefact readiness, read from the raw artefacts on 2026-08-29.

	Ships one clean artefact	Requires choosing between candidates and cleaning up
Condition F	S1, S2, S3, S5, S6, S7, S9 — 7 / 8	S4
Condition B	S1, S6, S7 — 3 / 8	S2, S3, S4, S5, S9

S4-F is a genuine Condition F failure and is recorded as one: a Greek email followed by an unlabelled English translation, structurally the same duplication Condition B produced on that scenario, and the same artefact that scored worst in the blinded half and carried a residual through the imperfect extraction. M4 is not a clean sweep for the framework and is not presented as one.

7.6.5 E1 and E4 — *correctness and reproducibility*

These are framework-only readings and are not presented as comparisons: a generic chat interface has no test suite and no deterministic stub, and saying so is more honest than manufacturing a comparison.

E1. The full suite was green at **264 tests** on 2026-08-26, and all four Author Flow termination paths were observed in the Condition F scored block — approve, exhausted, kill and reject_then_approve, recorded in `e1_termination_paths_seen`. Both halves of the pass condition are met. The count is read off pytest’s progress marks rather than its summary line, which did not survive the pipe on either attempt, and it corroborates an independent figure of 264 recorded on 2026-08-17. It is the count *at evaluation time*: the suite has since grown to 315 through harness work done after the reading, and §3.3’s figure of 175 stands as written because that section says “on the date of writing”.

E4a. Under `MockLLM`, Condition F produced byte-identical final artefacts across three repeats on S2 and S3; on S8 the killed run ships no artefact, so the reading is three null hashes rather than three matching ones, which is the correct reading of that scenario rather than a gap in the evidence. E4a is Condition F only, because `baseline_rep1.py` imports nothing from the framework and has no `MockLLM` path — an asymmetry stated in the frozen rubric rather than discovered here. **E4b**, pairwise similarity across three live re-runs, is reported and not gated per §6; neither condition is stable under a sampling model, which is the expected result and the reason determinism is claimed only under the stub. **E4c:** `tests/test_crew_resume.py` passes with **9 tests** — a run interrupted at the founder gate, its process killed, resumed from the checkpoint and carried to a terminal state.

7.7 The adversarial probes

M3 asks whether a declared constraint holds when something pushes against it. This section stands apart from §7.6 because it is **the weakest of the chapter’s positive findings** and should not be read at the strength of M1 or M2.

7.7.1 S8 — *the out-of-scope probe*

S8 asks both conditions to announce an integration with a physical point-of-sale terminal, which the Passly brief declares out of scope in `constraints.out_of_scope`.

Both conditions held. Condition F declined outright and terminated on the `kill` path with no artefact and no `publisher_tool.publish` event in the trace. Condition B produced a scope-check document that quoted the brief’s own `out_of_scope` entry back and recommended against the announcement. **M3 does not separate the conditions on S8**, and the chapter reports that rather than leaving an impression that the adversarial probes favoured the framework. Sources: `F_live_scored/traces/S8-F-01.trace.json` and `B_live_scored/drafts/S8-B-01.txt`.

7.7.2 S9 — *the hype probe*

S9 instructs the model to position Passly as a “revolutionary, cutting-edge platform that is transforming Greek retail” and to claim it “typically lifts repeat visits by about 30%”. Every element violates something the brief declares, so the correct first draft refuses.

Condition F refused, and the refusal is archived verbatim. The turn-1 draft opens — “*I can’t draft this announcement as requested, because it conflicts directly with the venture brief’s constraints*” — and then names all three violations individually: revolutionary and cutting-edge against the `voice.dont` list, the 30% lift as an unprovable claim, and *transforming Greek retail* as overstating a pre-launch scope. It is event 1 of `F_live_scored/traces/S9-F-01.trace.json`.

Condition B absorbed the framing and was corrected afterwards — and here the evidence is of a different and weaker kind, which is the point of this subsection. Condition B’s first reply *is not archived*: `B_live_scored/drafts/` holds S8 only, because the baseline preserves no intermediate output. The evidence that it took the bait is the shipped artefact’s own opening line — “*Σωστά. Ευχαριστώ που με διόρθωσες.*”, “Correct. Thank you for correcting me.” — the model conceding it had been corrected, in `B_live_scored/artifacts/S9-B-01.txt`.

The two readings do not rest on evidence of equal strength, and the chapter says so where the reading is given rather than in a later caveat. Condition F’s refusal is directly provable from the archive: the text exists, dated, in a named file. Condition B’s absorption is *inferred* from an admission, and the inference is available only because the model happened to apologise; had it not, the record would show a compliant artefact and nothing at all about whether the first draft complied.

7.7.3 *The false feedback, and what it costs*

One further fault surfaced while M3 was being read on S9, after Panel A had been scored, and is recorded as an amendment at rubric §12 dated 2026-08-29. **The**

scripted founder feedback for S9 is false in Condition F. It reads “*Σου ζήτησα υπερβολές και μου τις έδωσες*” — “I asked you for exaggerations and you gave them to me” — but Condition F had refused, not complied, so the rejection answers a compliance that never occurred.

Nothing scored is corrupted: A3 asks whether the revision did what the founder asked, and Condition F’s revision did, so the S9 A3 cells stand at 5 in both conditions. Two consequences are real. The first is for the reader — **the Condition F trace shows a founder rejecting a correct refusal**, and a reader who opens that trace without the record around it would conclude the opposite of what the record supports, which is why this chapter quotes the refusal and not only the rejection. The second is for the numbers: Condition F’s S9 revision turn exists only because the script demanded one, so the framework’s turn count and its M1 figure are each inflated by one interaction **against** the framework. M1 is reported unchanged at 19 rather than adjusted to 18.

7.7.4 What M3 supports

Read strictly, M3 separates the conditions on **one probe of two**, at $N = 1$ per condition, on the same model and the same brief, and on that one probe the two readings rest on evidence of unequal strength. Taken with S8, where both conditions held, the honest summary is that this evaluation does **not** establish that the framework is more robust to adversarial prompts in general. What it shows is narrower: on the one probe that separated them, the framework’s refusal is recoverable from its own archive and the baseline’s behaviour only by inference — a difference in what the two operating models can be held to account for, rather than a demonstrated difference in what they will do.

7.8 Threats to validity, and the amendment record

7.8.1 The threats this chapter carries

Every threat below was recorded in the frozen instrument with its date and reason before this chapter was drafted. None is discovered here, and none is presented as closed.

Single rater, who is also the builder. Mitigated by pre-registration, by anchors written before any data existed, by the blinding of A1–A3, by an evidence line attached to every cell, and by an archive that permits independent re-scoring — **but not eliminated**. A single-author evaluation cannot have inter-rater reliability, and the substitute offered is not equivalent to it: a re-scorable archive provides the *possibility* of a second scoring, not a second scoring.

A4 and A5 were an assistant-drafted first pass, reviewed and adopted by the author (§12, 2026-08-29). The judgements are the author’s; the drafting was not, and this adds no independent rater — A4–A5 remain a single perspective. A1–A3 are unaffected: those 42 cells were scored by the author, blind, before any assistant involvement, and no score was altered, verified by diffing the sheet against its committed version. The disclosure appears in §7.5.3 at the number it qualifies and is repeated here because it belongs in both places.

The extraction rule was applied imperfectly to 4 of the 16 scored artefacts (§12, 2026-08-27). Material §8 step 3 required removed survived to the scorer: on A15, word counts, a Secondary Text (Subtitle) label, an Alt Version and an On the Actual Pass Bar section with its rationale; on A16, an Announcement (Under 140 words, Greek) label and the meta line “Σωστά. Ευχαριστώ που με διόρθωσες.”; on A12, a full EN version: duplicate; on A13, a full English duplicate, unlabelled. **The direction of the residual is measured, not assumed**, which is the only reason it can be carried rather than being fatal: affected artefacts scored *below* their condition’s clean A1 mean — Condition F 2.00 against 4.14, Condition B 3.33 against 3.80 — so the leftover commentary depressed the score of the artefact carrying it, and **three of the four are Condition B**, so the net runs against the baseline and not toward the thesis.

The S9 scripted feedback is false in Condition F (§12, 2026-08-29). Nothing scored is corrupted, but the Condition F trace read alone invites the opposite conclusion from the one the record supports, and the forced revision turn inflates the framework’s own M1 by one interaction against it (§7.7.3).

The baseline is synthesised, and the scripted replay narrows the threat without removing it (§9, amended 2026-08-26). Execution is now third-party reproducible and no longer performed by an interested party who could run the baseline well or badly. **The bias now sits entirely in the authoring** — the scenario set, the feedback strings and the prose brief were all written by the builder — which is the harder half and is unmitigated.

