Formatted: Greek

Πανεπιστήμιο Αιγαίου
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Τίτλος: Σύστημα Παγκόσμιου Ιστού για την Επιλογή Serverless Πλατφορμών και τη Διάταξη Serverless Συναρτήσεων

Επιβλέπων Καθηγητής: Κρητικός Κυριάκος

Τριμελής Επιτροπή: Κοκολάκης Σπυρίδων, Σκούτας Δημήτριος

Φοιτητής: Καμπερόπουλος Αναστάσιος

Ημερομηνία Παράδοσης: Αύγουστος 2025

Formatted: Greek

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή κ. Κρητικό για την καθοδήγηση και υποστήριξή του καθ' όλη τη διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας, καθώς και στην οικογένειά μου για την αμέριστη συμπαράσταση και ενθάρρυνση σε όλη τη διάρκεια των σπουδών μου.

Formatted: Greek

Commented [KK1]: Κάθε αρχικό μέρος και κεφάλαιο στην αναφορά θα πρεπει να αρχιζει σε νέα σελίδα

Formatted: Normal

Περίληψη

Η παρούσα διπλωματική εργασία περιγράφει τον σχεδιασμό και την υλοποίηση ενός σύγχρονου συστήματος Παγκόσμιου Ιστού, το οποίο επιτρέπει την αξιολόγηση και επιλογή serverless πλατφορμών, όπως AWS Lambda, Google Cloud Functions και Azure Functions, βάσει πολλαπλών τεχνικών και ποιοτικών κριτηρίων. Το σύστημα υποστηρίζει μηχανισμούς φιλτραρίσματος και ταξινόμησης με δυνατότητα καθορισμού σταθμισμένων κριτηρίων (weighted criteria), υποστήριξη πολλαπλών χαρακτηριστικών/κριτηρίων ανά πλατφόρμα καθώς και αυτόματη διάταξη serverless συναρτήσεων μέσω διασύνδεσης με σχετικά APIs των serverless πλατφορμών που επιλέγονται από τον χρήστη.

Η ανάπτυξη βασίστηκε σε τεχνολογίες React για το frontend και Spring Boot και PostgreSQL για το backend, το οποίο υλοποιήθηκε με τη μορφή ενός RESTful API. Επιπλέον, ενσωματώθηκε η δυνατότητα σύνδεσης με το Serverless Framework, διευκολύνοντας τη διάταξη συναρτήσεων σε πολλαπλά περιβάλλοντα/πλατφόρμες. Η εργασία συμβάλλει στην οργανωμένη και αντικειμενική σύγκριση serverless πλατφορμών, προσφέροντας μια ολοκληρωμένη διαδικτυακή λύση στους χρήστες για την αξιολόγηση και επιλογή της καταλληλότερης πλατφόρμας (προς διάταξη συναρτήσεων) βάσει των απαιτήσεων και προτιμήσεών τους. Επίσης, επιτρέπει την διάταξη των συναρτήσεων σε πολλαπλά περιβάλλοντα, συνεπώς συμβάλλει στην μη χρήση τρόπων διάταξης που είναι ειδικοί ως προς μια πλατφορμα ενώ παρέχεται στον χρηστη η ευελιξία να διατασσει την ίδια συνάρτηση σε διαφορετικά περιβάλλοντα, εφοσον αυτό χρειαστει, αποφεύγοντας το προβλημα του vendor lock-in.

Λέξεις-Κλειδιά: Υπολογισμός χωρίς Εξυπηρετητές (Serverless computing), Σύστημα Παγκόσμιου Ιστού (Web System), Πλατφόρμες Υπολογιστικού Νέφους (Cloud Platforms), AWS Lambda, Azure Functions, Google Cloud Functions, Σταθμισμένα Κριτήρια, Αυτοματοποίηση Διάταξης, Spring Boot, React, PostgreSQL, Serverless Framework.

Formatted: Greek

Commented [KK2]: Θα μπορούσατε να πειτε επίσης πως η εργασία επιτρέπει την διάταξη των συναρτήσεων σε πολλαπλά περιβάλλοντα, συνεπώς συμβάλλει στην μη χρήση τρόπων διάταξης που είναι ειδικοί ως προς μια πλατφορμα ενώ παρέχεται στον χρηστη η ευελιξία να διατασσει την ίδια συνάρτηση σε διαφορετικά περιβάλλοντα, εφοσον αυτό χρειαστει, αποφεύγοντας το προβλημα του vendor lock-in.

Abstract

This thesis presents the design and implementation of a modern web-based system that enables the evaluation and selection of serverless platforms, such as AWS Lambda, Google Cloud Functions, and Azure Functions, based on multiple technical and qualitative criteria. The system supports mechanisms for platform filtering and ranking using weighted criteria, allows multi-attribute platform descriptions, and provides automated function deployment through integration with the corresponding serverless platform APIs that the user selects.

The development was carried out using React for the frontend and Spring Boot with PostgreSQL for the backend, which was implemented in the form of a RESTful API. Furthermore, the system integrates with the Serverless Framework, simplifying multi-platform function deployment.

The proposed solution contributes to the systematic and objective comparison of serverless platforms, offering an integrated web solution that assists users in selecting the most suitable platform (for function deployment) according to their requirements and preferences. It also allows functions to be deployed in multiple environments, thus contributing to the avoidance of platform-specific deployment methods while providing the user with the flexibility to deploy the same function in different environments, if necessary, thus avoiding the problem of vendor lock-in.

Keywords: Serverless computing, Web System, Cloud Platforms, AWS Lambda, Azure Functions, Google Cloud Functions, Weighted Criteria, Deployment Automation, Spring Boot, React, PostgreSQL, Serverless Framework.

Formatted: Greek

Commented [KK3]: Θα πρέπει να ξεκινήσει σε νέα σελίδα

Commented [KK4]: Ας προσθέσετε κι αυτό που ζητησα για την ελληνική έκδοση εδώ.

Πίνακας Περιεχομένων

1. Εισαγωγή 1612

2. Θεωρητική και Τεχνολογική Ανασκόπηση 1814

2.1 Υπολογιστικό Νέφος (Cloud Computing) 1814

2.1.1 Ορισμός 1814

2.1.2 Βασικά Χαρακτηριστικά & Πλεονεκτήματα 1915

2.1.3 Μοντέλα Παράδοσης / Είδη Υπηρεσιών 2016

2.1.4 Μοντέλα Ανάπτυξης 2420

2.2 Υπολογιστική χωρίς Διακομιστές (Serverless Computing) 2824

2.2.1 Ορισμός και Βασικές Αρχές 2824

2.2.2 Κατάλληλες Χρήσεις και Workloads 2925

2.2.3 Πλεονεκτήματα του Serverless Computing 2925

2.2.4 Τυπικές Εφαρμογές Serverless 3026

2.2.5 Παράδειγμα Μοντέλου Χρέωσης — AWS Lambda 3026

2.3 Ιστορική Εξέλιξη και Ανάγκη Ανάπτυξης Serverless Πλαισίων 3127

2.4 Αντιμετώπιση Κλειδώματος στον Πάροχο 3228

2.5 Αρχιτεκτονική Serverless 3329

2.6 Διαχείριση Πόρων στο Serverless Περιβάλλον 4036

2.6.1 Provisioning Πόρων 4036

2.6.2 Χρονοπρογραμματισμός (Scheduling) 4238

2.6.3 Κλιμάκωση Πόρων (Scaling) 4945

2.6.4 Παρακολούθηση Πόρων (Monitoring) 5349

2.7 Σύγκριση Πλατφορμών και Πλαισίων Serverless Υπολογιστικής 5652

2.7.1 Κριτήρια Σύγκρισης 5652

2.7.2 Εμπορικές Πλατφόρμες Serverless 5854

2.7.3 Πλαίσια Ανοικτού Κώδικα (Open Source Frameworks) 6157

2.7.4 Καθολικά Συμπεράσματα 6359

2.8 Πλεονεκτήματα και Προκλήσεις της Serverless Υπολογιστικής 6561

2.9 Συμπεράσματα Κεφαλαίου 6964

3. Σχετικές Εργασίες 7267

3.1 Επιλογή και Αξιολόγηση Serverless Πλατφορμών 7267

3.2 Διάταξη Serverless Συναρτήσεων 7368

Formatted: Greek

Commented [KK5]: Οι ευχαριστίες, η περίληψη, ο πίνακας περιεχομένων, οι λίστες πινάκων και εικόνων, ο πίνακας ακρωνυμίων δεν θα πρέπει να φαινονται εδώ.

Επίσης, η εμφάνιση των κεφαλαιων θα πρέπει να γινεται σωστά στον τρέχοντα πίνακα (διαφοροποίηση παράθεσης κεφαλαιων σε διαφορετικό επίπεδο). Μάλλον αυτό είναι ζητημα με τα χρησιμοποιούμενα headings που θα πρέπει να διορθωθει όταν παρέχονται οι τιτλοι των κεφαλαιων εντός του κυρίως κειμένου της αναφοράς.

Commented [KK6R5]: Προσέξτε διότι υπάρχει ζήτημα με την εμφάνιση των τίτλων κεφαλαιων. Θα πρέπει οι τιτλοι των κεφαλαίων/ενοτήτων του ίδιου επιπέδου να είναι κάθετα ευθυγραμμισμένοι. Επίσης, θα πρεπει να υπάρχει διαφοροποίηση εμφανισης των τιτλων κεφαλαίων/ενοτήτων διαφορετικών επιπέδων (πχ. κατά ένα tab).

Η Εισαγωγή στο Κεφαλαιο 2 θα πρέπει να είναι μια αριθμημένη ενότητα (2.1 Εισαγωγή).

Τέλος, υπάρχει κεφάλαιο 4 & 4ο κεφάλαιο. Ουσιαστικά θα πρέπει να υπάρχει ενιαια κεφάλαιο που θα πρέπει να ονοματιστει: 4. Ανάπτυξη Συστήματος.

Commented [KK7R5]: Υπάρχει ακόμη το ζητημα με τα tabs για διαφορετικά επίπεδα κεφαλίδων - δειτε 2.5 & όλες τις υπο-ενότητές του (2.5.1-2.5.4). Παρομοια για 2.6 όπου μάλιστα το προβλημα γινεται ακόμη πιο σοβαρο ενώ λειπει και μια υπο-ενότητα ...
Επίσης η 4.2.3 υπαρχει δύο φορές ως αριθμός ενότητας ...

Τέλος, φαίνεται να λείπει το Σεναριο 3 στην 5.2 ενώ στην 5.2.4 φαίνεται πως λείπει η λεξη Σεναριο από τον τιτλο ενώ υπαρχει και εξτρα κενό στην τελευταία τελεία της αρίθμησης

4. Ανάπτυξη Συστήματος 7570
4.1 Ανάλυση Απαιτήσεων 7570
4.1.1 Σκοπός Συστήματος 7570
4.1.2 Actors & Εξωτερικά Συστήματα 7671
4.1.3 Λειτουργικές Απαιτήσεις 7671
4.1.4 Μη Λειτουργικές Απαιτήσεις 8176
4.1.5 Περιορισμοί 8680
4.2. Σχεδίαση 8781
4.2.1 Γενική Αρχιτεκτονική του Συστήματος 8882
4.2.2 Λογική Αρχιτεκτονική (Logical Components) 10297
4.2.3 Διάγραμμα Κλάσεων (Class Diagram – Backend) 116111
4.2.4 Διαγράμματα Ακολουθίας 117112
4.2.5 Διαγράμματα Δραστηριοτήτων (Activity Diagram) 127122
4.3 Υλοποίηση Συστήματος (System Implementation) 130125
4.3.1 Cross-Origin Resource Sharing (CORS) Configuration 130125
4.3.2 Τεκμηρίωση REST API (OpenApiConfig) 131126
4.3.3 Ενότητα Ασφάλειας (Security Module) 131126
4.3.4 Ενότητα Διαχείρισης Πλατφορμών (Platform Management Module) 134129
4.3.5 Ενότητα Διάταξης (Deployment Module) 140135
4.3.6 Frontend Υλοποίηση (ReactJS) 141136
4.3.7 Βάση Δεδομένων (PostgreSQL) 144139
4.3.8 Περίληψη Υλοποίησης 145140
5. Εγκατάσταση & Επίδειξη Συστήματος 146141
5.1 Εγκατάσταση 146141
5.1.1 Προαπαιτούμενα 146141
5.1.2 Δημιουργία Βάσης Δεδομένων 147141
5.1.3 Εγκατάσταση Backend 147142
5.1.4 Εγκατάσταση Frontend 148143
5.2 Επίδειξη & Σενάρια Χρήσης 148143
5.2.1 Σενάριο 1 — Εγγραφή & Σύνδεση Χρήστη 149144
5.2.2 Σενάριο 2 — Δημιουργία & Ενημέρωση Πλατφόρμας (Διαχειριστής) 154149
5.2.3 Σενάριο 3 — Αναζήτηση Πλατφορμών (User) 160155

Formatted: Greek

5.2.4 Σενάριο 4 — Κατάταξη Πλατφορμών (Ranking) 164159
5.2.5 Σενάριο 5 — Διάταξη Συνάρτησης (Deployment) 166161
5.2.6 Σενάριο 6 — Διαχείριση Χρηστών (Admin – Απενεργοποίηση χρήστη) 169164
5.2.7 Σενάριο 7 — Διαχείριση Χρηστών (Admin: Ενημέρωση στοιχείων/Διαγραφή Χρήστη) 171166
6. Συμπεράσματα & Μελλοντική Εργασία 177172
6.1 Συμπεράσματα 177172
6.2 Μελλοντική Εργασία 177172
Βιβλιογραφία 183178
1. Εισαγωγή 9
2. Θεωρητική και Τεχνολογική Ανασκόπηση 11
2.1 Υπολογιστικό Νέφος (Cloud Computing) 11
2.1.1 Ορισμός 11
2.1.2 Βασικά Χαρακτηριστικά & Πλεονεκτήματα 12
2.1.3 Μοντέλα Παράδοσης / Είδη Υπηρεσιών 13
2.1.4 Μοντέλα Ανάπτυξης 17
2.2 Υπολογιστική χωρίς Διακομιστές (Serverless Computing) 21
2.2.1 Ορισμός και Βασικές Αρχές 21
2.2.2 Κατάλληλες Χρήσεις και Workloads 22
2.2.3 Πλεονεκτήματα του Serverless Computing 22
2.2.4 Τυπικές Εφαρμογές Serverless 23
2.2.5 Παράδειγμα Μοντέλου Χρέωσης — AWS Lambda 23
2.3 Ιστορική Εξέλιξη και Ανάγκη Ανάπτυξης Serverless Πλαισίων 24
2.4 Αντιμετώπιση Κλειδώματος στον Πάροχο 25
2.5 Αρχιτεκτονική Serverless 26
2.6 Διαχείριση Πόρων στο Serverless Περιβάλλον 33
2.6.1 Provisioning Πόρων 33
2.6.2 Χρονοπρογραμματισμός (Scheduling) 35
2.6.3 Κλιμάκωση Πόρων (Scaling) 42
2.6.4 Παρακολούθηση Πόρων (Monitoring) 46
2.7 Σύγκριση Πλατφορμών και Πλαισίων Serverless Υπολογιστικής 49
2.7.1 Κριτήρια Σύγκρισης 49

Formatted: Greek

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted ...

Formatted ...

Formatted ...

Formatted ...

2.7.2 Εμπορικές Πλατφόρμες Serverless 51
2.7.3 Πλαίσια Ανοικτού Κώδικα (Open Source Frameworks) 54
2.7.4 Καθολικά Συμπεράσματα 55
2.8 Πλεονεκτήματα και Προκλήσεις της Serverless Υπολογιστικής 56
2.9 Συμπεράσματα Κεφαλαίου 60
3. Σχετικές Εργασίες 63
3.1 Επιλογή και Αξιολόγηση Serverless Πλατφορμών 63
4. Ανάπτυξη Συστήματος 66
4.1 Ανάλυση Απαιτήσεων 66
4.1.1 Σκοπός Συστήματος 66
4.1.2 Actors & Εξωτερικά Συστήματα 67
4.1.3 Λειτουργικές Απαιτήσεις 67
4.1.4 Μη Λειτουργικές Απαιτήσεις 72
4.1.5 Περιορισμοί 76
4.2. Σχεδίαση 77
4.2.1 Πληροφοριακό Υποσύστημα Πλατφορμών 80
4.2.2 Μηχανισμός Φιλτραρίσματος & Ταξινόμησης 81
4.2.3 Deployment Engine 88
4.2.4 Λογική Αρχιτεκτονική (Logical Components) 93
4.2.5 Διάγραμμα Λογικών Συστατικών (Block Diagram) 105
4.2.6 Διαγράμματα Ακολουθίας 107
4.2.7 Διάγραμμα Κλάσεων (Class Diagram – Backend) ..... **Error! Bookmark not defined.**
4.2.8 Διαγράμματα Δραστηριοτήτων (Activity Diagram) ..... **Error! Bookmark not defined.**
4.3 Υλοποίηση Συστήματος (System Implementation) 118
4.3.1 Cross-Origin Resource Sharing (CORS) Configuration 119
4.3.2 Ενότητα Ασφάλειας (Security Module) 120
4.3.3 Τεκμηρίωση REST API (OpenApiConfig) ..... **Error! Bookmark not defined.**
4.3.4 Ενότητα Διαχείρισης Πλατφορμών (Platform Management Module) 123
4.3.5 Ενότητα Διάταξης (Deployment Module) 129
4.3.6 Frontend Υλοποίηση (ReactJS) 130
4.3.7 Βάση Δεδομένων (PostgreSQL) 133
4.3.8 Περίληψη Υλοποίησης 134

Formatted: Greek

Formatted: Default Paragraph Font, Font: Check spelling and grammar

Formatted: Default Paragraph Font, Font: Check spelling and grammar

Formatted: Default Paragraph Font, Font: Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

5. Εγκατάσταση & Επίδειξη Συστήματος 135
5.1 Εγκατάσταση 135
5.1.1 Προαπαιτούμενα 135
5.1.2 Δημιουργία Βάσης Δεδομένων 135
5.1.3 Εγκατάσταση Backend 136
5.1.4 Εγκατάσταση Frontend 137
5.2 Επίδειξη & Σενάρια Χρήσης 137
5.2.1 Σενάριο 1 — Εγγραφή & Σύνδεση Χρήστη 138
5.2.2 Σενάριο 2 — Δημιουργία & Ενημέρωση Πλατφόρμας (Διαχειριστής) 143
5.2.3 Σενάριο 3 — Αναζήτηση Πλατφορμών (User) 149
5.2.4 Σενάριο 4 — Διάταξη Συνάρτησης (Deployment) 155
5.2.5 Σενάριο 5 — Διαχείριση Χρηστών (Admin – Απενεργοποίηση χρήστη) 158
5.2.5 Σενάριο 6 — Διαχείριση Χρηστών (Admin: Ενημέρωση στοιχείων/Διαγραφή Χρήστη) 160
6. Συμπεράσματα & Μελλοντική Εργασία 165
6.1 Συμπεράσματα 165
6.2 Μελλοντική Εργασία 166

Formatted: Greek

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: +Body (Aptos), English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: +Body (Aptos), English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, Font: +Body (Aptos), English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Formatted: Default Paragraph Font, English (United States), Check spelling and grammar

Commented [KK8]: Λειπουν λίστες/ευρετήρια εικόνων, πινάκων & ακρωνυμίων σε ξεχωριστές σελίδες. Πριν ξεκινήσει το εισαγωγικό κεφάλαιο της αναφοράς.

Λίστα Εικόνων

Εικόνα 1: Αρχική σελίδα(Landing Page) 149
Εικόνα 2: Φόρμα Εγγραφής (Sign Up Form) 150
Εικόνα 3: Φόρμα Σύνδεσης (Login Form) 151
Εικόνα 4: Κλείδωμα λογαριασμού μετά από 3 αποτυχημένες προσπάθειες σύνδεσης 152
Εικόνα 5: Επιλογή Admin Dashboard 152
Εικόνα 6: Εμφάνιση νέου χρήστη απενεργοποιημένου στη λίστα χρηστών (Admin View) . 153
Εικόνα 7: Εμφάνιση νέου, πλέον ενεργοποιημένου χρήστη στη λίστα χρηστών (Admin View) 153
Εικόνα 8: Αρχική Σελίδα (Landing Page) μετά από επιτυχή αυθεντικοποίηση 154
Εικόνα 9: Λίστα Πλατφορμών(Admin View) 156
Εικόνα 10: Φόρμα Νέας Πλατφόρμας 156
Εικόνα 11: Λίστα Πλατφορμών(Admin View) 157
Εικόνα 12: Φόρμα επεξεργασίας πλατφόρμας (Edit Platform Form). 158
Εικόνα 13: Φόρμα επεξεργασίας πλατφόρμας, μετά την αλλαγή (Edit Platform Form). 158
Εικόνα 14: Ενημερωμένη λίστα πλατφορμών με τα νέα στοιχεία της τροποποιημένης πλατφόρμας. 159
Εικόνα 15: Σελίδα Αναζήτησης Πλατφορμών (Search Platforms Page) 160
Εικόνα 16: Basic Search - Αναζήτηση με φίλτρα 161
Εικόνα 17: Basic Search - Ταξινόμηση με φίλτρα (Name Desc) 161
Εικόνα 18: Full-text Search — Αναζήτηση με keyword(fuzzy matching) 162
Εικόνα 19: Autocomplete - Αναζήτηση με προτάσεις 163
Εικόνα 20: Autocomplete - Αναζήτηση με προτάσεις (Αποτέλεσμα επιλογής suggestion) . 164
Εικόνα 21: Σελίδα Ranking (φόρμα κριτηρίων + λίστα αποτελεσμάτων) 165
Εικόνα 22: Αποτελέσματα κατάταξης (All Platforms Ranked + scores) 166
Εικόνα 23: Φόρμα Διατάξης Συνάρτησης 167
Εικόνα 24: Αποτέλεσμα Επιτυχούς Διάταξης (Endpoint View) 168
Εικόνα 25: Περιεχόμενο διάταξης 168
Εικόνα 26: Επιλογή Admin Dashboard 170
Εικόνα 27: Πίνακας Διαχείρισης Χρηστών (Admin View) 170
Εικόνα 28: Απενεργοποίηση Χρήστη 171
Εικόνα 29: User Management — Λίστα χρηστών (Admin View) 172
Εικόνα 30: Φόρμα Επεξεργασίας Χρήστη (Edit User) 173
Εικόνα 31: Λίστα χρηστών μετά την ενημέρωση (Admin View) 174
Εικόνα 32: Επιβεβαίωση Διαγραφής Χρήστη (Delete Confirmation) 175
Εικόνα 33: Λίστα χρηστών μετά τη διαγραφή (Admin View) 175
~~Εικόνα 1: Αρχική σελίδα(Landing Page) ..... 139~~
~~Εικόνα 2: Φόρμα Εγγραφής (Sign Up Form) ..... 140~~
~~Εικόνα 3: Φόρμα Σύνδεσης (Login Form) ..... 141~~
~~Εικόνα 4: Κλείδωμα λογαριασμού μετά από 3 αποτυχημένες προσπάθειες σύνδεσης ..... 142~~
~~Εικόνα 5: Επιλογή Admin Dashboard ..... 142~~

Formatted: Greek

Formatted: Font: +Body (Aptos), 14 pt, Bold, Font color: Accent 1

Formatted: Font: +Body (Aptos), 14 pt, Bold

Εικόνα 6: Εμφάνιση νέου χρήστη απενεργοποιημένου στη λίστα χρηστών (Admin View) . 143
Εικόνα 7: Εμφάνιση νέου, πλέον ενεργοποιημένου χρήστη στη λίστα χρηστών (Admin View) 143
Εικόνα 8: Αρχική Σελίδα (Landing Page) μετά από επιτυχή αυθεντικοποίηση 144
Εικόνα 9:Λίστα Πλατφορμών(Admin View)..... 146
Εικόνα 10: Φόρμα Νέας Πλατφόρμας 146
Εικόνα 11:Λίστα Πλατφορμών(Admin View)..... 147
Εικόνα 12:Φόρμα επεξεργασίας πλατφόρμας (Edit Platform Form). 148
Εικόνα 13:Φόρμα επεξεργασίας πλατφόρμας, μετά την αλλαγή (Edit Platform Form). 148
Εικόνα 14:Ενημερωμένη λίστα πλατφορμών με τα νέα στοιχεία της τροποποιημένης πλατφόρμας. 149
Εικόνα 15: Σελίδα Αναζήτησης Πλατφορμών (Search Platforms Page) 150
Εικόνα 16: Basic Search - Αναζήτηση με φίλτρα 151
Εικόνα 17: Basic Search - Ταξινόμηση με φίλτρα (Name Desc) 151
Εικόνα 18: Full-text Search — Αναζήτηση με keyword(fuzzy matching) 152
Εικόνα 19: Autocomplete - Αναζήτηση με προτάσεις..... 153
Εικόνα 20: Autocomplete - Αναζήτηση με προτάσεις (Αποτέλεσμα επιλογής suggestion) . 154
Εικόνα 21: Σελίδα Ranking (φόρμα κριτηρίων + λίστα αποτελεσμάτων)..... 155
Εικόνα 22: Αποτελέσματα κατάταξης (All Platforms Ranked + scores)..... 156
Εικόνα 23: Φόρμα Διατάξης Συνάρτησης..... 157
Εικόνα 24: Αποτέλεσμα Επιτυχούς Διάταξης (Endpoint View)..... 158
Εικόνα 25: Περιεχόμενο διάταξης..... 158
Εικόνα 26: Επιλογή Admin Dashboard 160
Εικόνα 27: Πίνακας Διαχείρισης Χρηστών (Admin View)..... 160
Εικόνα 28:Απενεργοποίηση Χρήστη 161
Εικόνα 29: User Management — Λίστα χρηστών (Admin View) 162
Εικόνα 30: Φόρμα Επεξεργασίας Χρήστη (Edit User) 163
Εικόνα 31: Λίστα χρηστών μετά την ενημέρωση (Admin View)..... 163
Εικόνα 32: Επιβεβαίωση Διαγραφής Χρήστη (Delete Confirmation) 164
Εικόνα 33: Λίστα χρηστών μετά τη διαγραφή (Admin View) 165
Εικόνα 1: Αρχική σελίδα(Landing Page)..... 142
Εικόνα 2:Φόρμα Εγγραφής (Sign Up Form)..... 143
Εικόνα 3: Φόρμα Σύνδεσης (Login Form)..... 144
Εικόνα 4: Κλείδωμα λογαριασμού μετά από 3 αποτυχημένες προσπάθειες σύνδεσης..... 145
Εικόνα 5: Επιλογή Admin Dashboard 145
Εικόνα 6: Εμφάνιση νέου χρήστη απενεργοποιημένου στη λίστα χρηστών (Admin View) . 146
Εικόνα 7: Εμφάνιση νέου, πλέον ενεργοποιημένου χρήστη στη λίστα χρηστών (Admin View) 146
Εικόνα 8: Αρχική Σελίδα (Landing Page) μετά από επιτυχή αυθεντικοποίηση 147
Εικόνα 9:Λίστα Πλατφορμών(Admin View)..... 149
Εικόνα 10: Φόρμα Νέας Πλατφόρμας 149
Εικόνα 11:Λίστα Πλατφορμών(Admin View)..... 150

Formatted: Greek

~~Εικόνα 12:Φόρμα επεξεργασίας πλατφόρμας (Edit Platform Form). ..... 151~~
~~Εικόνα 13:Φόρμα επεξεργασίας πλατφόρμας, μετά την αλλαγή (Edit Platform Form). ..... 151~~
~~Εικόνα 14:Ενημερωμένη λίστα πλατφορμών με τα νέα στοιχεία της τροποποιημένης πλατφόρμας. ..... 152~~
~~Εικόνα 15: Σελίδα Αναζήτησης Πλατφορμών (Search Platforms Page) ..... 153~~
~~Εικόνα 16: Basic Search - Αναζήτηση με φίλτρα ..... 154~~
~~Εικόνα 17: Basic Search - Ταξινόμηση με φίλτρα (Name Desc) ..... 154~~
~~Εικόνα 18: Full-text Search — Αναζήτηση με keyword(fuzzy matching) ..... 155~~
~~Εικόνα 19: Autocomplete - Αναζήτηση με προτάσεις ..... 156~~
~~Εικόνα 20: Autocomplete - Αναζήτηση με προτάσεις (Αποτέλεσμα επιλογής suggestion) . 157~~
~~Εικόνα 21: Σελίδα Ranking (φόρμα κριτηρίων + λίστα αποτελεσμάτων) ..... 158~~
~~Εικόνα 22: Αποτελέσματα κατάταξης (All Platforms Ranked + scores) ..... 159~~
~~Εικόνα 23: Φόρμα Διατάξης Συνάρτησης ..... 160~~
~~Εικόνα 24: Αποτέλεσμα Επιτυχούς Διάταξης (Endpoint View) ..... 161~~
~~Εικόνα 25: Περιεχόμενο διάταξης ..... 161~~
~~Εικόνα 26: Επιλογή Admin Dashboard ..... 163~~
~~Εικόνα 27: Πίνακας Διαχείρισης Χρηστών (Admin View) ..... 163~~
~~Εικόνα 28:Απενεργοποίηση Χρήστη ..... 164~~
~~Εικόνα 29: User Management — Λίστα χρηστών (Admin View) ..... 165~~
~~Εικόνα 30: Φόρμα Επεξεργασίας Χρήστη (Edit User) ..... 166~~
~~Εικόνα 31: Λίστα χρηστών μετά την ενημέρωση (Admin View) ..... 166~~
~~Εικόνα 32: Επιβεβαίωση Διαγραφής Χρήστη (Delete Confirmation) ..... 167~~
~~Εικόνα 33: Λίστα χρηστών μετά τη διαγραφή (Admin View) ..... 168~~

Λίστα Διαγραμμάτων

Διάγραμμα 1: Διάγραμμα Λογικών Συστατικών (Architectural Block Diagram) 116
Διάγραμμα 2: Διάγραμμα Κλάσεων (Class Diagram – Backend) 117
Διάγραμμα 3: Διάγραμμα ακολουθίας Ροής διάταξης (Deployment Flow) 120
Διάγραμμα 4: Διάγραμμα ακολουθίας Ροής Αυθεντικοποίηση (Authentication/Login flow) 121
Διάγραμμα 5: Διάγραμμα ακολουθίας Αναζήτησης και Ταξινόμησης 125
Διάγραμμα 6: Διάγραμμα ακολουθίας κατατάξης 127
Διάγραμμα 7: Διαγράμμα Δραστηριοτήτων (Activity diagram User Registration) 128
Διάγραμμα 8: Διάγραμμα δραστηριοτήτων ενεργοποίησης/απενεργοποίησης χρήστη από διαχειριστή. 129
~~Διάγραμμα 1: Διάγραμμα Λογικών Συστατικών (Architectural Block Diagram) ..... 108~~

Formatted: Greek

Formatted: English (United States)

Formatted: Font: +Body (Aptos), Bold, Font color: Accent 1

Formatted: Font: +Body (Aptos), Bold

Διάγραμμα 2: Διάγραμμα Κλάσεων (Class Diagram – Backend) 109
Διάγραμμα 3: Διάγραμμα ακολουθίας Ροής διάταξης (Deployment Flow)..... 111
Διάγραμμα 4: Διάγραμμα ακολουθίας Ροής Αυθεντικοποίηση (Authentication/Login flow)112
Διάγραμμα 5: Διάγραμμα ακολουθίας Αναζήτησης και Ταξινόμησης 114
Διάγραμμα 6: Διάγραμμα ακολουθίας κατατάξης..... 116
Διάγραμμα 7: Διαγράμμα Δραστηριοτήτων (Activity diagram User Registration) 117
Διάγραμμα 8: Διάγραμμα δραστηριοτήτων ενεργοποίησης/απενεργοποίησης χρήστη από διαχειριστή. 118
Διάγραμμα 1: Διάγραμμα Λογικών Συστατικών (Architectural Block Diagram) 109
Διάγραμμα 2: Διάγραμμα Κλάσεων (Class Diagram – Backend) 110
Διάγραμμα 3:Διάγραμμα ακολουθίας Ροής διάταξης (Deployment Flow) 113
Διάγραμμα 4:Διάγραμμα ακολουθίας Ροής Αυθεντικοποίηση (Authentication/Login flow) 114
Διάγραμμα 5: Διάγραμμα ακολουθίας Αναζήτησης και Ταξινόμησης 118
Διάγραμμα 6: Διάγραμμα ακολουθίας κατατάξης..... 120
Διάγραμμα 7 : Διαγράμμα Δραστηριοτήτων (Activity diagram User Registration) 121
Διάγραμμα 8: Διάγραμμα δραστηριοτήτων ενεργοποίησης/απενεργοποίησης χρήστη από διαχειριστή. 122

Formatted: Greek

Λίστα Πινάκων

Πίνακας 1: Σύγκριση AWS Lambda, Azure Functions, Google Cloud Functions 58
Πίνακας 2: Σύγκριση Serverless Frameworks 62
Πίνακας 3: Βασικές παράμετροι διαμόρφωσης deployment για διαφορετικές serverless πλατφόρμες 99

~~No table of figures entries found.~~

Formatted: Greek

Formatted: Font: +Body (Aptos), 14 pt, Bold, Font color: Accent 1

Formatted: Space After: 0 pt

Field Code Changed

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

1. Εισαγωγή

Τα τελευταία χρόνια, η ραγδαία εξέλιξη των τεχνολογιών του υπολογιστικού νέφους (cloud computing) έχει μετασχηματίσει ριζικά τον τρόπο με τον οποίο αναπτύσσονται, φιλοξενούνται και εκτελούνται οι εφαρμογές λογισμικού. Οι οργανισμοί και οι προγραμματιστές μπορούν πλέον να αξιοποιούν προσφερόμενες υποδομές (ως το μέσον εκτέλεσης των εφαρμογών), πλατφόρμες (ως τα περιβάλλοντα ανάπτυξης των εφαρμογών) και υπηρεσίες (ως μέρη λογισμικού που επαναχρησιμοποιούνται για την υλοποίηση απαιτούμενων λειτουργιών των εφαρμογών) χωρίς να απαιτείται η φυσική διαχείριση πόρων ή η επένδυση σε εξοπλισμό. Ένα από τα σημαντικότερα στάδια αυτής της εξέλιξης αποτελεί το serverless computing (ο υπολογισμός χωρίς διακομιστές), το οποίο προσφέρει τη δυνατότητα εκτέλεσης κώδικα χωρίς ρητή διαχείριση διακομιστών, επιτρέποντας στους χρήστες να επικεντρωθούν αποκλειστικά στην ανάπτυξη της λογική της εφαρμογής τους.

Το serverless μοντέλο παρουσιάζει σημαντικά πλεονεκτήματα, όπως αυτόματη κλιμάκωση, χρέωση βάσει πραγματικής χρήσης (pay-per-use) και απλοποιημένη ανάπτυξη λειτουργιών (functions), καθιστώντας το ιδανικό για εφαρμογές που απαιτούν ευελιξία και δυναμική απόδοση. Ωστόσο, η ραγδαία εξάπλωση πολλαπλών serverless πλατφορμών – όπως το AWS Lambda[1], το Google Cloud Functions[2] και το Azure Functions[3] – έχει δημιουργήσει νέες προκλήσεις για τους προγραμματιστές και τους ερευνητές. Κάθε πλατφόρμα διαθέτει διαφορετικά χαρακτηριστικά, περιορισμούς, μοντέλα κόστους και επιδόσεις, καθιστώντας δύσκολη τη σύγκριση και επιλογή της καταλληλότερης πλατφόρμας που να ταιριάζει στις εκάστοτε απαιτήσεις ενός οργανισμού ή ενός ατόμου.

Το κύριο πρόβλημα που επιχειρεί να αντιμετωπίσει η παρούσα εργασία είναι η απουσία ενός ενιαίου, αντικειμενικού και εύχρηστου εργαλείου που να επιτρέπει τη συγκριτική αξιολόγηση serverless πλατφορμών βάσει τεχνικών και ποιοτικών κριτηρίων. Σήμερα, η επιλογή πλατφόρμας βασίζεται κυρίως σε χειροκίνητη αναζήτηση, εμπειρική γνώση ή αποσπασματικές μετρήσεις, κάτι που συχνά οδηγεί σε λανθασμένες αποφάσεις ή αυξημένο χρόνο πειραματισμού. Η έλλειψη ενοποιημένων πληροφοριών για τις πλατφόρμες δημιουργεί επίσης δυσκολίες στην εκπαίδευση, στην έρευνα και στη βιομηχανική αξιολόγηση serverless λύσεων.

Η παρούσα διπλωματική εργασία προτείνει και υλοποιεί ένα πληροφοριακό σύστημα Παγκόσμιου Ιστού (web information system), το οποίο επιτρέπει τη αναζήτηση μέσω φίλτρων και την ταξινόμηση των (serverless) πλατφορμών προς διευκόλυνση της επιλογής τους καθώς και την αυτόματη διάταξη (deployment) serverless συναρτήσεων σε πραγματικές πλατφόρμες υπολογιστικού νέφους. Ειδικότερα, ο χρήστης μπορεί να

[1] https://aws.amazon.com/lambda
[2] https://cloud.google.com/functions
[3] https://azure.microsoft.com/services/functions

Formatted: Greek

Field Code Changed

εφαρμόζει πολλαπλά φίλτρα (όπως διαθέσιμη μνήμη, περιορισμοί χρόνου εκτέλεσης, υποστηριζόμενες γλώσσες προγραμματισμού, μοντέλο χρέωσης) καθώς και να καθορίζει σταθμισμένα κριτήρια (weighted criteria) για το φιλτράρισμα και την ταξινόμηση των πλατφορμών, αντίστοιχα. Με αυτό τον τρόπο, διευκολύνεται ως προς την επιλογή της καταλληλότερης πλατφόρμας με βάση τις απαιτήσεις και προτιμήσεις του. Επιπλέον, το σύστημα υποστηρίζει την αυτόματη διάταξη συναρτήσεων σε 3 εναλλακτικές πλατφόρμες, τις οποίες ενδέχεται να προτιμά ο χρήστης, μέσω διασύνδεσης με το Serverless Framework, προσφέροντας μια πρακτική και πλήρως λειτουργική προσέγγιση για την αξιολόγηση των πλατφορμών στην πράξη.

Η εργασία είναι δομημένη ως εξής. Στο Κεφάλαιο 2 παρουσιάζεται μια εννοιολογική και τεχνολογική ανασκόπηση, καλύπτοντας σημαντικές έννοιες, όπως το υπολογιστικό νέφος, καθώς και βασικές τεχνολογίες που χρησιμοποιήθηκαν, όπως οι serverless πλατφόρμες. Το Κεφάλαιο 3 παρουσιάζει τις σχετικές εργασίες και το θεωρητικό υπόβαθρο, αναλύοντας υφιστάμενες προσεγγίσεις και εργαλεία που συνδέονται με το αντικείμενο της παρούσας εργασίας. Στο Κεφάλαιο 4 περιγράφεται αναλυτικά η διαδικασία ανάπτυξης και υλοποίησης του προτεινόμενου πληροφοριακού συστήματος ιστού, συμπεριλαμβανομένης της αρχιτεκτονικής, των βασικών υποσυστημάτων/ενοτήτων και των τεχνολογιών που χρησιμοποιούνται. Στο Κεφάλαιο 5 παρέχονται οδηγίες εγκατάστασης και εκτέλεσης του συστήματος, καθώς και επίδειξη της λειτουργίας του με ενδεικτικά screenshots. Στο Κεφάλαιο 6 προτείνονται κατευθύνσεις για μελλοντική εργασία και έρευνα.

Formatted: Greek

2. Θεωρητική και Τεχνολογική Ανασκόπηση

Το κεφάλαιο αυτό παρουσιάζει το θεωρητικό υπόβαθρο της παρούσας διπλωματικής εργασίας, το οποίο επικεντρώνεται στην τεχνολογία του υπολογιστικού νέφους και ειδικότερα στη φιλοσοφία της serverless υπολογιστικής (serverless computing). Αρχικά, αναλύονται οι βασικές έννοιες και τα χαρακτηριστικά του υπολογιστικού νέφους, τα μοντέλα υπηρεσιών και ανάπτυξης που το διέπουν, καθώς και η τεχνολογική εξέλιξη που οδήγησε στη δημιουργία του serverless (υπολογιστικού) μοντέλου.

Στη συνέχεια, παρουσιάζονται οι αρχιτεκτονικές και οι μηχανισμοί διαχείρισης πόρων που υιοθετούνται σε serverless περιβάλλοντα, καθώς και οι κύριες εμπορικές και ανοικτού κώδικα πλατφόρμες serverless. Τέλος, επισημαίνονται τα πλεονεκτήματα και οι προκλήσεις που συνοδεύουν τη χρήση του μοντέλου αυτού, θέτοντας τη θεωρητική βάση πάνω στην οποία αναπτύχθηκε το πληροφοριακό σύστημα της παρούσας εργασίας.

2.1 Υπολογιστικό Νέφος (Cloud Computing)

2.1.1 Ορισμός

Το υπολογιστικό νέφος (cloud computing) αποτελεί ένα παράδειγμα κατανεμημένης υπολογιστικής, το οποίο προσφέρει υπολογιστικούς πόρους — όπως επεξεργαστική ισχύ, αποθήκευση, λογισμικό και υπηρεσίες — μέσω του Διαδικτύου, κατ' απαίτηση και με δυνατότητα δυναμικής κλιμάκωσης. Ειδικότερα, σύμφωνα με τον ορισμό του NIST (National Institute of Standards and Technology)[6][4], το υπολογιστικό νέφος είναι «ένα μοντέλο που επιτρέπει την κατ' απαίτηση πρόσβαση σε κοινόχρηστους, ρυθμιζόμενους υπολογιστικούς πόρους (όπως δίκτυα, servers, αποθήκευση και εφαρμογές), οι οποίοι μπορούν να παραχωρηθούν και να αποδεσμευθούν με ελάχιστη διαχειριστική προσπάθεια».

Ουσιαστικά, το νέφος επιτρέπει στις επιχειρήσεις και στους οργανισμούς να αποφεύγουν τη δαπανηρή επένδυση και συντήρηση ιδιόκτητων κέντρων δεδομένων (data centers), αντικαθιστώντας το κεφαλαιοουχικό κόστος (CAPEX) με λειτουργικό κόστος (OPEX) βάσει χρήσης (πόρων που προσφέρονται από τρίτα μέρη, τους παρόχους νέφους).

Η εικονικοποίηση (virtualization) αποτελεί τον θεμέλιο λίθο του υπολογιστικού νέφους, επιτρέποντας την ταυτόχρονη λειτουργία πολλών εικονικών μηχανών (VMs) πάνω στον

[4] National Institute of Standards and Technology (NIST), *The NIST Definition of Cloud Computing*, Special Publication 800-145, 2011.

Formatted: Greek

Commented [KK9]: Θέλουμε το κυριως κειμενο σε όλη την αναφορα να είναι πλήρως στοιχισμένο (fully justified).

Commented [KK10R9]: Το σχολιο δεν έχει ακόμη ικανοποιηθεί!!!

Commented [KK11]: Θα προτεινα η οργανωση του κεφαλαιου να είναι πιο σωστά δομημένη. Θέλουμε 2 βασικές ενότητες:
2.1 Υπολογιστικό Νέφος
2.2 Serverless Computing

Υπό αυτές τις ενότητες έπειτα, μπορούν να μπαινουν υπο-ενότητες για περαιτέρω οργάνωση των ενοτήτων.

Commented [KK12]: Προσέξτε διότι ουσιαστικά εδώ επαναλαμβάνετε ότι ειπώθηκε προηγουμένως στην προηγούμενη παράγραφο. Επίσης, υπάρχουν και ορισμενες ασυνέπειες μεταξύ του προηγ. και του τρεχοντος περιεχομενου.

Θα προτεινα να αφαιρέσετε την προηγούμενη παράγραφο και να διατηρήσετε μόνο τις 2 τρέχουσες. Δεν χρειάζεται αυτές να μπουν σε μια πρώτη ενότητα διότι θεωρούμε πως είναι εισαγωγικές (καθολικό επίπεδο).
Οπότε, έπειτα ξεκινάτε πλέον από την 2.1 που θα αφορά το Υπολογιστικό Νέφος.

Commented [KK13]: 2.1.1 Ορισμός
2.1.2 Βασικά Χαρακτηριστικά
2.1.3 Μοντέλα Παράδοσης / Είδη Υπηρεσιών
2.1.4 Μοντέλα Ανάπτυξης
....

Αυτή θα πρεπει να είναι η δομή της Ενότητας 2.1

Commented [KK14]: Λειπει η σχετική παραπομπή εδώ.

Commented [AK15R14]: Προστεθηκε στη βιβλιογραφία

Commented [KK16R14]: Οκ αλλά θα πρεπει να παραπέμπεται η βιβλιογραφική αναφορά εντός του κειμένου της αναφοράς.
Καλό είναι να σημειώνετε με κάποιο τροπο (πχ. μέσω σχολίου) τις τοποθεσιες όπου θα πρέπει να εισάγονται οι παραπομπές εφοσον σκοπεύετε να τις εισάγετε στην τελική έκδοση της αναφοράς σας.

Commented [KK17R14]: Δεν ζήτησα να βάλετε την βιβλιογραφική αναφορά σε μορφη υποσέλιδου. Ζήτησα να έχετε παραπομπή προς την αναφορά αυτή.

Υπάρχουν 2 στυλ παραπομπής:
(α) με βάση αριθμό [X]
(β) με βάση ονόματα συγγραφέων, πχ. Kritikos et al. (2024) όταν γίνεται άμεση αναφορά στους συγγραφείς και (Kritikos et al., 2024) όταν γίνεται έμμεση (αναφορά)

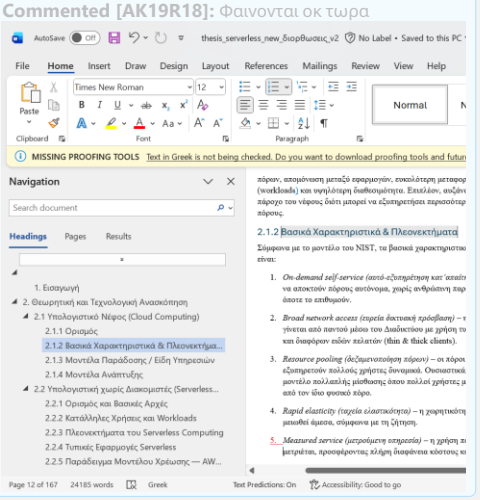
Formatted: Greek

ίδιο φυσικό εξυπηρετητή. Έτσι, επιτυγχάνεται καλύτερη εκμετάλλευση των διαθέσιμων πόρων, απομόνωση μεταξύ εφαρμογών, ευκολότερη μεταφορά φόρτων εργασίας (workloads) και υψηλότερη διαθεσιμότητα. Επιπλέον, αυξάνονται τα κέρδη για τον πάροχο του νέφους διότι μπορεί να εξυπηρετήσει περισσότερους πελάτες με λιγότερους πόρους.