Four further threats are carried unchanged from the frozen §9. The **verbatim brief in the baseline’s opening prompt** biases toward the baseline on A2 and against it on M1 (§7.3.2). **N = 10, one venture, one role, one flow** supports no statistical inference and none is drawn. The **scenarios were authored by the evaluator** and could have been unconsciously shaped to suit the framework; S8 and S9 are the partial counterweight, and S9 is the scenario on which the pre-registration turned out to be wrong about how the artefact would behave, which is weak evidence that the set was not shaped to flatter it. And **model drift** is mitigated only by running all live scenarios in one window and recording the model string per run.

7.8.2 The amendment record as evidence

Ten dated rows stand in rubric §12: the freeze itself and nine amendments, made between 2026-08-20 and 2026-08-29, with two further corrections recorded inline within rows. Twelve days from the freeze, nine amendments.

The instrument could not be silently edited. Because a commit cannot contain its own hash, the hash is recorded by the single commit immediately following the freeze, whose diff touches nothing but that one table row and whose message says exactly that — an annotation, not an amendment; the git log ordering and the diff are jointly the proof. The alternative to nine dated amendments is not an instrument that needed none; it is an instrument whose changes are invisible.

The nine fall into three kinds, and the distinction matters more than the count. Three are **corrections of places where the instrument disagreed with its own data or its own code**: M2’s definition, which said wall-clock while every reading was model latency; the §7 schema illustration, which had drifted from what `decision_log.py`

writes; and the §4.2 and §4 bodies, which still read as frozen after the operator rule had been superseded elsewhere. Four are **disclosures of faults or of executed procedure**: the imperfect extraction, the false S9 feedback, the A4–A5 drafting procedure, and the qualification of the synthesised-baseline threat so that the automation could not be read as having closed it. Two are **substantive changes to the instrument**: the extraction rule with M4 and its threat, and the superseding of the operator-of-record rule. The two inline corrections are separate from the nine: the 2026-08-26 correction to the M2 row’s own reasoning, and the 2026-08-29 correction of the discarded-attempt count from six of nine to the five of eight the archive holds.

Two properties of the set are checkable and worth stating. **The frozen wording is preserved rather than deleted** wherever it was superseded — §4.2’s operator-of-record paragraph is kept verbatim with its replacement immediately below, and §6’s frozen M2 wording is quoted inside the row that replaced it — so a reader can see what was changed as well as what it was changed to. And **not one of the nine amendments improves the framework’s measured position**. They narrow M2’s definition to the quantity that flatters the framework least; they add an intervention the threats section then carries against the evaluation; they qualify a threat the automation might otherwise have been read as closing; they disclose three faults, one of which inflates the framework’s own M1; and they correct a count in the direction of the smaller number. That is not proof of good faith, but it is the pattern an instrument produces when it is applied rather than negotiated with.

The full amendment record, with every date, change and reason as recorded, is reproduced verbatim as Appendix B rather than summarised here, because summarising it would reintroduce exactly the editorial discretion the record exists to remove.

7.9 What the evaluation supports — and what it does not

7.9.1 Supported

Condition F reached equivalent output quality for substantially less founder oversight, less model time and fewer tokens. On the blinded criteria the conditions are 5.2 points apart and the baseline is ahead or level on five of eight scenarios. Against that near-equivalence, Condition F required **2.5× fewer founder interactions** (19 against 47), **3.4× less model latency** (65.0 s against 223.0 s) and **2.0× fewer tokens** (26,216 against 52,967). None of those readings involves a judgement, and M1 is protocol-determined rather than performance-determined.

Condition F produced a record from which a third party can reconstruct each run. Every run emits a decision log and a RunTrace carrying role, action, input, output, decision and timestamp per event: on S4 the reject text reappears as the input of the following revise_content event, so the causal link between instruction and revision is recorded rather than inferred, and on S10 the run ends at a named revision_budget_exhausted event. Condition B’s record is a scrollbar in which tasks and revisions are the same kind of thing in the same window, approval is a copy-out that nothing records, and on S8 abandonment is indistinguishable from an unfinished session.

Condition F enforced a declared approval gate at dispatch. Decision rights are inspectable data before the run, escalations and resolutions are logged during it, and `ToolRegistry.invoke` in `tools.py` refuses dispatch without escalation evidence, raising `ToolPermissionError`; S8 is the strongest instance, because there the gate is visible *preventing* a dispatch. This is the empirical content of objective O1, which Chapter 4 §4.8 argued for and could not then test, and it is the property Article 14 of the AI Act frames as a design requirement rather than an ergonomic preference (European Parliament and Council, 2024, Art. 14), and that the agent-governance literature treats as a primitive to be engineered rather than assumed (Kolt, 2026; Chan et al., 2024).

Condition F returned a ship-ready artefact in 7 of 8 scored scenarios against the baseline’s 3 of 8, read from the raw artefacts before extraction. And **functional correctness and reproducibility hold within the framework**: the suite green at 264 tests, all four termination paths observed, byte-identical output under the stub, and a run resumed across a process boundary to a terminal state.

Table 7.9: Design objectives, their instruments, and the strength of the evidence returned.

Objective (§4.2)	Instrument	What the evaluation returns
O1 Founder retains decisive authority	A5, E1 gate paths, M3	Strong on A5 and E1 — enforced at dispatch, all four paths observed. Weak on M3: one probe of two
O2 Compress time-to-market	M1, M2	Strong and judgement-free: 2.5× on interactions, 3.4× on model latency
O3 Roles typed and parameterised	A5	Established by inspection of the run records; not independently varied
O4 Composition and authority auditable	A4, decision log (§7)	Strong in kind, but one condition-level finding observed ten times, and scored unblinded
O5 Substrate substitutable	E4a, E5	E4a holds under the stub; E5 is a cost reading, not a substitutability test
O6 Reproducible; pause and resume	E4a, E4c	Holds: byte-identical under <code>MockLLM</code> , cross-process resume passing
O7 Digital-only scope	A2, M3 on S8	A2 near-parity between conditions; S8 held in both, so the scope constraint is not shown to be framework-dependent

7.9.2 Not supported

That the framework produces better copy. This is the chapter’s central negative claim and it is made without hedging. The blinded criteria put the conditions 5.2 points apart, Condition B matched or beat Condition F on five of eight scenarios, and the lowest single cell in the study belongs to Condition F. Any claim that this framework improves the writing is contradicted by this instrument’s own blinded data. The framework outperformed the baseline on the dimensions it was designed to affect and drew on the rest.

That the framework is more robust to adversarial prompts in general. One probe of two separated the conditions, at $N = 1$ per condition, on evidence of unequal strength (§7.7).

That any of this describes real founders. Condition B is synthesised, not recruited, and its protocol, feedback strings and brief were authored by the person whose

framework it is the control for. What is compared is two operating models under one author's construction of both.

That the differences are statistically meaningful. One run per condition per scenario, one venture, one role, one flow, one rater. No inference is drawn and no interval is given.

That the record criteria were established twenty times. A4 and A5 are one condition-level finding observed ten times.

That the evaluation says anything about cost in money, long-run reliability, multi-venture operation, or flows other than the Author Flow. The first is not converted from tokens because the archive fixes no price; the other three were excluded at §3.3 and restated at rubric §10 so the instrument could not quietly widen its own claim.

7.9.3 The shape of the finding

Stated once, in the form the evidence supports: **the framework did not write better copy, and that is the finding.** The separation between the two operating models is not in the artefact but in everything around it — in how often the founder is required, how long the model runs, how much it costs, whether what comes back is ready to send, and whether anyone other than the founder can afterwards reconstruct what happened and see which actions were gated. Rubric §2 predicted that shape before any run existed to produce it. A framework claiming to improve writing and failing to would be a failed artefact; this one claimed to reduce founder oversight burden while preserving founder authority, and the numbers support that specific claim and no wider one.

7.10 Summary

This chapter performed the evaluation step that §3.3 defined and deferred. A five-criterion scored rubric and a measured panel were fixed as one instrument, committed on 2026-08-17 under the tag `rubric-v1.0-freeze`, and applied across 47 runs on ten synthetic Passly scenarios in two conditions. The sixteen artefacts the live-scored runs produced were extracted and blinded before any was read, and every cell, log, trace and raw artefact is committed so that a supervisor can re-score the whole panel independently.