2.1.2 Βασικά Χαρακτηριστικά & Πλεονεκτήματα

Commented [KK18]: Για κάποιο λόγο, όταν κάνω CTRL-F για να δω την δομή της αναφοράς, τα κεφαλαια 2.1.2 και 2.2.4 δεν φαινονται στην δομή. Ας το κοιτάξετε διότι μπορει να δημιουργήσει προβλημα στον πινακα περιεχομένων (ο οποίος παράγεται αυτομάτα).

Commented [AK19R18]: Φαινονται οκ τωρα



Σύμφωνα με το μοντέλο του NIST, τα βασικά χαρακτηριστικά του υπολογιστικού νέφους είναι:

1. *On-demand self-service (αυτό-εξυπηρέτηση κατ'απαίτηση)* – οι χρήστες μπορούν να αποκτούν πόρους αυτόνομα, χωρίς ανθρώπινη παρέμβαση από τον πάροχο, όποτε το επιθυμούν.

2. *Broad network access (ευρεία δικτυακή πρόσβαση)* – η πρόσβαση στις υπηρεσίες νέφους γίνεται από παντού μέσω του Διαδικτύου με χρήση τυποποιημένων πρωτοκόλλων και διαφόρων ειδών πελατών (thin & thick clients).

3. *Resource pooling (δεξαμενοποίηση πόρων)* – οι πόροι συγκεντρώνονται και εξυπηρετούν πολλούς χρήστες δυναμικά. Ουσιαστικά, υποστηρίζεται ένα μοντέλο πολλαπλής μίσθωσης όπου πολλοί χρήστες μπορούν να εξυπηρετούνται από τον ίδιο φυσικό πόρο.

4. *Rapid elasticity (ταχεία ελαστικότητα)* – η χωρητικότητα (πόρων) μπορεί να αυξηθεί ή να μειωθεί άμεσα, σύμφωνα με τη ζήτηση.

5. *Measured service (μετρούμενη υπηρεσία)* – η χρήση παρακολουθείται και μετριέται, προσφέροντας πλήρη διαφάνεια κόστους και κατανάλωσης.

Από την άλλη μεριά, τα κύρια πλεονεκτήματα του cloud computing συνοψίζονται ως εξής:

- *Ευελιξία*: Δυνατότητα άμεσης προσαρμογής πόρων ανάλογα με τις ανάγκες.
- *Κλιμάκωση*: Αυτόματη αύξηση ή μείωση πόρων χωρίς διακοπή λειτουργίας (ειδικότερα όταν αυτή πραγματοποιείται οριζόντια και όχι κάθετα) με βάση τις τρέχουσες συνθήκες και περιβάλλον (context).
- *Αξιοπιστία και διαθεσιμότητα*: Αυστηρά SLA (με κατάλληλους περιορισμούς διαθεσιμότητας προς εγγύηση) καθώς και πλεονασμός δεδομένων και πόρων μαζί με μηχανισμούς αυτόματης μετάπτωσης (failover) για περισσότερη αξιοπιστία και διαθεσιμότητα.
- *Οικονομία*: Μείωση κόστους συντήρησης και εξοπλισμού ιδιαίτερα στην πλευρά του πελάτη. Ο πάροχος διαλέγει ευκαιριακές περιοχές με μικρότερο κόστος

ενοικίασης χώρων, προσωπικού και ενέργειας ώστε να μειώσει τα σχετικά έξοδα λειτουργίας και συντήρησης/διαχείρισης των κέντρων δεδομένων του.

- *Προσβασιμότητα*: Δυνατότητα απομακρυσμένης εργασίας και συνεργασίας.

2.1.3 Μοντέλα Παράδοσης / Είδη Υπηρεσιών

Οι υπηρεσίες νέφους διακρίνονται σε διαφορετικά επίπεδα αφαίρεσης και ευθύνης διαχείρισης. Τα κυριότερα είδη υπηρεσιών (Service Models) (ονομάζονται επίσης και μοντέλα παράδοσης) νέφους είναι τα εξής:

Infrastructure as a Service (IaaS)

Περιγραφή: Το μοντέλο παράδοσης IaaS προσφέρει βασικές υποδομές πληροφορικής, όπως εικονικούς εξυπηρετητές, αποθηκευτικό χώρο και δίκτυα. Ο χρήστης έχει τον έλεγχο του λειτουργικού συστήματος, των εφαρμογών και των δεδομένων που εγκαθιστά στους προσφερόμενους εικονικούς εξυπηρετητές, χωρίς να χρειάζεται να διαχειρίζεται το φυσικό υλικό (πχ. ένα φυσικό εξυπηρετητή).

Παραδείγματα: Amazon EC2[5], Google Compute Engine[6], Microsoft Azure IaaS[7].
Πλεονεκτήματα:

- *Πλήρης έλεγχος λειτουργικού περιβάλλοντος* – ευελιξία διαμόρφωσής του.
- *Εύκολη κλιμάκωση* - Τα IaaS περιβάλλοντα προσφέρουν τη δυνατότητα οριζόντιας και κατακόρυφης κλιμάκωσης χωρίς την ανάγκη φυσικής παρέμβασης στη βασική υποδομή. Ο χρήστης μπορεί να αυξήσει ή να μειώσει τους διαθέσιμους πόρους (όπως CPU, RAM, δίσκο ή αριθμό VM instances) μέσω αυτοματοποιημένων μηχανισμών, όπως auto-scaling groups (ομάδες αυτό-κλιμάκωσης) και load balancers (εξισορροπητές φορτίου). Η διαδικασία ορίζεται (πλήρως) από το λογισμικό (software-defined), επομένως η κλιμάκωση πραγματοποιείται άμεσα, χωρίς καθυστέρηση εγκατάστασης ή προμήθειας νέου υλικού, κάτι που μειώνει σημαντικά το λειτουργικό κόστος και βελτιώνει την ευελιξία της υποδομής.
- *Μείωση κόστους* σε σχέση με επιτόπιες (on-premise) υποδομές λόγω της χρέωσης με βάση τη χρήση.

[5] https://aws.amazon.com/ec2
[6] https://cloud.google.com/compute
[7] Virtual machines in Azure - Azure Virtual Machines | Microsoft Learn

Formatted: Greek

Commented [KK20]: Ας γίνει πιο ευπαρουσίαστη η ανάλυση των μοντέλων. Πχ. χρηση bullets ή κατάλληλα διαμορφωμένων τιτλων (με υπογραμμιση ή πλάγια γράμματα).

Commented [KK21R20]: Δεν απαντήθηκε το σχόλιο - προχώρησα σε μορφοποίηση από την δική μου μεριά.

Για λόγους ομοιομορφιας, ας διατηρειται η ίδια μορφοποίηση (μη αριθμημένων άτυπων ενοτήτων) και στο υπολοιπο μέρος της αναφοράς σας.

Commented [KK22]: Θέλουμε συνδεσμους με τη μορφη footnotes εδώ.

Commented [KK23R22]: Δεν απαντήθηκε το σχόλιο.

Commented [KK24R22]: Οκ, πολύ ωραία αλλά υπάρχει ζήτημα με την αρίθμηση των footnotes διότι από τον αριθμό 3 πάμε ξαφνικά στον αριθμό 5.

Platform as a Service (PaaS)

Περιγραφή: Το PaaS παρέχει ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών που περιλαμβάνει εργαλεία, runtime, μεσισμικό (middleware) και υπηρεσίες βάσεων δεδομένων. Ο προγραμματιστής επικεντρώνεται αποκλειστικά στην ανάπτυξη του λογισμικού, ενώ ο πάροχος διαχειρίζεται την υποκείμενη υποδομή και το ίδιο το περιβάλλον ανάπτυξης ενώ μπορεί και να κλιμακώνει το περιβάλλον εφόσον αυτό απαιτείται.
Παραδείγματα: Google App Engine[8], AWS Elastic Beanstalk[9], Heroku[10].
Πλεονεκτήματα:

- *Απλοποιημένη ανάπτυξη και διάθεση εφαρμογών* χωρίς επίβαρο εγκατάστασης, διαχείρισης και συντήρησης του σχετικού περιβάλλοντος ανάπτυξης.
- *Υποστήριξη συνεργατικής και ευκίνητης ανάπτυξης (agile development)* μέσω χρήσης κατάλληλων εργαλείων όπως CI/CD και Git.
- *Δυνατότητα κλιμάκωσης* του περιβάλλοντος.

Software as a Service (SaaS)

Περιγραφή: Το SaaS παρέχει ολοκληρωμένες εφαρμογές προσβάσιμες μέσω φυλλομετρητή (web browser), χωρίς εγκατάσταση ή συντήρηση. Ο πάροχος είναι υπεύθυνος για όλη την υποδομή και το ίδιο το λογισμικό. Ο χρήστης μπορεί μόνο να διαμορφώσει ελάχιστα την προσφερόμενη υπηρεσία λογισμικού.
Παραδείγματα: Google Workspace[11], Microsoft 365[12], Zoom[13], Dropbox[14].
Πλεονεκτήματα:

- *Άμεση πρόσβαση από οπουδήποτε* με την προϋπόθεση διαδικτυακής σύνδεσης.
- *Αυτόματες ενημερώσεις και patching* χωρίς οποιασδήποτε μορφής επιβάρυνσης του πελάτη.
- *Ευέλικτα μοντέλα χρέωσης* όπως συνδρομής (subscription-based).
- *Υψηλή διαθεσιμότητα και ελαστικότητα* της προσφερόμενης εφαρμογής λογισμικού

[8] https://cloud.google.com/appengine
[9] https://aws.amazon.com/elasticbeanstalk
[10] https://www.heroku.com
[11] https://workspace.google.com
[12] https://www.microsoft.com/microsoft-365
[13] https://zoom.us
[14] https://www.dropbox.com

Formatted: Greek

Commented [KK25]: Θέλουμε συνδέσμους με τη μορφή footnotes για αυτά.

Commented [KK26R25]: Κι εδώ δεν φαινεται να έχουν μπει οι παραπομπές

Commented [KK27]: Θέλουμε συνδέσμους με τη μορφη footnotes κι εδώ.

Commented [KK28R27]: Κι εδώ δεν φαινεται να έχουν μπει οι παραπομπές (URL-based footnotes)

Container as a Service (CaaS)

Περιγραφή: Το CaaS επιτρέπει τη διαχείριση και εκτέλεση δοχειοποιημένων (containerized) εφαρμογών σε συστάδες μηχανών (machine clusters), αξιοποιώντας ενορχηστρωτές δοχείων (container orchestrators) όπως ο Kubernetes. Με το μοντέλο αυτό, ο χρήστης μπορεί να διατάσσει, να ενημερώνει και να κλιμακώνει εφαρμογές σε περιβάλλοντα μεγάλων συστάδων, διατηρώντας **υψηλό βαθμό ελέγχου** τόσο στη ρύθμιση των πόρων όσο και στον τρόπο εκτέλεσης των εφαρμογών.

Παραδείγματα:
Google Kubernetes Engine (GKE)[15], Amazon Elastic Kubernetes Service (EKS)[16], Azure Kubernetes Service (AKS)[17], AWS Fargate[18].

Είδη CaaS

Υπάρχουν δύο βασικές κατηγορίες Container as a Service:

(1) Managed Kubernetes CaaS

Σε αυτό το μοντέλο, ο πάροχος νέφους διαχειρίζεται το επίπεδο ελέγχου (control plane) του Kubernetes (master nodes – κύριοι κόμβοι), ενώ ο χρήστης έχει έλεγχο στους κόμβους εργασίας (worker nodes) και στη συμπεριφορά των εφαρμογών. Ο χρήστης διατηρεί **υψηλό βαθμό ελέγχου** ως προς:

- τη διαμόρφωση των κόμβων (τύπος VM, μέγεθος, CPU, RAM),
- Τα άνω και κάτω όρια ως προς την χρηση των πόρων των containers,
- τις στρατηγικές διάταξης (deployment strategies) (rolling updates, διατάξεις blue–green),
- τις πολιτικές δικτύωσης (networking policies), το μοντέλο ελέγχου πρόσβασης με βάση τους ρόλους (Role-Based Access Control – RBAC) και τις πολιτικές ασφάλειας (security policies),
- τον έλεγχο των κανόνων αυτοκλιμάκωσης (autoscaling) (HPA, VPA, Cluster Autoscaler).

[15] Google Kubernetes Engine (GKE): https://cloud.google.com/kubernetes-engine/docs
[16] Amazon Elastic Kubernetes Service (EKS): https://docs.aws.amazon.com/eks/
[17] Azure Kubernetes Service (AKS): https://learn.microsoft.com/azure/aks/
[18] https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.htmlOpenFaaS

Formatted: Greek

Commented [KK29]: Θέλουμε URLs με τη μορφή footnotes κι εδώ.

Commented [KK30R29]: Θέλουμε κανονικά links/URLs στα footnotes και όχι τιτλους με υπογράμμιση

Το μοντέλο αυτό είναι κατάλληλο για σύνθετες cloud-native αρχιτεκτονικές που απαιτούν λεπτομερή παραμετροποίηση.

(2) Serverless CaaS

Σε αυτό το μοντέλο (όπως AWS Fargate ή Google Cloud Run), ο χρήστης υποβάλλει απλώς τις εικόνες των δοχείων του (container images), και ο πάροχος αναλαμβάνει:

- την παροχή τροφοδοσίας (provisioning) των δοχείων,
- την κλιμάκωσή τους (scaling),
- την δικτύωσή τους (networking),
- και την υποκείμενη υποδομή τους (underlying infrastructure).

Ο χρήστης δεν χρειάζεται να διαχειρίζεται καθόλου κόμβους (nodes) ή συστάδες (clusters). Η υποδομή είναι εντελώς αφηρημένη (abstracted), κάτι που μειώνει το λειτουργικό βάρος αλλά και τον βαθμό ελέγχου.

Function as a Service (FaaS)

Το FaaS, που αποτελεί τη βάση της serverless υπολογιστικής (δείτε Ενότητα 2.2), όπου επιτρέπει την εκτέλεση μικρών συναρτήσεων (functions) ως απόκριση σε γεγονότα (events) ή HTTP αιτήσεις. Ο προγραμματιστής αναπτύσσει μόνο τη λογική της συνάρτησης χωρίς να ασχολείται με εξυπηρετητές ή την κλιμάκωση (δηλαδή το περιβάλλον λειτουργίας των συναρτήσεων καθώς και την εκτέλεση & κλιμάκωσή τους).

Παραδείγματα: AWS Lambda[19], Google Cloud Functions[20], Azure Functions[21].
Πλεονεκτήματα:

- *Αυτόματη κλιμάκωση* ανάλογα με τη ζήτηση.
- *Χρέωση αποκλειστικά βάσει πραγματικής χρήσης*.
- *Ιδανικό για event-driven εφαρμογές και μικρο-υπηρεσίες* με συγκεκριμένα μοτίβα χρήσης (πχ. σπάνια/αραιή εκτέλεση, μαζικά παράλληλος υπολογισμός).

---

[19] https://aws.amazon.com/lambda
[20] https://cloud.google.com/functions
[21] https://azure.microsoft.com/services/functions

Formatted: Greek

Commented [KK31]: Ας πειτε πως περισσοτερα πράγματα για το μοντέλο αυτό παρέχονται στην Ενότητα 2.2.

Commented [KK32]: Θέλουμε συνδέσμους με τη μορφή footnotes

Commented [KK33R32]: Δεν απαντήθηκε το σχόλιο αυτό

Formatted: Greek

2.1.4 Μοντέλα Ανάπτυξης

Τα μοντέλα ανάπτυξης (deployment models) καθορίζουν τον τρόπο με τον οποίο προσφέρονται και διαχειρίζονται οι πόροι του υπολογιστικού νέφους, λαμβάνοντας υπόψη την ιδιοκτησία, τον βαθμό ελέγχου και το κοινό στο οποίο απευθύνονται οι προσφερόμενες υπηρεσίες & πόροι νέφους.

Commented [KK34]: Πάλι εδώ θέλουμε μια πιο ωραια οπτικά παρουσιαση των μοντέλων ανάπτυξης αυτή την φορά.

Public Cloud (Δημόσιο Νέφος)

Στο δημόσιο νέφος, οι υπηρεσίες παρέχονται στον οποιοδήποτε μέσω διαδικτύου από τρίτους παρόχους, όπως οι Amazon Web Services (AWS), Microsoft Azure και Google Cloud Platform (GCP). Οι πόροι είναι κοινόχρηστοι ανάμεσα σε πολλούς χρήστες (multi-tenancy) με τη χρήση κατάλληλων μηχανισμών απομόνωσης (ιδιαίτερα για εικονικές μηχανές αλλά και δυνατότητες δημιουργίας ιδιωτικών εικονικών δικτύων).

Πλεονεκτήματα: χαμηλό κόστος, ευκολία πρόσβασης, υψηλή κλιμάκωση.

Μειονεκτήματα: περιορισμένος έλεγχος, ανησυχίες ασφάλειας σε ευαίσθητα δεδομένα, μεγάλη επιφάνεια επιθέσεων.

Private Cloud (Ιδιωτικό Νέφος)

Το ιδιωτικό νέφος ανήκει σε έναν οργανισμό και λειτουργεί είτε σε ιδία επιτόπια (on-premise) υποδομή είτε σε ιδιωτικό περιβάλλον τρίτου παρόχου. Προσφέρεται αποκλειστικά από τον οργανισμό αυτό στα τμήματα και τους υπαλλήλους του. Συνεπώς, ο πάροχος του νέφους είναι και ο μοναδικός του καταναλωτής / πελάτης.

Πλεονεκτήματα: αυξημένος έλεγχος, προσαρμοσμένη ασφάλεια εναρμονισμένη με τις πολιτικές ασφαλείας του παρόχου/οργανισμού, συμμόρφωση με κανονισμούς.

Μειονεκτήματα: υψηλότερο αρχικό κόστος, αυξημένο λειτουργικό κόστος (ιδιαίτερα όσο μεγαλώνει η ιδιωτική υποδομή), αυξημένες απαιτήσεις διαχείρισης και περιορισμένες δυνατότητες κλιμάκωσης και ικανοποίησης φόρτων εργασίας.

Hybrid Cloud (Υβριδικό Νέφος)

Το υβριδικό νέφος αποτελεί ένα μοντέλο παροχής υπολογιστικών υπηρεσιών που συνδυάζει **private cloud** και **public cloud**, επιτρέποντας τη ροή δεδομένων, εφαρμογών

και υπολογιστικών φορτίων μεταξύ των δύο περιβαλλόντων. Σε πιο γενικευμένη μορφή, το υβριδικό νέφος μπορεί να περιλαμβάνει και **πολλαπλά διαφορετικά νέφη (multi-cloud)**, ακόμη και διαφορετικών ειδών υπηρεσιών (IaaS, PaaS, CaaS, SaaS), τα οποία ενοποιούνται μέσω:

- κοινών διεπαφών (APIs),
- προτυποποιημένων τεχνολογιών (π.χ. Kubernetes, OpenAPI),
- ή ιδιόκτητων μηχανισμών μεταφοράς δεδομένων και φορητότητας φόρτων εργασίας (workload portability).

Στόχος του μοντέλου είναι να προσφέρει **ευελιξία**, **βελτιστοποίηση κόστους**, και **αυξημένη ασφάλεια**, αξιοποιώντας το δυνατότερο σημείο κάθε περιβάλλοντος.

Παραδείγματα χρήσης:

- Ευαίσθητα δεδομένα και κρίσιμες υπηρεσίες φιλοξενούνται στο private cloud, ενώ υπολογιστικά εντατικά workloads εκτελούνται στο public cloud για οικονομία και κλιμακωσιμότητα.
- Επιχειρήσεις που χρειάζονται disaster recovery (ανάνηψη από καταστροφές) στο public cloud αλλά θέλουν να διατηρούν το κύριο κέντρο δεδομένων τους (data center) σε ιδιωτική υποδομή.
- Χρήση edge & public cloud για εφαρμογές χαμηλού latency (π.χ. επεξεργασία δεδομένων κοντά στον χρήστη/αισθητήρες στο "άκρο" του δικτύου και αποστολή μόνο των απαραίτητων αποτελεσμάτων στο public cloud για αποθήκευση, ενδελεχή ανάλυση (analytics) ή κεντρική διαχείριση). "Edge" είναι η εκτέλεση/επεξεργασία κοντά στην πηγή των δεδομένων ή στον τελικό χρήστη — π.χ. σε gateway, router, base station ή τοπικό micro-data center, αντί σε απομακρυσμένο cloud.

Πλεονεκτήματα (Υβριδικού Νέφους)

1. Κόστος με βελτιστοποίηση

Οι οργανισμοί χρησιμοποιούν public cloud μόνο όπου χρειάζεται (π.χ. αιχμές ζήτησης), μειώνοντας κόστος σε σχέση με 100% private υποδομή.

2. Ευελιξία & δυναμική κλιμάκωση

Workloads μπορούν να μετακινούνται μεταξύ δημόσιου και ιδιωτικού νέφους, ανάλογα με τις ανάγκες σε φόρτο, απόδοση ή διαθεσιμότητα.

3. Ασφάλεια & συμμόρφωση

Formatted: Greek

Commented [KK35]: Μήπως εννοειτε edge-cloud για ...;

Επίσης, δεν είναι ξεκάθαρο τι είναι το edge. Θα μπορούσατε να το εξηγήσετε εδώ ή σε μορφή footnote (2 προτάσεις το πολύ είναι αρκετές).

Commented [KK36R35]: Δεν καλύψατε τον ορισμό του edge

Commented [KK37]: Προσεξτε εδώ την ευθυγράμμιση διότι έχει «χαλάσει» - θα πρέπει το επισημασμένο κειμενο να πάει στην αρχή της σελίδας ώστε να ευθυγραμμιστεί με το 2.

Formatted: Greek

Ευαίσθητα δεδομένα παραμένουν εντός του private cloud, ενώ μη ευαίσθητα ή προσωρινά workloads μπορούν να εκτελούνται στο public cloud.

4. Αποφυγή Vendor Lock-in

Η χρήση πολλαπλών νεφικών περιβαλλόντων μειώνει την εξάρτηση από έναν μόνο πάροχο.

5. Υψηλή διαθεσιμότητα & ανθεκτικότητα

Επιτρέπει disaster recovery μεταξύ private και public cloud, αυξάνοντας τη συνολική αξιοπιστία.

Μειονεκτήματα

1. Αυξημένη πολυπλοκότητα

Ο συνδυασμός πολλαπλών cloud περιβαλλόντων απαιτεί εξειδικευμένη αρχιτεκτονική και διαχείριση.

2. Προκλήσεις στην ασφάλεια

Η ασφαλής μεταφορά δεδομένων μεταξύ διαφορετικών περιβαλλόντων απαιτεί ισχυρές πολιτικές και παρακολούθηση (monitoring).

3. Κόστος δικτύωσης

Η διασύνδεση private και public cloud συχνά απαιτεί ειδικές συνδέσεις (VPN, Direct Connect), που αυξάνουν το λειτουργικό κόστος.

4. Απαιτείται συμβατότητα τεχνολογιών

Δεν υποστηρίζουν όλοι οι πάροχοι τα ίδια APIs ή πρότυπα μεταφοράς workload, κάτι που δυσχεραίνει την πλήρη φορητότητα (portability).

Community Cloud (Κοινοτικό Νέφος)

Το κοινοτικό νέφος αποτελεί ένα μοντέλο παροχής υπολογιστικών υπηρεσιών στο οποίο η υποδομή μοιράζεται μεταξύ οργανισμών που ανήκουν σε μια κοινότητα με **κοινές ανάγκες**, **πολιτικές**, **κανονιστικές απαιτήσεις** ή **στρατηγικούς στόχους**. Τέτοιες κοινότητες μπορεί να περιλαμβάνουν πανεπιστήμια, δημόσιους οργανισμούς, χρηματοπιστωτικούς φορείς ή οργανισμούς υγείας, όπου απαιτείται συνεργασία ή κοινά πρότυπα ασφαλείας.

Formatted: Greek

Η υποδομή μπορεί να διαχειρίζεται είτε από κοινού από τους συμμετέχοντες οργανισμούς, είτε από τρίτο πάροχο, και ακολουθεί συμφωνημένα μοντέλα διακυβέρνησης, συμμόρφωσης και ασφάλειας.

Πλεονεκτήματα

1. Κοινό/διαμοιραζόμενο κόστος

Το κόστος επένδυσης, λειτουργίας και συντήρησης κατανέμεται μεταξύ των μελών της κοινότητας, καθιστώντας τις υποδομές πιο οικονομικές.

2. Ενοποίηση/Εναρμόνιση πολιτικών ασφάλειας

Οι οργανισμοί ακολουθούν ενιαίες πολιτικές ασφαλείας, συμμόρφωσης και διακυβέρνησης, εξασφαλίζοντας ομοιομορφία και μειώνοντας τον κίνδυνο ασυμβατότητας (incompatibility).

3. Βελτιωμένη συνεργασία μεταξύ οργανισμών

Οι οργανισμοί μπορούν να μοιράζονται δεδομένα, εφαρμογές και υπηρεσίες σε ένα κοινό περιβάλλον με υψηλό επίπεδο εμπιστοσύνης.

4. Προσαρμοσμένο στις ανάγκες της κοινότητας

Το σύστημα μπορεί να ρυθμιστεί ειδικά για τις απαιτήσεις της συγκεκριμένης ομάδας (π.χ. αυξημένη ασφάλεια για δημόσιους φορείς, κοινά authentication frameworks σε πανεπιστήμια).

5. Μεγαλύτερος έλεγχος σε σχέση με το Public Cloud

Επειδή η υποδομή δεν μοιράζεται με «άγνωστους» τρίτους, υπάρχει αυξημένος έλεγχος στην πρόσβαση και στην ασφάλεια.

Μειονεκτήματα

1. Περιορισμένη ευελιξία σε σχέση με Public Cloud

Οι πόροι είναι κοινόχρηστοι αλλά όχι απεριόριστοι. Η κλιμάκωση ίσως απαιτεί επιπλέον επενδύσεις.

2. Πολυπλοκότητα διακυβέρνησης

Formatted: Greek

Η συνεργασία μεταξύ πολλών οργανισμών απαιτεί κοινά νομικά πλαίσια, συμβάσεις και μηχανισμούς διαχείρισης.

3. Κόστος μεγαλύτερο από το Public Cloud

Αν και το κόστος μοιράζεται, δεν μπορεί να επιτύχει την οικονομία κλίμακας των μεγάλων δημόσιων παρόχων.

4. Απαιτείται ευθυγράμμιση πολιτικών

Η ενοποίηση των πολιτικών ασφαλείας και συμμόρφωσης μεταξύ διαφορετικών οργανισμών συχνά απαιτεί χρόνο και πολύπλοκη διαδικασία συντονισμού.

2.2 Υπολογιστική χωρίς Διακομιστές (Serverless Computing)

Commented [KK38]: Καλό είναι να οργανώσετε δομικά την τρεχουσα ενότητα σε υπο-ενότητες.

2.2.1 Ορισμός και Βασικές Αρχές

Η serverless υπολογιστική αποτελεί ένα μοντέλο ανάπτυξης εφαρμογών στο οποίο ο προγραμματιστής επικεντρώνεται αποκλειστικά στη λογική του κώδικα, ενώ ο πάροχος αναλαμβάνει πλήρως τη διαχείριση της υποδομής, την ασφάλεια, τη διάταξη, την εκτέλεση και την κλιμάκωση του λογισμικού (δηλαδή των συναρτήσεων).

Το λογισμικό που αναπτύσσεται έχει τη μορφή συνάρτησης (function), η οποία ενεργοποιείται μόνο όταν προκύπτει ένα γεγονός (event), π.χ. HTTP αίτημα, μήνυμα από ουρά μηνυμάτων (message queue) ή αλλαγή σε αποθηκευτικό χώρο. Με αυτό το μοντέλο, η υποδομή παραμένει πλήρως αόρατη στον προγραμματιστή.

Ο πάροχος υπηρεσιών serverless αναλαμβάνει αυτόματα:

- τροφοδοσία των απαιτούμενων πόρων (CPU, μνήμη),
- διάταξη και εκτέλεση της συνάρτησης με ασφαλή απομόνωση,
- αυτόματη κλιμάκωση (scaling) σε πραγματικό χρόνο,
- παρακολούθηση, logging και απελευθέρωση πόρων μετά το πέρας εκτέλεσης.

Η χρέωση βασίζεται αποκλειστικά στη διάρκεια εκτέλεσης και στη μνήμη που δεσμεύεται (μοντέλο **pay-per-invocation**).
Οι δύο παράγοντες κόστους είναι:

1. **Execution time** (χρόνος εκτέλεσης) (ms)
2. **Memory allocation** (δέσμευση κύριας μνήμης) ανά εκτέλεση

Formatted: Greek

2.2.2 Κατάλληλες Χρήσεις και Workloads

Οι serverless συναρτήσεις ταιριάζουν ιδανικά σε workloads που έχουν:

1. Event-driven φύση

REST APIs, webhooks, IoT triggers και real-time events παράγουν διακριτές, αυτόνομες, σύντομες εκτελέσεις.
Οι συναρτήσεις μπορούν να ενεργοποιούνται ακαριαία χωρίς ανάγκη μόνιμης υποδομής.

2. Μικρή διάρκεια εκτέλεσης & ασταθή (burst) φόρτο

Οι serverless πλατφόρμες μπορούν να δημιουργούν χιλιάδες παράλληλα στιγμιότυπα μιας συνάρτησης μέσα σε λίγα milliseconds.
Κατάλληλο για αιχμές (spikes) όπως:

- χιλιάδες HTTP αιτήματα,
- κατ'απαίτηση επεξεργασία αρχείων,
- αγωγοί δεδομένων πραγματικού χρόνου (real-time data pipelines).

3. Ανεξάρτητα και παράλληλα βήματα επεξεργασίας

Οι συναρτήσεις αποδίδουν τέλεια σε μαζικά παράλληλους και ανεξάρτητους υπολογισμούς (**massively parallel independent computations)**, δηλαδή:

- υπολογισμούς χωρίς εξαρτήσεις (embarrassingly parallel tasks),
- batch επεξεργασία μεγάλων datasets (image processing, ML inference),
- map–reduce σενάρια όπου το "map" στάδιο μπορεί να εκτελεστεί από χιλιάδες ταυτόχρονα στιγμιότυπα της ίδιας συνάρτησης (function instances).

2.2.3 Πλεονεκτήματα του Serverless Computing

Τα βασικά πλεονεκτήματα του Serverless Computing είναι τα εξής:

- *Μηδενική διαχείριση υποδομής***:** ο προγραμματιστής δεν χρειάζεται να προβλέψει ή να συντηρεί διακομιστές (servers).
- *Αυτόματη κλιμάκωση*: οι συναρτήσεις δημιουργούνται και εκτελούνται σε μορφή όσων παράλληλων στιγμιοτύπων απαιτεί η ζήτηση.

- *Ψευδαίσθηση απεριόριστης κλιμάκωσης*: επειδή οι συναρτήσεις είναι μικρές μονάδες υπολογισμού, οι πάροχοι μπορούν να δημιουργούν πολύ περισσότερα στιγμιότυπα σε σχέση με εικονικές μηχανές (virtual machines).
- *Χρέωση βάσει πραγματικές χρήσης*: μηδενικό κόστος όταν η συνάρτηση δεν εκτελείται.
- *Free tier*: εκατομμύρια δωρεάν κλήσεις συναρτήσεων τον μήνα από παρόχους όπως AWS, GCP, Azure.
- *Ταχεία ανάπτυξη εφαρμογών*: εύκολη ενσωμάτωση σε αγωγούς (pipelines) CI/CD ώστε να πραγματοποιηθεί υποστήριξη για γρήγορη ενοποίηση αλλαγών και ταχείες διατάξεις (πχ. σε παραγωγικά περιβάλλοντα).

2.2.4 Τυπικές Εφαρμογές Serverless

Το serverless μοντέλο ενδείκνυται για:

- **RESTful APIs & webhooks**
- **Επεξεργασία αρχείων και ροών δεδομένων (file triggers, ETL tasks)**
- **IoT εφαρμογές** με υψηλό αριθμό μικρών γεγονότων
- **Real-time analytics (αναλυτική πραγματικού χρόνου) & event processing (επεξεργασία γεγονότων)**
- **Microservices (μικρο-υπηρεσίες) με ανεξάρτητη λογική**
- **Massively parallel tasks** (μαζικά παράλληλες λειτουργίες) χωρίς εξαρτήσεις

2.2.5 Παράδειγμα Μοντέλου Χρέωσης — AWS Lambda

Η AWS Lambda, μια serverless πλατφόρμα, χρεώνει ως εξής:

- Παράγοντες χρέωσης:
    - Τιμή ανά εκτέλεση: **$0.20 ανά 1 εκατ. αιτήσεις**
        - **Πρώτα 1.000.000 invocations/μήνα → δωρεάν**
    - Τυπικό κόστος όγκου δεδομένων ανά διάρκεια εκτέλεσης: **$0.0000166667 per GB–second (**GB–seconds used = Memory (GB) × Time (seconds)**)**
        - **Πρώτα 400,000 GB-seconds είναι δωρεάν**

Formatted: Greek

Formatted: Greek

Έτσι, μια συνάρτηση με:

- **256 MB μνήμη**,
- **1000 ms διάρκεια**,
- **1.600.000 εκτελέσεις**

κοστίζει **$0.12 τον μήνα (1000000 εκτελέσεις δωρεάν & 600000 εκτελέσεις με κόστος 0.20 ανά εκατομμύριο – το όριο των 400000 GB δεν ξεπερνιέται)**.

2.3 Ιστορική Εξέλιξη και Ανάγκη Ανάπτυξης Serverless Πλαισίων

Η εξέλιξη της serverless υπολογιστικής προέκυψε ως μέρος της ευρύτερης τάσης αφαίρεσης (abstraction) της πολυπλοκότητας της υποδομής και επιτάχυνσης των αναπτυξιακών κύκλων εφαρμογών στο υπολογιστικό νέφος.

Η πρώτη καταγεγραμμένη υλοποίηση μοντέλου **pay-as-you-go για εκτέλεση κώδικα** εμφανίστηκε το **2006** με την πλατφόρμα **Zimki** της Canon, η οποία επέτρεπε την εκτέλεση JavaScript συναρτήσεων στο cloud χωρίς διαχείριση υποδομής [Zimki, 2006].

Το **2008**, η **Google App Engine (GAE)** εισήγαγε μία πρώιμη μορφή *Platform-as-a-Service*, προσφέροντας εκτέλεση backend εφαρμογών χωρίς διαχείριση servers. Παρότι δεν ήταν πλήρως serverless, καθιέρωσε πρότυπα αυτόματης κλιμάκωσης και απομόνωσης πόρων [Google, 2008].

Η πραγματική θεμελίωση του serverless μοντέλου ήρθε το **2014** με την ανακοίνωση της **AWS Lambda**, της πρώτης πλήρως διαχειριζόμενης υπηρεσίας εκτέλεσης συναρτήσεων βάσει γεγονότων (event-driven functions) [AWS Lambda, 2014].

Ακολούθησαν:

- **Google Cloud Functions (2016)** – Ισχυρή ενσωμάτωση με Pub/Sub και Cloud Storage.
- **Azure Functions (2016)** – Εισήγαγε τα **Durable Functions**, ένα μοντέλο για **καταστατικές (stateful) ροές εργασίας (δηλ. σύνθετες συναρτήσεις)**, επιτρέποντας σύνθεση συναρτήσεων, ενοχρήστρωση (orchestration), επανεκτελέσεις (retries) και μακρόβιες ροές εργασίας (workflows).

- **Open-source λύσεις: OpenFaaS (2016)**[22], **Kubeless (2017)**[23], **Apache OpenWhisk (2016)**[24], που στόχευαν σε αυτο-φιλοξενούμενες (self-hosted) ή υβριδικές serverless υποδομές.

2.4 Αντιμετώπιση Κλειδώματος στον Πάροχο

Οι πρώτες τρεις πλατφόρμες (AWS Lambda, Google Cloud Functions, Azure Functions) **δεν** μειώνουν το vendor lock-in (κλείδωμα σε έναν πάροχο)· αντιθέτως, **το εντείνουν**, διότι:

- χρησιμοποιούν ιδιόκτητα (custom) triggers,
- απαιτούν ειδικές ρυθμίσεις,
- ακολουθούν διαφορετικά μοντέλα χρέωσης και APIs.

Η αντιμετώπιση του vendor lock-in έγινε **αργότερα**, κυρίως μέσω:

• Open-source serverless πλατφορμών (OpenFaaS, Kubeless, OpenWhisk)

• Πολυνεφικών (multi-cloud) εργαλείων όπως το Serverless Framework[25]

Multi-Cloud Περιβάλλοντα – Τι Είναι, Πλεονεκτήματα & Μειονεκτήματα

Τα **multi-cloud** (πολυνεφικά) περιβάλλοντα είναι αρχιτεκτονικές όπου ένας οργανισμός χρησιμοποιεί περισσότερους από έναν παρόχους cloud (π.χ. AWS + GCP + Azure). Μπορεί να περιλαμβάνουν:

- public clouds,
- private clouds,
- self-hosted serverless πλατφόρμες,
- υβριδικές συνθέσεις.

Πλεονεκτήματα Multi-Cloud

[22] https://www.openfaas.com/
[23] https://github.com/kubeless/kubeless
[24] https://openwhisk.apache.org/
[25] https://www.serverless.com/framework/docs

Formatted: Greek

Commented [KK39]: Δεν αναφέρετε κάποιο έτος αναφοράς ή κάποιο χρονικό εύρος αναφοράς για τις λύσεις αυτές (όπως έγινε στα προηγ. Bullets)

Commented [KK40R39]: Επίσης, θέλουμε και URLs με την μορφή footnotes για αυτές τις λύσεις

Commented [KK41R39]: Έχει μείνει μόνο η αναφορά στο έτος ή χρονική διάρκεια για να εισάγετε.

Commented [KK42]: Μου φαινεται πως από εδώ και πέρα θα πρέπει να ξεκινήσεις μια νέα ενότητα 2.4 (πριν την ενότητα της αρχιτεκτονικής) που να έχει ως τίτλο: 2.4 Αντιμετώπιση Κλειδώματος στον Πάροχο

Commented [KK43]: Αν δεν έχει δοθεί ήδη URL για το serverless framework ας το δωσετε εδώ με μορφη footnote

Formatted: Greek

- Αποφυγή vendor lock-in
- Καλύτερη διαθεσιμότητα και ανθεκτικότητα (high availability)
- Επιλογή του βέλτιστου παρόχου ανά υπηρεσία
- Μεγαλύτερη ευελιξία κόστους
- Δυνατότητα γεωγραφικής βελτιστοποίησης

Μειονεκτήματα Multi-Cloud

- Μεγαλύτερη πολυπλοκότητα διαχείρισης
- Δυσκολία στην ομογενοποίηση ασφάλειας και πολιτικών πρόσβασης
- Ανάγκη για ενιαίες πρακτικές παρακολούθησης/ορατότητας (observability)
- Ετερογένεια APIs των παρόχων

Ο Ρόλος του Serverless Framework στην Εξέλιξη

Για να αντιμετωπιστεί η πολυπλοκότητα ενός multi-cloud περιβάλλοντος, παρουσιάστηκε το **Serverless Framework**, ένα εργαλείο ανοικτού κώδικα που επιτρέπει:

- τη δήλωση serverless εφαρμογών μέσω ενιαίου αρχείου (serverless.yml),
- την αυτόματη διάταξη (deployment) συναρτήσεων/serverless εφαρμογών σε AWS, GCP, Azure και OpenFaaS,
- τη μεταφορά serverless εφαρμογών μεταξύ παρόχων,
- την απλοποίηση διαχείρισης triggers, events και permissions.

Έτσι, το Serverless Framework αποτελεί σήμερα ένα de facto πρότυπο για πολυνεφικές serverless εφαρμογές.

2.5 Αρχιτεκτονική Serverless

Η αρχιτεκτονική της serverless υπολογιστικής ακολουθεί ένα **event-driven** μοντέλο εκτέλεσης, στο οποίο οι συναρτήσεις ενεργοποιούνται μόνο ως απάντηση σε εξωτερικά ή εσωτερικά γεγονότα (events).

Formatted: Greek

Η διαχείριση της υποδομής (provisioning, scaling, monitoring, security) γίνεται εξ ολοκλήρου από τον πάροχο του νέφους, ενώ ο προγραμματιστής επικεντρώνεται αποκλειστικά στη λογική της συνάρτησης.

Παρότι κάθε πάροχος (AWS, GCP, Azure, OpenFaaS κ.λπ.) εφαρμόζει διαφορετικές τεχνολογικές υλοποιήσεις, οι περισσότερες σύγχρονες serverless πλατφόρμες ακολουθούν ένα κοινό αρχιτεκτονικό μοτίβο.

Η αρχιτεκτονική αυτή —όπως φαίνεται στην αντίστοιχη εικόνα— αποτελείται από **πέντε βασικά επίπεδα**, τα οποία συνεργάζονται μεταξύ τους.

---

1. Frontend Layer (CLI / SDK / Web Console / Deployment API)

Το επίπεδο αυτό περιλαμβάνει όλες τις διεπαφές ή τα εργαλεία (όπως SKDs) μέσω των οποίων ο χρήστης αλληλεπιδρά με την πλατφόρμα:

- CLI διεπαφές (π.χ. aws lambda, gcloud functions, faas-cli)
- Web Consoles
- Deployment APIs
- SDKs

Ρόλος:

- Υποβολή (upload) κώδικα και artifacts (π.χ. zip αρχεία function code).
- Διαμόρφωση triggers (HTTP, Pub/Sub, cron κ.λπ.)
- Παραμετροποίηση μνήμης, timeout (μέγιστη επιτρεπτή διάρκεια εκτέλεσης της συνάρτησης) , περιβαλλοντικών μεταβλητών.
- Διαχείριση εκδόσεων (versions) & διατάξεων (deployments).

Commented [KK44]: Δεν είναι ξεκάθαρο τι σημαίνει timeout ως προς τι.

Commented [AK45R44]: Timeout δηλαδή μέγιστη επιτρεπτή διάρκεια εκτέλεσης της συνάρτησης)

2. Control Plane (Master Node / Scheduler / API Gateway / Metadata Store / Security Service)

Στο διάγραμμα αρχιτεκτονικής, ο **Master** (κόμβος) αντιστοιχεί στο Control Plane (Επίπεδο Ελέγχου).
Το Control Plane δεν εκτελεί συναρτήσεις· συντονίζει, αποφασίζει, και διαχειρίζεται το σύστημα.

Formatted: Greek

Περιλαμβάνει:

• **API Gateway**: Δέχεται HTTP αιτήματα (requests) και τα μετατρέπει σε γεγονότα κλήσεων (συναρτήσεων) (invocation events).

• **Authentication & Authorization Service**: Επαληθεύει τα δικαιώματα πρόσβασης σε functions, queues, storage events κ.λπ.

• **Function Metadata Store**: Αποθηκεύει δομικά πληροφοριακά στοιχεία (δηλ. μεταδεδομένα) κάθε συνάρτησης:

- runtime
- μνήμη & timeout
- triggers
- versions
- άδειες (**δικαιώματα πρόσβασης και εκτέλεσης της συγκεκριμένης function**)

Commented [KK46]: Οκ αλλά ως προς τι;

• **Scheduler**: Αποφασίζει σε ποιον κόμβο (Invoker Node / Worker Node) θα εκτελεστεί η συνάρτηση.

Ρόλος Scheduler:

- αξιολογεί latency(δικτυακή καθυστέρηση), μνήμη, CPU φόρτο (load), και διαθέσιμα δοχεία
- βελτιστοποιεί επιδόσεις και κόστος
- αποτρέπει υπερφόρτωση κόμβων

Commented [KK47]: Με latency εννοούμε χρονο απόκρισης; Δικτυακή καθυστερηση;

Σχέση με Workers:
Ο Scheduler **δεν εκτελεί** τις συναρτήσεις — απλώς επιλέγει τον κατάλληλο κόμβο (εργασίας – worker).

3. Data Plane (Invoker Nodes / Worker Nodes / Sandboxed Runtimes)

Αυτό είναι το επίπεδο όπου "τρέχουν" (δηλ. γίνονται διαθέσιμες) οι συναρτήσεις.

Περιλαμβάνει:

- **Invoker Nodes / Worker Nodes**

Formatted: Greek

Αποτελούν τους φυσικούς ή εικονικούς κόμβους εκτέλεσης.

Ρόλος:

- Εκτελούν τη συνάρτηση όταν ο Scheduler του Control Plane τους αναθέσει ένα invocation.
- Διαχειρίζονται τα sandboxed περιβάλλοντα (containers / microVMs).
- Επικοινωνούν με το Control Plane για reporting (status, load, metrics).

Συνοπτικά:
είναι εκεί όπου τρέχει η συνάρτηση.

- **Sandboxed Runtimes (Containers / microVMs / Firecracker)**

  Είναι απομονωμένα περιβάλλοντα εκτέλεσης, σχεδιασμένα ώστε ο κώδικας κάθε χρήστη να τρέχει με ασφάλεια.

  Ρόλος:

  - Παρέχουν απομόνωση (isolation) και ασφάλεια.
  - Περιλαμβάνουν το απαραίτητο runtime (Node.js, Python, Java κ.λπ.).
  - Μπορούν να δημιουργηθούν ή να επαναχρησιμοποιηθούν για πιο γρήγορες εκτελέσεις (warm starts).

  Συνοπτικά:
  είναι το “sandbox” όπου πραγματικά εκτελείται ο κώδικας.

- **Runtime Manager**

  Ο Runtime Manager ελέγχει το περιβάλλον όπου θα εκτελεστεί η συνάρτηση.

  Ρόλος:

  - Φορτώνει το κατάλληλο runtime ανάλογα με τη γλώσσα.
  - Διαχειρίζεται initialization code, βιβλιοθήκες και dependencies.
  - Παρακολουθεί cold starts / warm starts και βελτιστοποιεί χρόνους εκκίνησης.

Συνοπτικά:
επιλέγει και ενεργοποιεί το "σωστό περιβάλλον εκτέλεσης".

- **Resource Allocator**

  Ο μηχανισμός που καθορίζει πόση μνήμη, CPU και δίκτυο θα διατεθούν σε μια συνάρτηση.

  Ρόλος:

  - Αναθέτει δυναμικά πόρους με βάση τις ανάγκες της συνάρτησης.
  - Εξασφαλίζει ότι καμία συνάρτηση δεν υπερκαταναλώνει πόρους εις βάρος άλλων.
  - Παρακολουθεί την κατανάλωση και ενημερώνει το Control Plane.

  Συνοπτικά:
  ρυθμίζει πόσους πόρους παίρνει το sandbox κατά την εκτέλεση.

Ρόλος:

1. Δημιουργεί ή επαναχρησιμοποιεί ένα sandbox για τη συνάρτηση.
2. Φορτώνει τον κώδικα (δηλ. διατάσσει την συνάρτηση).
3. Εκτελεί τη συνάρτηση (με βάση εισερχόμενο γεγονός).
4. Επιστρέφει το αποτέλεσμα.
5. Ελευθερώνει πόρους, **μόνο όταν δεν υπάρχουν ενεργές κλήσεις**.

---

4. Event System (Event Sources / Event Router / Event Bus)

Το Event System είναι ο μηχανισμός που **ενεργοποιεί** (trigger) τις συναρτήσεις.

• **Event Sources (Πηγές Γεγονότων)**: Οι πηγές των γεγονότων περιλαμβάνουν τις εξής:

- HTTP requests
- αλλαγές σε cloud storage (αποθηκευτικό χώρο νέφους)
- Pub/Sub topics (θέματα σε διαμορφώσεις τύπου publish-subscribe – οπότε ένα γεγονός που αφορά ένα θέμα έχει παραχθεί από έναν publisher)

Formatted: Greek

- message queues (ουρές μηνυμάτων)
- cron schedulers

• **Event Router**: Αποφασίζει **ποια συνάρτηση** αντιστοιχεί στο συμβάν.

• **Event Bus**: Μεταφέρει τα γεγονότα από τις πηγές προς τον Router και από εκεί στο Control Plane.

Το Event System **δεν εκτελεί τη συνάρτηση**, αλλά "πυροδοτεί" την διαδικασία εκτέλεσής της.
Η εκτέλεση γίνεται στο Data Plane (Worker Nodes), όχι στο Event System.

---

5. Monitoring, Billing & Security Layer

Περιλαμβάνει τα εξής συστατικά:

• **Monitoring Service**: Υπηρεσία παρακολούθησης (π.χ. AWS CloudWatch, Google Cloud Monitoring), η οποία είναι υπεύθυνη για:

- συλλογή logs (καταχωρήσεων), metrics (μετρικών) και traces
- ανίχνευση σφαλμάτων
- την δημιουργία triggers για alerts (συνεγαρμούς)

• **Billing Engine**: Υπολογίζει χρέωση βάσει:

- διάρκειας εκτέλεσης
- μνήμης που δεσμεύτηκε
- αριθμού εκτελέσεων (invocations)
- τον όγκο εξερχόμενης δικτυακής κίνησης

• **Security Layer**: είναι υπεύθυνο για:

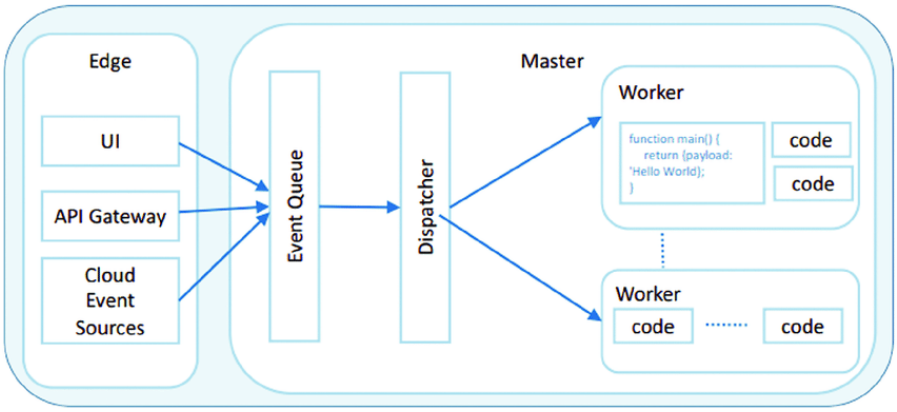
- διαχείριση IAM ρόλων
- εφαρμογή πολιτικών (policies) για συναρτήσεις και triggers
- έλεγχος αδειών (permissions) για πρόσβαση σε storage/queues

---

Formatted: Greek

Γενική Ροή Εκτέλεσης

1. Ο χρήστης υποβάλλει τον κώδικα μέσω CLI / API / Web Console.
2. Το Control Plane αποθηκεύει metadata & triggers.
3. Όταν συμβεί ένα event:
    - Το Event Source το παράγει
    - Το Event Bus το μεταφέρει
    - Ο Event Router επιλέγει συνάρτηση
4. Ο Scheduler επιλέγει Worker Node (για την διάταξη της συνάρτησης).
5. Ο Worker δημιουργεί ή επαναχρησιμοποιεί sandbox και εκτελεί τη συνάρτηση.
6. Το Monitoring καταγράφει τις μετρικές (metrics).
7. Το Billing Engine υπολογίζει το κόστος.
8. Το sandbox αποδεσμεύεται **μόνο όταν η συνάρτηση φτάσει σε μηδενική κίνηση** (scale-to-zero).



Εικόνα 1 – Αρχιτεκτονική Serverless Υπολογιστικής

Η συγκεκριμένη αρχιτεκτονική ακολουθείται με παραλλαγές από όλους τους μεγάλους παρόχους (στις serverless πλατφόρμες τους), με μικρές διαφοροποιήσεις ως προς το τρόπο

Formatted: Greek

Commented [KK48]: Οι λεζάντες θα πρεπει να μπαινουν αμέσως από κάτω από τις εικόνες.

Προτεινω για λόγους απλότητας να χρησιμοποιήσετε το προτυπο λεζάντας: Εικονα Χ - ....

Επίσης, ας εχετε απλή αρίθμηση και όχι βάση του εκαστοτε κεφαλαίου στο οποίο εντοπίζεται η εικόνα. Ούτως ή άλλως θα υπάρχει ευρετήριο/πίνακας εικόνων στην αρχή της αναφοράς ...

απομόνωσης των συναρτήσεων (isolation) (π.χ. AWS χρησιμοποιεί Firecracker microVMs[26], ενώ το GCP βασίζεται σε δοχεία).

Formatted: Greek

Commented [KK49]: Καλό είναι να έχουμε κάποιο συνδεσμο σε μορφή footnote ή παραπομπή προς αυτή την τεχνολογια.

2.6 Διαχείριση Πόρων στο Serverless Περιβάλλον

Η αποτελεσματική διαχείριση πόρων (resource management) αποτελεί κρίσιμο ζήτημα για την αποδοτική λειτουργία ενός serverless περιβάλλοντος. Περιλαμβάνει τις διαδικασίες προμήθειας/τροφοδοσίας (provisioning), χρονοπρογραμματισμού (scheduling), κλιμάκωσης (scaling) και παρακολούθησης (monitoring) των πόρων. Οι διαδικασίες αυτές διασφαλίζουν την τήρηση των Συμφωνιών Επιπέδου Υπηρεσίας (SLAs), τη βέλτιστη χρήση πόρων και τη σταθερή απόδοση.

2.6.1 Provisioning Πόρων

Το provisioning πόρων αποτελεί κρίσιμο μηχανισμό στις serverless πλατφόρμες, καθώς διασφαλίζει ότι κάθε συνάρτηση λαμβάνει τους απαραίτητους υπολογιστικούς πόρους (CPU, μνήμη RAM, I/O bandwidth) τη στιγμή που χρειάζεται να εκτελεστεί. Ωστόσο, η διαδικασία δεν αφορά μόνο την απλή δέσμευση πόρων· στην πράξη απαιτείται βέλτιστη διαχείριση των φυσικών πόρων και προετοιμασία κατάλληλων εκτελεστικών περιβαλλόντων πάνω σε αυτούς. Ένας φυσικός πόρος (π.χ. ένας φυσικός διακομιστής ή κόμβος στο cloud) μπορεί να "τεμαχιστεί" σε πολλούς επιμέρους εκτελεστικούς υπο-πόρους (containers, microVMs).

Το provisioning επομένως στο serverless περιβάλλον αφορά δύο επίπεδα:

(α) *Διαχείριση φυσικών πόρων (resource allocation)*: Ο πάροχος πρέπει να εξασφαλίσει ότι ο φυσικός πόρος (node) αξιοποιείται σωστά ώστε να αποφευχθούν τα παρακάτω 2 προβλήματα:

- *Υπερ-χρήση (over-utilization)*: Προκαλεί συμφόρηση, υψηλό latency και σφάλματα (failures), επειδή οι διαθέσιμες CPU/μνήμη δεν επαρκούν για όλα τα δοχεία.
- *Υπο-χρήση (under-utilization)*: Σημαίνει ότι ο πάροχος πληρώνει για υλικό (hardware) που μένει ανεκμετάλλευτο, οδηγώντας σε υψηλότερο κόστος λειτουργίας.

[26] https://firecracker-microvm.github.io/

Formatted: Greek

Ο στόχος είναι η βέλτιστη ισορροπία: κάθε κόμβος να εκτελεί όσο το δυνατόν περισσότερα δοχεία, χωρίς να επιβαρύνεται υπερβολικά.

(β) *Δέσμευση υπο-πόρων για εκτελέσιμα περιβάλλοντα (function environments)*: Πάνω στους φυσικούς πόρους δημιουργούνται sandbox περιβάλλοντα όπως:

- containers (π.χ. Docker-like)
- microVMs (π.χ. AWS Firecracker)

Τα περιβάλλοντα αυτά "φιλοξενούν" τις serverless συναρτήσεις.

Όταν ο πάροχος γνωρίζει —μέσω προγνωστικών αλγορίθμων— ότι μια συνάρτηση αναμένεται να κληθεί σύντομα, μπορεί:

- να δεσμεύσει εκ των προτέρων τους απαραίτητους υπο-πόρους (CPU slices, μνήμη),
- να προδημιουργήσει ένα "ζεστό" περιβάλλον (warm container),
- να φορτώσει εκ των προτέρων το runtime της συνάρτησης.

Αυτό μειώνει δραστικά τις κρύες εκκινήσεις (cold starts) των συναρτήσεων δηλαδή την αρχική καθυστέρηση που απαιτείται για να δημιουργηθεί από το μηδέν το εκτελέσιμο περιβάλλον μιας συνάρτησης. Οι επόμενες κλήσεις εκτελούνται χωρίς αυτή την καθυστέρηση, αφού το περιβάλλον παραμένει ήδη "ζεστό" (warm).

Commented [KK50]: Καλό είναι να εξηγήσετε τι είναι αυτός ο όρος. Πχ. αντιστοιχει στην καθυστέρηση για την αρχική κλήση της συνάρτησης. Οι επόμενες κλήσεις γινονται σε έτοιμο πλέον περιβάλλον οπότε αυτή η καθυστέρηση δεν θα υπάρχει .

Predictive Algorithms (Προγνωστικοί Αλγόριθμοι)

Οι πάροχοι χρησιμοποιούν ιστορικά δεδομένα κλήσεων για να ανιχνεύσουν μοτίβα ζήτησης:

- ημερήσιοι/εβδομαδιαίοι κύκλοι λειτουργίας
- αυξήσεις κίνησης σε συγκεκριμένες ώρες
- traffic bursts (αιφνίδια αιτήματα/κίνηση)

Με βάση αυτά, ο πάροχος μπορεί:

- να δημιουργήσει εκ των προτέρων επιπλέον περιβάλλοντα,

- να αυξήσει ή να μειώσει το πλήθος των ζεστών δοχείων (warm containers), (warm containers — **εκτελέσιμα περιβάλλοντα που έχουν ήδη αρχικοποιηθεί και παραμένουν ενεργά, ώστε οι συναρτήσεις να εκτελούνται άμεσα χωρίς καθυστέρηση "κρύας εκκίνησης"**),
- να μεταφέρει δυναμικά φορτίο σε άλλους κόμβους ώστε να αποφευχθούν bottlenecks.

Συνοψίζοντας, η τροφοδοσία πόρων στην serverless υπολογιστική δεν είναι απλή δέσμευση VM — είναι ένας συνδυασμός:

1. Ορθής χρήσης φυσικών πόρων ώστε να αποφεύγεται υπο-χρησιμοποίηση ή υπερ-χρησιμοποίηση.
2. Δέσμευσης και προετοιμασίας sandbox περιβαλλόντων, που φιλοξενούν τις συναρτήσεις.
3. Έξυπνης πρόβλεψης της ζήτησης, ώστε να διατηρούνται warm containers και να μειώνονται cold starts.

Έτσι, οι serverless πλατφόρμες μπορούν να ανταποκρίνονται **σε αιφνίδια traffic spikes χωρίς καθυστερήσεις και με βέλτιστο κόστος.**

2.6.2 Χρονοπρογραμματισμός (Scheduling)

Ο χρονοπρογραμματισμός (scheduling) καθορίζει **πού** και **πότε** θα εκτελεστεί μια serverless συνάρτηση. Αποτελεί κρίσιμο στάδιο της serverless αλυσίδας, καθώς επηρεάζει:

- τον χρόνο ενεργοποίησης (*activation latency*),
- το κόστος,
- την ενεργειακή απόδοση,
- και τη συνολική ποιότητα υπηρεσίας (QoS).

Ο όρος *activation latency* αναφέρεται στον χρόνο που μεσολαβεί από την υποβολή του αιτήματος μέχρι τη στιγμή που η συνάρτηση αρχίζει να εκτελείται. Η καθυστέρηση αυτή σχετίζεται άμεσα με το φαινόμενο του **cold start**, δηλαδή τη χρονική επιβάρυνση που απαιτείται όταν το εκτελέσιμο περιβάλλον (container ή microVM) πρέπει να δημιουργηθεί από την αρχή. Όταν υπάρχουν διαθέσιμα "ζεστά" περιβάλλοντα (warm containers), το activation latency μειώνεται σημαντικά.

Formatted: Greek

Commented [KK51]: Κι αυτό θα πρέπει να εξηγηθεί.

Commented [KK52]: Δεν είναι σωστή η μετάφραση του όρου latency κατά τη γνώμη μου. Ο προτυπος ελληνικός όρος είναι καθυστέρηση.

Ίσως αν το λέγατε activation latency εντός της παρένθεσης;
Κι ο ελληνικός όρος να γινει καθυστερηση ενεργοποίησης;
Επίσης, ποια η σχέση με cold start;

Formatted: Greek

Ο scheduler επιλέγει τον κατάλληλο **Invoker Node** για την εκτέλεση μιας συνάρτησης. Η επιλογή γίνεται με βάση αλγορίθμους κατανομής φορτίου, στόχος των οποίων είναι η **ισορροπία μεταξύ QoS και αποδοτικότητας πόρων**.

Σχέση Scheduling και Warm Environments

Ο Scheduler συνεργάζεται στενά με το σύστημα provisioning του Data Plane, το οποίο είναι υπεύθυνο για τη δημιουργία και διαχείριση των **warm containers**. Στο προηγούμενο αρχιτεκτονικό μοντέλο, αυτός ο ρόλος ανήκει στο **Runtime Manager και τον Resource Allocator** των Invoker/Worker Nodes.

Ρόλος των συστατικών

- **Resource Allocator**: δεσμεύει τους απαιτούμενους υπολογιστικούς πόρους (CPU slices, μνήμη) για τη δημιουργία ή διατήρηση ενός sandbox.
- **Runtime Manager**: προετοιμάζει ή διατηρεί ενεργά τα warm environments, φορτώνει runtimes και διαχειρίζεται την κατάσταση των containers/microVMs.

Άρα, το Data Plane —μέσω των επιμέρους αυτών συστατικών— είναι υπεύθυνο για την ύπαρξη warm containers που μπορούν να χρησιμοποιηθούν άμεσα.

Επίδραση στα cold starts

- **Αν υπάρχει διαθέσιμο warm environment:**
  Ο scheduler το επιλέγει άμεσα, με αποτέλεσμα **πολύ μικρή καθυστέρηση ενεργοποίησης** (activation latency) και ουσιαστικά **μηδενικό cold start**.
- **Αν δεν υπάρχει διαθέσιμο warm environment:**
  Ο scheduler ζητά από το Data Plane να δημιουργήσει νέο sandbox (container/microVM).
  Η διαδικασία αυτή **ισοδυναμεί με cold start**, επειδή απαιτείται:
  - εκκίνηση sandbox,
  - φόρτωση runtime,
  - ενδεχομένως και φόρτωση κώδικα.

Συνδυαστικός ρόλος

Το scheduling και το provisioning λειτουργούν ως **ενιαίο, συντονισμένο υποσύστημα του backend infrastructure της πλατφόρμας**, με κοινό στόχο:

- τη μείωση των cold starts,
- τη μείωση της καθυστέρησης ενεργοποίησης (activation latency),
- και την αύξηση του συνολικού **ρυθμού απόδοσης (system throughput)** του συστήματος serverless.

Με άλλα λόγια, η απόδοση **όλης της serverless πλατφόρμας** εξαρτάται από το πόσο αποδοτικά συνεργάζονται ο Scheduler και το provisioning του Data Plane στη διαχείριση και αξιοποίηση των warm environments.

Το βασικό trade-off: QoS vs Resource Efficiency

Ο Scheduler ισορροπεί ανάμεσα σε δύο αντίθετους στόχους:

1. Υψηλό QoS

- μικρό latency
- γρήγορη ανταπόκριση
- άμεση διάθεση warm containers → απαιτεί περισσότερους ενεργούς πόρους

2. Αποδοτικότητα Πόρων

- περιορισμένο (λειτουργικό) κόστος
- λιγότερα ενεργά δοχεία
- εξοικονόμηση ενέργειας

Σημειώνεται πως όσο λιγότερα warm containers και ενεργοί κόμβοι υπάρχουν, τόσο αυξάνεται η πιθανότητα **cold starts** σε επόμενες κλήσεις. Αυτό δικαιολογεί το προαναφερόμενο trade-off για το οποίο θα πρέπει να επιληφθεί ο Scheduler.

Οι σύγχρονες serverless πλατφόρμες εφαρμόζουν πιο προχωρημένες τεχνικές:

1. **Κόμβοι χαμηλής κατανάλωσης (low-power mode)**
   Κάποιοι worker nodes μπορούν να μειώσουν συχνότητα CPU, να περιορίσουν I/O activity ή να μπουν σε sleep/standby κατάσταση όταν δεν υπάρχουν ενεργές συναρτήσεις.

Formatted: Greek

Commented [KK53]: Οκ αλλά αυτό πως επιτυγχάνεται; Λιγότερος φορτος σημαινει λιγότερη ενέργεια; Ή κάποιοι κόμβοι ειτε περνάνε σε ποιο οικονομικό (αποδοτικό ως προς την ενέργεια) mode ή σβήνονται;

2. **Απενεργοποίηση κόμβων (node deactivation / scale-to-zero)** Σε χαμηλή ζήτηση, ο πάροχος "σβήνει" πλήρως ορισμένους κόμβους. Αυτό μειώνει δραστικά την κατανάλωση ενέργειας.

3. **Συμπίεση φορτίου (workload consolidation)** Οι λειτουργίες μεταφέρονται σε λιγότερους κόμβους για να εκμεταλλευτούν πλήρως τους διαθέσιμους πόρους, επιτρέποντας στους υπόλοιπους να απενεργοποιηθούν.

Ο Scheduler πρέπει να αποφασίζει πότε αξίζει:

- να **δημιουργήσει** νέο περιβάλλον (για QoS),
- ή να **συμπτυχθεί** σε λιγότερους κόμβους (για αποδοτικότητα).

Αλγόριθμοι Scheduling: Εδώ αναφερόμαστε σε συγκεκριμένους αλγορίθμους χρονοπρογραμματισμού που μπορεί να εφαρμόζονται από τον Scheduler.

(α) Round-Robin: Αναθέτει εκτελέσεις σε κόμβους με κυκλική σειρά. Ενώ προσφέρει ισορροπημένη κατανομή, αγνοεί το πραγματικό φορτίο κόμβων και δεν λαμβάνει υπόψη warm environments.

(β) Bin Packing – Ελαχιστοποίηση Κατακερματισμού Πόρων: Ο "κατακερματισμός πόρων" (resource fragmentation) συμβαίνει όταν:

- οι κόμβοι έχουν **μικρά, διάσπαρτα, αχρησιμοποίητα τμήματα RAM/CPU**,
- τα οποία δεν επαρκούν για να δεχτούν ένα νέο δοχείο.

Παράδειγμα:
Ένας κόμβος έχει 1GB RAM διαθέσιμο, αλλά μια συνάρτηση χρειάζεται 1.5GB → δεν μπορεί να εκτελεστεί εκεί, αν και υπάρχει ελεύθερο 1GB.

Ο Bin Packing αλγόριθμος προσπαθεί:

- να "πακετάρει" δοχεία σε όσο το δυνατόν λιγότερους κόμβους,
- ώστε να μένουν μεγάλα συνεχόμενα blocks πόρων διαθέσιμα.

Βασικά οφέλη:

- περιορίζει τα cold starts

Formatted: Greek

Commented [KK54]: Αν θυμηθειτε το προηγούμενο σχόλιό μου, η οριζόντια γραμμή δεν θα έπρεπε να ειχε τοποθετηθει εδώ. Είναι λάθος η θέση της.

Αλλά ας τις αφαιρέσετε τελείως.

Commented [KK55]: Προηγουμένως ειχατε αναφέρει και μειονεκτήματα στον round-robin. Ο τρεχων αλγοριθμος έχει κάποια;

- μειώνει ανάγκη για επιπλέον nodes
- επιτρέπει shutdown κόμβων για εξοικονόμηση ενέργειας

Μειονεκτήματα:

- απαιτεί υψηλότερο υπολογιστικό κόστος (πιο «βαριές» αποφάσεις)
- μπορεί να αυξήσει το activation latency σε υψηλό φορτίο
- δεν ανταποκρίνεται καλά σε ξαφνικά traffic bursts
- είναι πιο περίπλοκος στην υλοποίηση και συντήρηση

(γ) Load-Aware Scheduling: Λαμβάνει υπόψη τον **τρέχοντα φόρτο εργασίας** κάθε κόμβου:

- χρήση (utilization) CPU
- memory usage (χρήση κύριας μνήμης)
- active containers (ενεργά δοχεία)
- queue backlog (ο αριθμός των pending αιτημάτων που περιμένουν να εκτελεστούν σε έναν κόμβο)
- latency trends (η τάση μεταβολής της **καθυστέρησης ενεργοποίησης (activation latency)** με την πάροδο του χρόνου, π.χ. αν αυξάνεται ή μειώνεται λόγω φόρτου)

Υπάρχουν δύο στρατηγικές:

1. Load-minimizing:

Επιλογή κόμβου **με το μικρότερο τρέχον φορτίο** → ιδανικό για latency-sensitive (ευαίσθητες ως προς την καθυστέρηση) συναρτήσεις.

2. Load-consolidating:

Επιλογή κόμβου **που μπορεί να φορτωθεί περισσότερο**, ώστε:

- άλλοι κόμβοι να περάσουν σε κατάσταση χαμηλής κατανάλωσης,
- με μείωση στη συχνότητα του επεξεργαστή (CPU frequency scaling) ή πλήρη απενεργοποίηση αν είναι εφικτό.

Formatted: Greek

Commented [KK56]: Δεν είπατε τα θετικά & αρνητικά αυτού του scheduling

Commented [KK57]: Τι είναι αυτό; Ας το εξηγήσετε σε 1 προταση.

Commented [KK58]: Δειτε προηγούμενα σχολια

Formatted: Greek

Η επιλογή (μιας στρατηγικής εκ των δύο) εξαρτάται από τον βασικό στόχο της πλατφόρμας (QoS vs efficiency).

Πλεονεκτήματα (Load-Aware Scheduling):

- **Καλύτερο latency/QoS** όταν εφαρμόζεται load-minimizing, γιατί αποφεύγεται η δρομολόγηση σε "κορεσμένους" κόμβους.
- **Πιο σταθερή απόδοση** σε αιχμές, αφού αξιοποιεί πραγματικές μετρικές (CPU/memory/queue backlog) αντί για απλή round-robin.
- **Ενεργειακή αποδοτικότητα** με load-consolidating (συγκέντρωση φόρτου → περισσότεροι idle κόμβοι → δυνατότητα power saving).
- **Καλύτερη αξιοποίηση πόρων** (resource utilization) και μείωση σπατάλης υπολογιστικής ισχύος.

Μειονεκτήματα / Προκλήσεις:

- **Overhead παρακολούθησης μετρικών** (συλλογή/ενημέρωση CPU, memory, backlog, latency trends) που αυξάνει πολυπλοκότητα και κόστος ελέγχου.
- **Κίνδυνος "λανθασμένων" αποφάσεων λόγω καθυστερημένων ή θορυβωδών μετρήσεων**, ειδικά σε bursty workloads.
- **Load-minimizing:** μπορεί να οδηγήσει σε **κατακερματισμό φόρτου** σε πολλούς κόμβους → περισσότερα cold starts / περισσότερα ενεργά containers.
- **Load-consolidating:** μπορεί να αυξήσει **latency** ή να δημιουργήσει **hotspots** (συμφόρηση) αν η συγκέντρωση φόρτου γίνει υπερβολικά επιθετικά.
- **Trade-off QoS vs efficiency**: είναι δύσκολο να βρεθεί μία ρύθμιση που να είναι βέλτιστη για όλα τα είδη συναρτήσεων (latency-sensitive vs batch)

(δ) Machine Learning-Based Scheduling

Commented [KK59]: Κι εδώ δεν αναφέρατε τα θετικά & αρνητικά

Οι ML schedulers προβλέπουν το **μελλοντικό φορτίο**, όχι απλώς το τρέχον.

Χρησιμοποιούν:

- ιστορικά δεδομένα κλήσεων
- χρονικές σειρές αιχμών
- εποχικότητα (κίνησης) (π.χ. βραδινή αυξημένη κίνηση)

- μοτίβα για συγκεκριμένες συναρτήσεις

Αυτό επιτρέπει:

1. Πρόβλεψη φορτίου για όλο το σύστημα: όπως για παράδειγμα πόσοι κόμβοι θα χρειαστούν τις επόμενες ώρες.

2. Πρόβλεψη φορτίου για μια συγκεκριμένη συνάρτηση: το οποίο μπορεί να προσδιορίσει πόσα στιγμιότυπα (συνάρτησης) θα πρέπει να προ-δημιουργηθούν (warm containers).

Παράδειγμα:

Αν το μοντέλο προβλέψει ότι μια συνάρτηση που συνήθως εκτελείται 2 φορές/λεπτό θα ανέβει στις 50/λεπτό τα επόμενα 5 λεπτά:

- ο scheduler ζητά από το provisioning να δημιουργήσει **πολλαπλά warm instances**,
- ώστε να μην προκύψουν cold starts ή throttling(καθυστέρηση/απόρριψη κλήσεων όταν ξεπεραστεί ο μέγιστος επιτρεπόμενος ρυθμός εκτέλεσης)

Συνεπώς, ο Scheduler καλείται να αποφασίσει (χρησιμοποιώντας κατάλληλες στρατηγικές):

- Ποιον κόμβο να χρησιμοποιήσει;
- Πόσα environments (περιβάλλοντα) πρέπει να είναι warm;
- Πώς να ισορροπήσει QoS και κόστος;
- Πότε πρέπει να ενεργοποιήσει νέους κόμβους ή να απενεργοποιήσει υπάρχοντες;

Και όλα αυτά με βάση:

- τρέχον φορτίο
- προβλεπόμενο φορτίο
- διαθέσιμους πόρους
- τεχνικές κατανομής (Round-Robin, Bin Packing κ.λπ.)

Πλεονεκτήματα (Machine Learning-Based Scheduling):

- **Προληπτική κλιμάκωση (predictive scaling):** προετοιμάζει πόρους πριν την αιχμή, μειώνοντας **cold starts** και **throttling**.

Formatted: Greek

Commented [KK60]: Ας εξηγήσετε σε 1 προταση (κανονική ή σε παρενθέσεις) εδώ ή σε footnote τι σημαινει αυτό

- **Καλύτερο QoS σε bursty workloads:** όταν υπάρχουν απότομα spikes, ο scheduler δεν αντιδρά "εκ των υστέρων", αλλά προλαβαίνει το φόρτο.
- **Βελτιστοποίηση κόστους:** μπορεί να κρατά **λιγότερα warm instances** όταν προβλέπεται χαμηλή κίνηση και να αυξάνει μόνο όταν χρειάζεται.
- **Πιο "έξυπνη" πολιτική ανά συνάρτηση:** αξιοποιεί μοτίβα συγκεκριμένων functions (π.χ. βραδινές αιχμές) και όχι μόνο global load.

Μειονεκτήματα / Προκλήσεις:

- **Απαιτεί ιστορικά δεδομένα και σωστά features:** σε νέες συναρτήσεις (cold start του μοντέλου) οι προβλέψεις είναι συχνά αδύναμες.
- **Κίνδυνος σφαλμάτων πρόβλεψης:**
  - *Under-prediction* → καθυστερήσεις/cold starts/throttling,
  - *Over-prediction* → υπερδέσμευση πόρων και αυξημένο κόστος.
- **Υπολογιστικό/λειτουργικό overhead:** εκπαίδευση/ενημέρωση μοντέλων, αποθήκευση δεδομένων, monitoring, και πολυπλοκότητα συντήρησης.
- **Δυσκολία ερμηνείας (explainability):** δεν είναι πάντα ξεκάθαρο "γιατί" λήφθηκε μία απόφαση provisioning, άρα πιο δύσκολο debugging.
- **Drift / αλλαγή μοτίβων:** αν αλλάξει η συμπεριφορά (π.χ. νέο release, marketing campaign), το μοντέλο μπορεί να γίνει ανακριβές χωρίς retraining.

2.6.3 Κλιμάκωση Πόρων (Scaling)

Η κλιμάκωση (scaling) αφορά τη δυναμική προσαρμογή των διαθέσιμων υπολογιστικών πόρων ανάλογα με το τρέχον ή προβλεπόμενο φορτίο.
Στο serverless μοντέλο, η κλιμάκωση εφαρμόζεται **στο επίπεδο των στιγμιοτύπων εκτέλεσης (function instances)**, τα οποία φιλοξενούνται μέσα σε **sandboxed περιβάλλοντα**, συνήθως containers ή microVMs.

Υπάρχουν δύο μορφές κλιμάκωσης:

Formatted: Greek

(α) Οριζόντια Κλιμάκωση (Horizontal Scaling)

Η οριζόντια κλιμάκωση (scale-out / scale-in) δημιουργεί ή τερματίζει **επιπλέον στιγμιότυπα μιας συνάρτησης**, όταν το σύστημα εντοπίζει αυξημένη ή μειωμένη ζήτηση, αντίστοιχα.

Παραδείγματα ενεργειών:

- scale-out → δημιουργία πολλών παράλληλων στιγμιοτύπων (συνάρτησης) για να εξυπηρετήσουν αυξημένο ρυθμό κλήσεων
- scale-in → μείωση των στιγμιοτύπων (συνάρτησης) για εξοικονόμηση πόρων όταν η ζήτηση πέφτει

Η πρακτική αυτή συνδέεται άμεσα με το φαινόμενο:

Cold Start

Commented [KK61]: Αυτό κανονικά θα έπρεπε να εξηγηθει πιο νωρις. Ή διαφορετικά να υπήρχε παραπομπή προς το τρέχων σημείο από το προηγούμενο σημειο στο οποίο αναφέρεται ο τρέχων όρος.

Commented [AK62R61]: Τώρα έχει εξηγηθεί πιο νώρις

Αν μια συνάρτηση έχει κλιμακωθεί σε **0 στιγμιότυπα** (scale-to-zero) και απαιτηθεί να ξαναεκτελεστεί, ο πάροχος πρέπει να δημιουργήσει από την αρχή:

1. container ή microVM
2. να φορτώσει το runtime
3. να φορτώσει τον κώδικα της συνάρτησης

Αυτό προκαλεί καθυστέρηση (latency) στην πρώτη κλήση μετά το scale-in. Η μείωση των cold starts αποτελεί ενεργό πεδίο έρευνας, με τεχνικές όπως:

- warm containers
- pre-warming (δηλαδή προδημιουργία sandbox πριν από την πρώτη κλήση μιας συνάρτησης)
- predictive scaling (το οποίο έχει αναλυθεί στην προηγούμενη ενότητα, την 2.6.2, ως προγνωστική δημιουργία warm instances με βάση ιστορική ζήτηση)

Commented [KK63]: Αυτές αναφέρθηκαν προηγουμένως;

(β) Κάθετη Κλιμάκωση (Vertical Scaling)

Η κάθετη κλιμάκωση αναφέρεται στην **αύξηση ή μείωση των διαθέσιμων πόρων (CPU, RAM) (για ένα στιγμιότυπο συνάρτησης) μέσα στο ίδιο compute environment**.

Το "δοχείο (container)" στο serverless μοντέλο **είναι το περιβάλλον που φιλοξενεί την εκτέλεση ενός στιγμιοτύπου της συνάρτησης**.

Άρα η κάθετη κλιμάκωση σημαίνει αύξηση CPU / RAM που διατίθεται στο container και όχι αύξηση της χωρητικότητας του φυσικού μηχανήματος.

Παρότι λιγότερο διαδεδομένη (διότι το serverless βασίζεται περισσότερο στο scale-in/-out), η κάθετη κλιμάκωση χρησιμοποιείται:

- σε high-memory ή high-CPU functions
- όταν απαιτείται μεγαλύτερος υπολογιστικός χώρος εντός ενός μόνο στιγμιοτύπου. Ο χρήστης/διαχειριστής καθορίζει αυξημένους πόρους εκτέλεσης (π.χ. CPU ή RAM), τους οποίους στη συνέχεια εφαρμόζει η πλατφόρμα
-
- όταν ο **χρήστης αιτείται** το επιθυμητό εύρος πόρων (π.χ. 512MB, 1GB, 2GB RAM) όταν διαμορφώνει τη συνάρτηση.

Σε κάθε μία από τις παραπάνω περιπτώσεις, **το σύστημα (ο πάροχος)** εφαρμόζει την κατανομή, ελέγχοντας αν υπάρχει διαθέσιμο capacity και προσαρμόζοντας αναλόγως το περιβάλλον εκτέλεσης.

Ωστόσο ενέχει κινδύνους:

- **overprovisioning**: δέσμευση υπερβολικών πόρων για λίγη δουλειά
- **resource contention**: ανταγωνισμός πόρων μεταξύ δοχείων στον ίδιο κόμβο

Στόχοι της διαδικασίας Scaling

Ανεξάρτητα από το είδος της κλιμάκωσης, ο συνολικός στόχος είναι:

1. **σταθερός χρόνος απόκρισης** ακόμα και σε αιχμές
2. **αποφυγή σπατάλης πόρων**
3. **εξισορρόπηση cold starts και αποδοτικότητας**
4. **βελτιστοποίηση κόστους "pay-per-use"**

Οι στόχοι αυτοί δεν επιτυγχάνονται από ένα μόνο υποσύστημα· αποτελούν **συνδυαστική λειτουργία** των επιπέδων της πρότυπης serverless αρχιτεκτονικής:

---

1. Control Plane – Scheduling & Policy Decisions

Formatted: Greek

Commented [KK64]: Αυτό ποιος το αποφασιζει; Το συστημα ή το αιτειται ο χρηστης και το εφαρμοζει το συστημα;

Commented [KK65R64]: Το σχόλιο τελικά απαντάται; Με βάση την προταση μετά την λίστα των bullets; Και ποια bullets ακριβώς καλύπτει σε μια τέτοια περιπτωση;

Commented [AK66R64]: ο χρήστης το καθορίζει έμμεσα κατά τη διαμόρφωση της συνάρτησης), ενώ ο πάροχος/σύστημα το εφαρμόζει κατά την εκτέλεση

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Tab stops: Not at 1,27 cm

Commented [KK67]: Σε κάθε περίπτωση; Ή μιλάτε για αυτην στο τελευταιο bullet;

Commented [KK68]: Αυτός ο στοχος υλοποιείται από ποιο ή ποια συστατικά της πρότυπης αρχιτεκτονικής; Πχ. ένα συστατικό αιταται παραπάνω στιγμιότυπα και ένα άλλο εφαρμόζει με τον καλύτερο δυνατό τροπο το αίτημα αυτό; Το ρωτώ διότι η αποφυγή σπατάλης πόρων μοιάζει να είναι κάτι για το οποιο φροντιζει ο scheduler, σωστά;

Εκτος και αν εδώ (ειδικά για την αποφυγή σπατάλης πόρων που αναφερεται ως υπο-στόχος) εννοούμε τον σωστο αριθμό στιγμιοτύπων που θα πρέπει να έχει μια συνάρτηση για να αντιμετωπίσει τον τρέχων φορτο εργασιας.

Formatted: Greek

Αποφεύγει σπατάλη πόρων και διασφαλίζει σωστή κατανομή.

- Ο **Scheduler** αποφασίζει πόσα instances χρειάζεται μια συνάρτηση (scale-out / scale-in).
- Το **Provisioning subsystem** αποφασίζει αν θα δημιουργήσει ή θα καταστρέψει warm environments.
- Το **Metadata Store** διατηρεί τις παραμέτρους scaling (timeouts, memory limits, concurrency settings).
- Το **API Gateway** εφαρμόζει throttling ώστε να προστατεύει από υπερβολικά αιτήματα.

Ρόλος: αποφασίζει πόσα environments *πρέπει* να υπάρχουν ανά πάσα στιγμή.

---

2. Data Plane – Worker Nodes / Invokers

Εκτελεί τις αποφάσεις του Control Plane.

- Δημιουργεί ή καταστρέφει containers.
- Φορτώνει ή αποδεσμεύει μνήμη / CPU.
- Εκτελεί τα instances που ζητά ο scheduler.
- Εφαρμόζει το πραγματικό scaling σε επίπεδο container/microVM.

Ρόλος: κάνει την πραγματική κλιμάκωση (δημιουργεί/τερματίζει instances).

---

3. Predictive Scaling Algorithms (στο Control Plane)

Ελαχιστοποιούν cold starts και βελτιώνουν χρόνο απόκρισης.

- Προβλέπουν φόρτο εργασίας.
- Προ-δημιουργούν warm containers.
- Μειώνουν καθυστερήσεις (activation latency).

Ρόλος: προλαμβάνει spikes πριν συμβούν.

Formatted: Greek

Στην πράξη, οι πάροχοι εφαρμόζουν **συνδυασμό στρατηγικών scaling**, όπως reactive scaling (με βάση το τρέχον φορτίο), predictive scaling (με βάση προβλεπόμενο φορτίο), consolidation (για περιορισμό ενεργών κόμβων) και threshold-based policies. Οι στρατηγικές αυτές συνεργάζονται με τους μηχανισμούς scheduling και provisioning για να επιτύχουν τους στόχους της κλιμάκωσης.

2.6.4 Παρακολούθηση Πόρων (Monitoring)

Η παρακολούθηση (monitoring) αποτελεί κρίσιμο στοιχείο για τη διασφάλιση της αξιοπιστίας και της συμμόρφωσης με SLA (Service Level Agreements) σε serverless περιβάλλοντα. Σε αντίθεση με τις παραδοσιακές αρχιτεκτονικές, όπου ο προγραμματιστής έχει συνεχή πρόσβαση σε διακομιστές και σταθερά στιγμιότυπα (συναρτήσεων/υπηρεσιών), στο serverless μοντέλο οι εκτελέσεις είναι σύντομες, απομονωμένες και ενδέχεται να εκτελούνται σε διαφορετικούς κόμβους κάθε φορά. Συνεπώς, απαιτείται ένα πιο εξελιγμένο οικοσύστημα παρακολούθησης, βασισμένο σε μετρικές που καταγράφουν τη συμπεριφορά των συναρτήσεων και του υποκείμενου περιβάλλοντος.

Μετρικές που συλλέγονται και η σημασία τους

Οι πάροχοι προσφέρουν APIs και dashboards για την παρακολούθηση κρίσιμων μετρικών εκτέλεσης. Ενδεικτικά:

• **Execution Time (Χρόνος Εκτέλεσης)**: Ο συνολικός χρόνος εκτέλεσης μιας συνάρτησης. Αποτελεί βασικό δείκτη απόδοσης και επηρεάζει άμεσα το κόστος (pay-per-invocation μοντέλο).

• **Memory Consumption (Κατανάλωση Μνήμης)**: Η μέγιστη μνήμη που καταναλώνεται ανά invocation. Υψηλή τιμή μπορεί να οδηγήσει σε throttling ή αποτυχία εκτέλεσης.

• **Latency (Χρόνος Απόκρισης)**: Το latency αναφέρεται στο end-to-end χρονικό διάστημα από την ενεργοποίηση της συνάρτησης έως την ολοκλήρωση της απάντησης. Περιλαμβάνει:

- χρόνο ενεργοποίησης (συμπεριλαμβανομένων πιθανών cold starts),
- καθυστερήσεις δικτύου,
- εσωτερικό overhead της πλατφόρμας.

Ως μετρική είναι ιδιαίτερα σημαντική για SLA και εφαρμογές πραγματικού χρόνου (real-time).

• **Error Rate (Ποσοστό Σφαλμάτων)**: Αντιπροσωπεύει το ποσοστό αποτυχημένων κλήσεων και αποτελεί βασικό δείκτη αξιοπιστίας.

• **Throughput (Ρυθμός Απόδοσης)**: Ο αριθμός εκτελέσεων ανά δευτερόλεπτο. Συνδέεται στενά με την ικανότητα της πλατφόρμας να κλιμακώνει τις συναρτήσεις.

Εργαλεία Παρακολούθησης από Παρόχους

Οι κύριες serverless πλατφόρμες διαθέτουν ενσωματωμένα συστήματα/υπηρεσίες παρακολούθησης που περιλαμβάνουν:

- **AWS CloudWatch** (AWS Lambda)
- **Azure Monitor** (Azure Functions)
- **Google Cloud Monitoring** (Google Cloud Functions)

Οι υπηρεσίες αυτές παρέχουν:

- αυτόματη συλλογή καταχωρήσεων (logs),
- real-time dashboards,
- μετρικές εκτέλεσης,
- δυνατότητες alerting,
- tracing για εντοπισμό bottlenecks.

Εργαλεία Τρίτων (Prometheus, Datadog, New Relic κ.ά.)

Εκτός από τις προαναφερόμενες υπηρεσίες που δεσμεύονται σε συγκεκριμένο πάροχο, υπάρχουν και εργαλεία παρακολούθησης τρίτων, τα οποία δεν λειτουργούν αυτόματα· πρέπει να εγκατασταθούν και να διαμορφωθούν από τον ίδιο τον πελάτη (δηλαδή τον πάροχο/δημιουργό μιας συνάρτησης).

Πώς λειτουργούν στην πράξη:

1. Ο πάροχος (AWS/GCP/Azure) εκθέτει τις μετρικές μέσω monitoring APIs.
2. Το εργαλείο τρίτων (π.χ. Prometheus, Datadog) αντλεί (fetch/scrape) τις μετρικές από αυτά τα APIs.
3. Ο χρήστης μπορεί να δημιουργήσει custom dashboards, alerts και triggers.

Formatted: Greek

Formatted: Greek

Βασικό πλεονέκτημα εργαλείων τρίτων:

Μπορούν να παρακολουθούν και **custom application-level metrics** (δηλαδή μετρικές σε επίπεδο εφαρμογής), π.χ.:

- μετρικές (επιχειρησιακών) ροών εργασίας (workflows),
- latency σε όλα τα επίπεδα (API → Function → Storage),

Επιπλέον, τα εργαλεία αυτά υποστηρίζουν και tracing σε κατανεμημένα συστήματα. Το tracing δεν αποτελεί μετρική· είναι ξεχωριστός μηχανισμός που επιτρέπει την παρακολούθηση της ακολουθίας εκτέλεσης (execution flow) μέσα από πολλαπλές υπηρεσίες, συμπληρώνοντας τις μετρικές και προσφέροντας πιο ολοκληρωμένη εικόνα για την πραγματική συμπεριφορά του συστήματος.

Ορατότητα σε Serverless Περιβάλλοντα

Η παρακολούθηση σε serverless συστήματα είναι πιο απαιτητική για τους εξής λόγους:

- **Δεν υπάρχει σταθερός διακομιστής** για συνεχή παρακολούθηση.
- **Κάθε κλήση είναι απομονωμένη**, άρα τα δεδομένα πρέπει να συσχετίζονται μέσω tracing.
- **Η κλιμάκωση είναι δυναμική**, και ο αριθμός των στιγμιοτύπων μεταβάλλεται συνεχώς.
- **Ο χρόνος απόκρισης (latency)** επηρεάζεται και από cold starts.

Η παρατηρησιμότητα/ορατότητα (observability) σε serverless περιλαμβάνει:

- συλλογή **μετρικών** (metrics),
- καταγραφή **logs**,
- παρακολούθηση **traces**,
- συσχέτιση γεγονότων σε distributed περιβάλλον.

Σύγκρισης Παρακολούθησης από Πάροχο και Από Πελάτη

Monitoring από τον Πάροχο

- πλήρως διαχειριζόμενο,

- ιδανικό για standard (τυπικές) μετρικές,
- άμεση ενσωμάτωση με triggers και alarms.

Monitoring από τον Πελάτη

- γίνεται μέσω εργαλείων όπως Prometheus ή Datadog,
- υποστηρίζει custom, domain-specific / application-level μετρικές,
- επιτρέπει ενιαία παρακολούθηση (unified monitoring) σε multi-cloud περιβάλλον.

Το τελευταίο πλεονέκτημα είναι ιδιαίτερα σημαντικό για εφαρμογές που εκτελούνται σε πολλαπλές serverless πλατφόρμες.

2.7 Σύγκριση Πλατφορμών και Πλαισίων Serverless Υπολογιστικής

Οι serverless πλατφόρμες παρουσιάζουν σημαντικές διαφοροποιήσεις ως προς τα runtime περιβάλλοντα, τα triggers, τα μοντέλα κλιμάκωσης και τιμολόγησης, καθώς και τις δυνατότητες παρακολούθησης και ενορχήστρωσης. Για να δομηθεί μια σαφής και ομοιογενής σύγκριση, παρατίθενται πρώτα τα βασικά **κριτήρια σύγκρισης** και στη συνέχεια δύο πίνακες σύγκρισης/αξιολόγησης για:

(1) εμπορικές πλατφόρμες (AWS, Azure, Google) &
(2) πλαίσια ανοικτού κώδικα, αντίστοιχα.