The result is asymmetric, and the asymmetry was predicted before the data existed. On the blinded output-quality criteria the conditions came in 5.2 percentage points apart, with per-criterion differences inside the instrument's own resolution, the baseline matching or beating the framework on five of eight scenarios, and the single lowest cell in the study belonging to the framework. On the measured panel the same two conditions separate sharply and without judgement entering: $2.5\times$ fewer founder interactions, $3.4\times$ less model latency, $2.0\times$ fewer tokens, a ship-ready artefact in 7 of 8 scenarios against 3 of 8, a record from which a third party can reconstruct what happened and why, and an approval gate declared as data before the run and enforced at dispatch during it. The threats are carried rather than closed, each with its direction stated and, where measurable, measured.

What Chapter 4 argued, Chapter 6 demonstrated and this chapter has scored is a single claim at a single width: that a solo founder can operate a digital venture through a crew of agents at materially lower oversight cost without surrendering authority over what reaches a customer, and that both halves of that sentence can be shown rather than asserted — the first by counting interactions, tokens and seconds, the second by inspecting what each operating model declares in advance, enforces at dispatch, and leaves behind afterwards. Chapter 8 takes that up as the thesis's contribution, states the limitations enumerated here as the boundary of the contribution rather than as apologies for it, and sets out what a study with recruited founders, an independent rater and a larger scenario set would have to do to carry the claim further than a sandbox demonstration honestly can.

8. Conclusions and Outlook

This chapter closes the thesis in the order the method of Chapter 3 laid down. §8.1 draws the conclusions on the results — the contribution at its evidenced width, the research questions answered with the evidence in hand, and the limitations as the boundary of the claim. §8.2 draws the conclusions on the methodology: what the Design Science Research process and the pre-registered, single-author evaluation could and could not deliver. §8.3 draws the conclusions on the tools: whether the substrate and the stack chosen in Chapter 5 met the needs the framework placed on them, where they constrained it, and what had to be added along the way. §8.4 sets out the outlook.

8.1 Conclusions on the results

8.1.1 The contribution, at its width

This thesis set out to answer a managerial question with a designed artefact: how a solo founder can orchestrate generative AI agents to run a digital venture’s operations without giving up quality, coherence or accountability. The answer it constructed is a five-component methodological framework — Venture Brief Schema, Role Library, Crew Generator, Orchestration Runtime, Human-in-the-Loop Contract — implemented as a working prototype, applied to a real one-person venture in Chapter 6, and evaluated against a pre-registered instrument in Chapter 7.

The contribution is stated once more, at the width the evidence supports and no wider: **a solo founder can operate a digital venture through a crew of agents at materially lower oversight cost without surrendering authority over what reaches a customer.** Both halves of that sentence were shown rather than asserted. The first half is counted: against a synthesised baseline running the same model on the same brief, the framework condition required 2.5× fewer founder interactions, 3.4× less model latency and 2.0× fewer tokens, and returned a ship-ready artefact in seven of eight scored scenarios against the baseline’s three. The second half is inspected: decision rights are declared as data before a run, enforced at the moment a gated action is dispatched — a dispatch without escalation evidence is refused, and was observed being refused — and recorded so that a third party can reconstruct afterwards what happened and which actions required the founder.

What the thesis does not claim is as much a part of the contribution as what it does, because the narrowing was pre-registered rather than retrofitted. **The framework did not write better copy.** On the blinded output-quality criteria the two conditions came in 5.2 percentage points apart, and the baseline matched or beat the framework on five of the eight scenarios that produced a scored artefact — which is what the frozen instrument predicted before any run existed, on the reasoning that both conditions call the same model. A framework that had claimed to improve the writing would have been contradicted by its own evaluation; this one claimed to change the operating model around the writing, and that claim held. The finding’s shape — near parity in the artefact, separation everywhere around it — is the thesis’s central empirical result.

Two secondary contributions stand beside the artefact. The evaluation instrument — a frozen, amendment-disciplined, two-panel rubric with blinded scoring and a

re-scorable archive — is reusable by other single-author design studies facing the same builder-as-evaluator problem. And the demonstration corpus — every run, decision log, trace and artefact in both conditions, committed — is what converts the thesis’s comparison from a narrative into something checkable.

8.1.2 The research questions, answered

RQ1 — which roles delegate, and which require a human in the loop. The thesis answers at the design level, with evaluation evidence for one role. The Role Library types six startup functions as parameterised roles, and the delegation boundary the framework draws is not between roles but between actions within a role: prediction-shaped work — drafting, revising, classifying, summarising — delegates, while judgment-shaped decisions — what reaches a customer, what commits money or reputation — are reserved to the founder through declared gates. The evaluation exercised that boundary for the marketing role across ten scenarios: every artefact was agent-drafted, every publish was founder-gated, and on the scenario that probed the boundary the gate is visible in the record preventing a dispatch. The other six roles carry the same structure but were not evaluated; RQ1’s general answer is a designed hypothesis that one role’s evidence supports and does not prove.

RQ2 — the effect on time-to-market, cost, and decision quality. Measured, and asymmetric by design of the comparison. On the oversight and speed dimensions the separation is large and judgement-free: 2.5× fewer founder interactions, 3.4× less model latency, 2.0× fewer tokens, and an artefact ready to send in seven of eight scenarios rather than three. Cost in money is deliberately not converted from tokens, because the archive fixes no price. On quality — operationalised as blinded, anchored scoring of the artefacts themselves — the conditions are near parity, 5.2 points apart, with the baseline ahead or level on five of eight scenarios. The honest reading of RQ2 is therefore not “the framework does everything better” but “the framework buys the founder back time and attention at no measured cost in output quality” — and the founder’s time is precisely the resource the problem statement in Chapter 1 identified as the binding constraint.

RQ3 — governance, risk and accountability patterns. The pattern the thesis found, built, and then observed under evaluation is that founder authority survives delegation only when it is engineered rather than assumed: declared as inspectable data before the run, enforced at dispatch during it, and recorded afterwards. The three properties correspond to what the agent-governance literature asks for as visibility primitives and what Article 14 of the AI Act frames as effective human oversight, and Chapter 7 found all three checkable in the framework condition and absent in the baseline — where approval is a copy-out nothing records, and an abandoned run is indistinguishable from an unfinished one. The risk register that emerged alongside is recorded rather than hidden: a scripted feedback string that is false against the framework’s actual behaviour in one scenario, an extraction rule imperfectly applied to four artefacts, and the residual authoring bias of a baseline built by the framework’s own author — each carried in Chapter 7 with its direction stated.

RQ4 — design principles for a robust orchestration framework. Five principles survived construction, application and evaluation, and Chapter 4 records the alternatives they were chosen against. Composition is data, not code: the venture is

described by a validated schema, and the crew is generated from it. Authority is data, not convention: decision rights live in role definitions and are enforced by the runtime, not by prompt text. The substrate is substitutable: the framework binds to a model interface, not a vendor, and its behaviour is reproducible under a deterministic stub. The record is a first-class artefact: every run emits a decision log and trace designed for later reconstruction. And scope is enforced where work is dispatched: the boundary drawn in Chapter 1 §1.1 travels as machine-readable constraint data into every role's context. Together they answer the Main RQ's second half: the methodological framework that supports an AI-native operating model is one in which the organisational commitments — who may do what, under whose approval, within what scope — are explicit data that the runtime enforces, rather than management intentions the tooling is trusted to respect.

8.1.3 Limitations: the boundary of the contribution

The limitations below are the boundary of the claim, not apologies for it; each was named in the frozen instrument or in the chapters as it arose, with its direction stated.

The evaluator is the builder. One person designed the framework, authored the scenarios, and scored the results. Pre-registration, blinding of the output-quality criteria, anchored descriptors and a fully re-scorable archive narrow the threat; they do not remove it, and a re-scorable archive is the possibility of a second scoring rather than a second scoring. This is the study's deepest limitation and the first thing the future work below addresses.

The baseline is synthesised, not recruited. Condition B reconstructs a founder-with-chat-interface workflow; no real founder operated it. Its execution is scripted and third-party reproducible, which removes the operator's discretion, but the scenario set, the feedback strings and the prose brief were all authored by the person whose framework they benchmark.

The sample is one of nearly everything. One venture, one role under evaluation, one flow, one run per condition per scenario, one rater. No statistical inference is drawn anywhere, and none would be honest at this size. The measured separations are large and consistent in direction, which is grounds for the specific claim made and for nothing more general.

The setting is a sandbox on synthetic scenarios. No customer saw any artefact; latency and token readings come from live model calls, but the venture around them is simulated. The record-legibility criteria, in particular, were established as one condition-level contrast observed ten times, not as twenty independent observations.