2.7.1 Κριτήρια Σύγκρισης

Πριν παρουσιαστούν οι πίνακες, ορίζονται συνοπτικά τα κριτήρια αξιολόγησης. Τα πρώτα 6 αφορούν τις serverless πλατφόρμες, το 7ο είναι κοινό και τα 2 τελευταία αφορούν τα serverless πλαίσια:

1. **Υποστηριζόμενες Γλώσσες Προγραμματισμού**
   Περιγράφουν την ευκολία ανάπτυξης και τη συμβατότητα με υπάρχοντα συστήματα.
2. **Triggers / Events**
   Το σύνολο των γεγονότων που μπορούν να ενεργοποιήσουν μια συνάρτηση (HTTP requests, queues, storage events, schedulers).
3. **Durable / Stateful Functions και Orchestration**
   - *Stateful orchestration* (καταστατική σύνθεση συναρτήσεων)

Formatted: Greek

Formatted: Greek

Commented [KK69]: Προτεινω να έχετε ξεχωριστές υπο-ενότητες για τις πλατφόρμες και τα πλαισια. Ας εξηγήσετε και τα κριτήρια σύγκρισης πριν την εμφάνιση των πινάκων. Καλό είναι να σχολιαστει έπειτα το περιεχομενο των πινάκων για να αναφερθούν ορισμένα συμπεράσματα.

- *Durable/stateful functions* (δηλαδή συναρτήσεις που διατηρούν κατάσταση εγγενώς).

4. **Μοντέλο Κλιμάκωσης (Scaling)** Περιλαμβάνει scale-to-zero, min/max instances, όρια ταυτοχρονισμού (concurrency limits) και warm instances.
5. **Μοντέλο Χρέωσης (Billing Model)** Αξιολογεί τη χρέωση ανά κλήσης, διάρκεια εκτέλεσης, μνήμη και δωρεάν επίπεδα.
6. **Monitoring & Observability** Αναφέρεται σε dashboards, logs, tracing και δυνατότητες alerts.
7. **Μοναδικά Πλεονεκτήματα / Ιδιαιτερότητες** Χαρακτηριστικά που διαφοροποιούν μια πλατφόρμα / πλαίσιο από μια άλλη / άλλο.
8. **Υποδομή:** το είδος της υποδομής που υποστηρίζεται για την διάταξη των συναρτήσεων
9. **Κλιμάκωση**: ο τρόπος που πραγματοποιείται η κλιμάκωση με βάση τη χρήση ενός serverless πλαισίου

Επεξήγηση όρων που χρησιμοποιούνται στους πίνακες

- **Scale-to-zero:** δυνατότητα η πλατφόρμα να μειώνει τα ενεργά instances στο 0 όταν δεν υπάρχει φόρτος, ώστε να μη χρεώνεται υπολογιστικός χρόνος.
- **Min/Max instances:** όρια στον αριθμό instances που διατηρούνται ενεργά.
  - *min instances*: κρατά "ζεστά" instances για μείωση cold start.
  - *max instances*: περιορίζει την κλιμάκωση για έλεγχο κόστους/φόρτου.
- **Warm instances:** instances που παραμένουν ενεργά (π.χ. όταν min > 0), άρα μειώνονται τα cold starts.
- **Invocations / Executions:** αριθμός κλήσεων/εκτελέσεων της συνάρτησης.
- **GB-sec (ή GB-s):** χρέωση βάσει μνήμης (GB) × χρόνου εκτέλεσης (sec).
- **Gen1 / Gen2 (Google Cloud Functions):** διαφορετικές "γενιές" της υπηρεσίας Cloud Functions (όχι γενιές συναρτήσεων). Η Gen2 βασίζεται σε υποδομή Cloud Run, προσφέροντας επιπλέον δυνατότητες (π.χ. καλύτερο έλεγχο instances/κλιμάκωσης και διαφοροποιημένο μοντέλο πόρων/χρεώσεων).

Formatted: Greek

Commented [KK70]: Στον ένα ή και στους δύο;

2.7.2 Εμπορικές Πλατφόρμες Serverless

~~**Πίνακας 2.1 Σύγκριση AWS Lambda, Azure Functions, Google Cloud Functions**~~

Πίνακας 1: Σύγκριση AWS Lambda, Azure Functions, Google Cloud Functions

Χαρακτηριστικό	**AWS Lambda**	**Azure Functions**	**Google Cloud Functions**
Υποστηριζόμενες Γλώσσες	Node.js, Python, Java, Go, C#, Ruby, PowerShell, Custom Runtimes	C#, Java, Python, JavaScript/TypeScript, PowerShell, Custom Runtimes	Node.js, Python, Go, Java, .NET, PHP, Custom Runtimes
Triggers / Events	API Gateway, S3, DynamoDB, SNS/SQS, EventBridge, CloudWatch Events	HTTP, Event Grid, Service Bus, Storage, Timer, Queue	HTTP, Pub/Sub, Cloud Storage, Eventarc (90+ event sources), Cloud Scheduler
Durable / Stateful Functions	*Όχι* (stateful orchestration μέσω Step Functions)	**Ναι** (Durable Functions — stateful functions)	*Όχι* (stateful orchestration μέσω Google Workflows)
Κλιμάκωση	Αυτόματη, min/max concurrency, provisioned	Consumption: scale-to-zero, στο πακέτο Premium	Min/max instances, scale-to-zero, warm instances αν min > 0

Formatted: Greek

Formatted: Greek

Formatted: Greek

Commented [KK71]: Καλό είναι να φαινονται τα όρια του πίνακα ώστε αυτός να ξεχωρίζει από την λεζάντα του.

Formatted: English (United Kingdom)

Formatted: English (United Kingdom)

Formatted: Caption, Keep with next

Commented [KK72]: Η σημασιολογια και επεξήγηση των τιμων των κριτηρίων ας γινει προηγουμένως στην παρουσιαση των κριτηριων και όχι εντος του πίνακα. Αυτό που λέτε εδώ δεν αφορά μόνο την συγκεκριμένη πλατφορμα αλλά οποιαδήποτε πλατφόρμα που υποστηρίζει min instances.

Χαρακτηριστικό	AWS Lambda	Azure Functions	Google Cloud Functions
	concurrency, scale-to-zero	επιτρέπεται η διαμόρφωση min instances	
Διαθέσιμη Μνήμη	128 MB – 10 GB	128 MB – 14 GB	128 MB – 16 GB (Gen1), μεγαλύτερη ευελιξία στο Gen2 (32 GB)
Μοντέλο Χρέωσης	GB-sec + invocations, 1M free / month	Consumption: GB-sec + invocations, Premium πακέτο: vCPU/GB-s	GB-sec + invocations, 2M free / month, διαφορές Gen1/Gen2 πακετών
Monitoring	CloudWatch	Azure Monitor	Cloud Monitoring
Μοναδικά Πλεονεκτήματα	Firecracker microVMs (ισχυρή απομόνωση – χαμηλά cold starts)	Stateful functions + Ευελιξία διάταξης λόγω πολλαπλών hosting μοντέλων	Άριστη ενσωμάτωση με Cloud Run (GCF Gen2)

Σχολιασμός των Αποτελεσμάτων

Από τον πίνακα προκύπτουν τα εξής:

- **Υποστηριζόμενες γλώσσες:** Και οι τρεις πλατφόρμες καλύπτουν τον βασικό πυρήνα runtimes (Node.js, Python, Java, Go), ενώ διαφέρουν σε δευτερεύοντα runtimes (π.χ. Ruby/PowerShell στο AWS, TypeScript/PowerShell στο Azure, PHP/.NET στο GCP). Επιπλέον, και οι τρεις υποστηρίζουν **custom runtimes**, επιτρέποντας μεγαλύτερη φορητότητα.
- **Triggers / Events:** Και οι τρεις υπηρεσίες παρέχουν event-driven μοντέλο, όμως διαφοροποιούνται στο οικοσύστημα γεγονότων: η AWS συνδέεται ισχυρά με υπηρεσίες AWS (S3/SQS/SNS/EventBridge), η Azure με Event Grid/Service Bus/Storage, ενώ η Google μέσω Eventarc μπορεί να ενοποιεί μεγάλο πλήθος πηγών γεγονότων (το ίδιο όμως προσφέρει και η AWS με το EventBridge και η Azure με το Event Grid).

Formatted: Greek

Commented [KK73]: Ποια η διαφορά με invocations; Ας χρησιμοποιήσετε τον ίδιο όρο.

Commented [KK74]: Για τι ειδους διαφορές μιλάμε εδώ; Στους συντελεστές του μοντέλου χρέωσης ή κάτι άλλο;

Formatted: Greek

Commented [KK75]: Ο σχολιασμός βασιζεται κυρίως στα μοναδικά χαρακτηριστικά. Δεν λογαριάζει την απόδοση των πλατφορμών στα υπόλοιπα κριτήρια. Υπάρχει κάποια υπεροχή πλατφορμας σε κάποιο από τα προηγούμενα κριτήρια;

- **Durable / Stateful:** Μόνο η **Azure Functions** προσφέρει "native" stateful function orchestration μέσω **Durable Functions** (δηλαδή σύνθετες καταστατικές συναρτήσεις). Αντίθετα, AWS και GCP βασίζονται σε ξεχωριστές υπηρεσίες orchestration (AWS Step Functions και Google Workflows).

- **Κλιμάκωση:** Και οι τρεις πλατφόρμες υποστηρίζουν αυτόματη κλιμάκωση και scale-to-zero. ωστόσο διαφέρουν στον βαθμό ελέγχου που προσφέρουν και στο πώς μπορεί να μειωθεί το φαινόμενο των *cold starts*:

  - **AWS Lambda:** παρέχει τον πιο άμεσο έλεγχο για σταθεροποίηση της απόδοσης μέσω **Provisioned Concurrency**, δηλαδή προ-διατήρηση "ζεστών" instances ώστε να μειώνονται οι *cold starts*. Επιπλέον, επιτρέπει ρυθμίσεις που σχετίζονται με **concurrency** (όρια/διαχείριση ταυτόχρονων εκτελέσεων), κάτι που βοηθά στη διαχείριση αιχμών φορτίου και στην προβλεψιμότητα απόδοσης.

  - **Azure Functions:** υποστηρίζει *scale-to-zero* στο **Consumption plan**, ενώ στο **Premium plan** δίνει δυνατότητα **ελάχιστων instances (pre-warmed instances)** ώστε να μειώνονται τα *cold starts* και να βελτιώνεται η σταθερότητα. Η πλατφόρμα προσφέρει διαφορετικά *hosting plans*, τα οποία επηρεάζουν άμεσα τη συμπεριφορά κλιμάκωσης και το κόστος.

  - **Google Cloud Functions (Gen2):** κληρονομεί μηχανισμούς κλιμάκωσης από το **Cloud Run**, παρέχοντας ρυθμίσεις όπως **min/max instances**. Όταν οριστεί *min instances > 0*, διατηρούνται "warm" instances, μειώνοντας τους *cold starts*, ενώ με *min=0* επιτυγχάνεται πλήρες *scale-to-zero*. Έτσι, προσφέρεται πρακτικός έλεγχος ισορροπίας μεταξύ κόστους και καθυστέρησης εκκίνησης.

  - Συνεπώς, η βασική διαφοροποίηση δεν είναι αν υπάρχει auto-scaling, αλλά **πώς** μπορεί ο χρήστης/διαχειριστής να ρυθμίσει την ύπαρξη "warm" instances και τα όρια κλιμάκωσης, ώστε να ελέγξει το trade-off *κόστος ↔ καθυστέρηση εκκίνησης (cold start) ↔ προβλεψιμότητα απόδοσης*.

- **Διαθέσιμη μνήμη:** Και οι τρεις ξεκινούν από 128MB, όμως η μέγιστη μνήμη διαφοροποιείται, με την **Azure** και τη **Google (Gen2)** να προσφέρουν μεγαλύτερα όρια σε σχέση με το AWS στα συγκεκριμένα νούμερα του πίνακα.

- **Μοντέλο χρέωσης:** Και οι τρεις υπηρεσίες στηρίζονται σε χρέωση τύπου **GB-sec + invocations/executions** με free tier, όμως η Azure εμφανίζει περισσότερη πολυπλοκότητα λόγω διαφορετικών hosting plans (π.χ. Consumption vs Premium). Η Google διαφοροποιείται επιπλέον μεταξύ **Gen1 και Gen2**, επειδή η Gen2 βασίζεται σε διαφορετικό execution environment (Cloud Run), άρα διαφοροποιούνται και οι παράμετροι/λογική τιμολόγησης. Όμως, η τιμολόγηση είναι καλύτερη στο Gen2 διότι οδηγεί σε μικρότερο κόστος για διαφορετικά είδη workload.

Formatted: Greek

Commented [KK76]: Οπότε είναι καλύτερη στην Gen2;

- **Monitoring:** Και οι τρεις προσφέρουν πλήρη ενσωμάτωση monitoring με το αντίστοιχο οικοσύστημα του παρόχου (CloudWatch, Azure Monitor, Cloud Monitoring).
- **Μοναδικά πλεονεκτήματα:**
  - Το **AWS Lambda** ξεχωρίζει για ισχυρή απομόνωση και αρχιτεκτονικές επιλογές που σχετίζονται με την ασφάλεια και τον τρόπο εκτέλεσης (π.χ. Firecracker).
  - Το **Azure Functions** ξεχωρίζει λόγω **Durable Functions** και των **πολλαπλών hosting models**, τα οποία επιτρέπουν διαφορετικά trade-offs κόστους/απόδοσης/ελέγχου.
  - Το **Google Cloud Functions (Gen2)** ξεχωρίζει λόγω της στενής ενοποίησης με **Cloud Run**, η οποία επηρεάζει δυνατότητες κλιμάκωσης και το execution environment. Ουσιαστικά προσφέρεται ένα πιο ευέλικτο μοντέλο κλιμάκωσης καθώς και περισσότερα εύρη σε πόρους σε σχέση με τις ανταγωνιστικές πλατφόρμες

Ειδικά συμπεράσματα για τις εμπορικές πλατφόρμες

1. Οι εμπορικές πλατφόρμες προσφέρουν ώριμα και πλήρως διαχειριζόμενα οικοσυστήματα serverless, με σημαντικές διαφοροποιήσεις ως προς την απομόνωση συναρτήσεων, το μοντέλο χρέωσης και τον τρόπο κλιμάκωσης.
2. Η Azure Functions ξεχωρίζει επειδή υποστηρίζει εγγενώς durable/stateful functions μέσω του Durable Functions model, ενώ η AWS Lambda και το Google Cloud Functions βασίζονται σε εξωτερικές υπηρεσίες ενορχήστρωσης, όπως τα AWS Step Functions και Google Workflows αντίστοιχα.
3. Το Google Cloud Functions Gen2 προσφέρει ιδιαίτερα ευέλικτες επιλογές κλιμάκωσης λόγω της υποδομής Cloud Run. Ωστόσο, και η AWS Lambda και η Azure Functions παρέχουν προχωρημένες δυνατότητες ελέγχου κλιμάκωσης και πόρων, ανάλογα με το πλάνο και τις διαθέσιμες ρυθμίσεις. Επομένως, η καταλληλότερη επιλογή εξαρτάται από τις απαιτήσεις της εκάστοτε εφαρμογής ως προς την καθυστέρηση εκκίνησης, το κόστος, τον έλεγχο της κυκλοφορίας και την προβλεψιμότητα της απόδοσης.

2.7.3 Πλαίσια Ανοικτού Κώδικα (Open Source Frameworks)

Τα πλαίσια ανοικτού κώδικα επιτρέπουν τη δημιουργία serverless λειτουργιών σε ιδιόκτητη ή πολυ-νεφική υποδομή και συχνά χρησιμοποιούνται για αποφυγή vendor lock-in.

~~**Πίνακας 2.2 — Σύγκριση Serverless Frameworks**~~

Formatted
Commented [KK77]: Οπότε έχουμε καλύτερες
Formatted
Formatted
Formatted

Formatted: Greek

Formatted: Justified

Πίνακας 2: Σύγκριση Serverless Frameworks

Formatted: Caption, Keep with next

Πλαίσιο	Υποδομή	Κλιμάκωση	Ιδιαιτερότητες / Περιορισμοί
OpenFaaS	Docker + Kubernetes	Autoscaling βάσει μετρικών (Prometheus) μέσω Kubernetes	Δεν υποστηρίζει scale-to-zero· απαιτεί Docker/K8s υποδομή
Kubeless	Kubernetes (CRDs)	Native Kubernetes scaling	Functions ως Custom Resources (οι συναρτήσεις δηλώνονται ως 'πρώτης τάξης' αντικείμενα στο Kubernetes)· χρειάζεται εξειδικευμένη γνώση ή εκπαίδευση σε Kubernetes
Apache OpenWhisk	Docker + Kubernetes + CouchDB	Warm containers + event triggers · κλιμάκωση (event-driven) μέσω εσωτερικού scheduler/controllers	Χρησιμοποιείται από IBM Cloud Functions· υποστηρίζει *sequences*[27] (ακολουθίες συναρτήσεων)
Serverless Framework	Πολύ-νεφικό εργαλείο (AWS, GCP, Azure)	Εξαρτάται από την πλατφόρμα	Ενοποιημένη διαμόρφωση (serverless.yml) και αυτοματισμοί CLI για διάταξη σε πολλαπλούς παρόχους+ CLI automation – πρότυπο multi-cloud πλαίσιο

[27] Sequences είναι **αλυσίδες συναρτήσεων** όπου η έξοδος της μίας περνά αυτόματα ως είσοδος στην επόμενη *χωρίς* ανάγκη ξεχωριστού orchestrator. Συνεπώς, οι sequences στο OpenWhisk αποτελούν μηχανισμό σύνθεσης συναρτήσεων, επιτρέποντας απλές ροές χωρίς εξωτερικό orchestrator για βασικά σενάρια.

Πλαίσιο	Υποδομή	Κλιμάκωση	Ιδιαιτερότητες / Περιορισμοί
AWS Chalice	AWS Lambda	Εξαρτάται από AWS	Python micro-framework για serverless APIs (ειδικό για AWS)

Ειδικά συμπεράσματα για τα πλαίσια ανοικτού κώδικα

1. Το Serverless Framework αποτελεί το πλέον διαδεδομένο εργαλείο ανάπτυξης σε πολλαπλούς παρόχους νέφους, καθώς προσφέρει ενοποιημένο αρχείο παραμετροποίησης (serverless.yml), αυτοματισμούς CLI, υποστήριξη CI/CD και πλούσιο οικοσύστημα προσθέτων.
2. Πλαίσια όπως τα OpenFaaS, Kubeless και Apache OpenWhisk είναι καταλληλότερα όταν απαιτείται ανάπτυξη σε ελεγχόμενη υποδομή, ιδίως σε περιβάλλοντα βασισμένα σε Kubernetes, όπου ζητείται μεγαλύτερος βαθμός ελέγχου πάνω στη διάταξη και στην εκτέλεση των συναρτήσεων.
3. Για περιπτώσεις στις οποίες ο προγραμματιστής στοχεύει αποκλειστικά σε έναν μόνο πάροχο, συχνά προτιμώνται εξειδικευμένα πλαίσια που έχουν σχεδιαστεί ειδικά για το συγκεκριμένο οικοσύστημα. Χαρακτηριστικό παράδειγμα αποτελεί το AWS Chalice, το οποίο υποστηρίζει την ταχεία ανάπτυξη serverless RESTful APIs αποκλειστικά στην AWS και αξιοποιεί native δυνατότητες του οικοσυστήματος Lambda και API Gateway.
4. Παρότι τα multi-cloud εργαλεία μειώνουν το κόστος μετάβασης και διευκολύνουν τη φορητότητα στο επίπεδο της διάταξης, δεν εξαλείφουν πλήρως τις διαφοροποιήσεις που προκύπτουν από τα triggers, τα IAM μοντέλα, το observability και τις ειδικές υπηρεσίες κάθε παρόχου.

2.7.4 Καθολικά Συμπεράσματα

1. Η επιλογή μεταξύ εμπορικής serverless πλατφόρμας και πλαισίου ανοικτού κώδικα εξαρτάται κυρίως από το επιθυμητό ισοζύγιο μεταξύ ευκολίας διαχείρισης, ελέγχου υποδομής, φορητότητας και κόστους.
2. Οι εμπορικές πλατφόρμες υπερτερούν συνήθως σε επίπεδο διαχειριζόμενων υπηρεσιών, ενοποίησης με το οικοσύστημα του παρόχου και επιχειρησιακής

Formatted ...

ωριμότητας, ενώ τα ανοικτού κώδικα πλαίσια παρέχουν μεγαλύτερη ευελιξία και έλεγχο σε περιβάλλοντα ιδιόκτητης ή πολυ-νεφικής υποδομής.

3. Το vendor lock-in παραμένει κρίσιμο ζήτημα σε όλο το φάσμα της serverless υπολογιστικής. Αν και τα multi-cloud εργαλεία περιορίζουν εν μέρει την εξάρτηση από έναν πάροχο, η πλήρης φορητότητα επηρεάζεται από διαφορές σε μηχανισμούς ενεργοποίησης, πολιτικές πρόσβασης, παρατηρησιμότητα, πακετάρισμα και υποστηρικτικές υπηρεσίες.

1. ~~**Οι εμπορικές πλατφόρμες προσφέρουν ώριμα οικοσυστήματα**, με σημαντικές διαφορές σε απομόνωση συναρτήσεων, μοντέλο/τρόπο χρέωσης και κλιμάκωση.~~

1. ~~**Η Azure είναι μοναδική στην υποστήριξη durable/stateful functions**, ενώ AWS και Google προσφέρουν ενορχήστρωση συναρτήσεων σε επίπεδο workflows.~~

1. ~~Η Google Cloud Functions Gen2 προσφέρει ιδιαίτερα ευέλικτες επιλογές κλιμάκωσης λόγω της υποδομής Cloud Run. Ωστόσο, η AWS και η Azure παρέχουν επίσης προχωρημένες δυνατότητες ελέγχου κλιμάκωσης/πόρων ανάλογα με το πλάνο (π.χ. πολιτικές concurrency/provisioning ή premium hosting). Άρα η 'καλύτερη' επιλογή εξαρτάται από τις απαιτήσεις (latency, κόστος, έλεγχος traffic, προβλεψιμότητα).~~

1. ~~**Το Serverless Framework αποτελεί το δημοφιλέστερο multi-cloud πλαίσιο**, ενώ πλαίσια όπως OpenFaaS και Kubeless αποτελούν επιλογές για την υλοποίηση ιδιόκτητων Kubernetes υποδομών.~~

1. ~~Το **vendor lock-in** παραμένει πρόβλημα τόσο εκτός όσο και εν μέρει **εντός** λύσεων multi-cloud, καθώς η φορητότητα επηρεάζεται από διαφορές σε triggers, IAM, observability, packaging και provider specific υπηρεσίες. Εργαλεία όπως το Serverless Framework μειώνουν το κόστος μετάβασης στο επίπεδο διάταξης, αλλά δεν εξαλείφουν πλήρως την εξάρτηση από τις ιδιαιτερότητες κάθε παρόχου.~~

1. ~~Το **Serverless Framework** αποτελεί το πλέον διαδεδομένο εργαλείο οριζόντιας ανάπτυξης (cross provider deployment), επιτρέποντας σε προγραμματιστές να αναπτύσσουν συναρτήσεις ταυτόχρονα σε πολλαπλά νέφη και να αποφεύγουν το φαινόμενο του vendor lock-in. Παρέχει unified αρχείο παραμετροποίησης (serverless.yml), παροχή ενοποιημένου workflow διάταξης σε πολλαπλούς παρόχους, παρότι η υποστήριξη και οι διαθέσιμες ρυθμίσεις διαφέρουν ανά πλατφόρμα, δυνατότητα αυτόματης δημιουργίας API endpoints, καθώς και άμεση ενοποίηση με CI/CD pipelines. Επιπλέον, διαθέτει ένα πλούσιο οικοσύστημα προσθέτων (plugins) για λειτουργίες όπως bundling, secrets management, offline emulation και staged deployments, κάτι που το καθιστά ιδιαίτερα ευέλικτο σε μεγάλα περιβάλλοντα ανάπτυξης. Ωστόσο, όταν απαιτείται χρήση **ελεγχόμενης** υποδομής (π.χ. βασισμένη σε Kubernetes,~~

Formatted: Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: Greek

Formatted: Normal

Commented [KK81]: Όπως;
Ανέφερα προηγουμένως πως λογικά και τα άλλα πλαισια θα πρέπει να έχουν κάποιο CLI.
Μήπως η διαφορά είναι πως το Serverless framework επιτρεπει την διάταξη σε πολλαπλά περιβάλλοντα ενώ τα άλλα εργαλεία την διαταξη & εκτέλεση σε συγκεκριμένες πλατφόρμες μόνο;

Επίσης, το Serverless framework φαίνεται να υπερέχει στα εξης: παροχή χαρακτηριστικών αυτοματισμού (CI/CD, staged deployments) και μεγαλύτερο οικοσύστημα προσθέτων (bundling, secrets management, offline emulation),

~~είτε επιτόπιας είτε διαχειριζόμενης από πάροχο νέφους), είναι συνήθως προτιμότερο να χρησιμοποιηθεί ένα από τα τρία πρώτα πλαίσια του πίνακα (OpenFaaS, Kubeless, Apache OpenWhisk), τα οποία επιτρέπουν granular έλεγχο πάνω στην συστάδα διακομιστών της υποδομής.~~

1. ~~Για περιπτώσεις όπου ο προγραμματιστής στοχεύει σε **μία μόνο πλατφόρμα**, το βέλτιστο αποτέλεσμα συνήθως επιτυγχάνεται με τη χρήση πλαισίων που έχουν σχεδιαστεί **ειδικά** για αυτή την πλατφόρμα. Παρότι δεν εμφανίστηκαν συστηματικά στον προηγούμενο πίνακα σύγκρισης, τέτοια πλαίσια υπάρχουν. Ένα χαρακτηριστικό παράδειγμα είναι το **AWS Chalice**, ένα Python micro-framework που επιτρέπει την ταχεία ανάπτυξη serverless RESTful APIs αποκλειστικά στην AWS. Το Chalice παρέχει native routing, αυτόματη δημιουργία IAM roles, πακετάρισμα (packaging) συναρτήσεων και ταχύ τροφοδοσία (provisioning) σε AWS Lambda & API Gateway, αποτελώντας εξαιρετική επιλογή όταν ο στόχος είναι το οικοσύστημα AWS.~~

2.8 Πλεονεκτήματα και Προκλήσεις της Serverless Υπολογιστικής

Πλεονεκτήματα

Η serverless υπολογιστική προσφέρει σημαντικά οφέλη τόσο σε οργανισμούς όσο και σε προγραμματιστές:

1. **Αυτόματη Κλιμάκωση:** Οι πόροι προσαρμόζονται δυναμικά, χωρίς ανάγκη ανθρώπινης παρέμβασης. Αυτό εξασφαλίζει υψηλή διαθεσιμότητα ακόμη και σε αιχμές φόρτου (traffic spikes), καθώς η πλατφόρμα μπορεί να δημιουργήσει δεκάδες ή εκατοντάδες στιγμιότυπα της ίδιας συνάρτησης.

2. **Χρέωση Βάσει Χρήσης (Pay-per-Use):** Ο χρήστης πληρώνει μόνο για τις εκτελέσεις που πραγματοποιούνται και μόνο για την πραγματική διάρκεια εκτέλεσης (συνάρτησης) (σε ms) και τη μνήμη που δεσμεύεται.
   ➜ Σε αντίθεση με εικονικές μηχανές που χρεώνουν ωριαία, το serverless μοντέλο μηδενίζει το κόστος όταν η εφαρμογή δεν εκτελείται.

3. **Απλότητα και Ταχύτητα Ανάπτυξης:** Ο προγραμματιστής επικεντρώνεται στη λογική της εφαρμογής, ενώ η πλατφόρμα αναλαμβάνει τα πάντα (servers, scaling, provisioning, patching του περιβάλλοντος εκτέλεσης (runtime environment)).
   ➜ Η ανάπτυξη επιταχύνεται σημαντικά χάρη σε εργαλεία όπως **Serverless Framework, AWS SAM, Zappa, Chalice**, που αυτοματοποιούν το πακετάρισμα, τη μεταφόρτωση και τη δημιουργία των σχετικών πόρων.

Formatted: Greek

4. **Επεκτασιμότητα και Ενσωμάτωση με Cloud Υπηρεσίες:** Οι serverless συναρτήσεις μπορούν να συνδεθούν απευθείας με άλλες υπηρεσίες του νέφους (βάσεις δεδομένων, αποθήκες αρχείων, messaging queues, authentication APIs).

   ➜ Αυτό διευκολύνει τον σχεδιασμό event-driven αρχιτεκτονικών.

5. **Ανθεκτικότητα (Resilience):** Κάθε κλήση εκτελείται απομονωμένα (σε container ή microVM).

   ➜ Ένα σφάλμα σε μία συνάρτηση δεν επηρεάζει τα υπόλοιπα στιγμιότυπα (της συνάρτησης) ούτε άλλες συναρτήσεις της εφαρμογής. Εφόσον το σφάλμα είναι κριτικό, οι κλήσεις θα δρομολογούνται πλέον στα υπόλοιπα στιγμιότυπα, προσφέροντας υψηλό επίπεδο ανθεκτικότητας.

Προκλήσεις

Παρά τα σημαντικά πλεονεκτήματα, η serverless υπολογιστική συνοδεύεται από τεχνικά και λειτουργικά μειονεκτήματα:

1. **Καθυστέρηση Cold Start:** Όταν μια συνάρτηση είναι "κρύα" (scale-to-zero), η πλατφόρμα χρειάζεται χρόνο για να δημιουργήσει το κατάλληλο περιβάλλον εκτέλεσης (container/microVM). Αυτό μπορεί να προκαλέσει αυξημένο latency στα πρώτα (ταυτόχρονα) invocations μετά από περίοδο αδράνειας.

2. **Vendor Lock-In:** Κάθε πάροχος χρησιμοποιεί διαφορετικά APIs, event systems και μηχανισμούς τροφοδοσίας, χρονοπρογραμματισμού και κλιμάκωσης. Η μεταφορά εφαρμογών μεταξύ παρόχων είναι δαπανηρή ή δύσκολη. **Μερική λύση**: Δια-νεφικά πλαίσια όπως το Serverless Framework ή το Terraform μειώνουν την εξάρτηση, αλλά **δεν εξαλείφουν το vendor lock-in πλήρως**, διότι τα πιο εξειδικευμένα χαρακτηριστικά (π.χ. proprietary triggers, IAM policies, event bridges, monitoring hooks) παραμένουν μοναδικά για κάθε πάροχο.

3. **Περιορισμένη Παρακολούθηση και Παρατηρησιμότητα:** Η αφαιρετικότητα του μοντέλου και η βραχύβια εκτέλεση των συναρτήσεων δυσκολεύουν την αποσφαλμάτωση (debugging), διότι:

   - οι συναρτήσεις δεν έχουν σταθερό διακομιστή,
   - τα logs διασπείρονται σε διαφορετικά στιγμιότυπα,
   - δεν υποστηριζονται custom μετρικές ενώ δεν υπάρχει και κοινό σύνολο πρότυπων μετρικών που να υποστηριζονται από όλους τους παροχους.

Formatted: Greek

Commented [KK82]: Πρόσθεσα ταυτόχρονα διότι λογικά μόλις δημιουργηθει το περιβάλλον, οι επόμενες κλήσεις δεν θα έχουν το πρόβλημα. Μόνο οι πρώτες, σχεδον ταυτόχρονες.

Formatted: Indent: Left: 1,9 cm, Tab stops: 2,54 cm, List tab + Not at -1,27 cm

- χρειάζεται distributed tracing για συσχέτιση events.

Λύσεις / Εργαλεία:

- **Prometheus**: συλλογή και αποθήκευση μετρικών, υποστηρίζει alerts.
- **OpenTelemetry**: ενιαίο πρότυπο για traces, logs και metrics· επιτρέπει πλήρη παρατηρησιμότητα σε κατανεμημένες ροές εργασίας.
- **Datadog / New Relic**: εμπορικά εργαλεία με dashboards, traces και automatic correlation.

➜ Τα εργαλεία αυτά **βελτιώνουν σημαντικά** την παρατηρησιμότητα, αλλά δεν την βελτιστοποιούν πλήρως, καθώς περιορίζονται από το κλειστό λειτουργικό μοντέλο των serverless παρόχων.

4. **Κίνδυνοι Ασφάλειας Πολλαπλής Μίσθωσης (Multi-Tenancy Risks):** Οι συναρτήσεις πολλών πελατών εκτελούνται στους ίδιους φυσικούς κόμβους (shared infrastructure). Συνεπώς, απαιτούνται ισχυρά επίπεδα απομόνωσης (isolation layers) (microVMs, hardened containers, secure sandboxes). Οι κίνδυνοι που αναδύονται **δεν είναι "νέοι"**, αλλά αποτελούν γνωστές προκλήσεις που εμφανίζονται σε κάθε περιβάλλον νέφους όπου εκτελείται λογισμικό πελατών (cloud multi-tenant execution). Εντούτοις, το serverless προσθέτει **ιδιαιτερότητες**, λόγω της βραχύβιας, event-driven και αυτόματα κλιμακούμενης φύσης των συναρτήσεων.

Οι κίνδυνοι μπορούν να μπορούν να χωριστούν σε 2 κατηγόρίες,

- ~~1.~~ **γενικοί κίνδυνοι multi-tenancy: side-channel attacks** μεταξύ απομονωμένων εκτελέσεων (π.χ. cache timing leaks), **ανασφαλείς βιβλιοθήκες τρίτων (third-party dependencies)** που μπορεί να περιέχουν τρωτότητες, **shared hardware risks** (π.χ. Spectre/Meltdown-like vulnerabilities),
- ~~2.~~ **κίνδυνοι που είναι ιδιαίτεροι στο serverless μοντέλο: malicious event injection** (Κακόβουλος χρήστης μπορεί να δημιουργήσει ψεύτικα events (π.χ. fake SQS messages, crafted HTTP triggers), πυροδοτώντας ανεξέλεγκτες εκτελέσεις)**, over-privileged functions** (οι συναρτήσεις συχνά εκτελούνται με υπερβολικά διευρυμένες IAM permissions, γεγονός που αυξάνει τον αντίκτυπο μιας πιθανής παραβίασης)**, Economic Denial of Sustainability (EDoS)** (ο αντίπαλος δεν ρίχνει την υπηρεσία — την αφήνει να δουλεύει συνεχώς, προκαλώντας υπερχρέωση μέσω τεράστιου αριθμού invocations)**, state handling risks** (λόγω stateless εκτέλεσης, οι προγραμματιστές αποθηκεύουν δεδομένα σε εξωτερικές υπηρεσίες (DB,

Formatted: Greek
Formatted: Indent: Left: 1,9 cm, Tab stops: 2,54 cm, List tab + Not at 0,63 cm
Formatted: Indent: Left: 1,9 cm, Tab stops: 2,54 cm, List tab + Not at 1,9 cm
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Bold
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Bold
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, Bold
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

caches). Οπότε, η λανθασμένη παραμετροποίηση αυτών μπορεί να οδηγήσει σε διαρροή δεδομένων).

5. **Κλιμάκωση Υπό Όρους & Latency Spikes:**
Η αυτόματη κλιμάκωση δεν είναι στιγμιαία. Οπότε, ενδεχομένως ορισμένα προβλήματα μπορεί να εμφανιστούν, όπως:

   - προσωρινή έλλειψη επαρκών στιγμιοτύπων (για την αντιμετώπιση του τρέχοντος φόρτου εργασίας),
   - υπερφόρτωση πηγών γεγονότων (π.χ. Pub/Sub, SQS),
   - χρήση αλγορίθμων κλιμάκωσης που δεν προβλέπουν σωστά τη ζήτηση.

Όλα αυτά μπορεί να οδηγήσουν σε latency spikes ή throttling, ειδικά σε εφαρμογές πραγματικού χρόνου. Για να αντιμετωπιστεί αυτό, οι σύγχρονοι πάροχοι προσφέρουν διάφορους μηχανισμούς:

- **pre-warmed containers / min instances** (διαθέσιμα σε AWS, Azure και GCP),
- **Provisioned Concurrency (AWS Lambda)**, το οποίο παρέχει εγγυημένο αριθμό warm instances και υποστηρίζει *custom autoscaling policies* αποκλειστικά για αυτόν τον μηχανισμό,
- **Advanced Scaling Plans (Azure Functions)**, τα οποία επιτρέπουν πιο λεπτομερή παραμετροποίηση (π.χ. Elastic Premium Plan),
- **min/max instances (GCP Cloud Functions)** για μείωση cold starts.

Σε γενικές γραμμές, **custom autoscaling policies δεν εφαρμόζονται οριζόντια σε όλες τις πλατφόρμες**· είναι διαθέσιμες κυρίως στο πλαίσιο του Provisioned Concurrency στην AWS. Η Azure, αντίθετα, παρέχει πιο εξελιγμένα scaling plans ενσωματωμένα στη λειτουργία της πλατφόρμας, ενώ η GCP επιτρέπει την απλοποιημένη προθέρμανση μέσω min instances.

Formatted: Greek
Formatted: Indent: Left: 1,9 cm, Tab stops: 2,54 cm, List tab + Not at 1,9 cm
Formatted: Indent: Left: 1,9 cm, Tab stops: 2,54 cm, List tab + Not at 1,9 cm
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted: Font: (Default) Times New Roman, 12 pt, English (United Kingdom)
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted
Formatted: Font: (Default) Times New Roman, 12 pt
Formatted
Formatted: Greek

Formatted: Greek

2.9 Συμπεράσματα Κεφαλαίου

Το υπολογιστικό νέφος και, ειδικότερα, η serverless υπολογιστική, αποτελούν σήμερα τη βάση της σύγχρονης ανάπτυξης cloud εφαρμογών. Το serverless computing μοντέλο προσφέρει ευελιξία, αυτόματη κλιμάκωση, ταχύτητα και οικονομικά αποδοτική λειτουργία, καθιστώντας το ιδανικό για microservices, APIs και workloads με μεταβλητή ζήτηση.Παράλληλα, η ύπαρξη πολλών εμπορικών και ανοικτού κώδικα λύσεων δημιουργεί ένα πλούσιο οικοσύστημα εργαλείων που υποστηρίζει διαφορετικές ανάγκες ανάπτυξης.

Η σύγκριση των πλατφορμών κατέδειξε ότι η επιλογή παρόχου εξαρτάται από παράγοντες όπως:

- οι υποστηριζόμενες γλώσσες προγραμματισμού,
- το μοντέλο χρέωσης,
- το εύρος των διαθέσιμων triggers/events,
- ο βαθμός ενσωμάτωσης με άλλες cloud υπηρεσίες (databases, messaging, storage),
- η δυνατότητα διαμορφωσιμότητας (configurability) και ελέγχου της εκτέλεσης.

Όσον αφορά **την κλιμάκωση και τη χρονική καθυστέρηση (latency)**, αυτά προκύπτουν έμμεσα από τις διαφορές των παρόχων στα εξής σημεία:

- **Η Azure Functions διαθέτει διαφορετικά *πλάνα* (Consumption, Premium)** που επιτρέπουν χειροκίνητα κάτω/πάνω όρια instances — επομένως ο χρήστης έχει μεγαλύτερο έλεγχο στη συμπεριφορά scaling και στα cold starts. Επιπλέον, παρέχεται ένα πιο ευέλικτο μοντέλο κλιμάκωσης. Ειδικότερα, πέρα από min/max limits, επιτρέπονται:
    - custom scaling rules μέσω **Azure Monitor Autoscale** (π.χ. scale-out με βάση CPU, queue length, custom metrics),
    - επιλογή διαφορετικών σχεδίων λειτουργίας (Consumption, Premium, Dedicated),
    - δυνατότητα διαμόρφωσης διαφόρων scaling thresholds, cooldown periods και κανόνων ενεργοποίησης.

Commented [KK83]: Μπορειτε να αναλύσετε λίγο παραπάνω τις δυνατοτητες scaling που προσφερει η Azure Functions. Επιτρέπει τον προσδιορισμό custom policies εκτος από απλά όρια στιγμιοτύπων;

- **Η Google Cloud Functions (2η γενιά)** προσφέρει “granular” έλεγχο γιατί οι ρυθμίσεις scaling (minInstances, maxInstances) εφαρμόζονται **σε επίπεδο συνάρτησης**, όχι σε επίπεδο εφαρμογής ή service plan.

  Ο χρήστης μπορεί να ορίσει:

- min instances (για μείωση cold starts),
- max instances (για αποφυγή ανεξέλεγκτου scale-out ή κόστους),
- concurrency per instance,
- scaling limits per function.

- **Το AWS Lambda βασίζεται αποκλειστικά στο αυτόματο scaling**, αλλά επιτρέπει τον προσδιορισμό custom πολιτικών κλιμάκωσης για ορισμένες περιπτώσεις. Ειδικότερα, προσφέρει:
  - **Provisioned Concurrency**, το οποίο υποστηρίζει custom scaling policies ειδικά για αυτό τον μηχανισμό,
  - **fine-grained traffic shaping** (ο πάροχος επιτρέπει στον χρήστη να κατανέμει ελεγχόμενα το ποσοστό της εισερχόμενης κίνησης (requests) σε διαφορετικές *εκδόσεις* της ίδιας συνάρτησης (π.χ. v1, v2), ώστε να κάνεις αναβαθμίσεις με ασφάλεια.) (π.χ. weighted aliases, gradual rollouts),
  - **reserved concurrency** και throttling control.

Η καθυστέρηση (latency) **δεν περιλαμβάνεται ως πρωτεύον κριτήριο**, ωστόσο επηρεάζεται έμμεσα από:

- τις πολιτικές scaling που εφαρμόζει κάθε πάροχος,
- τη δυνατότητα προθέρμανσης (pre-warming) ή provisioned concurrency,
- τα min/max instances,
- τον χρόνο δημιουργίας και απόδοσης runtime περιβάλλοντος.

Επομένως, η σύγκριση των παρόχων ως προς το scaling **είναι εφικτή**, αλλά απαιτεί κατανόηση των διαφορετικών μηχανισμών που χρησιμοποιεί κάθε πλατφόρμα για να περιορίσει latency και cold starts. Από τη σύγκριση προκύπτει ότι οι δύο πιο εξελιγμένες πλατφόρμες σε αυτό το κριτήριο είναι:

- **Azure Functions**, λόγω των πολύπλευρων autoscaling policies,

Formatted: Greek

Commented [KK84]: Ναι, αλλά επιτρέπει τον προσδιορισμό custom πολιτικών κλιμάκωσης για ορισμένες περιπτώσεις ...
Καθώς και υποστηρίζει fine-grained traffic shaping>

Commented [KK85]: Θα μπορούσατε να εξηγήσετε τι είναι αυτό.

Formatted: Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Formatted: Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Formatted: Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Formatted: Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

- **AWS Lambda**, λόγω της provisioned concurrency και των μηχανισμών traffic shaping.

Η Google Cloud Functions παρέχει μεν σημαντικό granular έλεγχο σε επίπεδο συνάρτησης, αλλά λιγότερες συνολικές στρατηγικές σε σύγκριση με Azure και AWS.

Η παρούσα εργασία αξιοποιεί αυτό το θεωρητικό υπόβαθρο για να σχεδιάσει ένα σύστημα επιλογής serverless πλατφορμών και διάταξης συναρτήσεων σε αυτές, επιτρέποντας συγκριτική αξιολόγηση των παρόχων και αυτόματη ανάπτυξη συναρτήσεων σε πολλαπλά νέφη.

Formatted: Greek

Formatted: Greek

3. Σχετικές Εργασίες

Σκοπός του παρόντος κεφαλαίου είναι η παρουσίαση και ανάλυση σχετικών ερευνητικών και τεχνολογικών προσεγγίσεων που αφορούν τη σύγκριση και αξιολόγηση serverless πλατφορμών καθώς και τη διάταξη serverless συναρτήσεων σε πολλαπλά υπολογιστικά νέφη (multi-cloud περιβάλλοντα).

Δεδομένου ότι το προτεινόμενο σύστημα της παρούσας εργασίας συνδυάζει και τις δύο αυτές λειτουργίες, η ανάλυση οργανώνεται σε δύο ενότητες:

0. εργαλεία που εστιάζουν στην επιλογή και αξιολόγηση serverless πλατφορμών, και
1.
2. εργαλεία που εστιάζουν στη διάταξη (deployment) serverless συναρτήσεων σε πολλαπλά περιβάλλοντα.

3.1 Επιλογή και Αξιολόγηση Serverless Πλατφορμών

Το **FaaStener** αποτελεί ένα από τα γνωστότερα έργα που στοχεύουν στη συστηματική σύγκριση πλατφορμών Function-as-a-Service (FaaS) και είναι διαθέσιμο ως web εφαρμογή[28]. Το FaaStener παρέχει γραφικό περιβάλλον (GUI) που επιτρέπει στον χρήστη να **περιηγηθεί στα χαρακτηριστικά μιας πλατφόρμας**, να πραγματοποιήσει **πολυκριτηριακή/πολυ-χαρακτηριστική αναζήτηση (multi-attribute search) με βάση απαιτήσεις** (επιχειρησιακές, τεχνικές ή συνδυασμό), καθώς και να εκτελέσει **σύγκριση πολλαπλών πλατφορμών ανά χαρακτηριστικό (feature-wise comparison)**. Συνεπώς, πρόκειται για διαδραστικό εργαλείο υποστήριξης επιλογής πλατφόρμας και όχι αποκλειστικά για εργαλείο πειραματικού benchmarking.

Παρότι το FaaStener και το παρόν σύστημα κινούνται στην ίδια κατεύθυνση (διαδραστική αναζήτηση και σύγκριση), η παρούσα εργασία διαφοροποιείται και επεκτείνεται στα εξής σημεία: (α) ενσωματώνει **πλήρη ροή "selection-to-deployment"**, καθώς πέρα από την αναζήτηση/αξιολόγηση υποστηρίζει **αυτόματη διάταξη (deployment) serverless συναρτήσεων** σε επιλεγμένο πάροχο μέσω Deployment Engine (Serverless Framework), (β) υλοποιεί **σταθμισμένη πολυκριτηριακή κατάταξη** με ρητά βάρη που παρέχει ο χρήστης και υπολογισμό **συνθετικού score** ανά πλατφόρμα, (γ) υποστηρίζει **λογιστική/λειτουργική καταγραφή της διαδικασίας διάταξης** (deployment status, endpoint, logs, χρόνοι), ώστε η αξιολόγηση να συνδέεται με πρακτική χρήση και παρακολούθηση, και (δ) περιλαμβάνει **ρόλους και λειτουργίες διαχείρισης** (π.χ.

[28] https://faastener.github.io/home

Formatted: Greek

Commented [KK86]: Κάθε κεφάλαιο θα πρεπει να ξεκινά σε νέα σελίδα (από την αρχή της, όχι στην μέση).

Commented [KK87]: ΠΡΟΣΕΞΤΕ ΔΙΟΤΙ ΤΟ ΠΕΡΙΘΩΡΙΟ ΑΡΙΣΤΕΡΑ ΕΙΝΑΙ ΛΑΘΟΣ ΚΑΙ ΤΡΩΤΕ ΧΩΡΟ ΧΩΡΙΣ ΛΟΓΟ. ΤΟ ΠΡΟΒΛΗΜΑ ΑΡΧΙΣΕ ΑΠΟ ΤΟ ΤΗΝ ΕΝΟΤΗΤΑ 2.8 ΚΑΙ ΣΥΝΕΧΙΖΕΤΑΙ ΣΤΑ ΥΠΟΛΟΙΠΑ ΚΕΦΑΛΑΙΑ. ΑΣ ΤΟ ΔΙΟΡΘΩΣΕΤΕ!

Formatted: Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0,63 cm + Tab after: 1,27 cm + Indent at: 1,27 cm

Commented [KK88]: Οκ οπότε ας δώσετε και το σχετικό URL με τη μορφή footnote

Field Code Changed

διαχειριστής που εισάγει/ενημερώνει πλατφόρμες), οι οποίες επιτρέπουν τη συντήρηση και επέκταση του αποθετηρίου πλατφορμών στο χρόνο.

Με βάση τα παραπάνω, το FaaStener αποτελεί ένα ώριμο εργαλείο τεχνολογικής επισκόπησης/ταξινόμησης και σύγκρισης πλατφορμών, ενώ το σύστημα της παρούσας εργασίας αντιμετωπίζει το πρόβλημα ως **ολοκληρωμένο πληροφοριακό σύστημα λήψης απόφασης και υλοποίησης**, συνδυάζοντας αναζήτηση, κατάταξη και πραγματική διάταξη συναρτήσεων

3.2 Διάταξη Serverless Συναρτήσεων

Η δεύτερη κατηγορία εργαλείων αφορά τη διάταξη serverless συναρτήσεων σε πολλαπλά νέφη. Ένα χαρακτηριστικό παράδειγμα αποτελεί το έργο Multi-Cloud Serverless Deployment[29]. Το εργαλείο αυτό επιτρέπει την αυτόματη διάταξη συναρτήσεων σε διαφορετικούς παρόχους cloud (AWS, Azure, GCP), μέσω ενός ενιαίου configuration αρχείου, αξιοποιώντας AWS CLI, Azure CLI, και Google Cloud SDK. Στόχος του είναι να αυτοματοποιήσει τη διαδικασία ανάπτυξης χωρίς να απαιτείται η επαναγραφή κώδικα για κάθε πλατφόρμα.

Παρότι η λύση αυτή είναι πρακτική, παρουσιάζει ορισμένους περιορισμούς:

- Απαιτεί τοπική εγκατάσταση και ρύθμιση CLI εργαλείων για κάθε πάροχο.
-
- Δεν διαθέτει γραφικό περιβάλλον χρήστη (UI) ούτε δυνατότητα επιλογής ή παραμετροποίησης από φυλλομετρητή (browser).
-
- Δεν υποστηρίζει μηχανισμό αναζήτησης ή αξιολόγησης πλατφορμών πριν τη διάταξη.

Το σύστημα που αναπτύχθηκε στην παρούσα εργασία ενσωματώνει τη δυνατότητα διάταξης συναρτήσεων μέσω του Serverless Framework, επιτρέποντας multi-cloud ανάπτυξη με ενιαία περιγραφή (YAML configuration), όπως ισχύει και στο συγκρινόμενο εργαλείο. Αλλά προσφέρει επίσης web-based διαχείριση ενώ παρέχονται στον χρήστη οι δυνατότητες επιλογής της πλατφόρμας (μετά από ενδεχόμενη σύγκρισή τους), παροχής παραμέτρων λειτουργίας (π.χ. μέγεθος μνήμης, runtime περιβάλλον) και διάταξης απευθείας από το περιβάλλον του συστήματος, χωρίς εξωτερικά εργαλεία.

[29] https://github.com/ansjin/multi-cloud-serverless-deployment

Formatted: Greek

Formatted: Greek

Formatted: Justified

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Field Code Changed

Ειδικότερα, σε σχέση με το εργαλείο Multi-Cloud Serverless Deployment, η προτεινόμενη λύση προσφέρει:

- Εύχρηστη διαδικτυακή διεπαφή με ευρεία καθοδήγηση χρήστη,
- Ενοποιημένη προσέγγιση αξιολόγησης και διάταξης,
- Ενσωμάτωση με backend που αποθηκεύει και αναλύει δεδομένα πλατφορμών,
- Δυνατότητα επέκτασης σε νέους παρόχους με ελάχιστη παραμετροποίηση.

Formatted: Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0,63 cm + Indent at: 1,27 cm

Formatted: Greek

4. Ανάπτυξη Συστήματος

Commented [KK89]: Θα μπορούσατε να πείτε πως χρησιμοποιήθηκε το μοντέλο του καταρράκτη για την ανάπτυξη της εφαρμογής με διακριτές φάσεις ανάπτυξης. Και πως η δομή του τρεχοντος κεφαλαιου ακολουθεί τις φασεις αυτές, προσδιοριζοντας & αναλύοντας τα αποτελέσματά τους.

Το παρόν κεφάλαιο ακολουθεί τη λογική του μοντέλου ανάπτυξης **Καταρράκτη (Waterfall)**, στο οποίο η ανάπτυξη ενός πληροφοριακού συστήματος οργανώνεται σε **διακριτές και διαδοχικές φάσεις**, όπου κάθε φάση ολοκληρώνεται πριν ξεκινήσει η επόμενη. Συγκεκριμένα, η δομή του κεφαλαίου αντιστοιχεί στις φάσεις: **(α) Ανάλυση Απαιτήσεων**, όπου προσδιορίζονται οι λειτουργικές και μη λειτουργικές απαιτήσεις και οι περιορισμοί του συστήματος, **(β) Σχεδίαση**, όπου καθορίζεται η αρχιτεκτονική και το πληροφοριακό/λογικό μοντέλο (π.χ. υπηρεσίες, endpoints, διαγράμματα, δομές δεδομένων), και **(γ) Υλοποίηση**, όπου παρουσιάζεται πώς τα παραπάνω υλοποιήθηκαν στο backend και στο frontend (τεχνολογίες, δομή κώδικα, βασικοί μηχανισμοί). Με αυτόν τον τρόπο, το κεφάλαιο δεν περιγράφει μόνο τα επιμέρους τεχνικά στοιχεία, αλλά **προσδιορίζει και αναλύει τα παραδοτέα/αποτελέσματα κάθε φάσης**, συνθέτοντας μια συνεκτική εικόνα της διαδικασίας ανάπτυξης του συστήματος «Serverless Platform Selector» που κατασκευάσαμε στο πλαίσιο της διπλωματικής μας εργασίας.

4.1 Ανάλυση Απαιτήσεων

Η ενότητα αυτή παρουσιάζει τις λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος «Serverless Platform Selector». Η ανάλυση απαιτήσεων αποτυπώνει τις ανάγκες του χρήστη, τους στόχους της εφαρμογής και τους περιορισμούς της υλοποίησης, ώστε να θεμελιωθεί σωστά η φάση/δραστηριότητα της σχεδίασης που ακολουθεί.

4.1.1 Σκοπός Συστήματος

Ο κύριος σκοπός του συστήματος είναι η παροχή μιας ενιαίας πλατφόρμας που θα επιτρέπει:

- Αναζήτηση και σύγκριση serverless πλατφορμών βάσει τεχνικών χαρακτηριστικών/κριτηρίων.
- Πολυκριτηριακή κατάταξη (ranking) πλατφορμών με βάρη που αναθέτει ο χρήστης στα τεχνικά κριτήρια.
- Αυτόματη διάταξη (deployment) serverless functions σε AWS, Azure ή Google Cloud.
- Ασφαλή διαχείριση και πρόσβαση χρηστών με χρήση JWT.

Το σύστημα υποστηρίζει εξίσου τεχνικούς και μη τεχνικούς χρήστες, καθώς όλη η διαδικασία —αναζήτηση, σύγκριση, επιλογή πλατφόρμας και αυτόματη διάταξη (deployment) serverless functions— πραγματοποιείται μέσω γραφικού περιβάλλοντος χωρίς να απαιτούνται εξειδικευμένες γνώσεις.

Formatted: Greek

Ο χρήστης αρκεί να επιλέξει τα επιθυμητά χαρακτηριστικά (runtime, πάροχο, περιοχή κ.λπ.) και να ανεβάσει το αρχείο της συνάρτησης, ενώ το σύστημα αναλαμβάνει αυτόματα τη δημιουργία του configuration και την εκτέλεση του deployment στους παρόχους νέφους (AWS, Azure, GCP). Έτσι, η πλατφόρμα παραμένει προσβάσιμη σε όλους, ανεξάρτητα από τεχνικό υπόβαθρο.

4.1.2 Actors & Εξωτερικά Συστήματα

- **User (Εγγεγραμμένος Χρήστης)**: εκτελεί αναζήτηση, φιλτράρισμα και κατάταξη (serverless) πλατφορμών, μπορεί να βλέπει την περιγραφή της κάθε πλατφόρμας ενώ έχει και τη δυνατότητα διάταξης συναρτήσεων σε μία από αυτές τις πλατφόρμες.
- **Administrator**: διαθέτει επιπλέον δυνατότητες δημιουργίας, ενημέρωσης και διαγραφής πλατφορμών καθώς και διαχείρισης εγγεγραμμένων χρηστών (CRUD + αναζήτηση).
- **Serverless Platforms (AWS Lambda, Azure Functions, Google Cloud Functions)**: εξωτερικές υπηρεσίες στις οποίες πραγματοποιείται η διάταξη των συναρτήσεων.

Commented [KK90]: Ας εξηγήσετε αν αυτος είναι ένας εγγεγραμμένος χρηστης του συστήματος.
Αν ναι, τότε υπάρχει και ο επισκέπτης; Κι αν υπάρχει, τι δυνατοτητες έχει αυτός;

Commented [KK91]: Αν έχουμε και εγγεγραμμένους χρηστες, τοτε λογικά θα πρέπει να υποστηρίζεται και η διαχειριση χρηστών (crud + search).

Commented [KK92]: Προτιμώ να τις πειτε Serverless Platforms άρα μιλάμε για πλατφορμες όπου εκεί μπορει να γίνει η διάταξη των συναρτήσεων.

4.1.3 Λειτουργικές Απαιτήσεις

Οι λειτουργικές απαιτήσεις ομαδοποιούνται ανά επιχειρησιακή οντότητα και συνοδεύονται από περιγραφή εισόδων–εξόδων, ώστε να καταγράφεται ξεκάθαρα η σημασιολογία κάθε λειτουργίας

A. Απαιτήσεις που αφορούν τη Διαχείριση Χρηστών

1. Εγγραφή Χρήστη (Registration)

- **Είσοδος:** email, password, full name.
- **Έξοδος:** επιβεβαίωση εγγραφής
- **Σκοπός:** εγγραφή χρήστη στο σύστημα ώστε αυτός να έχει έπειτα πρόσβαση στις λειτουργίες αναζήτησης, κατάταξης και διάταξης συναρτήσεων.

2. Σύνδεση Χρήστη (Login) με JWT

- **Είσοδος:** email, password.
- **Έξοδος:** JWT token για stateless εξουσιοδότηση.

- **Ρόλος:** παρέχει ασφαλή πρόσβαση του χρήστη σε προστατευμένα endpoints του συστήματος.

3. Αλλαγή Στοιχείων Προφίλ

- **Είσοδος:** νέο όνομα, νέα στοιχεία λογαριασμού.
- **Έξοδος:** ενημερωμένο προφίλ χρήστη.

4. Αλλαγή Κωδικού

- **Είσοδος:** current password, new password.
- **Έξοδος:** επιβεβαίωση αλλαγής του κωδικού.

5. Επαναφορά Κωδικού (Password Reset)

- **Είσοδος:** διεύθυνση email χρήστη.
- **Έξοδος:** token επαναφοράς μέσω του οποίου παρέχεται η δυνατότητα ορισμού νέου κωδικού.

6. ~~6.~~ Διαγραφή Χρήστη (User Deletion)

- **Είσοδος:** userId.
- **Έξοδος:** επιβεβαίωση διαγραφής χρήστη.

7. ~~7.~~ Λήψη Λίστας Χρηστών (User Listing – Admin)

- **Είσοδος:** αίτημα από χρήστη με ρόλο ADMIN.
- **Έξοδος:** λίστα χρηστών του συστήματος.

8. ~~8.~~ Προβολή προφίλ Χρήστη

- **Είσοδος:** αίτημα από αυθεντικοποιημένο χρήστη
- **Έξοδος:** στοιχεία προφίλ χρήστη

Formatted: Greek

Commented [KK93]: Λείπουν απαιτήσεις για την διαγραφή χρηστών, την εμφανιση του προφίλ τους, την ενεργοποίηση/απενεργοποίησή τους και ενδεχομένως για την αναζήτηση / λήψη λίστας χρηστών (για τον διαχειριστή).

Commented [KK94R93]: OK, βλέπω πως εισάγατε τις απαιτήσεις αλλά προσέξτε την ευθυγράμμιση διότι έχει σπάσει (δειτε που αρχιζει η 5. ... και που η 6., 7. & 8.

Formatted: Font: (Default) Times New Roman, 12 pt, Bold

Formatted: List Paragraph, Left, Indent: Left: 3,17 cm, Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0,63 cm + Tab after: 1,27 cm + Indent at: 1,27 cm, Tab stops: 3,81 cm, List tab + Not at 1,27 cm

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Bold

Formatted: List Paragraph, Outline numbered + Level: 4 + Numbering Style: Bullet + Aligned at: 3,81 cm + Indent at: 4,44 cm

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: English (United States)

Formatted: List Paragraph, Indent: Left: 3,81 cm, Hanging: 0,63 cm, Outline numbered + Level: 4 + Numbering Style: Bullet + Aligned at: 3,81 cm + Indent at: 4,44 cm

Formatted: Font: (Default) Times New Roman, 12 pt, Bold

Formatted: List Paragraph, Indent: Left: 4,44 cm

Formatted: Font: (Default) +Body (Aptos), 11 pt, Not Bold, Greek

Formatted: List Paragraph, Indent: Left: 4,44 cm, No bullets or numbering

Formatted ...

Formatted ...

Formatted ...

Formatted ...

Formatted ...

Formatted: Font: Bold

Formatted ...

Formatted ...

Formatted: English (United States)

9. ~~9.~~ **Ενεργοποίηση / Απενεργοποίηση Χρήστη**

- **Είσοδος:** κατάσταση ενεργοποίησης.

Έξοδος: ενημερωμένη κατάσταση χρήστη.

-
-

Β. Απαιτήσεις Διαχείρισης Πλατφορμών (μόνο για ADMIN)

1. ~~10.~~ **Δημιουργία Πλατφόρμας (Create Platform)**
   - **Είσοδος:** όνομα πλατφόρμας, περιγραφή, κατηγορία, τεχνικά χαρακτηριστικά (JSON).
   - **Έξοδος:** επιβεβαίωση της δημιουργίας της πλατφόρμας.
2. ~~11.~~ **Ενημέρωση Πλατφόρμας (Update Platform)**
   - **Είσοδος:** νέα χαρακτηριστικά/περιγραφή.
   - **Έξοδος:** ενημερωμένη εμφάνιση πλατφόρμας.
3. ~~12.~~ **Διαγραφή Πλατφόρμας (Delete Platform)**
   - **Είσοδος:** platform id (αναγνωριστικό πλατφόρμας)
   - **Έξοδος:** επιβεβαίωση διαγραφής.
4. ~~13.~~ **Αναζήτηση Πλατφορμών από Administrator**
   - **Είσοδος:** φίλτρα αναζήτησης.
   - **Έξοδος:** λίστα πλατφορμών.
5. ~~14.~~ **Προβολή Πλατφορμών (View Platforms) — USER & ADMIN**
   - **Είσοδος:** αίτημα χρήστη ή διαχειριστή.
   - **Έξοδος:** λίστα διαθέσιμων serverless πλατφορμών.
   -
6. ~~15.~~ **Προβολή Συγκεκριμένης Πλατφόρμας (View Platform Details) — USER & ADMIN**

Formatted: Greek

Formatted: Font: (Default) Times New Roman, 12 pt, Bold

Formatted: Left, Indent: Left: 3,81 cm, Outline numbered + Level: 1 + Numbering Style: Bullet + Aligned at: 1,27 cm + Tab after: 1,9 cm + Indent at: 1,9 cm, Tab stops: 3,81 cm, List tab + Not at 1,9 cm

Commented [KK95]: Λογικά έχετε υλοποιήσει την εμφάνιση πλατφορμας, οπότε θα πρέπει να μπει η σχετική απαιτηση. Σημειώνω πως η εμφάνιση θα πρεπει να γινεται και σε χρηστες και σε διαχειριστές.

Commented [KK96R95]: OK - όμως έχουμε πάλι ζητημα ευθυγράμμισης

Formatted: Font: (Default) Times New Roman, 12 pt, Bold

Formatted: List Paragraph, Indent: First line: 0,08 cm, Outline numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 2,54 cm + Tab after: 3,17 cm + Indent at: 3,17 cm, Tab stops: 4 cm, List tab + Not at 3,17 cm

Formatted: Outline numbered + Level: 1 + Numbering Style: Bullet + Aligned at: 3,81 cm + Tab after: 4,44 cm + Indent at: 4,44 cm, Tab stops: Not at 4,54 cm

Formatted: Outline numbered + Level: 3 + Numbering Style: Bullet + Aligned at: 3,81 cm + Indent at: 4,44 cm, Tab stops: Not at 4,54 cm

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

Formatted

- **Είσοδος:** platform id.
- **Έξοδος:** πλήρη στοιχεία της πλατφόρμας (όνομα, περιγραφή, κατηγορία, τεχνικά χαρακτηριστικά).

C. Μηχανισμός Αναζήτησης & Φιλτραρίσματος Πλατφορμών

16. Full-Text Search

- **Σκοπός:** αναζήτηση πλατφορμών βάσει ονόματος, περιγραφής και λέξεων-κλειδιών.
- **Είσοδος:** search query (επερώτηση αναζήτησης).
- **Έξοδος:** κατάλογος/λίστα σχετικών πλατφορμών ταξινομημένων ως προς την σχετικότητά τους.

217. Fuzzy Matching

- **Σκοπός:** παρέχει ανθεκτικότητα σε ορθογραφικά λάθη και υποστηρίζει μη ακριβείς αναζητήσεις.
- **Είσοδος:** search query με πιθανές αποκλίσεις.
- **Έξοδος:** αποτελέσματα ταξινομημένα με βάση το σκορ ομοιότητάς τους (similarity score).

D. Μηχανισμός Πολυκριτηριακής Κατάταξης (Ranking)

18. Πολυκριτηριακή Κατάταξη με Σταθμισμένα Βάρη

- **Είσοδος:** βάρη κριτηρίων που ορίζει ο χρήστης (π.χ. μνήμη, timeout, κόστος, runtime).
- **Έξοδος:** ταξινομημένη λίστα πλατφορμών με συνολικό score.
- **Λειτουργία:**
  - κανονικοποίηση αριθμητικών τιμών,
  - αντιστοίχιση μη αριθμητικών τιμών (score mapping) σε αριθμητικές,

Formatted: Greek

Formatted: Indent: Left: 3,81 cm, Outline numbered + Level: 1 + Numbering Style: Bullet + Aligned at: 1,27 cm + Tab after: 1,9 cm + Indent at: 1,9 cm, Tab stops: 3,81 cm, List tab + Not at 1,9 cm

Formatted: Indent: Left: 3,81 cm, Outline numbered + Level: 1 + Numbering Style: Bullet + Aligned at: 1,27 cm + Tab after: 1,9 cm + Indent at: 1,9 cm, Tab stops: 3,81 cm, List tab + Not at 1,9 cm

Commented [KK97]: Τι σημαινει αυτό; Αντιστοιχιζονται μη αριθμητικές τιμές σε αριθμητικές;

Commented [AK98R97]: αναφέρεται στη μετατροπή κατηγορικών/μη αριθμητικών χαρακτηριστικών (π.χ. runtime, provider, scaling mode) σε αριθμητικά σκορ

- υπολογισμός τελικής βαθμολογίας.

Ε. Απαιτήσεις Διάταξης Serverless Συναρτήσεων

1~~9~~. Μεταφόρτωση Συνάρτησης (Upload Function)

- **Είσοδος:** Base64-encoded ZIP με τον κώδικα της συνάρτησης και τα metadata (runtime, provider, function name).

  Η μεταφόρτωση μπορεί να πραγματοποιείται είτε:

  - ως **Base64-encoded string**, είτε
  - μέσω **multipart/form-data upload**.

- **Έξοδος:** Επιβεβαιωση μεταφόρτωσης
- **Σκοπός:** αποστολή ενιαίου αιτήματος από τον χρήστη χωρίς multipart parsing.

2~~0~~. Δημιουργία και Παραμετροποίηση serverless.yml

- **Είσοδος:** runtime, provider, memory, timeout, events.
- **Έξοδος:** serverless.yml έτοιμο για deployment.

3~~21~~. Εκτέλεση Deployment / Διάταξης

- **Είσοδος:** configuration, package (όπως έχουν ήδη διαμορφωθεί από τον χρήστη – δείτε προηγούμενες 2 απαιτήσεις).
- **Έξοδος:**
  - URL endpoint της συνάρτησης,
  - logs εκτέλεσης,
  - status (success/failure).
- **Περιγραφή:**
  το backend εκτελεί το Serverless Framework και δρομολογεί τη διάταξη στην επιλεγμένη πλατφόρμα (AWS, Azure, GCP).

4~~22~~. Αποθήκευση Ιστορικού Διατάξεων

- **Είσοδος:** deployment result, timestamp, provider, endpoint.

Formatted: Greek

Commented [KK99]: Οπότε τα μεταδεδομένα παρέχονται μέσα στο ZIP σε κάποιο αρχειο; Ή είναι μέρος της φορμας; Το ρωτω διότι δεν έχουμε multipart parsing, σωστά;

Commented [AK100R99]: Εχουμε 2 επιλογες απευθειας string base64 ή multipart

Commented [KK101R99]: Ποιο κάτω όμως στο σκοπό αναφέρεται κατι για μη multi-part parsing, το οποίο μπερδεύει. Μήπως να το αφαιρέσετε; Επίσης, μπορείτε να πείτε στην απαίτηση πως θα πρεπει να υποστηρίζονται και τα δύο

- **Έξοδος:** καταχώρηση deployment στη βάση. Η καταγραφή εκτελείται αυτόματα από το backend ως side-effect της διάταξης, χωρίς πρόσθετη ενέργεια του χρήστη

5~~23~~. Προβολή Ιστορικού & Αποτελεσμάτων

- **Είσοδος:** αίτημα χρήστη ή διαχειριστή, με δυνατότητα προαιρετικών φίλτρων (π.χ. κατάσταση διάταξης – status).
- **Έξοδος:** λίστα διατάξεων που περιλαμβάνει:
  - την κατάσταση εκτέλεσης (PENDING, SUCCESS, FAILED),
  - το URL της αναπτυγμένης συνάρτησης,
  - καθώς και τα σχετικά logs της διαδικασίας deployment.
- **Σκοπός:** επιτρέπει την παρακολούθηση του ιστορικού διατάξεων των serverless συναρτήσεων.
  - Οι απλοί χρήστες (USER) μπορούν να βλέπουν μόνο τις διατάξεις που έχουν πραγματοποιήσει οι ίδιοι.
  - Οι διαχειριστές (ADMIN) έχουν δυνατότητα προβολής όλων των διατάξεων του συστήματος.

4.1.4 Μη Λειτουργικές Απαιτήσεις

Οι μη λειτουργικές απαιτήσεις συγκροτούν το πλαίσιο ποιότητας του συστήματος και ομαδοποιούνται ανά κατηγορία (ασφάλεια, απόδοση, χρηστικότητα, φορητότητα, επεκτασιμότητα, συντηρησιμότητα).

Α. Ασφάλεια (Security)

Στόχος: Υψηλό επίπεδο ασφάλειας και προστασίας δεδομένων.

Σχετικές Απαιτήσεις:

- **Ασφαλής αυθεντικοποίηση** μέσω Spring Security και JWT.
  - *Εξασφαλίζει stateless έλεγχο πρόσβασης και προστατευμένα endpoints.*

Formatted: Greek

Commented [KK102]: Αυτό είναι side-effect. Επισης, ας ξεκαθαρισετε αν αυτή η λειτουργια επιτελειται από το συστημα στο background χωρις ενεργή συμμετοχή του χρηστη.

Commented [KK103]: Θα μπορούσατε να πειτε και κάτι για την φορητοτητα του συστήματος εφοσον αυτό είναι εφικτο (και έχει προφανώς νόημα με βάση και τις τεχνολογιες ανάπτυξης που χρησιμοποιειτε).

Επιπλέον, δεν λέτε κάτι για την χρηστικότητα του συστήματος ενώ την εκθειάζατε στην σύγκριση με σχετικές εργασίες.

- **Κρυπτογράφηση/Κατακερματισμός κωδικών πρόσβασης** (BCrypt hashing).
- **Role-Based Access Control (RBAC)**:
    - Χρήστες με ρόλο *USER* έχουν πρόσβαση σε βασικές λειτουργίες.
    - Χρήστες με ρόλο *ADMIN* διαχειρίζονται πλατφόρμες & χρήστες.
- **Προστασία από επιθέσεις web** μέσω Spring Security και validation επιπέδου εφαρμογής:
    - προστασία από SQL Injection,
    - προστασία από XSS,
    - CSRF-disabled σε JWT περιβάλλον (ορθό λόγω stateless design).
- **Secure data exchange:** όλα τα δεδομένα (αυθεντικοποίησης) μεταφέρονται σε JSON και πιστοποιούνται μέσω JWT. Γίνεται χρήση TLS/HTTPS για την ασφαλή ανταλλαγή όλων των δεδομένων που διαχειρίζεται η εφαρμογή.

Β. Απόδοση

Στόχος: Καλή απόδοση συστήματος υπό φυσιολογικό και αυξημένο φορτίο.

Σχετικές Απαιτήσεις:

- **Χρόνος απόκρισης αναζήτησης** (Full-Text & Fuzzy Matching):
    - *< 200 ms για έως 10 ταυτόχρονους χρήστες.*
- **Χρόνος απόκρισης κατάταξης (ranking):**
    - *< 300 ms για λίστα έως 200 πλατφορμών.*
- **Χρόνος διάταξης (deployment) συνάρτησης:**

Formatted: Greek

Commented [KK104]: Γίνεται χρηστη TLS/HTTPS για την ασφαλή ανταλλαγή των δεδομένων;

- **Άμεση απόκριση** (acknowledgement): το backend επιστρέφει επιβεβαίωση έναρξης (π.χ. "deployment started" + deploymentId) εντός 1–2 sec
- **Ολοκλήρωση διάταξης:** Επειδή ο συνολικός χρόνος ολοκλήρωσης εξαρτάται από τον πάροχο (AWS/Azure/GCP) και το μέγεθος του πακέτου, δεν μπορούν να εξασφαλιστούν συγκεκριμένοι περιορισμοί. Αλλά το σύστημα καταγράφει τον χρόνο ολοκλήρωσης και επιστρέφει τελικό status (SUCCESS/FAILED), endpoint URL (αν παραχθεί) και logs.

- **Υποστήριξη ταυτόχρονων χρηστών (concurrent users):**
  - το σύστημα πρέπει να υποστηρίζει *τουλάχιστον 50* ταυτόχρονες αναζητήσεις χωρίς performance degradation.
- **Χρόνοι απόκρισης για διαχείριση πλατφορμών (ADMIN – CRUD):** Δημιουργία/Ενημέρωση/Διαγραφή πλατφόρμας: < 300 ms για τυπική εγγραφή (χωρίς βαριά επεξεργασία).

Ανάκτηση λίστας πλατφορμών για διαχείριση: < 200 ms με pagination.

- **Χρόνοι απόκρισης για διαχείριση χρηστών (USER/ADMIN):** Login: < 300 ms (έκδοση JWT). Registration: < 500 ms (δημιουργία χρήστη + validation). Προβολή λίστας χρηστών (ADMIN, αν υφίσταται): < 300 ms με pagination.

Γ. Stateless (Μη Καταστατική) Επικοινωνία

Σχετική Απαίτηση:

- Όλες οι κλήσεις UI → backend πρέπει να περιλαμβάνουν:
  - το JWT token για πιστοποίηση,
  - την πλήρη αναπαράσταση των δεδομένων της εκάστοτε λειτουργίας.
- Το backend **δεν διατηρεί session state**, επομένως κάθε αίτημα πρέπει να περιέχει όλα τα απαραίτητα δεδομένα.

Formatted: Greek

Commented [KK105]: Μπορειτε να πειτε κάτι και για την διάταξη;

Επίσης, εν γένει, έχετε αφήσει τελείως έξω άλλες λειτουργίες (διαχειριση χρηστων & πλατφορμών)

Οφέλη: καλύτερη κλιμάκωση / ανεξαρτησία κόμβων backend.

Δ. Χρηστικότητα (Usability)

Στόχος: Φιλικότητα προς τεχνικούς και μη τεχνικούς χρήστες.

Σχετικές Απαιτήσεις:

- **Καθαρό και απλό UI** με προσανατολισμό στη ροή "Search → Compare → Rank → Deploy".
- **Μη τεχνικοί χρήστες** μπορούν να εκτελέσουν deployment χωρίς γνώση CLI/Cloud Providers.
- **Auto-complete και dynamic filtering** για μείωση λαθών στην αναζήτηση.
- **Εμφάνιση κατανοητών μηνυμάτων σφάλματος** (π.χ. σε login, invalid weights, invalid package).
- **Responsive UI** για χρήση από κινητές και desktop συσκευές.

Ε. Κλιμακωσιμότητα, Επεκτασιμότητα & Αρθρότητα (Scalability, Extensibility & Modularity)

Στόχος: εύκολη προσθήκη νέων χαρακτηριστικών, παρόχων και κριτηρίων.

Σχετικές Απαιτήσεις:

- **Modular backend με CSR αρχιτεκτονική (Controller–Service–Repository)**:
    - *Επιτρέπει εύκολη συντήρηση και επέκταση, καθώς κάθε επίπεδο έχει σαφώς καθορισμένες ευθύνες.*
    - *Η αλλαγή σε ένα επίπεδο δεν επηρεάζει τα υπόλοιπα.*
- **Δυνατότητα προσθήκης νέων providers** (π.χ. Cloudflare Workers, OpenFaaS) χωρίς αλλαγές στον βασικό κορμό του backend.
- **Δυνατότητα επέκτασης των κριτηρίων ranking** με νέες κατηγορίες δεδομένων (μέσω JSONB).

- **Κλιμάκωση backend σε πολλαπλά στιγμιότυπα** λόγω stateless αρχιτεκτονικής.

. Φορητότητα (Portability)

Στόχος: Εύκολη μεταφορά και εκτέλεση του συστήματος σε διαφορετικά περιβάλλοντα.

Σχετικές Απαιτήσεις:

- Το backend λειτουργεί σε οποιοδήποτε περιβάλλον που υποστηρίζει:
    - Java 17+
    - Spring Boot
    - PostgreSQL
- Η εφαρμογή μπορεί να τρέξει εξ ολοκλήρου σε Docker containers χωρίς αλλαγές στον κώδικα.
- Το frontend (React) μπορεί να αναπτυχθεί σε οποιοδήποτε static hosting (Netlify, Vercel, S3, custom Nginx).
- Το σύστημα δεν εξαρτάται από συγκεκριμένο πάροχο cloud.

Z. Συντηρησιμότητα (Maintainability)

Στόχος: εύκολη εξέλιξη, εντοπισμός και επίλυση προβλημάτων.

Μη Λειτουργικές Απαιτήσεις:

- **Διαχωρισμός ευθυνών (CSR)** για ευκολότερη εντοπισμό σφαλμάτων και ανεξάρτητη ανάπτυξη υπηρεσιών.
- **Καθαρή δομή κώδικα**, ομοιόμορφη σύμβαση ονοματοδοσίας (naming convention) και JavaDoc.
- **Έλεγχος εξαρτήσεων μέσω Maven** για σταθερότητα και εύκολη ενημέρωση.
- **Log traceability**: όλα τα σημαντικά event (login, ranking, deployment) καταγράφονται.

- **Unit tests & Integration tests** για βασικές υπηρεσίες (ranking, auth, platform management).

4.1.5 Περιορισμοί

A. Πληροφοριακοί Περιορισμοί

1. Ενιαίο Πληροφοριακό Μοντέλο Πλατφορμών

Το σύστημα υποστηρίζει **ενιαίο πληροφοριακό σχήμα** (schema) για την περιγραφή των serverless πλατφορμών.

Οι ιδιότητες των πλατφορμών που αποθηκεύονται σε μορφή JSONB πρέπει:

- να αντιστοιχούν σε **επιτρεπτά κλειδιά**,
- και οι τιμές τους να ανήκουν σε **προκαθορισμένες απαριθμήσεις (Enums)**.

B. Περιορισμοί Ανάπτυξης και Αρχιτεκτονικής

2. Ανάπτυξη βάσει του μοντέλου Καταρράκτη (Waterfall Model)

Η συνολική ανάπτυξη του συστήματος ακολουθεί το μοντέλο του καταρράκτη, δηλαδή τις φάσεις:

1. ανάλυση απαιτήσεων,
2. σχεδιασμός,
3. υλοποίηση,
4. έλεγχος,
5. παράδοση.

Ο περιορισμός αυτός αποτυπώνει τη μεθοδολογική προσέγγιση ανάπτυξης, όπου κάθε φάση ολοκληρώνεται πριν την επόμενη.

3. RESTful Backend με συγκεκριμένο επίπεδο ωριμότητας (Richardson Maturity Model)

Το backend υλοποιείται ως **RESTful API** με ωριμότητα Επιπέδου 2 του μοντέλου Richardson:

Formatted: Greek

Commented [KK106]: & Integration tests?

Commented [KK107]: Θα προτεινα να προσθέσετε 2 επιπλέον περιορισμούς:
(α) ανάπτυξη με βάση το μοντέλο του καταρράκτη
(β) υλοποίηση backend με τη μορφή RESTful API με συγκεκριμένο επίπεδο ωριμότητας βάσει το μοντέλο ωριμότητας του Richardson

- χρήση διαφορετικών HTTP μεθόδων (GET, POST, PUT, DELETE) ανά είδος/φύση λειτουργίας,
- χρήση κατάλληλων endpoints & ονομάτων πόρων (/api/platforms, /api/deployments κ.λπ.),
- stateless επικοινωνία με JWT authentication.

4.2. Σχεδίαση

Η σχεδίαση του συστήματος βασίζεται σε μια **πολυστρωματική (multi-layer)** αρχιτεκτονική, η οποία επιλέχθηκε ώστε να καλύπτει τις απαιτήσεις αρθρωτότητας, επεκτασιμότητας και ευκολίας συντήρησης. Η συγκεκριμένη αρχιτεκτονική αποτελεί ενδεδειγμένη προσέγγιση για web-based συστήματα, καθώς επιτρέπει τον σαφή διαχωρισμό ευθυνών μεταξύ των στρωμάτων και περιορίζει την αλληλεξάρτηση μεταξύ των συστατικών.

Το σύστημα οργανώνεται σε τέσσερα κύρια στρώματα:

1. **Στρώμα Παρουσίασης (Presentation Layer – Frontend)** Παρέχει τη γραφική διεπαφή αλληλεπίδρασης του χρήστη με το σύστημα. Όλη η επικοινωνία με το backend γίνεται μέσω REST API κλήσεων.
2. **Στρώμα Επιχειρησιακής Λογικής (Business Logic Layer – Backend Services)** Ακολουθεί το αρχιτεκτονικό μοτίβο **Controller–Service–Repository (CSR)**. Το μοτίβο αυτό επιτρέπει τον καθαρό διαχωρισμό ευθυνών:
   - οι Controllers διαχειρίζονται HTTP αιτήσεις,
   - τα Services υλοποιούν την επιχειρησιακή λογική,
   - τα Repositories χειρίζονται την πρόσβαση στα δεδομένα.
3. **Στρώμα Δεδομένων & Υποδομής (Data Layer & Infrastructure)** Περιλαμβάνει το σχεσιακό σύστημα βάσης δεδομένων (**Database**), το οποίο αποθηκεύει χρήστες, πλατφόρμες, ιστορικό διατάξεων και τα τεχνικά χαρακτηριστικά των πλατφορμών σε μορφή JSONB καθώς και την μηχανή διάταξης (**Deployment Engine**) με την μορφή ενός τοπικά εγκατεστημένου εργαλείου / πλαίσιου (**Serverless Framework CLI**), το οποίο επιλέχθηκε λόγω της δυνατότητάς του να διατάσσει συναρτήσεις σε πολλές servrless πλατφόρμες και το οποίο καλείται από την υπηρεσία διάταξης.

Formatted: Greek

Commented [KK108]: Προτεινω η σχεδιαση να αρχίσει με την αρχιτεκτονική και μετά να αναφερθεί σε σχεδιαστικές λεπτομέρειες των συστατικών ή της λειτουργικότητάς τους.

Καλό είναι επίσης να προσδιοριστει ρητά και το πληροφοριακό μοντέλο του συστήματος (πχ. μέσω class/ER diagram). Εκτος φυσικά από το μοντέλο αρχιτεκτονικής που θα πρέπει οπωσδήποτε να υπάρχει (πχ. μέσω block ή component diagram)

Commented [KK109]: Θα προτεινα αυτό να το λέγατε Στρώμα Δεδομένων & Υποδομής. Εδώ ουσιαστικά υπάρχει η βάση δεδομένων, το συστημα αρχειων και το Deployment Engine ως συστατικό που το επαναχρησιμοποιειτε και εκτελειτε μέσω του command line.

Formatted: Greek

4.2.1 Γενική Αρχιτεκτονική του Συστήματος

Η συνολική αρχιτεκτονική απεικονίζεται μέσω διαγράμματος συστατικών (component diagram), όπου φαίνεται η αλληλεπίδραση μεταξύ:

- Frontend,
- Backend (CSR services),
- Database,
- Deployment Engine (αυτό και το προηγούμενο απαρτίζουν το τρίτο στρώμα στην αρχιτεκτονική μετά το backend & το frontend),
- FaaS Providers (εξωτερικό σύστημα).

Το σύστημα είναι **client–server**, stateless ως προς το backend, και βασίζεται σε JSON-based REST επικοινωνία.

Εξειδικευμένη Αρχιτεκτονική Backend (CSR)

Στο backend εφαρμόζεται εξειδικευμένη αρχιτεκτονική CSR.

Controller Layer

Υπεύθυνο για:

- λήψη αιτημάτων από το frontend,
- έλεγχο πρόσβασης (μέσω JWT roles),
- προώθηση των αιτημάτων στις κατάλληλες υπηρεσίες.

Service Layer

Περιλαμβάνει επιμέρους υπηρεσίες όπως:

- User Management Service,
- Platform Service,
- Ranking Service,
- Deployment Service.

Καθεμία υλοποιεί μια συγκεκριμένη επιχειρησιακή περιοχή.

Repository Layer

Επικοινωνεί με την σχεσιακή βάση δεδομένων (επιλέχθηκε η PostgreSQL) μέσω Spring Data JPA και διαχειρίζεται:

- χρήστες,
- πλατφόρμες,
- deployments (δηλ. διατάξεις συναρτήσεων),
- JSON χαρακτηριστικά των πλατφορμών.

Η υλοποίηση JPA ενισχύει την αρθρωτότητα, καθώς οποιαδήποτε αλλαγή βάσης δεδομένων δεν επηρεάζει τα ανώτερα στρώματα.

Στο επίπεδο του backend, η λειτουργικότητα οργανώνεται σε βασικές ενότητες (services/modules): (α) AuthModule: Διαχείριση Χρηστών & Ασφάλεια (User Management & Security), (β) Πλατφορμών (Platform Module), (δ) Διάταξης (Deployment Module), η οποία διασυνδέεται με εξωτερικά συστήματα (FaaS providers) μέσω της Deployment Engine (Μηχανής Διάταξης).

4.2.1.1 Ενότητα (Διαχείρισης) Πλατφορμών

Η Ενότητα Πλατφορμών αποτελεί ένα από τα κεντρικές ενότητες του backend του συστήματος και είναι υπεύθυνη για τη διαχείριση των τεχνικών χαρακτηριστικών των serverless πλατφορμών που υποστηρίζονται. Σκοπός της είναι να προσφέρει ένα **ενιαίο, συνεκτικό και επεκτάσιμο πληροφοριακό μοντέλο**, το οποίο θα χρησιμοποιείται από την ενότητα αναζήτησης & ταξινόμησης πλατφορμών. Αρχιτεκτονικά, η ενότητα αυτή αποτελείται από:

1. **Μοντέλο Πλατφόρμας (Platform Model):** Ορίζει τα βασικά μεταδεδομένα της πλατφόρμας (όνομα, πάροχος, περιγραφή κ.λπ.).
2. **Σύνολο Τεχνικών Χαρακτηριστικών (Platform Features Model):** Περιλαμβάνει τα τεχνικά χαρακτηριστικά που μπορούν να χρησιμοποιηθούν για φιλτράρισμα και κατάταξη.

   Για να διασφαλιστεί ενιαία περιγραφή των πλατφορμών, το σύστημα υιοθετεί:

   - **Enum για τα επιτρεπτά ονόματα χαρακτηριστικών**, ώστε να καθορίζονται ρητά ποια attributes αποτελούν αποδεκτά κριτήρια (π.χ. runtime, region, timeout, scalingPolicy, memory).

Formatted: Greek

Commented [KK110]: Σας ανέφερα πως θα πρέπει να αναλύσετε τις ενότητες του backend. Συνεπώς, εδώ έχουμε την ενότητα της διαχειρισης πλατφορμών.

Commented [KK111]: Πάλι εδώ έχουμε το προβλημα με το αριστερό περιθώριο.

Μάλλον θέλατε η πρωτη προταση κάθε παραγράφου να αρχίζει πιο δεξιά αλλά το κάνατε για όλο το περιεχομενο της παραγράφου, πραγμα που είναι λάθος.

Formatted: Left, Indent: Left: 0 cm

Formatted: English (United States)

- **Enum για τις επιτρεπτές τιμές** συγκεκριμένων χαρακτηριστικών όπου απαιτείται (π.χ. υποστηριζόμενα runtimes ή κατηγορίες περιοχών).

Με αυτόν τον τρόπο το σύστημα εξασφαλίζει ότι οι πλατφόρμες ακολουθούν **ενιαίο και προκαθορισμένο πληροφοριακό σχήμα**.

3. **Μηχανισμό Επικύρωσης (Validation Engine):** Ο οποίος λειτουργεί τόσο κατά την εισαγωγή / ενημέρωση πλατφορμών όσο και κατά την αναζήτηση.

   Συγκεκριμένα:

   - Ελέγχει ότι τα χαρακτηριστικά που εισάγει ο διαχειριστής ανήκουν στα επιτρεπτά μέλη των Enums.
   - Ελέγχει ότι οι τιμές τους είναι συμβατές με τον αντίστοιχο τύπο (numeric, categorical, boolean κ.λπ.). Για τα κατηγορικά (categorical) κριτήρια, η εφαρμογή μπορεί να περιορίζει τις τιμές σε προκαθορισμένο σύνολο επιτρεπτών επιλογών (enums).
   - Ελέγχει επίσης τα ζεύγη **key–value** που παρέχει ο χρήστης κατά το φιλτράρισμα, ώστε να αποφεύγονται μη έγκυρες αναζητήσεις.

Με τον τρόπο αυτό, η υποενότητα διασφαλίζει:

- **Συνεκτικότητα δεδομένων:** όλες οι πλατφόρμες περιγράφονται με κεντρικά καθορισμένα και τυποποιημένα χαρακτηριστικά.
- **Επεκτασιμότητα:** μπορούν να προστεθούν νέα χαρακτηριστικά στο μέλλον, αρκεί να ενσωματωθούν στα αντίστοιχα Enums.
- **Αξιοπιστία αναζήτησης και ranking:** επειδή όλα τα χαρακτηριστικά επιβεβαιώνονται πριν αποθηκευτούν ή χρησιμοποιηθούν, αποφεύγονται ασυμβατότητες στις διαδικασίες φιλτραρίσματος και πολυκριτηριακής ταξινόμησης.

Σε επόμενο επίπεδο (στην Ενότητα 4.3 που περιγράφει την υλοποίηση του συστήματος), αναλύεται ο τρόπος με τον οποίο τα χαρακτηριστικά αποθηκεύονται στη βάση δεδομένων.

4.2.1.1.1 Υποενότητα Αναζήτησης & Κατάταξης

Ο υπο-ενότητα αυτή εντοπίζεται εντός της ενότητας Platform Module και ενσωματώνει μηχανισμό φιλτραρίσματος και κατάταξης, οποίος αποτελεί μια κρίσιμη λειτουργικότητα του backend του συστήματος, καθώς επιτρέπει στον χρήστη να εντοπίζει, να συγκρίνει και

Formatted: Greek

Commented [KK112]: Οπότε έχουμε συντακτικούς τύπους με δυνατοτητα χρησης και κατηγορικών που καλύπτουν την προαναφερομενη περιπτωση χρησης enum για συγκεκριμένα χαρακτηριστικά;

να αξιολογεί serverless πλατφόρμες βάσει τόσο τεχνικών όσο και περιγραφικών κριτηρίων.

Η σχεδίαση της υπο-ενότητας ακολουθεί μια πολυστρωματική προσέγγιση ως εξής (1) **Application layer (Backend Services):** ερμηνεία κριτηρίων, σύνθεση query (full-text/fuzzy), εφαρμογή sorting/pagination. (2) **Data layer (PostgreSQL):** full-text search (tsvector/tsquery), trigram similarity (pg_trgm) και indexes για απόδοση.