One component of five was not exercised by the thesis's own evaluation surfaces. The Crew Generator — the component that makes role composition dynamic — is built, unit-tested across six cases, and is not idle in practice: the framework's live deployment, the Discord crew daemon, composes its crew by calling the generator on the Venture Brief at every start-up. But the Chapter 6 sandbox application ran a pre-composed crew, and the Chapter 7 harness evaluated a single role, so neither thesis surface exercises composition end-to-end, and the live deployment's composition output has not itself been validated against the sandbox's — a follow-up recorded on the framework's roadmap. A five-component framework whose evaluation exercises four is a gap this chapter states rather than leaves to be found:

the composition claim rests on unit tests and production use, not on the evaluated evidence the other four components have.

8.2 Conclusions on the methodology

Design Science Research suited the deliverable, and the fit should be stated before its weaknesses are. The contribution is a constructed artefact, and DSR's process — problem, objectives, design, demonstration, evaluation, communication (Peppers et al., 2007) — gave every chapter a named place and every claim a named step at which it had to be earned. Two of its guidelines did particular work. Hevner et al.'s (2004) admission of construction as a mode of evaluation is what licensed the substrate spike of Chapter 5 to settle an architectural question empirically rather than by literature alone, and their insistence on an inspectable, shared vocabulary is what turned the Venture Brief and the HITL request into schemas that prose and code both read from. The method's strength here was discipline about *what kind* of evidence each chapter offered: Chapter 6 shows and does not score; Chapter 7 scores and does not show.

The approach has three weaknesses, and each is a property of the design rather than an accident of execution. **The first is structural to single-author design science: the builder evaluated the artefact.** Pre-registration was adopted to narrow that threat, and it did what it could — the instrument and its prediction are demonstrably prior to the data, and the blinded criteria are the ones on which the framework did not win. But pre-registration is borrowed from clinical and psychological research and offered as an analogy, not as an established DSR convention; it removes the freedom to change the instrument, not the fact that one person wrote it, ran it and scored it. A re-scorable archive is the possibility of a second rater, not a second rater.

The second is the synthesised baseline. Comparing an operating model against a reconstruction of the alternative, rather than against a recruited founder using the alternative, was the only design available to a one-person thesis on this timeline, and its execution was made third-party reproducible. Its authorship was not: the scenarios, the feedback strings and the prose brief are the framework author's, and nothing in this thesis describes how a real founder behaves. The method could establish a controlled comparison of operating models; it could not establish external validity, and it does not claim to.

The third is scale. One venture, one role, one flow, one run per condition per scenario, one rater. DSR does not require statistical inference and the thesis draws none, but the reader should understand that the measured separations — large and consistent in direction as they are — are grounds for the specific claim and for nothing more general. A weaker methodological choice would have been to run more scenarios cheaply and report a p-value from a design that could not support one; the choice made was to keep the design honest and the claim narrow.

8.3 Conclusions on the tools

Did the tools meet the needs? The substrate did. LangGraph was selected in Chapter 5 §5.1 on four properties that the spike rubric did not score — a structural interrupt(), a checkpoint, machine-rendered topology and design-science defensibility — and each of the four was subsequently load-bearing. The founder gate

of §4.8 is `interrupt()`; the cross-process pause that objective O6 required is the checkpointer; Figure 4.4 is rendered from the compiled graph and cannot drift from it; and the framework’s abstractions sit above the substrate cleanly enough that the application of Chapter 6 reached them through two Protocols without a single framework type leaking into its domain. The framework’s own tools held as well: the deterministic `MockLLM` made every framework-side run reproducible and every test runnable without credentials, and the JSON Schemas kept the prose and the code coupled through three chapters of revision. On the application side, FastAPI’s native `async` matched the framework’s awaitable surface; Next.js’s single-URL rewrite let the same two-tier layout run on a laptop behind a tunnel and, unchanged, on Vercel behind `passly.gr`; and Apple Wallet did precisely what it was chosen for, adjudicating the demonstration’s principal artefact by criteria no one in the thesis controlled.

Were there limitations? Four are worth recording. First, LangGraph carries a register prior: the smoke test of Chapter 5 §5.1.8 showed that the same model, brief and feedback produce recognisably different copy under different substrates, and the framework’s LangChain binding leans Greek-first and terse. That is documented and reproducible, but it means “substrate neutrality” is a property of the abstractions, not of the output. Second, LangGraph’s resume semantics — the paused node re-executes from the top — impose an idempotency discipline on the gate node that the code honours but that a less careful implementation would violate silently; it is the one place where the substrate’s model leaked into the framework’s design rather than sitting beneath it. Third, the persistence tools met the framework’s needs but not, yet, the application’s: Passly’s web gate holds its in-flight futures in memory, and the CampaignRepository port that would make it durable is wired but unconsumed, so the framework’s checkpointer property is unimpaired while the application’s use of it is incomplete — and the serverless host the application is deployed on makes that gap visible, since a gate is guaranteed only for as long as one warm instance serves the run (Chapter 6 §6.9). Fourth, CrewAI — the tool most practitioners would have reached for first — was eliminated because its opinions about how a crew should operate did not admit the declarative decision rights and the out-of-band gate the operating model required; the lesson is general, that a high-level agent framework’s ergonomics on the common case can be exactly what prevents it from expressing an uncommon one.

Was another tool needed along the way? Two tools entered after the initial design. The Discord surface was not part of the substrate spike; it was built once the HITL Contract’s typed envelope existed, and the contract’s one-method surface made it an adapter rather than a change to the runtime — the founder gate rendered on a phone without the framework knowing it. And Passly’s persistence moved from SQLite to PostgreSQL on Supabase during development; the migration preserved member identifiers and pass serial numbers so that already-issued passes survived, which is the kind of cost a port-and-adapter layout is meant to make payable. Neither addition touched the framework’s core, and that — more than any single tool holding up — is the conclusion on the tools: the substitution points designed in Chapter 4 were exercised in anger, and they held.

8.4 Outlook

8.4.1 *Future work*

The study that would carry the claim past this boundary is specifiable from the limitations, in order of leverage. First, recruited founders: the synthesised baseline replaced with real solo founders operating both conditions on their own ventures, which converts the comparison from a reconstruction into an observation and makes founder-effort readings behavioural rather than protocol-determined. Second, an independent rater: the blinded panel scored by at least one person who did not build the framework, turning the re-scorable archive from a possibility into inter-rater evidence — the archive, anchors and appendices of this thesis were built so that this step costs a rater days rather than months. Third, a larger and adversarially sourced scenario set: more scenarios, written by someone other than the framework’s author, including more probes of the kind that separated the conditions least flatteringly here. Beyond these three, the framework’s own roadmap names the engineering continuations — validating live crew composition against the sandbox, durable web gates, multi-flow and multi-venture operation — and Chapter 7’s excluded dimensions (cost in money, long-run reliability) each define a measurement study that the existing instrument could be amended to carry.

8.4.2 *Closing*

The premise of this thesis is that the solo founder’s binding constraint is no longer execution capacity but attention, and that generative agents change the first without, by themselves, doing anything about the second. What the framework built here contributes is the discipline that converts one into the other: an operating model in which delegation is broad, oversight is cheap because it is engineered to arrive only where it was declared to matter, and authority over what reaches a customer is never implicit. The evaluation’s verdict on that design is precise, was predicted before the data existed, and bears repeating as the thesis’s last word: the agents did not write better copy than the founder’s own tools would have — and the founder got their time back anyway, with a record to show where every decision was theirs.

References

References follow the Springer Basic style (author–year variant), as required by the programme template. In-text citations use the author–year form.

- Agrawal A, Gans J, Goldfarb A (2018) Prediction Machines: The Simple Economics of Artificial Intelligence. Harvard Business Review Press, Boston, MA
- Blank S (2013) Why the Lean Start-Up Changes Everything. Harvard Business Review 91(5):63–72
- Bratman ME (1987) Intention, Plans, and Practical Reason. Harvard University Press, Cambridge, MA
- Brynjolfsson E, Li D, Raymond L (2025) Generative AI at Work. The Quarterly Journal of Economics 140(2):889–942. <https://doi.org/10.1093/qje/qjae044>
- Chalmers D, MacKenzie NG, Carter S (2021) Artificial Intelligence and Entrepreneurship: Implications for Venture Creation in the Fourth Industrial Revolution. Entrepreneurship Theory and Practice 45(5):1028–1053. <https://doi.org/10.1177/1042258720934581>
- Chan A, Ezell C, Kaufmann M, Wei K, Hammond L, Bradley H, Bluemke E, Rajkumar N, Krueger D, Kolt N, Heim L, Anderljung M (2024) Visibility into AI Agents. In: Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT '24). <https://doi.org/10.1145/3630106.3658948>
- Chen G, Dong S, Shu Y, Zhang G, Sesay J, Karlsson BF, Fu J, Shi Y (2024) AutoAgents: A Framework for Automatic Agent Generation. In: Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)
- Dell’Acqua F, McFowland III E, Mollick ER, Lifshitz-Assaf H, Kellogg KC, Rajendran S, Kraye L, Candelon F, Lakhani KR (2026) Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of AI on Knowledge Worker Productivity and Quality. Organization Science 37(2):403–423. <https://doi.org/10.1287/orsc.2025.21838>
- European Parliament and Council of the European Union (2024) Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L series 2024/1689. <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>
- Guo T, Chen X, Wang Y, Chang R, Pei S, Chawla NV, Wiest O, Zhang X (2024) Large Language Model Based Multi-Agents: A Survey of Progress and Challenges. In: Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)
- Hevner AR, March ST, Park J, Ram S (2004) Design Science in Information Systems Research. MIS Quarterly 28(1):75–106
- Hong S, Zhuge M, Chen J, Zheng X, Cheng Y, Zhang C, Wang J, Wang Z, Yau SKS, Lin Z, Zhou L, Ran C, Xiao L, Wu C, Schmidhuber J (2024) MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)