Ενώ συνδέεται και με το στρώμα της διεπαφής, δημιουργώντας ένα ολοκληρωμένο υπο-σύστημα αναζήτησης/φιλτραρίσματος & ταξινόμησης. Στο στρώμα αυτό υλοποιούνται τα εξής: φόρμες αναζήτησης, φίλτρα, sorting, autocomplete και βασικό validation.

Το ολοκληρωμένο υποσύστημα προσφέρει τα εξής:

- **περιγραφική αναζήτηση (semantic/textual search)**,
- **φιλτράρισμα βάσει δομημένων χαρακτηριστικών**,
- **μηχανισμό fuzzy matching για ανοχή σφαλμάτων**,
- **μηχανισμό πολυκριτηριακής ταξινόμησης με βάρη χρήστη**,
- **μηχανισμό αυτόματης συμπλήρωσης (auto-complete)**.

Στην συνέχεια αναλύουμε τις πιο σημαντικές από αυτές τις λειτουργικότητες.

Full-Text Search (Πλήρης Κειμενική Αναζήτηση)

Ο μηχανισμός αναζήτησης αποτελεί κρίσιμο υποσύστημα της εφαρμογής, καθώς επιτρέπει στον χρήστη να εντοπίζει serverless πλατφόρμες βάσει συνδυασμού τεχνικών χαρακτηριστικών και περιγραφικής πληροφορίας. Στο επίπεδο της σχεδίασης, η λειτουργία αυτή βασίζεται σε μία αφηρημένη διαδικασία επεξεργασίας και αναπαράστασης κειμένου, η οποία υποστηρίζει αναζητήσεις τόσο ακριβείας όσο και περίπου-αντιστοίχισης.

Η βασική ιδέα του Full-Text Search στηρίζεται στις **τιμές** των κειμενικών πεδίων (όπως ονομασία, περιγραφή, κατηγορία και ενδεχομένως άλλα πεδία που περιλαμβάνονται στη μεταδεδομένη περιγραφή της πλατφόρμας – ουσιαστικά καλύπτονται τόσο τεχνικά χαρακτηριστικά όσο και μεταδεδομένα) κάθε πλατφόρμας καί στη μετατροπή τους σε μια ενιαία αναπαράσταση «διανύσματος λέξεων» (*word vector representation*). Η αναπαράσταση αυτή επιτρέπει:

- την ανάλυση του περιεχομένου σε επίπεδο όρων,
- την κανονικοποίηση των λέξεων (π.χ. stemming ή lemmatization),

Formatted: Greek

Commented [KK113]: Όλα αυτά αναλύονται παρακάτω;

Commented [AK114R113]: Ναι σωστά

Commented [KK115]: Οκ αλλά εσείς λογαριάζετε μόνο τις τιμές των πεδιων ή και τα ονόματά τους; Ας το προσδιορισετε αυτό εντος του κειμένου.

Commented [KK116]: Άρα τα τεχνικά χαρακτηριστικά τελικά αποκλειονται ή όχι; Το ρωτλω διότι προηγουμένως αναφερθήκατε σε συνδυασμό τεχνικών χαρακτηριστικών και μεταδεδομένων που υποστηρίζεται. Οπότε, εδώ υπάρχει μια ασυνέπεια που θα πρεπει να διορθωθει.

Commented [AK117R116]: Οχι τα τεχνικα χαρακτηριστικα δεν αποκλειονται

- την εξαγωγή βαρών σημασίας (πχ. TFIDF) για κάθε όρο.

Κατά την αναζήτηση, ο όρος που εισάγει ο χρήστης μετατρέπεται αντίστοιχα σε ένα διανυσματικό ερώτημα (*query vector*). Η σύγκριση των δύο διανυσμάτων γίνεται με βάση τη χρήση των μετρικών της θεωρίας της ανάκτησης πληροφορίας (Information Retrieval), όπως **cosine similarity**, ή άλλες συναφών μετρικών.

Με τον τρόπο αυτό, το σύστημα μπορεί να υπολογίσει έναν βαθμό «σχετικότητας» (relevance score) μεταξύ της εισαγόμενης αναζήτησης και κάθε πλατφόρμας. Η σχετικότητα εκφράζει την εγγύτητα των δύο διανυσμάτων και συνήθως υπολογίζεται βάσει του βαθμού αλληλεπικάλυψης μεταξύ των σημαντικότερων όρων τους.

Για να διασφαλιστεί ότι ένα αποτέλεσμα θεωρείται όντως σχετικό, εφαρμόζεται ένα κατώφλι (threshold), το οποίο καθορίζει το ελάχιστο επίπεδο ομοιότητας που πρέπει να υπάρχει μεταξύ του ερωτήματος και του κειμένου μιας πλατφόρμας.

Συνολικά, ο μηχανισμός πλήρους κειμενικής αναζήτησης:

- επιτρέπει στον χρήστη να πραγματοποιεί αναζητήσεις φυσικής γλώσσας χωρίς να απαιτείται απόλυτη αντιστοίχιση όρων,
- παρέχει σημασιολογική συνάφεια μεταξύ των αποτελεσμάτων,
- υποστηρίζει σύνθετα ερωτήματα που βασίζονται σε πολλαπλά στοιχεία της περιγραφής των πλατφορμών,
- ενισχύει την ακρίβεια και την ποιότητα της κατάταξης των αποτελεσμάτων.

Σε επίπεδο υλοποίησης μπορούν να χρησιμοποιηθούν συγκεκριμένες τεχνικές ή δομές (όπως tsvector, tsquery ή άλλες), αλλά αυτές αποτελούν ζήτημα του τεχνικού επιπέδου και όχι της σχεδιαστικής περιγραφής. Εδώ επικεντρωνόμαστε στη λογική αρχιτεκτονική: τη μετατροπή του κειμένου σε δομημένη αναπαράσταση και την εύρεση της σχετικότητας με βάση κλασικές μετρικές της βιβλιογραφίας.

Υποστήριξη Fuzzy Matching (Συντακτική Ομοιότητα μέσω N-grams)

Για να ενισχυθεί η ευχρηστία και η ανθεκτικότητα του μηχανισμού αναζήτησης, το σύστημα ενσωματώνει έναν μηχανισμό **fuzzy matching** βασισμένο σε συγκεκριμένη βιβλιογραφική τεχνική συντακτικής ομοιότητας: **trigram similarity** (n-gram similarity analysis).

Η τεχνική αυτή ανήκει στην κατηγορία των *approximate string matching methods* και λειτουργεί ως εξής:

Formatted: Greek

Commented [KK118]: TFIDF βάρη; Αυτά χρησιμοποιείτε, σωστά;

Commented [AK119R118]: στην PostgreSQL χρησιμοποιώ tsvector ranking

Τεχνική: Trigram-based Similarity (n-gram model)

Κάθε τιμή μεταδεδομένου ή τεχνικού χαρακτηριστικού (π.χ. όνομα πλατφόρμας ή χαρακτηριστικό) μετατρέπεται σε ένα σύνολο από **trigrams** — δηλαδή διαδοχικές ακολουθίες 3 χαρακτήρων. Έτσι, η ομοιότητα δεν περιορίζεται σε μεμονωμένες λέξεις, αλλά μπορεί να υπολογιστεί και για φράσεις πολλών λέξεων, επιτρέποντας ανθεκτικότητα σε τυπογραφικά λάθη ή μερικώς πληκτρολογημένους όρους.

Παράδειγμα

Ο όρος:
"memory"
παράγει τα trigrams:

" mem", "mem", "emo", "mor", "ory", "ry "

Ο όρος του χρήστη:
"memor"
παράγει τα trigrams:

" mem", "mem", "emo", "mor", "or "

Μέθοδος ομοιότητας (Similarity Metric)

Η ομοιότητα των δύο λέξεων υπολογίζεται βάσει της **Jaccard similarity** των δύο συνόλων n-grams:

$$similarity(A, B) = \frac{| A \cap B |}{| A \cup B |}$$

Όπου:

- **A** = σύνολο trigrams της λέξης της πλατφόρμας
- **B** = σύνολο trigrams του όρου αναζήτησης του χρήστη.

Αυτό επιτρέπει την εύρεση σχετικών αποτελεσμάτων ακόμη και όταν υπάρχουν:

- μικρά ορθογραφικά λάθη,
- αλλαγές στη σειρά χαρακτήρων,

Formatted: Greek

Commented [KK120]: Αν έχουμε μια τιμή μεταδεδομένου/χαρακτηριστικού με πολλές λέξεις, τοτε τι ακριβώς γινεται;
Το λέτε αυτό παρακάτω;

Commented [KK121]: Επειδή μιλάτε για ομοιότητα μεταξύ 2 λέξεων, τότε θα πρέπει να συγκριθεί μια λέξη από την πλατφορμα και μια λέξη από τον επερωτηση του χρηστη.

- αποκοπές ή μερικά πληκτρολογημένοι όροι.

Επιπρόσθετα, σε multi-word πεδία η similarity εφαρμόζεται στο πλήρες string (με trigrams που περιλαμβάνουν και κενά), ενώ προαιρετικά μπορεί να χρησιμοποιηθεί word-level similarity για πιο ακριβή ταίριασμα σε μικρά queries.

Συνδυασμός Fuzzy Matching με Full-Text Search

Ο μηχανισμός fuzzy matching **δεν αντικαθιστά** τον μηχανισμό πλήρους κειμενικής αναζήτησης.
Αντίθετα, λειτουργεί **συνδυαστικά**:

1. **Πρώτα** εφαρμόζεται full-text search για τον εντοπισμό των πιο σχετικών εγγραφών (πλατφορμών) ως προς τη σημασία (semantic relevance).
2. **Στη συνέχεια**, εφαρμόζεται η μετρική trigram similarity για να διορθώσει ή να ενισχύσει τα αποτελέσματα όταν:
    - ο χρήστης χρησιμοποιεί όρους που δεν υπάρχουν ακριβώς στη βάση,
    - υπάρχουν τυπογραφικά λάθη,
    - ο χρήστης δεν θυμάται την ακριβή τεχνική ονομασία (για μια τιμή πεδίου).

Η ομοιότητα υπολογίζεται **μεταξύ ολόκληρων strings** και όχι μόνο μεμονωμένων λέξεων. Επομένως, όταν το ερώτημα ή το πεδίο της βάσης δεδομένων περιέχει πολλές λέξεις, το σύστημα δημιουργεί το σύνολο των trigrams από ολόκληρο το κείμενο και υπολογίζει μια συνολική ομοιότητα με βάση τα κοινά trigrams μεταξύ των δύο strings. Στην παρούσα υλοποίηση, το fuzzy matching χρησιμοποιείται ως **μηχανισμός ανοχής σε λάθη** και όχι ως δεύτερο σύστημα βαθμολόγησης που συγχωνεύεται με το full-text score. Μια πλατφόρμα θεωρείται υποψήφια και επιστρέφεται στα αποτελέσματα όταν ισχύει **τουλάχιστον μία** από τις παρακάτω συνθήκες:

1. **Full-Text match:** τα πεδία name, description, category ταιριάζουν με το ερώτημα του χρήστη μέσω PostgreSQL full-text (@@ plainto_tsquery(...)).
2. **Fuzzy match:** ένα από τα πεδία κειμένου της πλατφόρμας (*π.χ. name, description ή category*) ~~το name~~ είναι "αρκετά κοντά" στο keyword με βάση trigrams, με κατώφλι ομοιότητας π.χ. similarity(name, keyword) > 0.25 (PostgreSQL pg_trgm).

Ο συνδυασμός γίνεται με **λογική OR** (full-text **ή** fuzzy).

Άρα:

Formatted: Greek

Formatted: Greek

Commented [KK122]: Ναι, αλλά η τεχνική αφορά την σύγκριση/ομοιότητα 2 λέξεων ενώ μια πλατφορμα ή μια επερώτηση αποτελούνται από πολλές λέξεις. Τι γινεται σε αυτή την περιπτωση; Λογαριάζουμε τα τριγράμματα για όλες τις λέξεις της πλατφορμας/επερώτησης και έπειτα υπολογίζουμε μια συνολική ομοιότητα;
Θα πρέπει να διευκρινιστει εντος του κειμένου.

Commented [KK123]: Για μια τιμή φαντάζομαι ... Διότι λογικά επικεντρώνεστε μόνο σε τιμές και όχι σε ονόματα πεδιων, σωστά;

Commented [AK124R123]: Σωστα

Formatted: Greek

Commented [KK125]: Μόνο το όνομα λογαριάζεται;

Προτείνω να αναφέρετε όλα τα πεδια που υποστηριζονται όπως κάνατε και στο 1. full-text match.

Formatted: Greek

Formatted: Greek

Formatted: Greek

- το fuzzy matching έχει **ένα ρυθμιζόμενο threshold** (π.χ. 0.25),
- το full-text επιστρέφει αποτελέσματα με βάση το match και μπορεί να παρέχει και σχετική συνάφεια (ts_rank_cd(...)).

Πλεονεκτήματα της προσέγγισης n-gram fuzzy matching

- **Ανθεκτικότητα σε ορθογραφικά λάθη**
- **Ανεξαρτησία από γλώσσα** (λειτουργεί σε αυθαίρετο κείμενο)
- **Δεν απαιτεί λεξικά ή NLP**
- **Εξαιρετικά γρήγορο** με indexed n-grams (ευρετήριο τύπου inverted index που χαρτογραφεί κάθε n-gram → λίστα εμφανίσεων, ώστε το query να επιστρέφει άμεσα υποψήφιους και να γίνεται scoring μόνο σε αυτούς)
- **Επιτρέπει approximate search χωρίς machine learning**

Συνοπτικά

Ο συνδυασμός των δύο μετρικών:

- **Full-text relevance**
- **Trigram similarity**

επιτρέπει στο σύστημα να προσφέρει:

- ρεαλιστική και «ανεκτική» αναζήτηση,
- υψηλή χρηστικότητα για μη τεχνικούς χρήστες,
- ορθή κατάταξη ακόμη και σε ελλιπή ή μη ακριβή δεδομένα εισόδου.

Τέλος **οι δύο μετρικές δεν συγχωνεύονται σε ένα ενιαίο score**. Το fuzzy matching χρησιμοποιείται κυρίως για τον εντοπισμό επιπλέον υποψήφιων αποτελεσμάτων όταν το ερώτημα περιέχει σφάλματα ή μη ακριβείς όρους. Η τελική κατάταξη των αποτελεσμάτων βασίζεται κυρίως στο score συνάφειας που προκύπτει από τον μηχανισμό full-text search.

Συνολική Λειτουργικότητα

Formatted: Greek

Formatted: Outline numbered + Level: 1 + Numbering Style: Bullet + Aligned at: 0,63 cm + Tab after: 1,27 cm + Indent at: 1,27 cm

Commented [KK126]: Ας το εξηγήσετε αυτό λίγο παραπάνω.

Commented [KK127]: Δεν έχετε εξηγήσει πως γινεται η κατάταξη. Αν έχουμε 2 scores ομοιότητας συνδυάζονται ή θεωρούμε πως ακολουθούμε το σκορ που προκύπτει από την πρώτη ομοιότητα μόνο;

Ο μηχανισμός αναζήτησης συνδυάζει δύο συμπληρωματικές τεχνικές: **Full-Text Search** και **Fuzzy Matching**. Κάθε μία επιτελεί διαφορετικό ρόλο και η συνεργασία τους οδηγεί σε πιο ακριβή και φιλική προς τον χρήστη αναζήτηση.

Πολυκριτηριακή κατάταξη (Multi-criteria ranking)

Για τη φάση της ταξινόμησης (ranking), κάθε πλατφόρμα αξιολογείται με βάση ένα σύνολο κριτηρίων που μπορεί να ορίσει ο χρήστης (π.χ. χρόνος εκτέλεσης, κόστος, υποστηριζόμενες γλώσσες, διαθέσιμη μνήμη).

Σε κάθε κριτήριο αντιστοιχίζεται ένα βάρος (weight) από 0.0 έως 1.0, με το άθροισμα όλων των βαρών να ισούται με 1.0. Τα βάρη μπορούν να υπολογιστούν με βάση την τεχνική Analytic Hierarch Process.[30]

Ο αλγόριθμος υπολογίζει ένα τελικό σκορ (score) για κάθε πλατφόρμα, μέσω του τύπου:

$$Scoreplatform = \sum_{i=1}^{n} \left(w_i \times normalized(value_i) \right)$$

όπου w_i είναι το βάρος και normalized($value_i$) η κανονικοποιημένη τιμή του χαρακτηριστικού/κριτηρίου. Τα κριτήρια είναι επιτρεπτά χαρακτηριστικά των πλατφορμών, όπως αυτά έχουν οριστει στη σχετική απαρίθμηση (enum).

Αυτή η προσέγγιση επιτρέπει τη δυναμική προσαρμογή των αποτελεσμάτων σύμφωνα με τις προτεραιότητες του χρήστη (π.χ. άλλος μπορεί να δίνει έμφαση στο μέγεθος, άλλος στις υποστηριζομενες γλώσσες).

Κανονικοποίηση Τιμών (Value Normalization)

Η κανονικοποίηση πραγματοποιείται για να εξασφαλιστεί ότι όλα τα κριτήρια συνεισφέρουν αναλογικά στο τελικό σκορ, ανεξάρτητα από τη φύση ή το εύρος των τιμών τους.
Για κάθε κριτήριο υπολογίζεται γραμμική κανονικοποίηση στο διάστημα [0,1]:

[30] ~~(PDF) The Analytic Hierarchy Process~~https://www.researchgate.net/publication/362349026_The_Analytic_Hierarchy_Process

Formatted: Greek

Commented [KK128]: Τα κριτήρια αυτά δεν είναι χαρακτηριστικά/ιδιότητες των πλατφορμών; Αν ναι, τότε δεν θα πρέπει να εξασφαλιστει πως το συνολο των κριτηρίων είναι ενιαιο και πως οι περιγραφές των πλατφορμών γίνονται με βάση τα κριτήρια αυτά;

Commented [KK129]: Λείπει σε αυτό το σημειο παραπομπή προς αυτή την τεχνική.

Commented [KK130]: Θέλουμε full URL στο footnote και όχι περιγραφικό. Επίσης, καλό είναι για άρθρα/ερευνητικές προσεγγισεις να χρησιμοποιειται κανονικές παραπομπές και όχι συνδέσμους. Παραπομπές σε βιβλιογραφικές αναφορές που παρατίθενται στο τελευταιο κεφάλαιο της αναφορας σας.

Commented [KK131]: ¨Αρα τα κριτήρια είναι επιτρεπτά χαρακτηριστικά των πλατφορμών, όπως αυτά έχουν οριστει στη σχετική απαριθμηση (enum); Ας το ξεκαθαρισετε εντός του κειμένου.

Commented [KK132]: Να θεωρήσουμε πως όλα τα κριτήρια ταξινόμησης είναι αριθμητικά;
Το ρωτώ διότι δεν καλύπτετε την περιπτωση μη αριθμητικών κριτηριων. Αυτά πώς θα μπορούσαν να κανονικοποιηθούν;
Μπορούν να αντιστοιχισθούν σε αριθμητικές τιμές εφοσον οι (πραγματικές) τιμές τους είναι μελη απαρίθμησης;

Formatted: Greek
Field Code Changed
Formatted: Greek
Formatted: Greek
Formatted: Greek
Formatted: Greek
Formatted: Greek
Formatted: Greek
Formatted: Greek
Formatted: Greek
Formatted: Greek

- **min**: η μικρότερη τιμή που εμφανίζεται για το συγκεκριμένο κριτήριο *μεταξύ όλων των πλατφορμών* που συμμετέχουν στη σύγκριση.
- **max**: η μεγαλύτερη τιμή του κριτηρίου στο ίδιο σύνολο πλατφορμών.

Αν το κριτήριο έχει **θετική κατεύθυνση** (π.χ. μνήμη, ταχύτητα), τότε

$$normalized(value) = \frac{value - min}{max - min}$$

Διαφορετικά, αν το κριτήριο έχει **αρνητική κατεύθυνση** (π.χ. κόστος, latency), τότε

$$normalized(value) = \frac{max - value}{max - min}$$

Μη αριθμητικά κριτήρια (categorical)

Για κριτήρια που δεν έχουν αριθμητική μορφή (π.χ. runtime, supported languages), η αξιολόγηση βασίζεται σε σύγκριση των δυνατοτήτων της πλατφόρμας με τις απαιτήσεις του χρήστη.

Συγκεκριμένα, ο χρήστης ορίζει ένα σύνολο επιθυμητών τιμών (π.χ. λίστα από runtime ή γλώσσες προγραμματισμού), και κάθε πλατφόρμα αξιολογείται με βάση τον βαθμό στον οποίο υποστηρίζει αυτές τις τιμές.

Η βαθμολογία υπολογίζεται ως:

score = (αριθμός τιμών που υποστηρίζονται από την πλατφόρμα) / (αριθμός τιμών που έχει ορίσει ο χρήστης)

Με τον τρόπο αυτό, η αξιολόγηση προσαρμόζεται δυναμικά στις προτιμήσεις του χρήστη και δεν βασίζεται σε προκαθορισμένες ιεραρχίες τιμών. Επιπλέον, η προσέγγιση αυτή είναι ανεξάρτητη από τη σημασιολογία των χαρακτηριστικών και μπορεί να εφαρμοστεί ομοιόμορφα σε διαφορετικούς τύπους δεδομένων, συμπεριλαμβανομένων και δυαδικών χαρακτηριστικών.

Με αυτόν τον τρόπο, τόσο τα αριθμητικά όσο και τα μη αριθμητικά κριτήρια παράγουν συγκρίσιμες βαθμολογίες, οι οποίες μπορούν να συνδυαστούν στον τελικό υπολογισμό ranking.

4.2.1.2 Deployment Module

Η Deployment Module αποτελεί την ενότητα εκείνη που αναλαμβάνει τη διάταξη (deployment) των serverless συναρτήσεων στις επιλεγμένες πλατφόρμες νέφους, αξιοποιώντας το **Serverless Framework**. Η διαδικασία της διάταξης περιλαμβάνει τρία

Formatted: Greek

Commented [KK133]: Δεν εξηγειτε τι σημαίνει min & max

Commented [KK134]: Θα μπορούσατε να την αποτυπώσετε με κάποιο UML διάγραμμα όπως ακολουθίας.

Commented [KK135R134]: Εκτος και πειτε πως αυτά τα στάδια είναι διαδοχικά και δεν περιλαμβάνουν πολλαπλά βηματα Οπότε δεν χρειάζεται σε μια τέτοια περιπτωση κάποιο διάγραμμα (ακολουθίας ή δραστηριοτήτων) για την αποτύπωση της διαδικασιας αυτής.

Αλλά από ότι βλέπω από την περιγραφή, το κάθε στάδιο εχει πολλα βήματα, οπότε εχει νόημα η μοντελοποίηση του σχετικου διαγράμματος.

Commented [AK136R134]: Το διαγραμμα ακολουθιας για την διαταξη ειναι στην ενοτητα μετα 4.2.6

στάδια: μεταφόρτωση πακεταρισμένου κώδικα, δημιουργία αρχείου ρυθμίσεων, εκτέλεση της διάταξης.

---

1. Μεταφόρτωση και Αποκωδικοποίηση Αρχείου

Ο χρήστης ανεβάζει το πακέτο της συνάρτησης μέσω του frontend. Το πακέτο αποστέλλεται στο backend ως **Base64 string** μέσα σε JSON request.

Σχήμα αιτήματος (JSON payload). Το αίτημα διάταξης περιλαμβάνει τόσο το πακέτο κώδικα όσο και τα απαραίτητα μεταδεδομένα, σύμφωνα με το DeploymentRequestDTO. Ενδεικτικά:

- *platformId*: αναγνωριστικό πλατφόρμας (AWS ή GCP)
- *functionName*: όνομα συνάρτησης
- *userId*: αναγνωριστικό χρήστη (για ανάκτηση credentials & καταγραφή ιστορικού)
- *credentialsId* (προαιρετικό / υλοποιείται μέσω mapping credentials ανά χρήστη–πλατφόρμα)
- *functionPackageBase64*: Base64-encoded ZIP (artifact)
- *runtime* (προαιρετικό): π.χ. nodejs18.x ή nodejs18
- *handler* (προαιρετικό): π.χ. index.handler
- *region* (προαιρετικό): π.χ. us-east-1 / us-central1

Με τον τρόπο αυτό, το backend λαμβάνει **σε ένα ενιαίο request** τόσο το αρχείο όσο και τις παραμέτρους που απαιτούνται για τη δυναμική δημιουργία του serverless.yml και την εκτέλεση του deploy.

Η επιλογή JSON αντί multipart upload έγινε για τους εξής λόγους:

- Επιτρέπει την **ενιαία διαχείριση** μεταδεδομένων και αρχείου σε μία δομή.
- Απλοποιεί την **REST αρχιτεκτονική**, χωρίς ανάγκη ενεργοποίησης multipart parsing.
- Αποφεύγεται η ανάγκη προσωρινού external storage ή streaming κατά το upload.

Formatted: Greek

Commented [KK137]: Το πέμπτο στάδιο καταγράφεται ως μέρους του τεταρτου. Οπότε θα πρέπει να το ξεχωρίσετε και να του δώσετε και τον αριθμό 5.

Commented [KK138]: Αυτή μοιάζει με τεχνική λεπτομέρεια. Εκτος και αν την θεωρειτε ως σχεδιαστική επιλογή ...

Commented [KK139R138]: Μπορειτε να διατηρήσετε την επιλογή αυτή - ας θεωρηθεί σχεδιαστική. Αλλά θα πρέπει να προσδιορισετε ποιο είναι το σχημα του payload σε αυτή την περίπτωση διότι δεν περιλαμβάνεται μόνο το πακέτο αλλά και ορισμένα μεταδεδομένα.

Το backend αποκωδικοποιεί το περιεχόμενο και το εξάγει σε **προσωρινό φάκελο με μοναδική ονομασία**, ο οποίος δημιουργείται ανά διάταξη, π.χ. serverless-deployment-{deploymentId} μέσα στο java.io.tmpdir. Η χρήση του deploymentId εξασφαλίζει ότι κάθε εκτέλεση έχει **διακριτό workspace**, αποτρέποντας σύγκρουση/επικάλυψη αρχείων μεταξύ ταυτόχρονων χρηστών ή διατάξεων.

Η πρόσβαση στο συγκεκριμένο directory γίνεται αποκλειστικά από την εφαρμογή (backend process) και ο φάκελος **καθαρίζεται μετά το τέλος** της διάταξης (cleanup), ώστε να ελαχιστοποιείται η παραμονή προσωρινών δεδομένων στο σύστημα αρχείων, ώστε η διαδικασία να παραμένει **απομονωμένη και ασφαλής**, χωρίς κοινή χρήση φακέλων μεταξύ χρηστών.

---

2. Δημιουργία Configuration (serverless.yml)

Με βάση τις επιλογές του χρήστη (πάροχος, runtime, όνομα συνάρτησης, περιοχή), το σύστημα παράγει δυναμικά το αρχείο serverless.yml. Επιπλέον, προσδιορίζει **ειδικές ρυθμίσεις ανά πάροχο**, όπως:

~~Πίνακας 3: Βασικές παράμετροι διαμόρφωσης deployment για διαφορετικές serverless πλατφόρμες~~

Πίνακας 3: Βασικές παράμετροι διαμόρφωσης deployment για διαφορετικές serverless πλατφόρμες

Πάροχος	Runtime[31]	Handler[32]	Πρότυπο Deployment
AWS Lambda	Node / Python / Java	index.handler	HTTP API trigger
Google Cloud Functions	Node / Python	function handler	HTTP trigger

Τι γίνεται με τα events (triggers);

[31] Runtime ονομάζεται το περιβάλλον εκτέλεσης της συνάρτησης, δηλαδή η γλώσσα προγραμματισμού και η αντίστοιχη έκδοση που χρησιμοποιείται για την εκτέλεση του κώδικα.
[32] Handler είναι το σημείο εισόδου (entry point) της serverless συνάρτησης, δηλαδή η μέθοδος που καλείται από την πλατφόρμα όταν ενεργοποιείται η function.

Formatted: Greek

Commented [KK140]: Οκ, ούτως ή άλλως το συστημα θα αναλαμβάνει την αποθήκευση αυτή εκ μέρους των χρηστών.
Απλώς θα πρέπει να διασφαλιστει η απομόνωση των προσωρικών ευρετηριων/φακέλων.
Πχ. αν όλα μπαινουν σε ένα κοινό υπερ-φάκελο, όπως τον temp, θα πρέπει να εξασφαλιστει η διαφορετική/μοναδική ονομασια τους.
Επίσης, θα πρέπει να επιτρέπεται η πρόσβαση μόνο σε εξουσιοδοτημένους «χρηστες» (όπως είναι η εφαρμογή/σύστημα).

Commented [KK141]: Αυτές οι επιλογές πραγματοποιούνται σε αυτή την φάση (παρέχονται τότε από τον χρηστη) ή ΄χουν δοθει σε προηγούμενη φαση (πχ. την πρώτη); Δεν είναι ξεκάθαρο.

Commented [KK142]: Αυτές οι επιλογές αφορούν χρηστη με άρτιες τεχνικές γνώσεις. Επομένως, τουλάχιστον η διάταξη θα πρέπει να πραγματοποιειται από έμπειρους χρήστες (πχ. Devops ενός οργανισμού).

Η αναζητηση θα μπορούσε να γινεται και από απλούς χρηστες εφοσον υπάρχει σχετική υποστήριξη και αυτό έχει νόημα - πχ. αν η αναζητηση εστιάζει σε τεχνικά χαρακτηριστικά, αυτά δεν θα ενδιαφέρουν τον απλό χρήστη διότι δεν θα τα καταλαβαινει. Απεναντίας αν έχουμε να κάνουμε με επιχειρησιακό ειδικό, τοτε αυτος θα ενδιαφέρετε να κάνει συγκριση πλατφορμών σε υψηλότερο επίπεδο μέσω χρησης επιχειρησιακών/ποιοτικών/υψηλού επιπέδου κριτηριων.

Άρα, με την ανάλυση που σας κάνω, ίσως τελικά η πλατφορμα να απευθύνεται κυρίως σε τεχνικούς χρηστες.

Commented [KK143]: Αυτές μπορείτε να τις εξηγήσετε λίγο παραπάνω; Πχ. τι είναι ο handler; Το runtime πρέπει να περιλαμβάνει και εκδόσεις γλωσσών προγραμματισμού; Ή μόνο η επιλογή γλώσσας είναι αρκετή;
Τέλος, ποια είναι η διαφορά μεταξύ HTTP trigger & HTTP API trigger; Στην πρώτη περιπτωση, η συνάρτηση δεν είναι διαθέσιμη ως RESTful υπηρεσια αλλά μια απλή ιστοσελίδα;

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Caption, Keep with next

Field Code Changed

Commented [KK144]: Επειδή αυτός είναι πίνακας, θα πρέπει να έχει λεζάντα από πάνω του.

Commented [KK145R144]: Δεν έχετε βάλει ακόμη την λεζάντα πάνω από τον πίνακα!

- Στην παρούσα έκδοση το σύστημα χρησιμοποιεί **προεπιλεγμένο trigger τύπου HTTP**, ώστε κάθε συνάρτηση να είναι άμεσα προσβάσιμη μέσω endpoint.
- Η διαφορά μεταξύ **HTTP trigger** και **HTTP API trigger** σχετίζεται κυρίως με τον τρόπο διαχείρισης του HTTP endpoint από την πλατφόρμα:
    - **HTTP trigger**: η συνάρτηση ενεργοποιείται απευθείας από ένα HTTP request και επιστρέφει μια απάντηση. Η λειτουργία αυτή χρησιμοποιείται συχνά για απλά web endpoints ή microservices.
    - **HTTP API trigger**: χρησιμοποιείται συνήθως μέσω ενός API Gateway και επιτρέπει την έκθεση της συνάρτησης ως πλήρους REST API endpoint, παρέχοντας επιπλέον δυνατότητες όπως routing, authentication και rate limiting.
- Και στις δύο περιπτώσεις η συνάρτηση μπορεί να λειτουργήσει ως backend endpoint μιας web εφαρμογής.
- Ωστόσο, το configuration (παραμετροποίηση / διαμόρφωση) έχει σχεδιαστεί **παραμετρικά**, επιτρέποντας μελλοντικά να προστεθούν:
    - Cron schedules
    - Queue triggers (SQS / PubSub)
    - Storage event triggers
- Σε μελλοντική επέκταση ο χρήστης θα μπορεί να ορίζει events μέσω UI επιλογών (πχ. dropdowns / presets).

---

3. Εκτέλεση Deployment

Το backend εκτελεί το Serverless Framework σε απομονωμένο περιβάλλον στο σημείο όπου υπάρχει το πακέτο της συνάρτησης και το αρχειο διαμόρφωσης που δημιουργήθηκε, με:

serverless deploy --verbose

Το πλαίσιο αυτό:

- Μεταφορτώνει το πακέτο (package) της συνάρτησης στον πάροχο,
- Δημιουργεί τους απαιτούμενους πόρους νέφους,

Formatted: Greek

Commented [KK146]: Αυτό είναι τεχνική λεπτομέρεια;

Commented [KK147]: Στο σημειο όπου υπάρχει το πακέτο της συνάρτησης και το αρχειο διαμόρφωσης που δημιουργήθηκε;

- Και επιστρέφει **public endpoint** (που περιτυλίγει την διαταγμένη συνάρτηση), το οποίο αποθηκεύεται στη βάση.

Η υπηρεσία διάταξης εφαρμόζει συνεχές tracking της κατάστασης με χρήση της οντότητας DeploymentRecord. Σε κάθε διάταξη:

- Καταγράφεται αρχικά ως PENDING
- Μετά το πέρας της διαδικασίας ενημερώνεται ως SUCCESS ή FAILED
- Σε κάθε περίπτωση καταγράφονται τα πλήρη logs της εκτέλεσης

Έτσι το σύστημα:

- γνωρίζει ανά πάσα στιγμή σε ποιο στάδιο βρίσκεται η διάταξη,
- μπορεί να ειδοποιήσει άμεσα τον χρήστη,
- και υποστηρίζει πλήρες ιστορικό για debugging ή benchmarking.

Formatted: Greek

4.2.2 Λογική Αρχιτεκτονική (Logical Components)

Frontend

Το frontend αποτελεί το γραφικό περιβάλλον αλληλεπίδρασης του χρήστη με το σύστημα και παρέχει λειτουργίες εγγραφής, σύνδεσης, αναζήτησης, φιλτραρίσματος και αξιολόγησης πλατφορμών, καθώς και εκτέλεσης της διάταξης συνάρτησης (σε μία από τις πλατφόρμες) μέσω του *Deployment Wizard*.

Πριν αποστείλει οποιοδήποτε αίτημα προς το backend για την εκτέλεση αντίστοιχης βασικής λειτουργίας του συστήματος, ελέγχει την εγκυρότητα της εισόδου του χρήστη (π.χ. πλήρωση πεδίων, μορφή αρχείου).

Η επικοινωνία με το backend πραγματοποιείται αποκλειστικά μέσω **REST API κλήσεων** που περιλαμβάνουν περιεχόμενο μηνύματος σε μορφή **JSON**. Ο χρήστης δεν έρχεται σε επαφή με CLI ή σύνθετες εντολές· όλες οι ενέργειες εκτελούνται αποκλειστικά μέσω του web γραφικού περιβάλλοντος του συστήματος.

Backend (Spring Boot Application)

Το backend, υλοποιημένο σε **Spring Boot**, αποτελεί το επιχειρησιακό επίπεδο του συστήματος και λειτουργεί ως ενδιάμεσος μεταξύ του frontend και της βάσης δεδομένων. Ακολουθεί αυστηρά το μοτίβο **Controller–Service–Repository (CSR)**, όπου:

- **Κάθε λειτουργική περιοχή του συστήματος υλοποιείται από ένα ξεχωριστό Service**,
- ενώ **κάθε Service εκτίθεται μέσω αντίστοιχου Controller** (δηλαδή, υπάρχει 1-προς-1 αντιστοίχιση Controller ↔ Service).

Με αυτό τον τρόπο η αρχιτεκτονική παραμένει επεκτάσιμη, αρθρωτή και εύκολη στη συντήρηση.

Το backend περιλαμβάνει τις εξής κύριες (επιχειρησιακές) ενότητες (ουσιαστικά υλοποιούν μια RESTful υπηρεσία / API / endpoint), κάθε μία από τις οποίες ακολουθεί το μοτίβο CSR:

Formatted: Greek

Commented [KK148]: Η αρχιτεκτονική θα πρέπει να παρέχεται από την αρχη της 4.2 κι όχι στο τέλος της.

Επίσης, το διαγραμμα της αρχιτεκτονικής θα πρέπει να παρέχετε σε αυτή την σελίδα και όχι Ν σελίδες μετά.

Commented [KK149R148]: Εφόσον εδώ μιλάμε για την λεπτομερή αρχιτεκτονικής της εφαρμογής, ας αποτυπωθεί αυτή σε αυτό το σημειο.

Αλλά η αναλυση της αρχιτεκτονικης περιλαμβανει πολλε ΄ς τεχνικε ΄ς λεπτομερειες (πχ. κλασεις υλοποίησης για μια ενότητας) που θα πρεπει να μεταφερθουν στην Ενότητα 4.3. Εδώ μιλάμε για λογική αρχιτεκτονική και όχι για τον τρόπο υλοποίησης ενός συστατικου ή υπηρεσίας ή ενότητας. Θέλουμε να δούμε την δομή του συστήματος σε ενότητες και συστατικά και ποια είναι η υπευθυνότητα της κάθε ενότητας και συστατικου.

Commented [KK150]: Το μεγάλο μου προβλημα είναι πως κατά την ανάλυση εμπλέκετε και λεπτομέρειες υλοποίησης που θα πρεπει να μπουν στην ενότητα 4.3.

Εδώ μιλάμε για λογική αρχιτεκτονική. Οπότε, προσδιοριζετε την βασική λειτουργικότητα για κάθε συστατικό και την σχεδίαση της διεπαφής της (άρα και τα endpoints). Τίποτε άλλο πέρα από αυτό.

Αν έχετε χρονο, ας αντιμετωπίσετε το σχόλιο αυτό.

Commented [KK151]: Η εγγραφή δηλαδή δεν έχει καλυφθεί;

Commented [AK152R151]: Έχει καλυφθει

Formatted: Greek

Commented [KK153]: Χρησιμοποιώ τον όρο module ώστε να μην υπάρχει σύγχυση με τον όρο υπηρεσια/service στο CSR. Όμως, σε λογικό επίπεδο, μπορούμε να πούμε πως ένα module αντιστοιχει σε μια RESTful υπηρεσια.

Formatted: Greek

Formatted: Greek

1. Ενότητα Διαχείρισης Χρηστών & Ασφάλειας (AuthModule)

Η ενότητα διαχείρισης χρηστών και ασφάλειας διασφαλίζει την αυθεντικοποίηση, εξουσιοδότηση και προστασία όλων των REST endpoints της εφαρμογής καθώς και επιτρέπει την πλήρη διαχείριση των χρηστών. Η υλοποίησή της βασίζεται στο Spring Security και αποτελείται από ένα σύνολο κλάσεων που εφαρμόζουν τις πολιτικές ασφαλείας σε οριζόντιο επίπεδο.

Όσον αφορά το κομμάτι της ασφάλειας, η ενότητα αποτελείται από τις ακόλουθες βασικές κλάσεις:

- **SecurityConfig**: καθορίζει όλες τις πολιτικές ασφαλείας (πρόσβαση ανά ρόλο, stateless sessions, προστασία endpoints, CORS/CSRF).
- **JWTAuthFilter**: φίλτρο που εκτελείται πριν από κάθε αίτημα, επικυρώνει το JWT token (που περιλαμβάνεται στην κεφαλίδα του) και θέτει το Authentication Context.
- **OurUserDetailsService**: service που φορτώνει τα στοιχεία των χρηστών από τη βάση για την διαδικασία αυθεντικοποίησης.
- **AuthenticationProvider (DaoAuthenticationProvider)**: μηχανισμός που επαληθεύει τα credentials του χρήστη χρησιμοποιώντας τον UserDetailsService και τον (BCrypt)PasswordEncoder.

Commented [KK154]: Αυτό δεν αναλύεται σε ξεχωριστο bullet.

- **BCryptPasswordEncoder:** Χρησιμοποιείται για την ασφαλή αποθήκευση και επαλήθευση κωδικών πρόσβασης, παράγοντας κρυπτογραφικό αποτύπωμα (hash) με BCrypt και συγκρίνοντάς το κατά το login (χωρίς ποτέ να αποθηκεύεται ή να διακινείται ο κωδικός σε απλή μορφή).

Αυτές οι κλάσεις αποτελούν το βασικό security pipeline της εφαρμογής. Μάλιστα, οι πολιτικές ασφάλειας εφαρμόζονται *οριζόντια* από το SecurityConfig και από τα φίλτρα του Spring Security.

Λειτουργίες του μέρους της Ασφάλειας του AuthModule

Commented [KK155]: Προσέξτε λίγο με την αριθμηση διότι εδώ μιλάτε για λειτουργιες της ενότητας της ασφάλειας ενώ πριν είχατε αρχίσει στο ίδιο επίπεδο την αριθμηση για τις ενότητες του backend.

Οπότε, ας βάζετε bullets όταν μιλάτε για τις λειτουργιες μιας ενότητας

Αυθεντικοποίηση χρηστών μέσω JWT

Τα tokens δημιουργούνται από το JWTUtils και επικυρώνονται στο JWTAuthFilter. Κάθε αίτημα περνά από JWT token validation πριν δοθεί πρόσβαση σε προστατευμένο endpoint.

Εξουσιοδότηση ανά ρόλο (USER / ADMIN)

Formatted: Greek

Η κλάση SecurityConfig καθορίζει:

- ποια endpoints είναι δημόσια
- ποια endpoints απαιτούν authentication
- ποια endpoints είναι προσβάσιμα μόνο για ADMIN
- ποια endpoints είναι προσβάσιμα μόνο για USER

Υποστηρίζονται επίσης μέθοδοι με @PreAuthorize λόγω @EnableMethodSecurity.

Stateless έλεγχος πρόσβασης

Η εφαρμογή λειτουργεί αποκλειστικά σε **stateless mode**:

.sessionManagement(manager ->
manager.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

Αυτό σημαίνει:

- δεν υπάρχει session state στον server
- κάθε αίτημα πρέπει να περιέχει έγκυρο JWT
- ο server δεν διατηρεί πληροφορίες προηγούμενων αιτημάτων
- η κλιμάκωση και το load balancing γίνονται ευκολότερα

Authentication Provider

Ορίζεται ένας **DaoAuthenticationProvider**, ο οποίος:

- φορτώνει χρήστες μέσω του OurUserDetailsService
- ελέγχει τον κωδικό με χρήση BCryptPasswordEncoder

Ο AuthenticationProvider είναι ο πυρήνας που αποφασίζει αν το login είναι έγκυρο.

Το μέρος της διαχείρισης χρηστών του AuthModule παρέχει όλες τις βασικές λειτουργίες που σχετίζονται με την εγγραφή, σύνδεση και διαχείριση προφίλ. Αποτελεί το σημείο αλληλεπίδρασης του χρήστη με το σύστημα ταυτοποίησης (authentication) και συνεργάζεται στενά με την ενότητα της ασφάλειας, χωρίς όμως να επιβάλλει από μόνο του κανόνες εξουσιοδότησης.

Formatted: Greek

Υλοποίηση (συγκεκριμένες κλάσεις)

- **UserManagementService**: βασική υπηρεσία που υλοποιεί όλες τις λειτουργίες διαχείρισης χρηστών.
- **OurUserDetailsService**: χρησιμοποιείται από το Spring Security ώστε να φορτώνει χρήστες από τη βάση δεδομένων.
- **JWTUtils**: χειρίζεται την παραγωγή και επαλήθευση JWT tokens.

Controller

- **UserManagementController** – **/api/auth/**
  Παρέχει τα endpoints για:
  - register (εγγραφή χρήστη)
  - login (σύνδεση χρήστη)
  - refresh token (ανανέωση token αυθεντικοποίησης)
  - get profile (λήψης προφίλ χρήστη)
  - update user (ενημέρωσης προφίλ χρήστη)
  - delete user (διαγραφής χρήστη)

Λειτουργίες διαχείρισης χρηστών

Εγγραφή νέου χρήστη (registration)

Endpoint: /api/auth/register

- Δημιουργία λογαριασμού.
- Κρυπτογράφηση κωδικού.
- Αντιστοίχιση ρόλου (USER/ADMIN).
- Αποθήκευση στη βάση.

Σύνδεση χρήστη + δημιουργία JWT token

Endpoint: /api/auth/login

Formatted: Greek

Η υπηρεσία:

- Επαληθεύει email & password.
- Ελέγχει αν ο χρήστης είναι ενεργοποιημένος.
- Εκτελεί authentication μέσω AuthenticationManager.
- Παράγει:
    - JWT token
    - Refresh token

Αυτή η λειτουργία αφορά **authentication (αυθεντικοποίηση)**, δηλαδή επαλήθευση ταυτότητας. **Δεν περιλαμβάνει authorization** (διαχείριση δικαιωμάτων). Η εξουσιοδότηση (authorization) εφαρμόζεται σε άλλο επίπεδο (SecurityConfig).

Formatted: Greek

Ανανεώση (refresh) JWT token

Endpoint: /api/auth/refresh

- Το **access token** έχει συνήθως μικρότερη διάρκεια ζωής για λόγους ασφάλειας. Το **refresh token** έχει μεγαλύτερη διάρκεια ζωής και χρησιμοποιείται **μόνο** για να εκδοθεί νέο access token. Η υπηρεσία επαληθεύει το refresh token.

Commented [KK156]: Δεν έχετε εξηγήσει τι είναι αυτό και για ποιο λόγο χρησιμοποιείται

- Εκδίδει νέο access JWT token.

Ανάκτηση στοιχείων προφίλ (get profile)

Endpoint: /api/auth/adminuser/get-profile

Εντοπίζει τον χρήστη βάσει email από το JWT.

Ενημέρωση λογαριασμού (update user)

Endpoint: /api/auth/profile

- Αλλαγή email, ονόματος, πόλης.
- Αλλαγή ρόλου (για διαχειριστές).
- Αναβάθμιση/ενημέρωση κωδικού με hashing.

Διαγραφή χρήστη (delete user)

Endpoint /api/auth/admin/get-all-users

Υποστηρίζεται μέσω:

public ReqRes deleteUser(Integer userId)

Ενεργοποίηση/Απενεργοποίηση χρήστη

Υποστηρίζεται μέσω:

public ReqRes toggleActive(Integer id)

Αλλαγή / Επαναφορά κωδικού

Η τρέχουσα ενότητα του backend υποστηρίζει αλλαγή κωδικού μέσα από το update user, αλλά δεν παρέχει μηχανισμό επαναφοράς κωδικού μέσω email, ο οποίος μπορεί να αποτελέσει μελλοντική επέκταση.

2. Ενότητα (Διαχείρισης) Πλατφορμών (Platform Module)

Υλοποιείται από: PlatformService
Controller: PlatformController (/api/platforms/**)

Λειτουργίες:

- **CRUD** λειτουργίες για τις serverless πλατφόρμες.
- **Αναζήτηση** πλατφορμών με κριτήρια (filtering).
- **Ταξινόμηση (Sorting)** αποτελεσμάτων Με βάση κριτήρια, όπως τιμή, όνομα, κατηγορία κ.λπ. Ο χρήστης μπορεί να προσδιορίζει ποιο κριτήριο θα χρησιμοποιηθεί κι έπειτα η ταξινόμηση πραγματοποιείται αυτόματα. Το κριτήριο ταξινόμησης μπορεί έπειτα να αλλάζει από τον χρήστη όποτε το επιθυμεί.

Η Ενότητα Διαχείρισης Πλατφορμών αποτελεί τον βασικό μηχανισμό μέσω του οποίου το σύστημα χειρίζεται τις πληροφορίες των serverless πλατφορμών (AWS Lambda, Azure Functions, Google Cloud Functions κ.λπ.). Το υποσύστημα επιτρέπει την καταχώριση, επεξεργασία και αναζήτηση των πλατφορμών, καθώς και την

Formatted: Greek

Commented [KK157]: Επειδή εν τέλει υπάρχει ξεχωριστή υπηρεσια για την κατάταξη, αυτό σημαινει πως μπορει να υπάρχει και για την ταξινόμηση;
Αν ναι, τοτε θα πρέπει αυτή να εμφαρνίζεται ως άλλη/διαφορετική υπηρεσια του συστήματος.
Αν όχι, τοτε σωστά η λειτουργια εμπίπτει στην τρέχουσα υπηρεσια.

Commented [KK158]: Με βάση κριτήρια, όπως ...

Commented [KK159]: Η κατευθυνση ταξινόμησης μπορει να αλλάξει ή είναι fixed; Αν μπορει να αλλάξει, ας το τονίσετε στο κειμενο.

Commented [KK160]: Ενοποίηση των δύο bullets διότι μιλούν για το ίδιο πραγμα.

περαιτέρω επεξεργασία τους μέσω ειδικού μηχανισμού ranking που περιγράφεται σε επόμενη ενότητα.

Υλοποίηση (συγκεκριμένες κλάσεις)

Το υποσύστημα υλοποιείται από:

- **PlatformService** → περιέχει όλη τη λειτουργική λογική για CRUD, αναζητήσεις και ranking.
- **SearchService** → επιτρέπει την αναζήτηση και ταξινόμηση των πλατφορμών
- **RankingService** → επιτρέπει την κατάταξη των πλατφορμών
- **PlatformRepository** → παρέχει πρόσβαση στη βάση δεδομένων και υποστηρίζει δυναμικά φίλτρα (JPA Specifications), full-text search και auto-complete.
- **PlatformController (/api/platforms/)**** → εκθέτει όλα τα διαθέσιμα endpoints στο frontend.

Λειτουργίες της ενότητας

CRUD λειτουργίες

Η υπηρεσία υποστηρίζει πλήρη κύκλο ζωής των εγγραφών serverless πλατφορμών:

- **Create**: δημιουργία νέας πλατφόρμας με όνομα, περιγραφή, κατηγορία και τεχνικά χαρακτηριστικά (διαθέσιμη μόνο για ADMIN χρήστες).
- **Read**: ανάκτηση όλων των πλατφορμών ή επιλεκτικά βάσει ID.
- **Update**: ενημέρωση υπαρχόντων στοιχείων (διαθέσιμη μόνο για ADMIN χρήστες).
- **Delete**: διαγραφή πλατφόρμας (διαθέσιμη μόνο για ADMIN χρήστες).

Όλες οι λειτουργίες υλοποιούνται στην κλάση **PlatformService**, ενώ η πρόσβαση ελέγχεται από την ενότητα ασφάλειας μέσω ρόλων.

Αναζήτηση Πλατφορμών (Filtering)

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Commented [KK161]: Ποιο είναι το endpoint σε αυτή την περιπτωση;

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Υλοποιείται από: SearchService

Το σύστημα επιτρέπει στον χρήστη να εφαρμόσει πολλαπλά κριτήρια αναζήτησης:

- **όνομα πλατφόρμας**
- **κατηγορία**
- **περιγραφή**
- **γενικό keyword** (full-text search)
- **συνδυασμούς των παραπάνω**

Η αναζήτηση υλοποιείται με δύο τρόπους:

a) Απλή Αναζήτηση (JPA Specifications) (Ο όρος JPA Specifications αναφέρεται σε μηχανισμό της Spring Data JPA που επιτρέπει την δυναμική σύνθεση συνθηκών (WHERE clauses) με βάση προαιρετικά φίλτρα.)

Endpoint: /api/platforms/search

Για αναζήτηση με φίλτρα και keyword matching σε name, category και description, δηλαδή στα μεταδεδομένα των πλατφορμών.

b) Εμπλουτισμένη Αναζήτηση (Full-Text Search + Fuzzy Matching)

Endpoint: /api/platforms/search/fulltext

Η ενότητα αναζήτησης αξιοποιεί ειδικό μηχανισμό πλήρους κειμενικής αναζήτησης και fuzzy similarity, ώστε:

- να βρίσκει αποτελέσματα ακόμη και με ορθογραφικά λάθη,
- να υποστηρίζει πιο φυσική αναζήτηση (π.χ. “gogle” → “google cloud functions”),
- να εκτελεί βαθμολόγηση σχετικότητας.

Ο ακριβής τρόπος λειτουργίας αναλύεται στη σχετική υποενότητα του μηχανισμού αναζήτησης (Ενότητα 4.2.2).

Ταξινόμηση αποτελεσμάτων (Sorting)

Formatted: Greek

Commented [KK162]: Υπάρχει κάποιο endpoint σε αυτή την περιπτωση;

Formatted: Greek

Formatted: Greek

Formatted: Greek

Commented [KK163]: Τι εννοειτε εδώ; Δεν είναι ξεκάθαρο.

Commented [KK164]: Οκ αλλά πως ξεχωρίζουμε σε επίπεδο backend/endpoint ποιο ειδος αναζήτησης απαιτεί ο χρήστης;

Commented [KK165]: Ποιο είναι το endpoint σε αυτή την περιπτωση; Τι λαμβάνει ως εισοδο και τι παράγει ως έξοδο;

Η ταξινόμηση δεν υλοποιείται ως ξεχωριστό endpoint. Εφαρμόζεται ως παράμετρος στα endpoints αναζήτησης:

- **Στο GET /api/platforms/search μέσω των query params sortBy και sortDirection.**
- **Στο POST /api/platforms/search ή /search/fulltext μέσω των αντίστοιχων πεδίων στο FilterRequestDTO.**
- **Είσοδος:**
  - **sortBy (π.χ. name, category, κ.λπ.)**
  - **sortDirection (ASC / DESC)**
- **Έξοδος: Η ίδια σελίδα αποτελεσμάτων (Page<...>) ταξινομημένη.**

Το σύστημα επιτρέπει ταξινόμηση ως εξής:

- **sortBy**: name, category, createdAt, rank κ.λπ.
- **sortDirection**:
  - ASC (αύξουσα)
  - DESC (φθίνουσα)

Η κατεύθυνση ταξινόμησης είναι δυναμική και ορίζεται από τον χρήστη.

Αν ο χρήστης αλλάξει το κριτήριο ταξινόμησης, η ταξινόμηση εφαρμόζεται άμεσα χωρίς επανυπολογισμό των αποτελεσμάτων. Το backend είναι **stateless** και δεν διατηρεί "μνήμη" προηγούμενων αποτελεσμάτων αναζήτησης. Επομένως, κάθε αλλαγή σε sorting / filters / page οδηγεί σε **νέο request** από το frontend με ενημερωμένες παραμέτρους, και το backend επιστρέφει νέα ταξινομημένη σελίδα αποτελεσμάτων.

Κατατάξη Πλατφορμών

Υλοποιείται από: RankingService (**endpoint**: /api/platforms/rank) **Controller:** χρησιμοποιείται από PlatformController (π.χ. /api/platforms/rank). Συνεπώς, κατ'εξαίρεση, εδώ έχουμε τον ίδιο controller για 2 διαφορετικές υπηρεσίες.

Λειτουργίες:

Formatted: Greek

Commented [KK166]: Οκ αλλά σε επίπεδο backend, δεν διατηρειται κάποιο state όσον αφορά τα αποτελέσματα, σωστά; Άρα, είναι δουλειά του frontend να τα θυμάται και να ζητάει από το backend να τα αναταξινομήσει, σωστά; Ας το διευκρινήσετε εντος του κειμένου.

Formatted: Greek

Formatted: Greek

Formatted: Greek

Commented [KK167]: Υπάρχει κάποιο endpoint σε αυτή την περιπτωση;

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Commented [KK168]: Σε λογικό επίπεδο, έχει νόημα να πούμε εν τέλει πως έχουμε μια ενότητα που να διαχειριζεται τα πάντα για τις πλατφορμες (+ αναζητηση & κατάταξη); Αυτό βγάζει περισσοτερο νόημα κατά τη γνώμη μου.
Άρα, θα ενοποιούσα την Ενότητα Κατάταξης στην Ενότητα Διαχειρισης Πλατφορμών

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: Greek

- Υπολογισμός συνολικής βαθμολογίας κάθε πλατφόρμας.
- Υποστήριξη δυναμικών βαρών ανά χρήστη.
- Υποστήριξη πολλαπλών κριτηρίων (π.χ. μνήμη, latency, τιμή GB-sec).
- Παραγωγή λίστας πλατφορμών ταξινομημένων βάσει συνολικού σκορ.

4. Ενότητα (Διαχείρισης) Διατάξεων (Deployment Module)

Η Ενότητα Διαχείρισης Διατάξεων αυτοματοποιεί την ανάπτυξη (deployment) των serverless functions του χρήστη σε επιλεγμένους παρόχους νέφους. Η ενότητα δεν αντιμετωπίζεται ως ξεχωριστό υποσύστημα ή στρώμα της αρχιτεκτονικής, αλλά ως **μια ενότητα backend** που ακολουθεί τη CSR δομή (Controller–Service–Repository).

Η παρεχόμενη λειτουργικότητα (διάταξης) υλοποιείται από:

- **DeploymentService** (πυρήνας λογικής διάταξης)
- **DeploymentController** (/api/deployments) — εκθέτει τα endpoints προς το frontend
- Entity & Repository: **DeploymentRecord / DeploymentRepository**

Το Deployment Service χρησιμοποιεί επιμέρους βοηθητικές υπηρεσίες με τη μορφή **συστατικών πρόσβασης δεδομένων** (data-access component) του backend όπως:

- **PlatformRepository** για ανάκτηση στοιχείων πλατφόρμας
- **UserCredentialsService** για παροχή κρυπτογραφημένων credentials του χρήστη για μια serverless πλατφόρμα

  Τα credentials διάταξης **δεν αποστέλλονται μαζί με το αίτημα διάταξης** από τον χρήστη (ώστε να αποφεύγεται έκθεση μυστικών). Αντίθετα, το σύστημα διατηρεί credentials **ανά χρήστη και ανά πάροχο**, και κατά τη διάταξη:

  - Ο χρήστης επιλέγει πάροχο/πλατφόρμα.
  - Το backend ανακτά τα αντίστοιχα credentials μέσω της **UserCredentialsService**.
  - Τα credentials αποθηκεύονται **κρυπτογραφημένα** (EncryptionUtil) και χρησιμοποιούνται προσωρινά από τη διαδικασία διάταξης.

- **EncryptionUtil** για ασφαλή διαχείριση μυστικών

Formatted: Greek

Formatted: Greek

Commented [KK169]: Αυτή είναι υπηρεσια ή συστατικό ή απλώς κλάση υλοποίησης;

Commented [KK170]: Τι γινεται όμως με τα credentials που έχει ο χρηστης για συγκεκριμένη πλατφορμα; Αυτά τα διαχειρίζεστε με κάποιο τροπο ή τα παρέχει ο χρηστης στην αίτησή του; Στην τρεχουσα περιπτωση, για ποια credentials μιλάμε;

- **ObjectMapper** για ανάλυση JSON (π.χ. Google Cloud credentials)

Επιπλέον, το Deployment Service εκμεταλλεύεται την "Μηχανή Διάταξης" (Deployment Engine) για να υλοποιήσει την **εσωτερική λογική** του, δηλαδή να μπορεί να διατάσσει serverless συναρτήσεις. Αλλά το Deployment Engine σημειώνεται πως είναι **χαλαρά ενσωματωμένο (loosely coupled)** στο backend (δηλ. με το DeploymentService). Αυτό σημαίνει πως:

- χρησιμοποιεί κοινές υπηρεσίες υποδομής (DB, filesystem), αλλά
- **μπορεί να αντικατασταθεί** από διαφορετικό μηχανισμό (π.χ. πλήρως AWS SDK-based, Google Cloud Functions API, Azure Functions Core Tools) χωρίς να επηρεαστεί ο υπόλοιπος κώδικας της εφαρμογής.

Λειτουργίες της Υπηρεσίας

Endpoint: /api/deployments
Είσοδος: στοιχεία διάταξης + function package
Έξοδος: αποτέλεσμα διάταξης (status + endpoint + logs)

Επεξεργασία εισερχόμενων δεδομένων διάταξης

Αποσυμπίεση και προετοιμασία του πακέτου

Αυτόματη δημιουργία του serverless.yml

Εκτέλεση της διάταξης μέσω Serverless Framework CLI

Παρακολούθηση κατάστασης & καταγραφή logs

Η επικοινωνία με το backend γίνεται μέσω προγραμματιστικής διεπαφής: το backend δημιουργεί δυναμικά αρχεία ρυθμίσεων (serverless.yml), προετοιμάζει προσωρινούς φακέλους εργασίας, και εκτελεί το Serverless Framework μέσω ProcessBuilder, συλλέγοντας logs, exit codes και public endpoints της συνάρτησης. Ο χρήστης δεν αλληλεπιδρά με το CLI· η διάταξη υλοποιείται αποκλειστικά από το backend σε συνέργεια με το DeploymentEngine.

5. Ενότητα Observability (Ορατότητας) (Observability Module)

Η ενότητα αυτή είναι **εσωτερική του Backend** και όχι ξεχωριστό ανεξάρτητο τμήμα της συνολικής αρχιτεκτονικής. Ανήκει στο backend στρώμα διότι:

- λειτουργεί αποκλειστικά μέσα στη Spring Boot εφαρμογή,
- παρακολουθεί μόνο το backend και τις υπηρεσίες του,
- δεν αφορά αυτόνομα στοιχεία (όπως το frontend ή το deployment engine).

Formatted: Greek

Σκοπός του είναι να παρέχει ορατότητα για την κατάσταση του backend και για κρίσιμες λειτουργίες, όπως τις διατάξεις (deployments).

Η ενότητα βασίζεται στο **Spring Boot Actuator**, το οποίο ενεργοποιεί προκαθορισμένα REST endpoints για:

- **health checks** (π.χ. /actuator/health)
- **metrics** (π.χ. /actuator/metrics/**)
- **application info**
- **monitoring του thread pool, HTTP requests, JVM**

Τα Actuator endpoints λειτουργούν ως μία ελαφριά υπηρεσία παρακολούθησηςπαρακολούθησης (lightweight monitoring service) και ενσωματώνονται με φυσικό τρόπο στο backend.

ΣΤΡΩΜΑ ΔΕΔΟΜΕΝΩΝ & ΥΠΟΔΟΜΗΣ

Το στρώμα αυτό εντοπίζεται στο χαμηλότερο μέρος της γενικής αρχιτεκτονικής του συστήματος με αρμοδιότητες την κατάλληλη διαχείριση των επιχειρησιακών δεδομένων, την διάταξη συναρτήσεων και την παρακολούθηση της διάταξης αυτής. Προς αυτό τον σκοπό, απαρτίζεται από τα ακόλουθα συστατικά.

1. Database (PostgreSQL)

Η βάση δεδομένων χρησιμοποιείται ως το **μόνιμο αποθετήριο** δεδομένων του συστήματος.
Αποθηκεύει:

- **Χρήστες** με ρόλους, πλήρες όνομα, στοιχεία πρόσβασης και κατάσταση ενεργοποίησης.
- **Πλατφόρμες** serverless, με τεχνικά χαρακτηριστικά αποθηκευμένα σε μορφή **JSONB** για ευελιξία.
- **Ιστορικό διατάξεων** (deployments), καταγραφή χρόνων, κατάστασης και endpoint URLs.

2. Deployment Engine (Serverless Framework)

Η Deployment Engine αποτελεί μια υπηρεσία υποδομής, αντίστοιχη σε επίπεδο με τη διαχείριση της βάσης δεδομένων αλλά με διαφορετικό σκοπό: αναλαμβάνει την αυτοματοποιημένη διάταξη (deployment) serverless συναρτήσεων σε cloud providers. Η πλήρης εκμετάλλευση της υπηρεσιας αυτής γινεται από την Ενότητα Διάταξης (Deployment Module) του backend.

Η υπηρεσία υποστηρίζει απομόνωση περιβάλλοντος, καταγραφή logs και αποθήκευση χρόνων εκτέλεσης στη βάση. Αν και το Serverless Framework δεν αποτελεί καθαρό IaC εργαλείο (όπως Terraform[33] ή Pulumi[34]), αξιοποιεί τεχνικές IaC και μπορεί, εφόσον απαιτηθεί, να αντικατασταθεί με άλλες τεχνολογίες διάταξης. Η εφικτότητα αντικατάστασης αφορά κυρίως τη διαδικασία της διάταξης, ενώ η ενσωμάτωση παρακολούθησης (monitoring) μετά τη διάταξη εξαρτάται από τον εκάστοτε serverless platform provider.

4.2.2.1 Διάγραμμα Λογικών Συστατικών (Block Diagram)

Το παρακάτω διάγραμμα αποτελεί **Architectural Block Diagram (Διάγραμμα Λογικών Μπλοκ)** και παρουσιάζει σε υψηλό επίπεδο τη λογική δομή της αρχιτεκτονικής του συστήματος και τον τρόπο με τον οποίο αλληλεπιδρούν τα κύρια δομικά στοιχεία του.

Το σύστημα χωρίζεται σε τρία κύρια στρώματα:

1. Στρώμα Παρουσίασης (Frontend)

Περιλαμβάνει το περιβάλλον χρήστη, το οποίο χειρίζεται την πλοήγηση, τα UI components και τη συλλογή/υποβολή δεδομένων προς το backend.

2. Στρώμα Εφαρμογής (Backend – Controllers, Services, Repositories)

Αποτελεί τον πυρήνα της επιχειρησιακής λογικής. Σε αυτό ανήκουν:

- οι **Controllers**, οι οποίοι εκθέτουν HTTP endpoints προς το frontend,
- οι **Services**, όπου υλοποιείται η επιχειρησιακή λογική (αναζήτηση, κατάταξη, authentication, deployment),

[33] https://www.terraform.io/
[34] https://www.pulumi.com/

Formatted: Greek

Commented [KK171]: Ας δώσετε συνδέσμους με μορφη footnote για αυτά τα 2 πλαισια

Formatted: Greek

Commented [KK172]: Δεν εφαρμόζετε στυλ κεφαλίδας εδώ.

Commented [KK173]: Πάλι παρεχετε την λεπτομέρεια υλοποίησης για την χρηση του React που δεν χρειάζεται στη αποτυπωση της φάσης της σχεδιασης.

Field Code Changed

Field Code Changed

- τα **Repositories**, τα οποία παρέχουν πρόσβαση στη βάση δεδομένων μέσω μηχανισμού ORM.

  Τα Repositories **δεν αντιστοιχούν 1-1 με τις υπηρεσίες**, αλλά με τις **βασικές πληροφοριακές οντότητες** της εφαρμογής, όπως:

  - PlatformRepository → Ενσωματωνει την διαχειριση των serverless πλατφορμών ως πληροφοριακές οντοτητες.
  - UsersRepo → Ενσωματωνει την διαχειριση των χρηστών ως πληροφοριακές οντοτητες
  - DeploymentRepository → Ενσωματωνει την διαχειριση των διατάξεων πλατφορμών ως πληροφοριακές οντοτητες.

  Κάθε service χρησιμοποιεί τα αντίστοιχα repositories για τις απαιτούμενες πράξεις CRUD, χωρίς να υπάρχει υποχρεωτική αντιστοίχιση 1-1 μεταξύ service και repository.

3. Στρώμα Δεδομένων & Υποδομής (Infrastructure Layer)

Αποτελεί το στρώμα που φιλοξενεί:

- το **Σύστημα Διαχείρισης Βάσης Δεδομένων (PostgreSQL)**,
- την **Μηχανή Διάταξης (Deployment Engine)** που εκτελεί εντολές του Serverless Framework για λογαριασμό του backend.

Εξωτερικό Περιβάλλον / Context

Οι πάροχοι serverless πλατφορμών (AWS Lambda, Google Cloud Functions, Azure Functions) **δεν αποτελούν συστατικά του συστήματος**, αλλά **εξωτερικά συστήματα** με τα οποία το backend αλληλεπιδρά στο πλαίσιο της διάταξης συναρτήσεων του χρήστη.

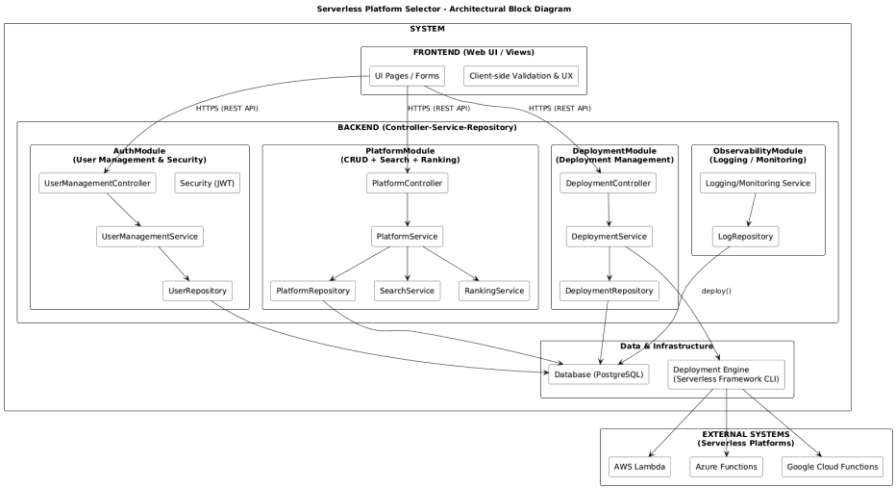
Formatted: Greek

Commented [KK174]: Σωστά! Αυτό σας έχω ζητήσει να πειτε και στην αρχή της 4.2. Οπότε, με αυτό τον τροπο θα υπάρχει συνέπεια στην αναφορά των 3 στρωμάτων της εφαρμογής.

Αλλά καλό είναι τα στρώματα να αναφερθούν από την αρχη και έπειτα να παρουσιαστει η λεπτομερής αρχιτεκτονική της εφαρμογής

Commented [KK175]: Ωραία αρχιτεκτονική εικόνα. Σας παρέχω ορισμένα σχόλια ώστε να την βελτιώσετε περαιτέρω:
(α) δεν χρειάζεται να δείξετε/έχετε το βέλος Results/Status
(β) δεν χρειάζεται να δείχνετε εντός των συστατικών πράγματα που δεν χρειάζονται ή είναι άσχετα. Για παράδειγμα, ο REST API Client δεν θεωρώ πως είναι μέλος του Front-end οπότε θα πρέπει να τον διαγράψετε. Το input validation είναι εσωτερικό χαρακτηριστικό του UI οπότε θα το αφαιρούσα κι αυτό
(γ) στο επίπεδο του Backend θα ήταν καλό να φαινονταν και τα repositories εφόσον εφαρμόζεται η CSR αρχιτεκτονική
(δ) καλό είναι να ξεχωρίσετε με κάποιο τρόπο το συστημά σας από τα εξωτερικά συστήματα (Cloud Functions). Θα πρότεινα προς αυτό τον σκοπό να εμφανίσετε ένα υπερ-συστατικό που να περιλαμβάνει όλα τα υπόλοιπα εκτός από το Cloud Functions
(ε) μετονομασια Cloud Functions σε FaaS Platforms ή Serverless Platforms
(στ) το βέλος μεταξύ Monitoring & Monitoring Integration είναι σωστό; Το ρωτώ διότι δεν είναι ξεκάθαρο αν θα έχουμε pull ή pushed των μετρικών. Λογικά, αν έχουμε pushed, θα έπρεπε το Deployment Engine να επικοινωνει με τον Controller (πράγμα που δεν το βλέπουμε). Αν είναι pull, τοτε το βέλος θα πρέπει να έχει ανάποδη φορά/κατεύθυνση (από το Monitoring προς το Monitoring Integration). Επίσης, το βέλος θα πρέπει να έχει ονομασία. Θα ήθελα να προσδιορίσω κάτι σε αυτό το σημείο: θεωρώ πως υπάρχουν 2 εναλλακτικές υλοποίησης στο pull-based μοντέλο. Η μία είναι αυτή που προδιέγραψα πριν. Η άλλη είναι το Monitoring Integration, αν είναι συστατικό που αντλεί μετρήσεις από το νέφος, αυτό να αποθηκεύει τις μετρήσεις στην βάση δεδομένων. ¨Επειτα, το Monitoring απλώς τις διαβάζει από την βάση. Συνεπώς, η επικοινωνία μεταξύ Monitoring & Monitoring Integration σε αυτή την περιπτωση θα είναι έμμεση.
(ζ) να πειτε το Monitoring σε Monitoring Service; Επειδή θεωρούμε πως ακολουθείτε το CSR μοτίβο. Παρομοίως, το JWT/Spring Security θα μπορούσε να μετονομαστει σε Security Service.
(η) το βέλος από το Monitoring στον Controller μου φαινεται πως είναι λάθος. Εν γένει, ας δείχνετε μόνο ποιο συστατικό εκκινεί τις αλληλεπιδράσεις με τα βέλη και όχι τις απαντήσεις που λαμβάνονται πίσω. Αν θέλετε να καλύψετε και τις απαντήσεις, ας χρησιμοποιείτε βέλη διαφορετικής μορφής (πχ. με διακεκομμένες γραμμές)

Formatted: Greek



Διάγραμμα 14: Διάγραμμα Λογικών Συστατικών (Architectural Block Diagram)

4.2.3 Διάγραμμα Κλάσεων (Class Diagram – Backend)

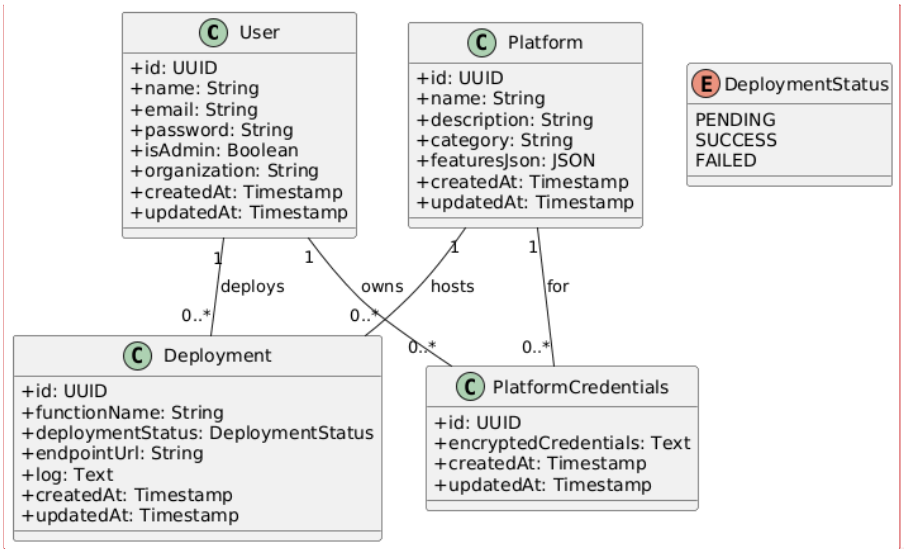
Το πληροφοριακό μοντέλο του συστήματος, το οποίο περιλαμβάνει τις βασικές οντότητες που διαχειρίζεται το σύστημα, αποθανατίζεται από το παρακάτω UML διάγραμμα κλάσεων. Το διάγραμμα αυτό παρουσιάζει τις βασικές κλάσεις του μοντέλου πεδίου (domain model) του backend και τις μεταξύ τους συσχετίσεις. Οι κλάσεις **User**, **Platform**, **Deployment** και **PlatformCredentials** ποτελούν τα κεντρικά δομικά στοιχεία της εφαρμογής, ενώ το enum **DeploymentStatus** χρησιμοποιείται για την τυποποιημένη αναπαράσταση της κατάστασης μιας διάταξης (PENDING, SUCCESS, FAILED). Οι κλάσεις αντιστοιχούν σε οντότητες που αποθηκεύονται στη βάση μέσω ORM, οπότε το διάγραμμα εστιάζει στη λογική δομή και όχι στη σχεσιακή αναπαράσταση πινάκων.

Η κλάση **User** περιλαμβάνει στοιχεία λογαριασμού και πρόσβασης (id, name, email, password, role, organization) καθώς και χρονικές σφραγίδες (createdAt, updatedAt). Η συσχέτιση **deploys** δηλώνει ότι ένας χρήστης μπορεί να εκτελέσει **0..*** διατάξεις (**Deployment**), ενώ κάθε διάταξη μπορεί να εκτελεστεί από **ακριβώς 1** χρήστη (πληθικότητα 1), κάτι που αποτυπώνει ότι μια διάταξη δεν μπορεί να υπάρξει χωρίς ιδιοκτήτη/αιτούντα χρήστη. Επιπλέον, η σχέση **owns** προσδιορίζει πως ένας χρήστης μπορεί να έχει πολλά διαπιστευτήρια σε serverless platforms (PlatformCredentials) (**0..***) αλλά ένα διαπιστευτήριο ανήκει πάντοτε σε έναν (**1**) μόνο χρήστη.

Η κλάση **Platform** περιγράφει μια serverless πλατφόρμα με πεδία όπως id, name, description, category και featuresJson (χαρακτηριστικά σε μορφή JSON), καθώς και

createdAt/updatedAt. Η συσχέτιση με την κλάση **Deployment** εκφράζεται με την σχέση **hosts**: μία πλατφόρμα μπορεί να φιλοξενεί **0..*** διατάξεις, ενώ κάθε διάταξη φιλοξενείται από **ακριβώς 1** πλατφόρμα. Από την άλλη μεριά, η συσχέτιση με την κλάση **PlatformCredentials** εκφράζεται με την σχέση **for**: μια πλατφόρμα μπορεί να αντιστοιχεί σε πολλά διαπιστευτήρια αλλά ένα διαπιστευτήριο ανήκει πάντοτε σε μια πλατφόρμα.

Τέλος, η κλάση **Deployment** λειτουργεί ως καταγραφή της διαδικασίας διάταξης μιας συνάρτησης, περιλαμβάνοντας το functionName, το endpointUrl που παράγεται μετά την επιτυχή διάταξη, το log (κείμενο εκτέλεσης/σφαλμάτων) και timestamps. Η τρέχουσα κατάσταση της διάταξης εκφράζεται μέσω της απαρίθμησης **DeploymentStatus**, επιτρέποντας στο σύστημα να παρακολουθεί αν η διάταξη βρίσκεται σε εξέλιξη, ολοκληρώθηκε επιτυχώς ή απέτυχε.



Διάγραμμα 2: Διάγραμμα Κλάσεων (Class Diagram – Backend)

4.2.4 Διαγράμματα Ακολουθίας

Παραθέτουμε παρακάτω ορισμένα διαγράμματα ακολουθίας που προσδιορίζουν τον τρόπο υλοποίησης σε υψηλό επίπεδο συγκεκριμένων βασικών λειτουργιών του συστήματος μέσω αλληλεπιδράσεων μεταξύ χρηστών και συστατικών του συστήματος.

1) Ροή διάταξης (Deployment Flow)

Formatted: Greek

Commented [KK176]: Το διάγραμμα θεωρώ πως είναι πλέον σε τελική μορφή. Απλώς επειδή μιλάμε για εικόνα, θα πρέπει να μπει λεζάντα από κάτω της.

Commented [KK177]: Εδώ θα πρεπει να ενσωματώσετε και να περιγράψετε όλα τα διαγράμματα ακολουθίας και όχι μόνο 1.
Λανθασμένα έχετε ενσωματώσει επιπροσθετα διαγράμματα ακολουθίας στην ενότητα 4.2.6 που αφορά διαγράμματα δραστηριοτήτων. Οπότε, όλα αυτά θα πρέπει να μεταφερθούν εδώ.

Επίσης, προτεινω εναλλαγή των ενοτήτων 4.2.4 & 4.2.5 με την λογική πως πρώτα ας δειξετε το πληροφοριακό μοντέλο και έπειτα τα διαγράμματα δραστηριοτήτων και ακολουθίας που καλύπτουν τις βασικές λειτουργιες του συστήματος.

Το παρακάτω διάγραμμα ακολουθίας περιγράφει με λεπτομέρεια τη διαδικασία διάταξης (deployment) μίας serverless συνάρτησης, από τη στιγμή που ο χρήστης υποβάλλει το Base64-encoded πακέτο μέχρι την ολοκλήρωση της διαδικασίας και την ενημέρωσή του για το αποτέλεσμα.

Το διάγραμμα αυτό επικεντρώνεται αποκλειστικά στη λειτουργικότητα Deploy Function, ενώ λειτουργίες όπως η αυθεντικοποίηση (login) και η αποσύνδεση (logout) καλύπτονται σε ξεχωριστά διαγράμματα, ώστε να αποφευχθούν άσκοπες επαναλήψεις.

Η ροή ξεκινά όταν ο χρήστης επισκεφθεί τη σχετική σελίδα. Πρώτα, εμφανίζεται η σχετική φόρμα και έπειτα αυτός υποβάλλει το Base64 αρχείο μέσω της διεπαφής (Frontend). Το αίτημα αποστέλλεται στο Backend, όπου το αντίστοιχο Controller εκτελεί βασική επικύρωση του request και το προωθεί στην υπηρεσία DeploymentService, η οποία αποτελεί το κεντρικό συστατικό της επιχειρησιακής λογικής.

Ακολουθεί σειρά ενεργειών: έλεγχος εγκυρότητας του Base64 περιεχομένου, αποκωδικοποίηση, εξαγωγή του πακέτου και δυναμική δημιουργία του αρχείου serverless.yml. Το συστατικό υποδομής Deployment Engine αναλαμβάνει να εκτελέσει τη διαδικασία διάταξης προς τον αντίστοιχο πάροχο (AWS, Azure ή GCP), μέσω του προσφερόμενου Cloud API του. Το διάγραμμα καλύπτει τόσο το επιτυχές όσο και το αποτυχημένο σενάριο (happy path & unhappy path), χρησιμοποιώντας κατάλληλα το UML στοιχείο alt για τις εναλλακτικές ροές.

Σε περίπτωση επιτυχημένης διάταξης, δημιουργείται και αποθηκεύεται ένα νέο Deployment μέσω του Repository καθώς και ενημερώνεται ο χρήστης. Σε περίπτωση αποτυχίας, ο χρήστης απλώς ενημερώνεται για το ατυχές αποτέλεσμα.

Το διάγραμμα αυτό αποκαλύπτει πλήρως τη συνεργασία μεταξύ των επιμέρους συστατικών του backend (Controller, Service, Repository, DeploymentEngine) και βοηθά στην κατανόηση της συμπεριφοράς του συστήματος σε μία από τις σημαντικότερες περιπτώσεις χρήσης (use cases) (δηλ. βασική λειτουργία) του έργου.

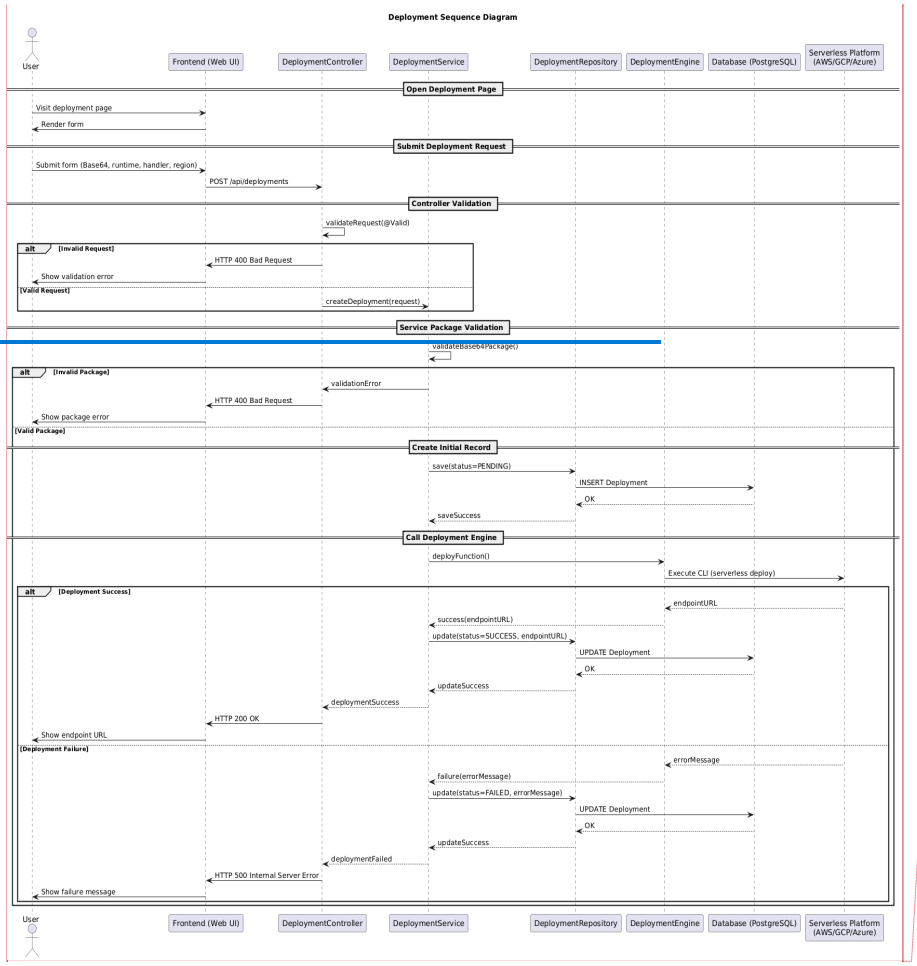
Formatted: Greek

Commented [KK178]: Το διάγραμμα είναι καλό αλλά έχει ορισμένα ζητήματα (δειτε επόμενο σχολιο). Επειδή του λείπει το DeploymentEngine ως σχετικό συστατικό του συστήματος, θα προτεινα να το ενσωματώσετε. Αν τώρα η πολυπλοκότητα του διαγράμματος αυξηθεί σημαντικά, μπορειτε να αφαιρέσετε ορισμένα unhappy paths.

Commented [KK179]: Λογικά πρώτα εμφανίζεται η σχετική φορμα εφοσον επισκεφθεί τη σχετική σελίδα ο χρηστης και έπειτα αυτός υποβάλλει το ZIP αρχειο.

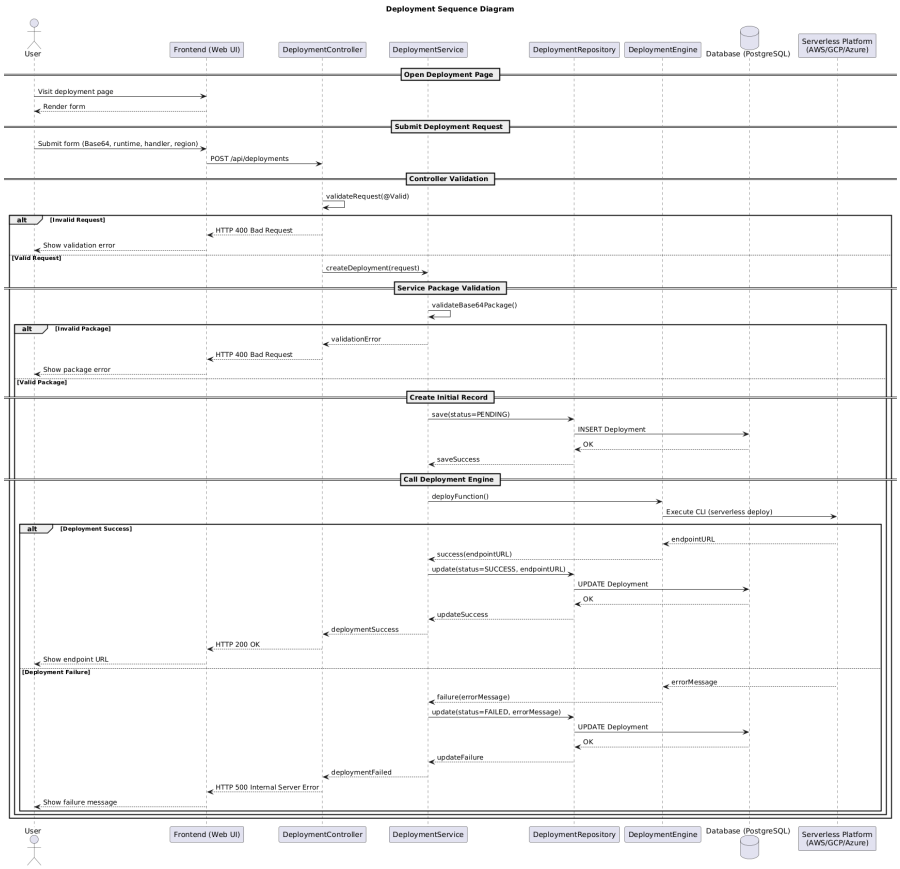
Commented [KK180]: Ορισμένα σχολια για το διάγραμμα:
(α) να μπει λεζάντα από κάτω του
(β) λειπει το DeploymentEngine, το οποίο είναι και υπεύθυνο για την διάταξη των συναρτήσεων. Οπότε, όταν το service θα πάει να κάνει deploy function, θα το κάνει μέσω του DeploymentEngine και όχι απευθειας με τη Serverless Platform. Από εκει και πέρα το DeploymentEngine θα είναι υπεύθυνο για την επικοινωνία με το ServerlessPlatform
(γ) συνήθως το validation ενός request γινεται από τον Controller μέσω του @Valid annotation. Αυτό λοιπόν θα πρέπει να εμφανιστει ως αυτό-αλληλεπίδραση στον Controller
(δ) κανονικά όταν έχουμε alt στοιχεια θα πρέπει να έχουμε 2 μέρη τους με συγκεκριμένες συνθήκες. Εσεις στα 2 πρώτα alt εμφανίζεται μόνο ένα μέρος που αντιστοιχει σε αρνητικές/unhappy συνθήκες.
Επιπλέον, κανονικά αν έχουμε λάθος του χρήστη, θα πρεπει να ολοκληρώνεται η ροή. Αυτό σημαινει πως στο πρώτο alt στο happy μέρος θα πρέπει να έχουμε το δεύτερο alt. Ενώ στο happy μέρος του δευτερου alt θα συνεχίζεται η ροή στο διάγραμμα.
(ε) όταν γινεται validate το πακέτο από την υπηρεσία, θα πρέπει να επιστρέφεται ένα μήνυμα πίσω στον controller και έπειτα να έχουμε το Alt
(στ) δεν καταλαβαινω για ποιο λογο χρειάζεται η append logs. Θα μπορούσε να αφαιρεθεί.
(ζ) θέλουμε μήνυμα επιστροφής από το repository στο service όταν είναι επιτυχής η ενημέρωση του deploymentrecord (επιτυχής διάταξη)
(η) αντιστοίχως, το service έπειτα θα πρέπει να ενημερώσει τον controller
(θ) τα (ζ) & (η) ζητήματα εμφανίζονται και στο unhappy path όταν η διάταξη αποτύχει.

Formatted: Greek



Commented [KK181]: Πολύ ωραία.
Έχουν μεινει μόνο 2 πράγματα:
(α) το δεύτερο καθολικό alt θα πρέπει να μπει εντος του happy path του πρώτου καθολικού alt
(β) στο κάτω μέρος του διαγραμματος, όταν έχουμε αποτυχία διάταξης, το updateSuccess μήνυμα θα πρέπει να μετονομαστεί σε updateFailure.

Formatted: Greek



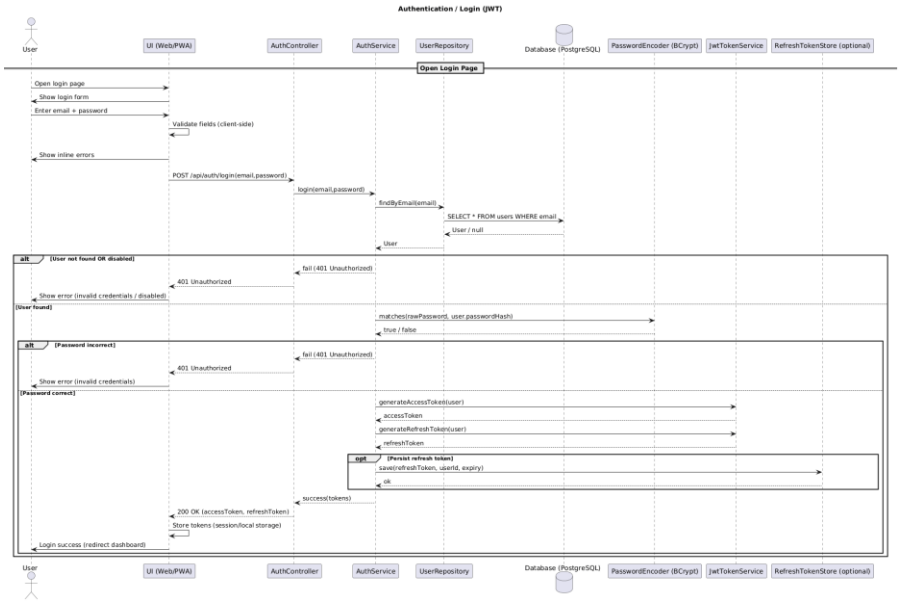
Διάγραμμα 33: Διάγραμμα ακολουθίας Ροής διάταξης (Deployment Flow)

2) Authentication / Login Flow (Ροή Αυθεντικοποίηση)

Το διάγραμμα “Authentication / Login” περιγράφει τη ροή αυθεντικοποίησης χρήστη και την έκδοση JWT. Ο χρήστης εισάγει email και κωδικό πρόσβασης, και το UI εκτελεί βασικό έλεγχο εγκυρότητας (π.χ. κενά πεδία). Στη συνέχεια αποστέλλεται αίτημα σύνδεσης στο backend, το οποίο αναζητά τον χρήστη στη βάση με βάση το email του. Αν ο χρήστης δεν υπάρχει ή ο λογαριασμός είναι απενεργοποιημένος, η διαδικασία τερματίζει με αποτυχία (π.χ. HTTP 401). Αν ο χρήστης είναι έγκυρος, γίνεται έλεγχος αντιστοίχισης του κωδικού με το αποθηκευμένο hash (BCrypt). Εφόσον επιβεβαιωθούν τα στοιχεία, εκτελείται η αυθεντικοποίηση μέσω AuthenticationManager και παράγονται JWT και

refresh token. Το UI αποθηκεύει τα tokens (π.χ. σε local storage/session) και ενημερώνει τον χρήστη για επιτυχή σύνδεση.