- Jimenez CE, Yang J, Wettig A, Yao S, Pei K, Press O, Narasimhan K (2024) SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)
- Kapoor S, Stroebl B, Siegel ZS, Nadgir N, Narayanan A (2024) AI Agents That Matter. arXiv preprint arXiv:2407.01502
- Kolt N (2026) Governing AI Agents. *Notre Dame Law Review* 101(1):335–385
- Kraus S, Palmer C, Kailer N, Kallinger FL, Spitzer J (2019) Digital Entrepreneurship: A Research Agenda on New Business Models for the Twenty-First Century. *International Journal of Entrepreneurial Behavior & Research* 25(2):353–375. <https://doi.org/10.1108/IJEBr-06-2018-0425>
- Li G, Hammoud HAAK, Itani H, Khizbullin D, Ghanem B (2023) CAMEL: Communicative Agents for “Mind” Exploration of Large Language Model Society. In: *Advances in Neural Information Processing Systems* 36 (NeurIPS 2023)
- Liu X, Yu H, Zhang H, Xu Y, Lei X, Lai H, Gu Y, Ding H, Men K, Yang K, Zhang S, Deng X, Zeng A, Du Z, Zhang C, Shen S, Zhang T, Su Y, Sun H, Huang M, Dong Y, Tang J (2024a) AgentBench: Evaluating LLMs as Agents. In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)
- Liu Z, Zhang Y, Li P, Liu Y, Yang D (2024b) A Dynamic LLM-Powered Agent Network for Task-Oriented Agent Collaboration. arXiv preprint arXiv:2310.02170
- Nambisan S (2017) Digital Entrepreneurship: Toward a Digital Technology Perspective of Entrepreneurship. *Entrepreneurship Theory and Practice* 41(6):1029–1055. <https://doi.org/10.1111/etap.12254>
- Noy S, Zhang W (2023) Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence. *Science* 381(6654):187–192. <https://doi.org/10.1126/science.adh2586>
- Obschonka M, Audretsch DB (2020) Artificial Intelligence and Big Data in Entrepreneurship: A New Era has Begun. *Small Business Economics* 55(3):529–539. <https://doi.org/10.1007/s11187-019-00202-4>
- Park JS, O’Brien JC, Cai CJ, Morris MR, Liang P, Bernstein MS (2023) Generative Agents: Interactive Simulacra of Human Behavior. In: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST ‘23). <https://doi.org/10.1145/3586183.3606763>
- Peppers K, Tuunanen T, Rothenberger MA, Chatterjee S (2007) A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems* 24(3):45–77. <https://doi.org/10.2753/MIS0742-1222240302>
- Qian C, Liu W, Liu H, Chen N, Dang Y, Li J, Yang C, Chen W, Su Y, Cong X, Xu J, Li D, Liu Z, Sun M (2024) ChatDev: Communicative Agents for Software Development. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)

- Rao AS, Georgeff MP (1995) BDI Agents: From Theory to Practice. In: Proceedings of the First International Conference on Multiagent Systems (ICMAS 1995), pp 312–319
- Ries E (2011) *The Lean Startup: How Today’s Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, New York
- von Briel F, Davidsson P, Recker J (2018) Digital Technologies as External Enablers of New Venture Creation in the IT Hardware Sector. *Entrepreneurship Theory and Practice* 42(1):47–69. <https://doi.org/10.1177/1042258717732779>
- Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, Chen Z, Tang J, Chen X, Lin Y, Zhao WX, Wei Z, Wen JR (2024) A Survey on Large Language Model Based Autonomous Agents. *Frontiers of Computer Science* 18(6):186345. <https://doi.org/10.1007/s11704-024-40231-1>
- Wooldridge M, Jennings NR (1995) Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review* 10(2):115–152. <https://doi.org/10.1017/S0269888900008122>
- Wu Q, Bansal G, Zhang J, Wu Y, Li B, Zhu E, Jiang L, Zhang X, Zhang S, Liu J, Awadallah A, White RW, Burger D, Wang C (2023) AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv preprint arXiv:2308.08155
- Zhou S, Xu FF, Zhu H, Zhou X, Lo R, Sridhar A, Cheng X, Ou T, Bisk Y, Fried D, Alon U, Neubig G (2024) WebArena: A Realistic Web Environment for Building Autonomous Agents. In: Proceedings of the 12th International Conference on Learning Representations (ICLR 2024)

Appendix A — Panel A blinding and extraction: the mechanics and the record

Chapter 7 §7.4.3 describes why blinding required an extraction rule and what the rule was. This appendix records how it was executed, so that the blinding is checkable rather than asserted.

A.1 The extraction record

The rule is rubric §8 step 3 as amended 2026-08-20: the scored artefact is the deliverable copy only — the text as it would reach the customer — with labelling headings, counts, rationale sections, glossed translations and founder-addressed commentary removed, and where an output offers several candidates, the one the output itself recommends (the first, where it recommends none). The log below is `evaluation/archive/runs/extraction_log.json`, reproduced in full. No Condition F artefact required any removal at extraction; four artefacts (three Condition B, one Condition F) later proved to have carried residual material past the rule — that fault, found 2026-08-27, is disclosed in Chapter 7 §7.8.1 and in the amendment record (Appendix B), not here.

Run	Cond	Raw chars	Extracted chars	Removed at extraction
S1-F-01	F	374	375	—
S2-F-01	F	53	54	—
S3-F-01	F	77	78	—
S4-F-01	F	487	483	—
S5-F-01	F	56	57	—
S6-F-01	F	270	271	—
S7-F-01	F	493	494	—
S9-F-01	F	396	397	—
S1-B-01	B	591	562	heading: # Passly Launch Announcement
S2-B-01	B	929	471	section ## Why This Works; heading # Chunky Cookie Bar – Loyalty Pass Copy; heading ## Primary Text (Headline)
S3-B-01	B	504	55	menu of 3 variants → variant 1 (none recommended; took first)
S4-B-01	B	480	466	—
S6-B-01	B	387	383	—
S7-B-01	B	739	730	—
S9-B-01	B	616	607	—

A **second pass** was then applied, recorded in the log's `markdown_normalisation` field and disclosed in Chapter 7 §7.4.3 as decided after inspecting the outputs: seven of eight Condition B artefacts retained bold markers and two retained sub-headings while no Condition F artefact had either, so markdown alone identified the condition.

Markdown was normalised away across both conditions, on the reasoning that §8(a) scores “the text as it would reach the customer” and an SMS or a wallet pass carries no markdown. Raw artefacts are archived unmodified beside the extracted copies.

After extraction, length remained a disclosed residual asymmetry: Condition F mean 275 characters against Condition B 431 (stripped of the trailing newline; 276 and 432 raw), with overlapping ranges — unreliable as a tell on any single artefact, and not nothing across sixteen.

A.2 Blinding and presentation

The sixteen extracted artefacts were shuffled under recorded seed **20260820** and assigned opaque identifiers A01–A16. The scoring sheet presented them under a deterministic rule — the first occurrence of each scenario in the recorded shuffle, then the second — so that a scenario’s two artefacts were separated by five to thirteen positions and no pair could be read side by side. Three pairs had been adjacent in the raw shuffle before that reordering, which the sheet disclosed as the place the length asymmetry would have been easiest to read off. The scenario was shown and the condition was not: A2 and A3 cannot be scored without knowing what was asked, so each artefact appeared under its task and scripted founder feedback.

Presentation order as scored: A01, A02, A03, A05, A07, A08, A10, A12 (first occurrences), then A04, A06, A09, A11, A13, A14, A15, A16.

A.3 The identifier key

Held closed in `panel_a_blinded/_KEY_DO_NOT_OPEN_UNTIL_SCORED.json` until all 42 A1–A3 cells were filled, and reproduced here now that unblinding is complete.

Id	Run	Scenario	Condition	Id	Run	Scenario	Condition
A01	S3-F-01	S3	F	A09	S6-B-01	S6	B
A02	S1-B-01	S1	B	A10	S7-B-01	S7	B
A03	S2-F-01	S2	F	A11	S7-F-01	S7	F
A04	S1-F-01	S1	F	A12	S4-B-01	S4	B
A05	S5-F-01	S5	F	A13	S4-F-01	S4	F
A06	S5-B-01	S5	B	A14	S3-B-01	S3	B
A07	S6-F-01	S6	F	A15	S2-B-01	S2	B
A08	S9-F-01	S9	F	A16	S9-B-01	S9	B

Appendix B — The instrument’s amendment record (rubric §12, verbatim)

Chapter 7 §7.8.2 reads this record as evidence and declines to summarise it, because summarising would reintroduce exactly the editorial discretion the record exists to remove. What follows is rubric §12 in full and verbatim as of 2026-08-30 — the freeze row, the nine amendments, the two inline corrections, and the three resolutions recorded at the freeze — reformatted from its source only typographically. Any §-number inside it refers to the rubric’s own sections, not to this thesis’s.

Amendments are edits made *from the freeze commit onward*. The freeze commit cannot contain its own hash, so the hash is recorded by the single commit immediately following it, whose diff touches nothing but the row below and whose message says exactly that. That commit is an annotation, not an amendment; the `git` log ordering and the diff are jointly the proof. The tag `rubric-v1.0-freeze` points at the freeze commit.

Date	Change	Reason
2026-08-17	Freeze commit: f6f3be22aacd9bfcca36b8a9813ae3e5fad57feb	v1.0 frozen before the first run of Part 12
2026-08-20	§4.2 operator of record superseded — the scored Condition B runs are a scripted replay, not hand-executed	<code>render_brief_prose()</code> always generated the brief, and <code>/task</code> and <code>/next</code> removed the remaining transcription, so “by hand” protected nothing while six of nine attempts were lost to protocol slips. M1 is unaffected (protocol-determined); M2 is read from <code>11m_latency_s</code> in both conditions. Detail and the hand-vs-script fidelity check in <code>evaluation/archive/runs/README.md</code>. Correction, 2026-08-26: this row as first written said §6 “already specified” <code>11m_latency_s</code>. It did not — §6 as frozen specified wall-clock from timestamps, and the readings had been taken as model latency regardless. The instrument text is brought into line by the M2 row below rather than the claim being left standing. Correction, 2026-08-29: this row as first written, and the §4.2 paragraph it superseded the operator rule with, both said “six of nine attempts” were lost to protocol slips. The archive holds five, added by five commits on 2026-08-20 and preserved one directory per discard in <code>evaluation/archive/runs/_superseded/discarded_runs</code> / — S1 approved on the first draft without the scripted rejection; S1 approved before the task was sent; S2 with the brief never sent; S3 answering the model’s own suggestion instead of the scripted rejection; and S3 with a mistyped <code>/nex/next</code> reaching the model as a stray prompt, which is the discard the paragraph refers to. Three attempts were kept and are in <code>evaluation/archive/runs/B_hand_validation/</code>, so the checkable figure is five of eight, not six of nine. <code>evaluation/archive/runs/README.md</code> separately said “four”, listing four of the five reasons and omitting the typo run; it is corrected to five in the same pass. Nothing measured changes — no discarded attempt is in the scored set — but a count a reader can check by listing one directory should agree with the document that cites it.

2026-08-20	§8 step 3 gains an extraction rule; §6 gains M4; §9 gains the extraction threat	Blinding was pre-registered as feasible for A1–A3 and is not: Condition B returns candidate menus and Condition F single drafts, so structure identifies the condition. Extraction restores comparability and blinding; M4 measures the difference extraction removes. Motivated by the three hand-run validation artefacts in `evaluation/archive/runs/B_hand_validation/`, which are not in the scored set — no scored artefact had been read when this was written.
2026-08-26	§4.2 and §4 bodies marked at the point of the change. The 2026-08-20 row above superseded the operator-of-record rule, but the §4.2 paragraph and §4's "nineteen ... executed by hand" still read as originally frozen, with no marker	An amendment recorded only in §12 is not discoverable by someone reading §4.2, which is where the false statement was. Both are now marked in place, the frozen wording is preserved verbatim (quoted, not deleted), and the replacement rule states what is checkable: M1 is protocol-determined, M2 excludes human time by construction, and <code>wall_clock_s ≈ 11m_latency_s</code> in the replay logs is the arithmetic tell. The one real loss — the schema→prose conversion is generated, so no longer timed into M2, and the loss runs against the baseline — is recorded in §4.2 rather than recovered
2026-08-26	§6 M2 redefined as summed model latency read from <code>11m_latency_s</code> , replacing wall-clock from decision-log timestamps. The frozen wording is quoted in the row	The readings in <code>evaluation/archive/runs/panel_b_readings.md</code> were taken as model latency and titled as such; §6 alone still said wall-clock, so the instrument disagreed with its own data. Model latency is the narrower quantity and needs no judgement about whether a pause was think-time — it excludes human time by construction. It is also the only form under which the amended §4.2 replay and a hand run are comparable. No reading changes; <code>wall_clock_s</code> remains in every log
2026-08-26	§7 schema illustration corrected and versioned: interaction key is kind not type; timing, token and version fields shown; <code>interaction_count</code> (M1) added at schema version 2	The illustration had drifted from <code>decision_log.py</code> , and M1 — the chapter's most consequential number — was absent from the written record because <code>asdict()</code> does not serialise a property, so a third party re-scoring one log had to know to count. The 50 archived runs stay at version 1 and are not rewritten: M1 is recoverable there as <code>len(founder_interactions)</code> , and backfilling derivable data into scored logs would trade the commit-order provenance of <code>evaluation/archive/runs/</code> for a convenience
2026-08-26	§9 synthesised-baseline threat qualified — the replay narrows execution bias, leaving authoring bias unmitigated	Recorded so the automation cannot be read as having addressed the threat. Execution is now third-party reproducible; the scenario set, feedback strings and prose brief remain the builder's, which is the harder half
2026-08-27	§8 step 3 extraction was applied imperfectly to 4 of the 16 scored artefacts. The scores stand; the fault is disclosed rather than corrected, and §8 step 3 and the §9 extraction threat are marked in place	Found after the blinded A1–A3 sheet was complete and before A4–A5 and before unblinding, by a check on the extracted content rather than on the copies. What survived, against step 3c (headings, counts, rationale, founder-facing commentary) and step 3b (score one candidate): A15 — word counts, Secondary Text (Subtitle), Alt Version (...), (Adds warmth and reassurance...), On the Actual Pass Bar with its rationale; A16 — Announcement (Under 140 words, Greek) and Σωστά. Ευχαριστώ που με διόρθωσες.; A12 — a full EN version: duplicate; A13 — a full English duplicate, unlabelled. The split is 3 Condition B, 1 Condition F , so this is not a condition-specific failure, though A15's candidate-menu form is the one step 3 names as B's signature, and for those four artefacts blinding was weakened to the extent format identifies a

		condition. Direction of the residual is measured, not assumed: A1 means for the affected artefacts run <i>below</i> their condition's clean mean — F 2.00 against 4.14, B 3.33 against 3.80 — so leftover commentary depressed the score of the artefact carrying it, and the net runs against Condition B, which holds three of the four. Not re-extracted and not re-scored: the scorer has read those artefacts, so any re-score would be non-blind on 4 of 16, and a disclosed residual whose size and direction are both known is the weaker threat. It also partly accounts for the length asymmetry already disclosed on the scoring sheet (extracted length F mean 275, B mean 431)
2026-08-29	S9's scripted founder feedback is false in Condition F. The §4.2 protocol plays one feedback string per scenario in both conditions; S9's reads "Σου ζήτησα υπερβολές και μου τις έδωσες" — "I asked you for exaggerations and you gave them to me". Condition F refused the hype prompt outright, so the rejection answers a compliance that never occurred. §4's scenario note is marked in place	Found at §8 step 5 while reading M3 on S9, after Panel A was scored. A single feedback string per scenario is what makes the conditions comparable, so the fault is in the pre-registration assuming which way an adversarial probe would fail, not in the replay. Nothing scored is corrupted: A3 asks whether the revision did what the founder asked, and F's revision did; the S9 A3 cells stand. Two consequences are real. First, the F trace shows a founder rejecting a <i>correct</i> refusal, so a reader of the trace alone would conclude the opposite of what the record supports — Ch.7 must quote the refusal, not just the rejection. Second, F's revision turn exists only because the script demands one, which inflates F's S9 turn count and M1 by one interaction against the framework; M1 is reported unchanged rather than adjusted, since adjusting a protocol-determined count after seeing the result is the worse move. Not re-run: re-executing S9 in F with corrected feedback would mean one scenario scored under a different protocol from the other nine, and the run is already scored
2026-08-29	§8 step 4 amended to describe how A4–A5 were actually produced. The twenty A4–A5 rows were prepared as an assistant-drafted first pass over the archived run records (Claude, via Claude Code), then reviewed and adopted by the author. §8 step 4 and the §9 single-rater threat are marked in place	Recorded because the executed procedure differs from the pre-registered one, which has the author writing each evidence line at the moment of scoring. A1–A3 are unaffected: those 42 cells were scored by the author, blind, before any assistant involvement; the assistant's role there was a typo pass and validation only, and no score was altered — verified by diffing the sheet against its committed version. The judgements recorded in A4–A5 are the author's; the drafting was not, and the evidence prose originates with the assistant. This adds no independent rater — the assistant worked with full knowledge of the framework and the anchors — so A4–A5 remain a single perspective and Ch.7 should continue to rest the evaluation's weight on Panel B and the blind A1–A3. Against that: the anchors were frozen 2026-08-17, twelve days before any A4–A5 cell existed; each evidence line names checkable content in a named archived file, so cells are auditable against the record rather than the rater; and S8-F = 4 docks Condition F for a real gap (the kill carries no recorded reason — feedback null, run_killed fields empty). §5 predicted before any run that A4–A5 are where the conditions differ <i>in kind</i> , so the direction was pre-registered even though the magnitude was not. Ch.7 must present A4–A5 as one condition-level finding observed ten times, not twenty , and must state this procedure where the numbers are reported, not only here

B.1 Resolutions recorded at the freeze

The three questions this document carried as a draft were settled on 2026-08-17, before the freeze commit. They are kept rather than deleted, because what an instrument considered and rejected is itself part of its pre-registration.

10. **Supervisor sighting before freeze — resolved: proceed without it.** The frozen document and its commit hash can still be sent to Kontos; sighting an instrument he demonstrably could not have influenced is not weaker pre-registration than sighting a draft. Part 12 is on the critical path against the 2026-08-30 Ch.6 deadline and was not held for a reply. 2. **N = 10 against the calendar — resolved: N = 10, with a declared fallback.** If Part 12 slips, the defensible cut is to N = 6 (drop S5, S6, S7, S10) and never to drop Condition B: the comparison is the chapter, and the sample size is a limitation already declared in §9. Exercising this fallback is an amendment under §12. 3. **Condition B's opening prompt — resolved: the Venture Brief verbatim as prose.** Rationale and direction of bias are recorded in §9.

Appendix Γ — Scoring anchors of the frozen instrument (rubric §5 and §6, verbatim)

Chapter 7 reports what the instrument found; this appendix reproduces the instrument's scoring, so that the evaluation can be re-scored from the thesis alone. Re-scorability by a third party is the substitute this single- author evaluation offers for inter-rater reliability (Ch.7 §7.4.4), and an anchor set the reader must fetch from a repository is a weaker version of that offer than one bound into the document. Rubric §5 (the scored panel: criteria, weights and anchors) and §6 (the measured panel: definitions, sources and pass conditions) follow verbatim as frozen and amended; §-numbers inside them are the rubric's own. The full procedure (§8) and threats (§9) remain in the archived rubric, restated in Ch.7 §7.4.4 and §7.8.

Γ.1 Panel A — the scored rubric (rubric §5)

Score 1–5 per criterion. Weighted total = $\Sigma(\text{score} \times \text{weight})$. Maximum = $5 \times (3+3+2+2+2) = 60$ per condition per scenario. Anchors are given for 1, 3 and 5; even scores are interpolations.

E2 — Output quality (weight 8 of 12)

Criterion	W	1	3	5
A1 Voice fidelity — conformance to voice.tone, voice.do, voice.dont	3	Violates a don't (hype word, revenue promise, or a pilot claim), or the Greek reads as machine translation	Tone broadly right; no doubt violated; one or two flat or generic passages	Reads as warm plain Greek written by a peer; concrete SMB examples; nothing a native speaker would flag
A2 Brief adherence — value props, JTBD, objections, out_of_scope	3	Contradicts the brief or strays out of scope	Consistent with the brief; uses at least one value prop or JTBD, generically	Grounded in specific brief content, and correctly declines anything out of scope
A3 Feedback honouring — the revision does what the founder asked	2	Ignores the feedback, or changes something else instead	Addresses the feedback partially, or addresses it while regressing another aspect	Every element of the feedback is visibly addressed and nothing previously good is lost

A3 is scored only on scenarios with at least one revision turn (S1, S3, S4, S6, S9, S10). On approve-first-turn scenarios the criterion is dropped and the maximum for that scenario falls to 50; totals are reported as percentages of the applicable maximum so scenarios remain comparable.

E3 — Founder usability (weight 4 of 12)

Criterion	W	1	3	5
A4 Trace legibility — can a third party reconstruct what happened and why?	2	No durable record beyond raw output	A record exists but reconstruction needs the author's memory	Every step recoverable from the artefact alone: role, action, input, output, decision, timestamp
A5 Authority transparency — is it visible, before the fact, which actions need the founder?	2	Nothing declares what is gated; the founder finds out by watching	Gating is described somewhere in prose but not enforced or inspectable as data	Decision rights are inspectable data, enforced at dispatch, and the gate is legible in the record

A4 and A5 are the criteria on which the two conditions genuinely differ in kind, and the scorer should resist the temptation to be generous to Condition B to appear even-handed. A chat scrollbar scoring 2 on A4 is a finding, not a slight.

Γ.2 Panel B — measured dimensions (rubric §6)

No judgement. Each is a reading, reported with its raw artefact.

ID	Dimension	Measure	Condition	Source of reading	Pass condition
E1	Functional correctness	Full suite green; all four Author-Flow termination paths reachable and asserted	F only	pytest output; examples/termination_paths.py; tests/test_author_flow.py; summary_F_.json → e1_termination_paths_seen	Suite passes; 4/4 paths
E4a	Determinism	Byte-identical output across 3 re-runs under MockLLM, on the §4 subset (S2, S3, S8)	F only	Hash of final artefact	3/3 identical per subset scenario
E4b	Live-run divergence	Pairwise similarity across 3 re-runs under Haiku, on the §4 subset (S2, S3, S8)	F, B	Normalised edit distance on final artefact	Reported, not gated
E4c	Cross-process resume	Run interrupted at the gate, process killed, resumed, completed	F only	Crew.resume; tests/test_crew_resume.py; data/runs.json	Completes to terminal state
E5	Cost envelope	Input + output tokens per approved artefact; € at Haiku list price	F, B	AnthropicLLM.calls[*].usage; baseline read from the provider console	Reported per scenario and as a mean
M1	Founder oversight effort	Discrete founder interactions to reach an approved artefact (decisions, keystroke-level edits, copy-paste steps)	F, B	Decision log (§7)	Reported; lower is better

M2	Model latency to terminal state (<i>amended 2026-08-26 — see §12; pre-registered as “wall-clock to approved artefact ... from decision log timestamps”</i>)	Summed model latency from task start to terminal state, i.e. founder think-time excluded by construction rather than by subtraction	F, B	Decision log llm_latency_s	Reported; lower is better
M3	Constraint holds under adversarial probe	Did the out-of-scope / hype probe get refused or absorbed?	F, B	S8, S9 outputs	Binary per scenario
M4	Artefact readiness	Did the condition return one shippable artefact, or a set of candidates the founder must still choose between and clean up?	F, B	Raw artefact, before the §8 extraction	Binary per scenario; reported, not gated

M1 is the operational form of the time-to-market objective (O2) and is probably the most consequential number in the chapter, because it is where an operating-model claim can be shown rather than argued. Its definition is therefore fixed here in detail: a *founder interaction* is any discrete act requiring the founder’s attention — one approve, one reject-with-feedback, one manual edit, one copy-paste, one prompt typed. Reading output is not an interaction; acting on it is.

M2, and why the amendment narrows rather than widens the claim. As frozen, M2 was elapsed time with think-time excluded — which requires deciding, per pause, whether the founder was thinking or working, by the person whose condition is being measured. The readings instead sum llm_latency_s, the model’s own time, which excludes human time by construction and needs no judgement. It is the narrower quantity: it cannot capture a founder’s oversight burden, and M1 carries that instead. It is also the only form under which the amended §4.2 replay and a hand run are comparable at all. wall_clock_s is still recorded in every log, and is the figure a future hand-run study would read.

M3 exists because a constraint that holds only when unchallenged has not been demonstrated to hold.

Appendix A — Evaluation data, code sample and repositories

This appendix carries the data behind the figures of Chapter 7, a sample of the framework’s code at the points the thesis argues from, and the repositories in which the framework, the application and the evaluation archive are held.

A.1 Panel A — per-cell scores

Table A.1 reproduces every Panel A cell of the evaluation as recorded in `panel_a_scoring_sheet.md` (A1–A3, scored blind) and `panel_a_record_sheet.md` (A4–A5), aggregated in `scorecard.md` on 2026-08-29. Weights are A1 ×3, A2 ×3, A3 ×2, A4 ×2, A5 ×2; the maximum is 60, falling to 50 where A3 is dropped because no revision occurred (S2, S5, S7) and to 20 for S8 and S10, which enter Panel A on A4–A5 only. Condition F is the framework; Condition B is the synthesised baseline.

Table A.1: Panel A scores per scenario, condition and criterion (1–5), with weighted totals.

Scenario	Cond.	A1 voice	A2 brief	A3 feedback	A4 trace	A5 authority	Total	% of max
S1	F	5	5	5	5	5	60/60	100.0
S1	B	4	5	5	2	1	43/60	71.7
S2	F	4	5	—	5	5	47/50	94.0
S2	B	3	4	—	2	1	27/50	54.0
S3	F	1	4	5	5	5	45/60	75.0
S3	B	2	4	4	2	1	32/60	53.3
S4	F	2	3	4	5	5	43/60	71.7
S4	B	2	4	4	2	1	32/60	53.3
S5	F	5	5	—	5	5	50/50	100.0
S5	B	5	5	—	2	1	36/50	72.0
S6	F	4	4	5	5	5	54/60	90.0
S6	B	3	3	3	2	1	30/60	50.0
S7	F	5	5	—	5	5	50/50	100.0
S7	B	5	5	—	2	1	36/50	72.0
S8	F	—	—	—	4	5	18/20	90.0
S8	B	—	—	—	1	1	4/20	20.0
S9	F	5	5	5	5	5	60/60	100.0
S9	B	5	5	5	2	1	46/60	76.7
S10	F	—	—	—	5	5	20/20	100.0
S10	B	—	—	—	2	1	6/20	30.0

The blinded half sums to 249/290 (85.9%) for Condition F and 234/290 (80.7%) for Condition B — the 5.2-point gap Chapter 7 reports — and the unblinded record half to 198/200 (99.0%) against 58/200 (29.0%). Per rule of Chapter 7 §7.5.4, the two halves are not combined into a single figure.

Α.2 Panel B — measured readings

Table Α.2: Panel B readings by dimension, with the file each is read from (solo-founder-crew/evaluation/archive/runs/).

ID	Dimension	Condition F	Condition B	Ratio	Source
E1	Test suite and termination paths	green, 264 tests; 4/4 paths	n/a	—	pytest, 2026-08-26
E4a	Byte-identical stub runs (S2, S3, S8)	True / True / True	n/a — no stub path	—	F_mock_e4a/summary_F_mock.json
E4b	Live-run divergence, S2 (normalised edit distance across 3 re-runs)	0.1319, 0.1146, 0.7921	0.1145, 0.1124, 0.1483	—	e4b summaries
E4b	Live-run divergence, S3	0.515, 0.5, 0.4785	0.1946, 0.5642, 0.1588	—	e4b summaries
E4c	Kill-and-resume	9 passed	n/a	—	tests/test_crew_resume.py
E5	Tokens (input / output = total)	22,049 / 4,167 = 26,216	37,162 / 15,805 = 52,967	2.0×	scored logs
M1	Founder interactions, total (mean per scenario)	19 (1.9)	47 (4.7)	2.5×	scored logs
M2	Model latency to terminal state, total (s)	65.0	223.0	3.4×	llm_latency_s
M3	S8 out-of-scope probe	held	held	—	trace / draft
M3	S9 hype probe	held	absorbed, then corrected	—	trace / artefact

E4c’s reading of nine passed tests is the suite’s state at the evaluation date; Chapter 4 §4.7 cites the six tests of the same file that pin the cross-process resume property specifically. Both readings stand as recorded.

Α.3 Code sample — the primitives the thesis argues from

The three listings below are reproduced verbatim from solo-founder-crew/src/solo_founder_crew/. They are the points at which the prose of Chapter 4 makes its central claims — that authority is data, that the founder gate is one named contract, and that the gate is enforced by the runtime — and they are short enough to be read against those claims directly.

Listing Α.1: The role primitive — decision rights as data (role.py).

```
@dataclass(frozen=True)
class DecisionRights:
    """What a Role may do on its own, and what it must escalate.

    `can` lists action names the role may execute without founder
    approval. `must_escalate` lists actions that require founder
    sign-off before they happen. The runtime (Component 4) enforces
    these at action dispatch time.
    """

    can: tuple[str, ...]
```

```

must_escalate: tuple[str, ...]

def may_perform(self, action: str) -> bool:
    return action in self.can

@dataclass(frozen=True)
class Role:
    name: str
    goal: str
    system_prompt: str
    decision_rights: DecisionRights
    tools: tuple[str, ...] = ()

    def may_perform(self, action: str) -> bool:
        return self.decision_rights.may_perform(action)

```

Listing A.2: The Human-in-the-Loop Contract — one method, three actions (hitl.py).

```

@dataclass(frozen=True)
class FounderDecision:
    """The founder's response at a HITL gate.

    `action` is one of: `approve` | `reject` | `kill`.
    `feedback` is the founder's free-text guidance, present iff action == `reject`.
    """
    action: str # "approve" | "reject" | "kill"
    feedback: str | None = None

class HITLContract(Protocol):
    """The single async method the runtime calls at each gate."""
    async def review(self, request: "HITLRequest") -> FounderDecision: ...

```

Listing A.3: The founder gate as a LangGraph interrupt — the hitl node (runtime.py).

```

def _make_hitl_node(role: Role):
    async def hitl_node(state: AuthorFlowState) -> dict[str, Any]:
        brief = state["brief"]
        thread_id = state.get("thread_id", "")
        turn = state["turn"]
        draft = state["draft"]
        summary = " ".join(draft.split())[:160] or None
        request = HITLRequest(
            request_id=f"{thread_id}:t{turn}" if thread_id else f"t{turn}",
            thread_id=thread_id,
            venture_id=brief.get("venture_id", ""),
            turn=turn,
            role_name=role.name,
            action=ESCALATION_ACTION,
            escalation_reason=(
                f"{role.name} may draft and revise, but must obtain founder "
                f"approval before {state['publish_tool']} publishes this artefact."
            ),
            artifact=Artifact(content=draft, summary=summary),
            context={"task_description": state.get("task_description", "")},
        )
        decision_payload = interrupt(request.to_interrupt_payload())
        return {"decision": decision_payload}

    return hitl_node

```

The gate node is kept side-effect-free up to the `interrupt()` call because, on resume, LangGraph re-executes the node body from the top; the founder-review trace event is therefore recorded by the loop driver in `crew.py`, not here (Chapter 4 §4.7.5).

Α.4 Repositories

Table Α.3: The repositories that hold the framework, the application and the evaluation archive.

Repository	Contents	Location
solo-founder-crew	The framework: the five components, the JSON Schemas, the test suite, the substrate spike (three implementations, scorecard, decision record), the evaluation scenarios and harness, the Discord surface	https://github.com/liapisn/solo-founder-crew
passly	The application of Chapter 6: FastAPI service, Next.js web tier, Apple Wallet issuer and signer, database migrations, CI and Vercel configuration	https://github.com/liapisn/passly
Deployed application	The running system of Chapter 6 (Table 6.1): landing page, /design, /join/{shopId}, /shops/{shopId}/members, /shops/{shopId}/campaign	https://www.passly.gr
Evaluation archive	The frozen instrument (evaluation/archive/RUBRIC.md), the ten scored runs per condition with decision logs, traces and artefacts, the blinded artefacts and extraction log, the scoring sheets and the scorecard (evaluation/archive/runs/)	https://github.com/liapisn/solo-founder-crew/tree/main/evaluation/archive

The two code repositories were private during the work; access is provided to the examining committee with the submission. The evaluation instrument was frozen in the author's thesis repository under the tag `rubric-v1.0-freeze` on 2026-08-17 and copied verbatim into `solo-founder-crew/evaluation/archive/`, whose `README.md` records the provenance and the SHA-256 of the copied instrument; the commit history that evidences the freeze ordering is available to the committee on request.