Σημειώνεται πως η αυθεντικοποίηση (authentication) αφορά την επιβεβαίωση της ταυτότητας, ενώ η εξουσιοδότηση (authorization) εφαρμόζεται έπειτα, όταν ο χρήστης επιχειρεί πρόσβαση σε προστατευμένα endpoints βάσει ρόλου/δικαιωμάτων.



Διάγραμμα 44: Διάγραμμα ακολουθίας Ροής Αυθεντικοποίηση (Authentication/Login flow)

Formatted: Greek

3) Search & Sorting (Αναζήτηση + Ταξινόμηση)

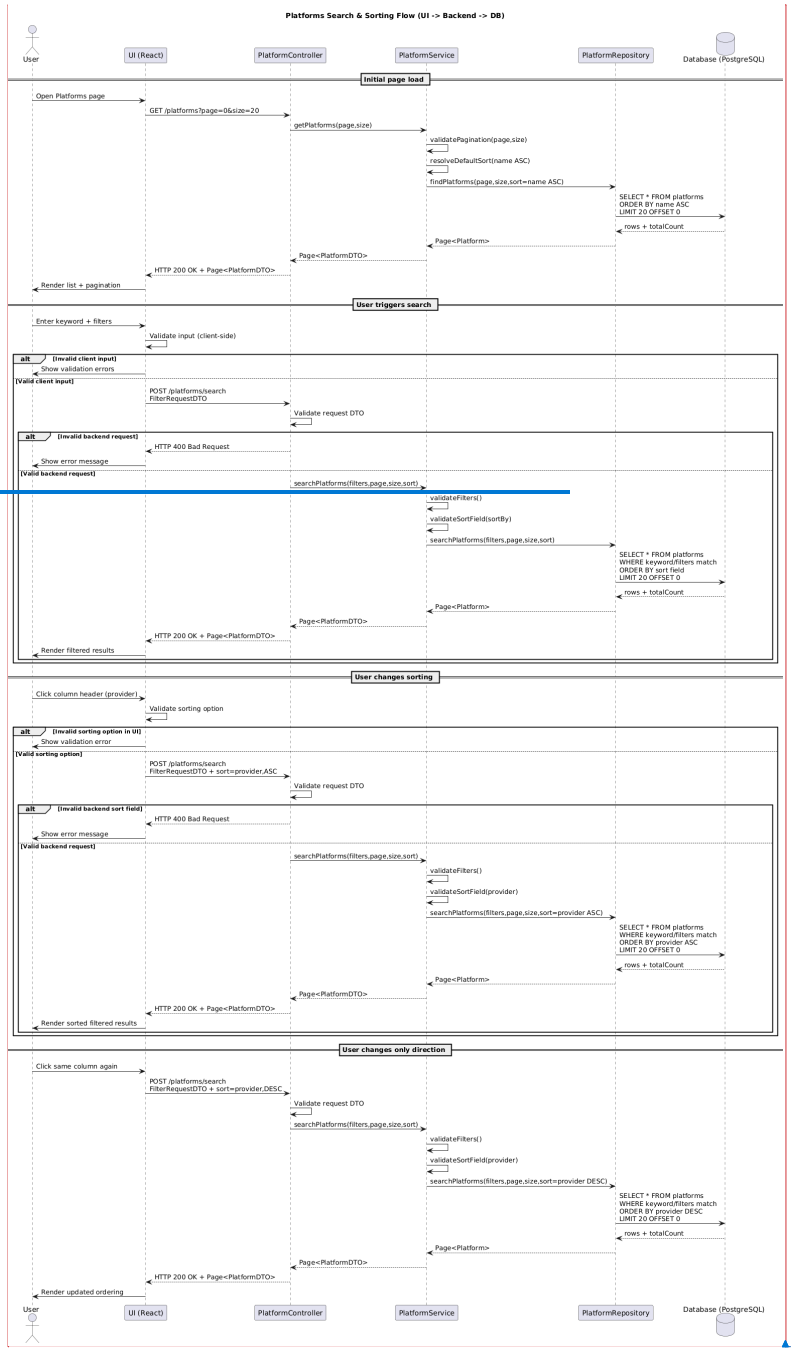
Το διάγραμμα “Search & Sorting” παρουσιάζει τη ροή αναζήτησης πλατφορμών και ταξινόμησης αποτελεσμάτων. Ο χρήστης εισάγει λέξεις-κλειδιά και φίλτρα (π.χ. κατηγορία, περιγραφή) και επιλέγει παραμέτρους ταξινόμησης (sortBy) και κατεύθυνση (sortDirection), καθώς και παραμέτρους σελιδοποίησης (page/size). Πριν την αποστολή του αιτήματος, το UI πραγματοποιεί βασικό έλεγχο εγκυρότητας (client-side validation), όπως έλεγχο κενών τιμών ή μη αποδεκτών επιλογών, με στόχο τη βελτίωση της εμπειρίας χρήστη και την αποφυγή περιττών αιτημάτων προς το backend. Ωστόσο, η κύρια και πλήρης επικύρωση πραγματοποιείται στο backend, όπου ελέγχεται η εγκυρότητα των παραμέτρων (π.χ. επιτρεπτά πεδία ταξινόμησης, φίλτρα, όρια σελιδοποίησης). Σε περίπτωση μη έγκυρων δεδομένων, το σύστημα επιστρέφει κατάλληλο μήνυμα σφάλματος (π.χ. HTTP 400 Bad Request), διασφαλίζοντας την ορθότητα και την ασφάλεια της λειτουργίας. Εφόσον το αίτημα είναι έγκυρο, το UI το αποστέλλει στο backend (FilterRequestDTO), όπου κατασκευάζεται δυναμικά το ερώτημα αναζήτησης/φιλτραρίσματος. Στη συνέχεια εφαρμόζεται ταξινόμηση πάνω στα αποτελέσματα με βάση το πεδίο που ζητήθηκε από τον χρήστη (π.χ. name) και την κατεύθυνση (ASC ή DESC), και τέλος εφαρμόζεται σελιδοποίηση για ελεγχόμενη επιστροφή δεδομένων. Το backend επιστρέφει Page<PlatformDTO>, και το UI εμφανίζει τη λίστα μαζί με controls πλοήγησης σελίδων και αλλαγής ταξινόμησης. Έτσι γίνεται σαφής διάκριση: η αναζήτηση/φιλτράρισμα καθορίζει “ποια” αποτελέσματα επιστρέφονται, ενώ η ταξινόμηση καθορίζει “με ποια σειρά” εμφανίζονται τα αποτελέσματα που ήδη επιλέχθηκαν.

Formatted: Greek

Commented [KK182]: Το διάγραμμα που παρέχετε καλύπτει μονο την ταξινόνηση και όχι την αναζητηση. Οποτε, θα πρέπει να επεκταθει για να καλύψει και αυτή την λειτουργια

Commented [KK183]: Χρειάζεται το UI να επικυρώσει το αιτημα πριν το στειλει στο backend;

Και το backend δεν θα πρεπει να προχωρήσει σε επικύρωση του αιτήματος που λαμβάνει;



Formatted: Greek

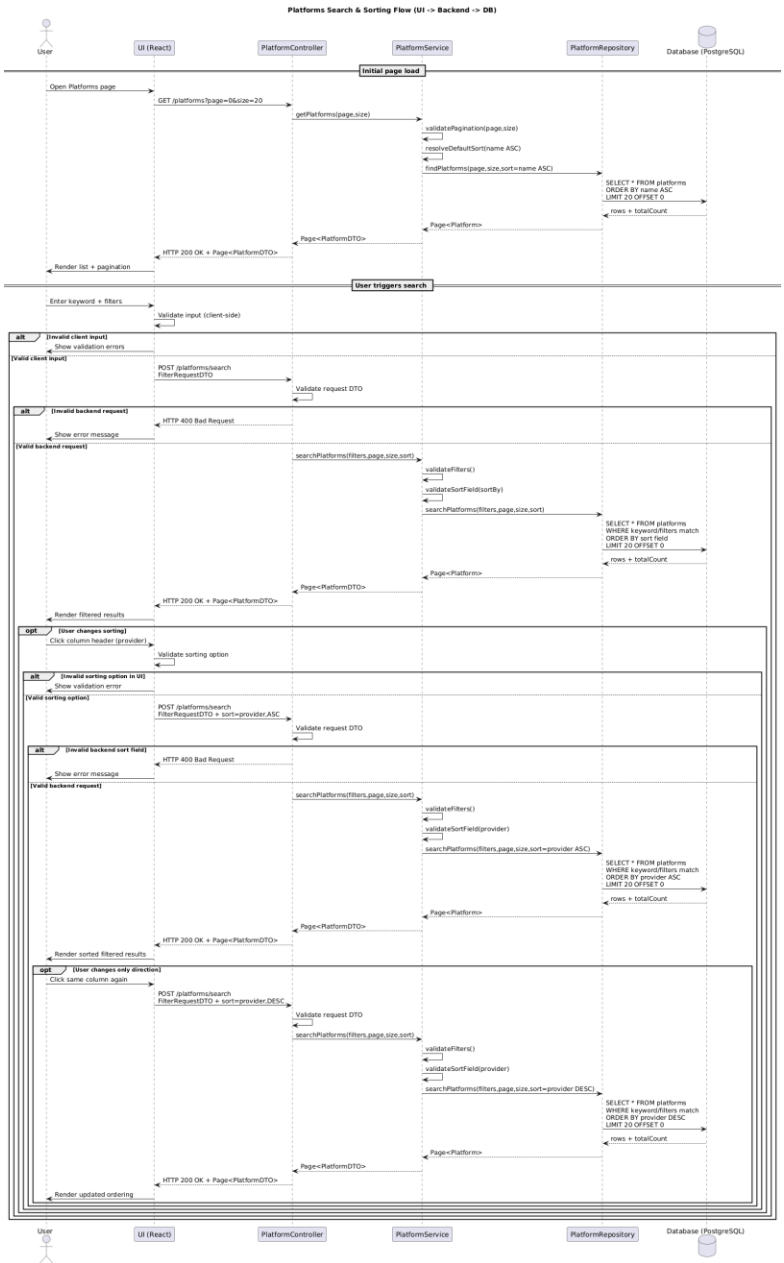
Commented [KK184]: Δύο σχολια εδώ:
(α) το user changes sorting κομμάτι/μέρος θα έπρεπε να μπει εντος του happy path στο πρώτο alt
(β) παρομοίως για τα υπόλοιπα 2 μέρη. Επίσης, τα 2 αυτά μέρη λογικά είναι προαιρετικά (δεν γινονται πάντοτε, οπότε θα μπορούσαν να μπουν μέσα σε ένα opt έκαστο.

Formatted: Greek

Formatted: Greek

Formatted: Greek

Formatted: English (United States)



Formatted: Greek

Formatted: Greek

Διάγραμμα 55: Διάγραμμα ακολουθίας Αναζήτησης και Ταξινόμησης

4) Ranking (Κατάταξη με κριτήρια & βάρη)

Το διάγραμμα "Ranking" αποτυπώνει τη ροή πολυκριτηριακής κατάταξης πλατφορμών με βάση κριτήρια και βάρη που παρέχει ο χρήστης. Ο χρήστης επιλέγει κριτήρια αξιολόγησης (π.χ. μνήμη, μέγιστος χρόνος εκτέλεσης, runtime κ.ά.) και προσδιορίζει βάρη (weights) που εκφράζουν τη σχετική προτεραιότητα κάθε κριτηρίου.

Σε επίπεδο διεπαφής χρήστη (UI), πραγματοποιείται βασικός έλεγχος εγκυρότητας (client-side validation), όπως έλεγχος απαιτούμενων πεδίων, αποδεκτών τιμών και ορθότητας των βαρών. Στη συνέχεια, το UI αποστέλλει ένα αντικείμενο τύπου RankingRequestDTO στο backend. Το backend πραγματοποιεί πλήρη επικύρωση του αιτήματος (server-side validation), διασφαλίζοντας την εγκυρότητα των κριτηρίων, των βαρών και των παραμέτρων που παρέχονται. Σε περίπτωση μη έγκυρων δεδομένων, επιστρέφεται κατάλληλο σφάλμα. Έπειτα, το backend ανακτά τις διαθέσιμες πλατφόρμες από τη βάση δεδομένων, μετατρέπει τα χαρακτηριστικά τους (featuresJson) σε κανονικοποιημένη δομή και υπολογίζει για κάθε πλατφόρμα ένα συνολικό score ως σταθμισμένο άθροισμα επιμέρους βαθμολογιών ανά κριτήριο.

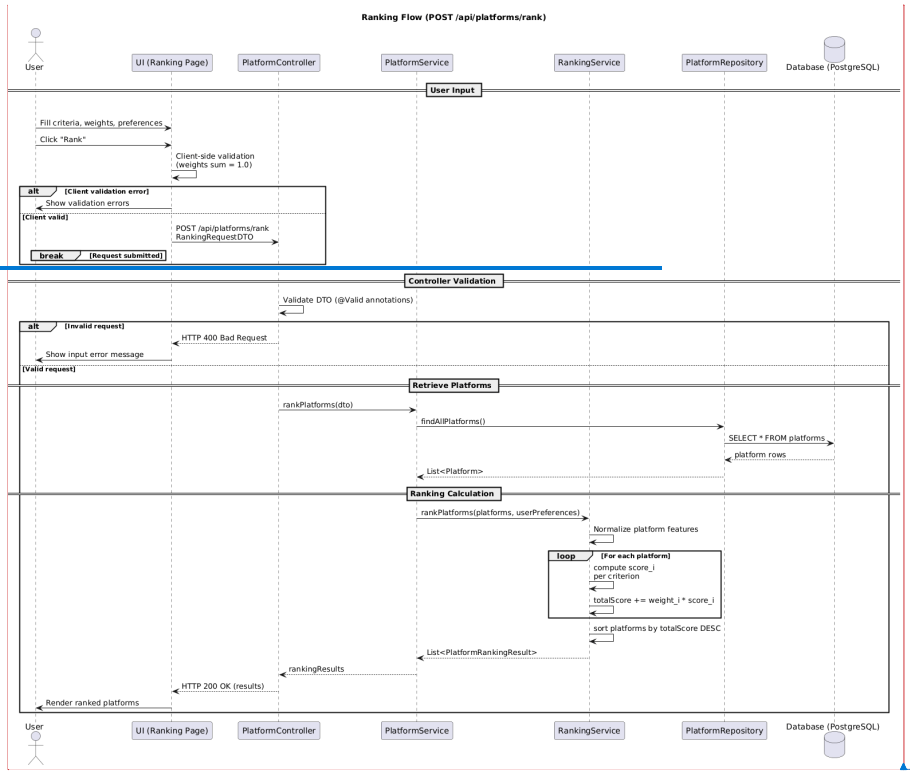
Η αξιολόγηση πραγματοποιείται με βάση τον τύπο των κριτηρίων:

- Για αριθμητικά κριτήρια εφαρμόζεται κανονικοποίηση min–max σε σχέση με τις τιμές όλων των πλατφορμών.

- Για κατηγορικά κριτήρια εφαρμόζεται σύγκριση συνόλων (set matching), υπολογίζοντας το ποσοστό των απαιτήσεων του χρήστη που καλύπτει η κάθε πλατφόρμα.

Τα αποτελέσματα επιστρέφονται ως ενιαία λίστα πλατφορμών, ταξινομημένη κατά φθίνουσα συνολική βαθμολογία (ranking score).

Με τον τρόπο αυτό, ο χρήστης λαμβάνει μία συνολική και συγκρίσιμη κατάταξη των διαθέσιμων πλατφορμών, όπου η σειρά εμφάνισης καθορίζεται αποκλειστικά από το υπολογισμένο score και όχι από απλά κριτήρια ταξινόμησης.

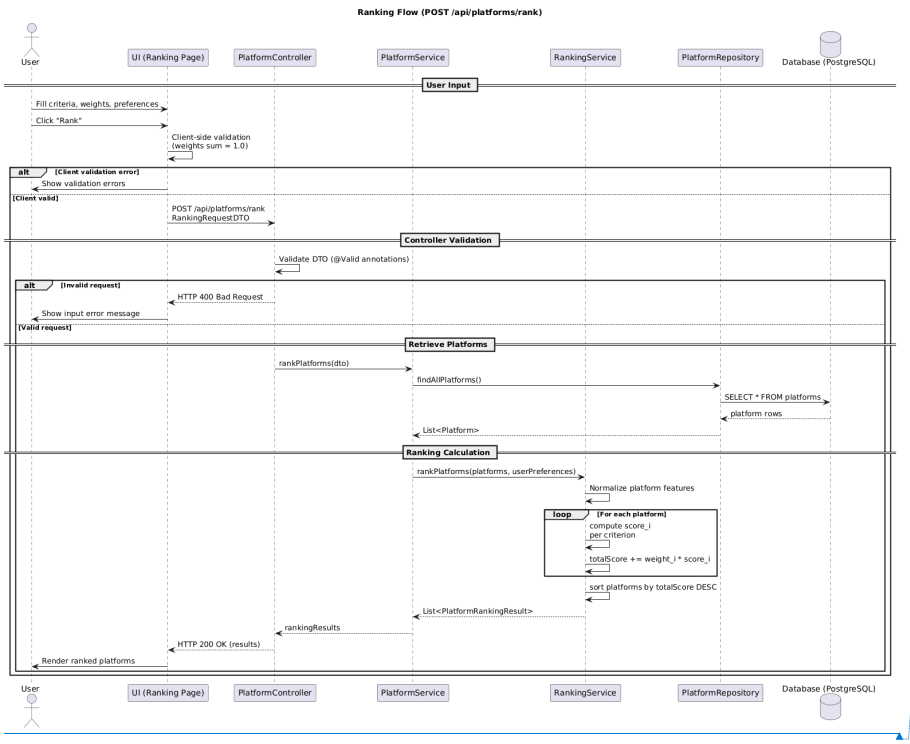


Formatted: Greek

Commented [KK185]: Ωραιο το διάγραμμα.
1 μονο σχολιο προς αντιμετώπιση: το μέρος από το Controller Validation και έπειτα θα πρεπει να μπει εντος του happy path στο πρώτο καθολικό alt

Formatted: Greek

Formatted: Greek



Διάγραμμα 66: Διάγραμμα ακολουθίας κατατάξης

4.2.5 Διαγράμματα Δραστηριοτήτων (Activity Diagram)

Παραθέτουμε παρακάτω τις διαδικασίες που υλοποιούν συγκεκριμένες βασικές λειτουργίες του συστήματός μας με την μορφή διαγραμμάτων δραστηριοτήτων.

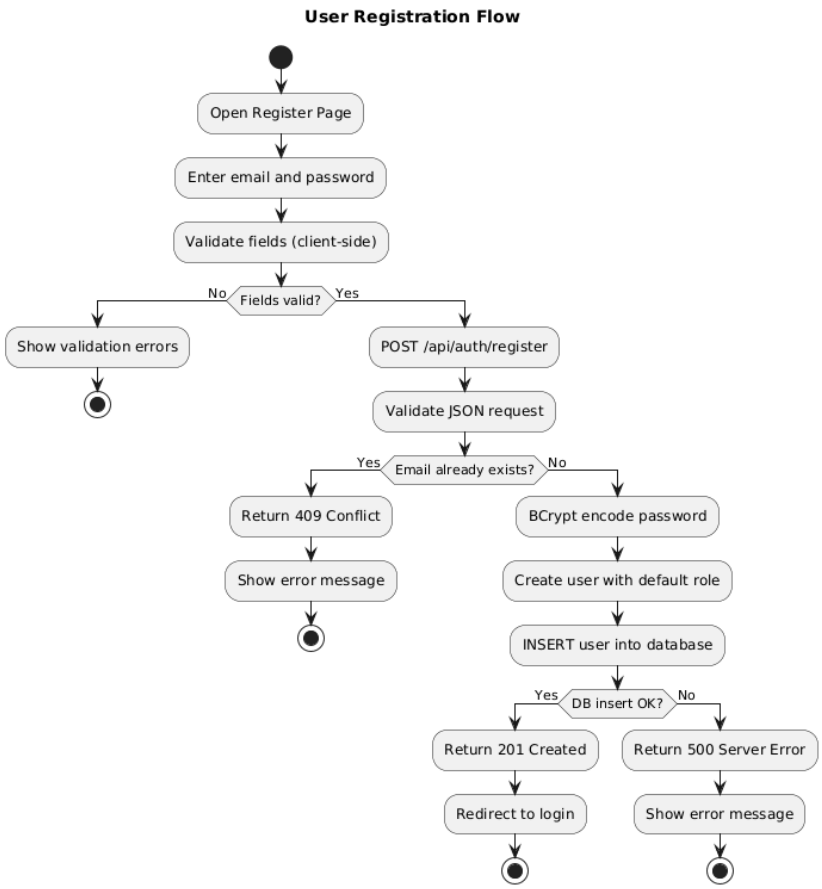
User Registration (Εγγραφή)

Το διάγραμμα δραστηριοτήτων “User Registration” αποτυπώνει τη ροή εγγραφής νέου χρήστη και τη συνεργασία μεταξύ του χρήστη, του UI και του backend. Η διαδικασία ξεκινά από τη συμπλήρωση της φόρμας εγγραφής (name/email/password) και έναν αρχικό έλεγχο εγκυρότητας στο frontend (client-side validation), ώστε να αποφευχθούν εμφανώς λανθασμένα αιτήματα. Εφόσον τα πεδία είναι έγκυρα, το UI αποστέλλει αίτημα εγγραφής στο endpoint του backend. Στο backend πραγματοποιείται επικύρωση του JSON request, έλεγχος ύπαρξης του email (ώστε να αποφευχθούν διπλές εγγραφές) και

Formatted: Greek

Formatted: English (United States)

κρυπτογράφηση/κατακερματισμός του κωδικού με BCrypt πριν την αποθήκευση. Σε περίπτωση επιτυχούς εισαγωγής στη βάση, επιστρέφεται επιβεβαίωση (π.χ. HTTP 201) και το UI ενημερώνει τον χρήστη. Σε περίπτωση σφαλμάτων (π.χ. ήδη υπάρχον email ή αποτυχία αποθήκευσης), επιστρέφονται κατάλληλα μηνύματα/κωδικοί σφάλματος και το UI εμφανίζει σχετική ειδοποίηση στον χρήστη.



Διάγραμμα 7-: Διαγράμμα Δραστηριοτήτων (Activity diagram User Registration)

Formatted: Greek

Commented [KK186]: Η εικόνα δεν έχει λεζάντα και είναι δυσκολα αναγνώσιμη οπότε το resolution της θα πρέπει να αυξηθεί.

Επίσης, παρεχω ορισμένα σχόλια για το διάγραμμα που εμφανίζεται:

1)Θα πρέπει να έχετε σωστή αναπαράσταση για αρχική και τελική κατάσταση όπως έγινε και στο προηγούμενο διάγραμμα

2)Εφοσον γινει εμφάνιση των σφαλμάτων, δεν χρειάζεται να τα δει ο χρηστης και πάμε αμέσως στην τελική κατάσταση

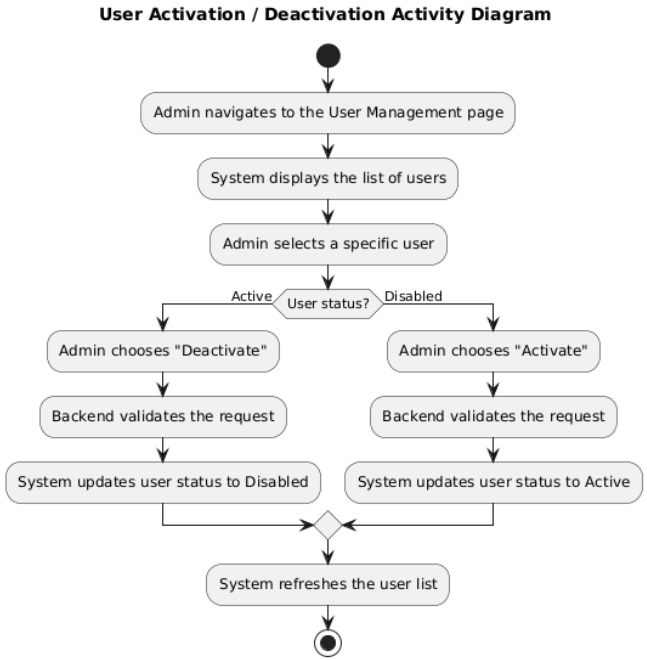
3)Ωραια η ομαδοποίηση των δραστηριοτήτων αλλά έχει δημιουργήσει και προβλήματα ορθότητας του διαγράμματος. Ειδικά εφοσον εστιάσουμε στην αλληλεπίδραση μεταξύ Database Operation & Server Processing. Δεν θα πρέπει να λογαριάζετε μια ομάδα ως μια σύνθετη δραστηριότητα. Συνεπώς, όταν έχουμε την δημιουργια του χρηστη στον server, έπειτα θα πάμε από αυτή την δραστηριότητα στην δραστηριότητα INSERT ... Επιπλέον, η επόμενη δραστηριότητα OK/FAIL δεν έχει νόημα οπότε θα πρέπει να διαγραφει. Θα προτεινα ως εξερχόμενη από την INSERT να πηγαινουμε στο decision point DB Insert OK?

Commented [KK187R186]: Θα προτιμούσα επίσης η μοντελοποίηση να γινει στο εργαλειο με το οποίο μοντελοποιήθηκε το προηγ. διάγραμμα για διάφορους λόγους: (α) το φόντο είναι λευκό, (β) χρησιμοποιούνται κατάλληλα στοιχεια διαγράμματος δραστηριοτήτων όπως αρχικές και τελικές καταστάσεις.

Formatted: Greek

Ένεργοποίηση/Απενεργοποίηση Χρήστη από διαχειριστή (user activation/deactivation performed by an administrator.)

Το διάγραμμα δραστηριοτήτων «Ενεργοποίηση / Απενεργοποίηση Χρήστη» αποτυπώνει τη ροή αλλαγής της κατάστασης ενός λογαριασμού χρήστη και τη συνεργασία μεταξύ του διαχειριστή, του UI και του backend. Η διαδικασία ξεκινά όταν ο διαχειριστής μεταβαίνει στη σελίδα διαχείρισης χρηστών, όπου το σύστημα εμφανίζει τη λίστα των εγγεγραμμένων χρηστών. Στη συνέχεια, ο διαχειριστής επιλέγει έναν συγκεκριμένο χρήστη και αποφασίζει αν θα ενεργοποιήσει ή θα απενεργοποιήσει τον λογαριασμό του, ανάλογα με την τρέχουσα κατάστασή του. Το αίτημα αποστέλλεται στο backend, όπου πραγματοποιείται επικύρωση της ενέργειας και ενημερώνεται η κατάσταση του επιλεγμένου χρήστη. Μετά την ολοκλήρωση της διαδικασίας, το σύστημα ανανεώνει τη λίστα χρηστών, ώστε να αποτυπώνεται η νέα κατάσταση του λογαριασμού.



Διάγραμμα 88: Διάγραμμα δραστηριοτήτων ενεργοποίησης/απενεργοποίησης χρήστη από διαχειριστή.

4.3 Υλοποίηση Συστήματος (System Implementation)

Η υλοποίηση του συστήματος πραγματοποιήθηκε με χρήση Java 21, Spring Boot, ReactJS και PostgreSQL, αξιοποιώντας σύγχρονες τεχνολογίες ανάπτυξης web εφαρμογών.

Στο επίπεδο του backend εφαρμόζεται το μοτίβο **Controller–Service–Repository (CSR)**, το οποίο εξασφαλίζει καθαρό διαχωρισμό ευθυνών, επεκτασιμότητα και οργανωμένη δομή κώδικα.

Στις επόμενες ενότητες παρουσιάζεται αναλυτικά:

- η λειτουργία των βασικών **υπηρεσιών (services)** του backend,
- οι **controllers** που εκθέτουν τις αντίστοιχες διεπαφές,
- καθώς και οι απαραίτητες **διαμορφώσεις συστήματος** (όπως CORS, ασφάλεια, logging και αρχεία ρυθμίσεων).

Η οργάνωση αυτή επιτρέπει η υλοποίηση να παραμείνει καθαρή, επεκτάσιμη και ευθυγραμμισμένη με τη συνολική αρχιτεκτονική του συστήματος.

4.3.1 Cross-Origin Resource Sharing (CORS) Configuration

Η διαχείριση CORS δεν αποτελεί αυτόνομο υποσύστημα του backend· είναι μια **διαμορφώσιμη πολιτική ασφαλείας** εντός του Spring Boot, καθώς αφορά τον τρόπο με τον οποίο εξουσιοδοτούνται εξωτερικά frontend περιβάλλοντα να επικοινωνούν με το API του συστήματος. Στο πλαίσιο αυτό, υλοποιήθηκε η κλάση **CorsConfig**, η οποία καθορίζει ποια domains έχουν δικαίωμα πρόσβασης στις HTTP διεπαφές / τελικά σημεία (endpoints) του Spring Boot backend.

Η ανάγκη για CORS προκύπτει επειδή το frontend (React) εκτελείται σε διαφορετική θύρα (π.χ. localhost:5173) από το backend (localhost:8080). Το πρόγραμμα πλοήγησης, για λόγους ασφαλείας, μπλοκάρει εξ' ορισμού cross-origin αιτήματα, και επομένως απαιτείται ρητή πολιτική που να επιτρέπει την επικοινωνία των δύο συστατικών.

Η υλοποίηση της πολιτικής CORS πραγματοποιείται με χρήση ενός bean τύπου **WebMvcConfigurer**, το οποίο ορίζει:

- **Τα επιτρεπόμενα origins**, όπως το περιβάλλον ανάπτυξης του React,
- **Τις επιτρεπόμενες HTTP μεθόδους**, συμπεριλαμβανομένων των GET, POST, PUT και DELETE,

Formatted: Greek

Commented [KK188]: Αυτά δεν τα ειπατε ήδη;

Commented [KK189]: Υποσχεθήκατε πως αυτό θα πρέπει να είναι ένα υπο-σύστημα εφοσον πάτε σε ανάλυση υπο-συστημάτων. Αλλά μοιάζει να είναι μια συγκεκριμένη λεπτομέρεια που αφορά την διαμόρφωση συστήματος, σωστά;

Συνεπώς, θα πρέπει να δειτε που θα πρέπει να ενταχθει η ανάλυση αυτής της διαμόρφωσης - ίσως σε ξεχωριστή ενότητα της 4.3 που να εστιάζει σε όλες τις απαραίτητες διαμορφώσεις του συστήματος;

- **Την υποστήριξη μεταφοράς credentials** (cookies, authorization headers),
- **Explicit allow-list & deny-by-default security policy:** Αντί γενικής χαρτογράφησης /**, ορίζονται ρητά τα δημόσια endpoints (π.χ. /api/auth/register, /api/auth/login, /api/auth/refresh, Swagger) και τα read-only endpoints των πλατφορμών (GET /api/platforms/**). Όλες οι υπόλοιπες διεπαφές απαιτούν αυθεντικοποίηση, ενώ οι λειτουργίες τροποποίησης δεδομένων (POST/PUT/DELETE) προστατεύονται με ρόλο ADMIN. Με αυτόν τον τρόπο αποτρέπεται η ακούσια έκθεση νέων endpoints σε μελλοντικές επεκτάσεις του backend.

Η διαμόρφωση αυτή αποτελεί μια **τυπική αλλά απαραίτητη ρύθμιση ασφαλείας**, που επιτρέπει στο React frontend να επικοινωνεί με ασφαλή και ελεγχόμενο τρόπο με το Spring Boot backend, διασφαλίζοντας παράλληλα ότι σε περιβάλλον παραγωγής μπορούν να οριστούν διαφορετικά, πιο περιοριστικά origins (π.χ. domain του πραγματικού συστήματος).

4.3.2 Τεκμηρίωση REST API (OpenApiConfig)

Η OpenApiConfig ενεργοποιεί το **Swagger UI** μέσω της βιβλιοθήκης **springdoc-openapi**, παρέχοντας ένα περιβάλλον για χειροκίνητες δοκιμές των τελικών σημείων (endpoints) και αυτόματη τεκμηρίωση του (προσφερόμενου) API (που προσφέρεται από το backend).

4.3.3 Ενότητα Ασφάλειας (Security Module)

Η ενότητα ασφάλειας αποτελεί μία από τις βασικές μονάδες του backend και αντιστοιχεί στην ενότητα ασφαλείας του συνολικού αρχιτεκτονικού μοντέλου της εφαρμογής. Στόχος της είναι να διασφαλίσει ότι μόνο αυθεντικοποιημένοι και εξουσιοδοτημένοι χρήστες μπορούν να αποκτούν πρόσβαση στις λειτουργίες του συστήματος, εφαρμόζοντας πολιτικές προστασίας σε όλα τα REST endpoints.

Η υλοποίηση βασίζεται στο **Spring Security**, το οποίο λειτουργεί ως κεντρικό πλαίσιο επιβολής πολιτικών ασφαλείας, και στη χρήση **JWT (JSON Web Tokens)** για stateless αυθεντικοποίηση. Με το μοντέλο αυτό, το backend δεν διατηρεί sessions· κάθε αίτημα φέρει το δικό του token, το οποίο επαληθεύεται πριν επιτραπεί η πρόσβαση στον εκάστοτε controller.

Η ενότητα αυτή αποτελείται από τέσσερα βασικά δομικά στοιχεία (ουσιαστικά κλάσεις κώδικα):

1. SecurityConfig – Κεντρική διαμόρφωση κανόνων ασφαλείας

Η κλάση SecurityConfig καθορίζει τις πολιτικές ασφαλείας σε όλο το backend:

Formatted: Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Normal, Indent: Left: 0,63 cm, No bullets or numbering

Commented [KK190]: Οκ - αλλά η ασφάλεια αντιστοιχει σε κάποια υπηρεσία; Από αυτές που έχουν μοντελοποιηθεί στην αρχιτεκτονική του συστήματος; Αν ναι, ποια είναι η υπηρεσια αυτή και τι λειτουργίες προσφέρει. Ποια είναι η σχεση των τεχνολογιών Spring Security με αυτή την κλάση και τις λειτουργιες της;

Commented [KK191]: Ας το πειτε απλώς υπηρεσια ασφάλειας όπως έχω αναφέρει και σε προηγούμενα σχόλιά μου. Δεν πρέπει να χρησιμοποιείτε τον όρο υποσύστημα ..Ας τον αντικαταστήσετε με τον όρο συστατικό.

Commented [KK192R191]: Ή μονάδα / ενότητα (module)

Commented [KK193]: Αν όλα αυτά είναι κλάσεις ας πειτε σε παρενθέσεις
(ουσιαστικά κλάσεις κώδικα).

- Ορίζει ποια endpoints είναι δημόσια (π.χ. /auth/**, Swagger UI, public APIs).
- Περιορίζει λειτουργίες, όπως δημιουργία/τροποποίηση πλατφορμών, σε χρήστες με ρόλο **ADMIN**.
- Ενεργοποιεί *stateless* λειτουργία μέσω SessionCreationPolicy.STATELESS.
- Προσθέτει το JWTAuthFilter στην αλυσίδα των filters, ώστε η επαλήθευση των tokens να εκτελείται πριν την πρόσβαση στους controllers.
- Καθορίζει τον μηχανισμό αυθεντικοποίησης μέσω DaoAuthenticationProvider, ο οποίος συνδέεται με το OurUserDetailsService (υπηρεσία που εντοπίζει χρήστες στη βάση με βάση το email τους) και το BCryptPasswordEncoder (αλγόριθμος hashing για ασφαλή αποθήκευση και έλεγχο κωδικών).

Με αυτόν τον τρόπο, η SecurityConfig διαμορφώνει σωστά όλες τις λειτουργίες του αρχιτεκτονικού στοιχείου Security Module.

2. JWTAuthFilter – Έλεγχος εγκυρότητας κάθε JWT token

Η κλάση JWTAuthFilter εκτελείται σε κάθε εισερχόμενο αίτημα ως OncePerRequestFilter (φίλτρο που ενεργοποιείται εφάπαξ ανά αίτημα χρήστη) και υλοποιεί τον πυρήνα της ροής JWT authentication:

1. Ανακτά το JWT από το Authorization header (κεφαλίδα αυθεντικοποίησης αιτήματος χρήστη).
2. Εξάγει από το token το email του χρήστη και τον ρόλο του (claim).
3. Ελέγχει την εγκυρότητα του token μέσω της JWTUtils (υπογραφή, διάρκεια, στοιχεία).
4. Δημιουργεί αντικείμενο Authentication με τον σωστό ρόλο (USER ή ADMIN).
5. Εισάγει το Authentication στο SecurityContextHolder (δομή-thread-local της Spring Security που κρατάει την ταυτότητα του χρήστη για όλη τη διάρκεια της επεξεργασίας του αιτήματος), ώστε το backend να αναγνωρίζει τον χρήστη κατά την εκτέλεση του controller.

Αν το token λείπει ή είναι άκυρο, το αίτημα είτε περνάει ως μη αυθεντικοποιημένο είτε απορρίπτεται με HTTP 401. Το φίλτρο απλώς **δεν τοποθετεί Authentication στο SecurityContextHolder**, και η τελική απόφαση λαμβάνεται από τους **κανόνες authorization** που έχουν οριστεί στο SecurityConfig.

Formatted: Greek

Commented [KK194]: Ας πειτε σε παρενθέσεις με μια συντομη φράση τι είναι αυτά - λογικά θα αναλύονται στην συνέχεια οπότε δεν χρειάζονται ολόκληρες προτάσεις ή παράγραφοι.

Commented [KK195]: Οκ - ας πειτε σε παρενθέσεις τι είναι αυτό.

Commented [KK196]: Οκ αλλά ποιος το αποφασιζει αυτό και σε ποιες περιπτώσεις;
Χρειάζεται η γνώση του τελικού σημείου της αιτησης ώστε να προκύψει η σχετική απόφαση;

Formatted: Greek

Formatted: Greek

3. OurUserDetailsService – Ανάκτηση και επαλήθευση χρηστών

Η κλάση OurUserDetailsService συνεργάζεται με τον authentication provider (πάροχο αυθεντικοποίησης) και έχει την ευθύνη φόρτωσης των χρηστών από τη βάση δεδομένων. Επιστρέφει αντικείμενα Spring Security που ανήκουν στην κλάση UserDetails, στα οποία περιλαμβάνονται οι εξής πληροφορίες:

- email χρήστη,
- hashed password,
- ρόλος (authority).

Αυτή η υπηρεσία χρησιμοποιείται κυρίως κατά την αυθεντικοποίηση χρήστη (login).

Formatted: Greek

4. JWTUtils – Μηχανισμός δημιουργίας και επαλήθευσης tokens

Η JWTUtils αποτελεί την «token engine» (μηχανή μαρκών αυθεντικοποίησης) του υποσυστήματος και υλοποιεί τις εξής λειτουργίες:

- δημιουργία JWT token κατά την αυθεντικοποίηση χρήστη,
- εξαγωγή claims (username, role, expiration),
- έλεγχο εγκυρότητας και υπογραφής των (δημιουργημένων) tokens,
- χειρισμός λήξεων (tokens) και εξαιρέσεων (exceptions).

Χάρη στην ύπαρξη αυτής της κλάσης, το σύστημα διαχωρίζει καθαρά τη λογική δημιουργίας tokens από τη λογική ασφάλειας του HTTP pipeline (δηλ. του αγωγού επεξεργασίας των αιτημάτων του χρήστη).

Formatted: Greek

Συνολικός ρόλος στην αρχιτεκτονική

Η Ενότητα Ασφάλειας λειτουργεί ως ενδιάμεσο επίπεδο ανάμεσα στο UI και στους controllers.
Πριν εκτελεστεί οποιαδήποτε επιχειρησιακή λειτουργία, κάθε αίτημα περνά από τον μηχανισμό JWT–based ελέγχου, ο οποίος:

- επαληθεύει την ταυτότητα του χρήστη,

- επιβεβαιώνει τον ρόλο του,
- αποτρέπει μη εξουσιοδοτημένη πρόσβαση,
- εξασφαλίζει stateless επικοινωνία.

4.3.4 Ενότητα Διαχείρισης Πλατφορμών (Platform Management Module)

Η Ενότητα Διαχείρισης Πλατφορμών είναι υπεύθυνη για την αποθήκευση, ενημέρωση, ανάκτηση και αναζήτηση των serverless πλατφορμών, καθώς και για τη διάθεση των σχετικών λειτουργιών μέσω REST API προς το frontend. Σε επίπεδο υλοποίησης, η ενότητα αυτή εκτίθεται μέσω της υπηρεσίας **PlatformService** και του αντίστοιχου ελεγκτή **PlatformController**, οι οποίοι συνεργάζονται με το **PlatformRepository** για την διαχείριση των δεδομένων των πλατφορμών στην υποκείμενη βάση δεδομένων μέσω ORM.

Η βασική λειτουργικότητα της υπο-ενότητας περιλαμβάνει CRUD πράξεις (Create, Read, Update, Delete) για τις πλατφόρμες, καθώς και πιο προχωρημένους μηχανισμούς αναζήτησης και αξιολόγησης. Το συστατικό δεν περιορίζεται μόνο σε CRUD λειτουργίες, αλλά υποστηρίζει τρεις διακριτές κατηγορίες λειτουργιών:

1. Αναζήτηση (Search): εύρεση σχετικών πλατφορμών

Endpoint: POST /api/platforms/search/fulltext

Η αναζήτηση πραγματοποιείται με συνδυασμό τεχνικών που παρέχει η PostgreSQL:

- **Full-text search** (χρήση tsvector, tsquery, ts_rank_cd)
    - **tsvector**: εσωτερική αναπαράσταση κειμένου "έτοιμη για αναζήτηση". Η PostgreSQL μετατρέπει ένα κείμενο (π.χ. description) σε **tokens/lexemes** (λέξεις σε κανονικοποιημένη μορφή), ώστε να μπορεί να γίνει αποδοτική full-text αναζήτηση.
    - **tsquery**: είναι η δομή που εκφράζει τι ψάχνει ο χρήστης (λέξεις/τελεστές), ώστε να ταιριάξει πάνω σε tsvector.
    - **ts_rank_cd**: συνάρτηση **βαθμολόγησης σχετικότητας**. Όταν ταιριάζει το tsquery σε tsvector, η PostgreSQL υπολογίζει ένα **rank** (score) που εκφράζει πόσο σχετικό είναι το αποτέλεσμα. Το ts_rank_cd δίνει βαρύτητα και στην "κάλυψη"/πυκνότητα αντιστοιχίσεων.

Formatted: Greek

Commented [KK197]: Δεν θα πρεπει να εξηγηθούν τι είναι όλα αυτά; Το ίδιο και για το pg_trgm & similarity()

- **Fuzzy matching** μέσω του PostgreSQL extension **pg_trgm** και της συνάρτησης similarity()
    - **pg_trgm**: extension της PostgreSQL που βασίζεται σε **trigrams** (ακολουθίες 3 χαρακτήρων). Χρησιμοποιείται για να μετρήσει ομοιότητα μεταξύ strings ακόμη κι όταν υπάρχουν μικρά λάθη.
    - **Similarity (a, b)**: συνάρτηση που επιστρέφει έναν αριθμό (συνήθως 0..1) που εκφράζει πόσο μοιάζουν δύο strings βάσει trigrams.

Έτσι, ο χρήστης μπορεί να εντοπίζει πλατφόρμες ακόμη και όταν το keyword δεν ταιριάζει ακριβώς με το όνομα ή την περιγραφή μιας πλατφόρμας. Οι τεχνικές αυτές επιστρέφουν ένα σύνολο «σχετικών» αποτελεσμάτων, πάνω στο οποίο εφαρμόζονται οι επόμενες λειτουργίες.

1. **Ταξινόμηση (Sorting): αναδιάταξη των αποτελεσμάτων**

    Endpoint: δεν υπάρχει ξεχωριστό sorting endpoint. Η ταξινόμηση δίνεται ως παράμετρος στα endpoints αναζήτησης:
    GET /api/platforms/search?sortBy=...&sortDirection=...

    Αφού εντοπιστούν οι υποψήφιες πλατφόρμες μέσω της αναζήτησης, ο χρήστης μπορεί να αλλάξει τον τρόπο με τον οποίο εμφανίζονται (π.χ. αύξουσα/φθίνουσα ταξινόμηση κατά όνομα, κατηγορία ή άλλο πεδίο).

    Η ταξινόμηση δεν επηρεάζει τα αποτελέσματα που επιστρέφονται — μόνο τη σειρά παρουσίασης.

2. **Κατάταξη (Ranking): πολυκριτηριακή βαθμολόγηση**

    Endpoint: POST /api/platforms/rank

    Η κατάταξη αποτελεί ξεχωριστή διαδικασία από την αναζήτηση και την ταξινόμηση.
    Σε αντίθεση με το full-text relevance, η κατάταξη βασίζεται:

    - Στα τεχνικά χαρακτηριστικά κάθε πλατφόρμας

Formatted: Greek

Formatted: Greek

Commented [KK198]: Θέλουμε κι εδώ να δούμε το endpoint

Formatted: Greek

- Στα κριτήρια που επιλέγει ο χρήστης
- Στα βάρη που αποδίδει ο χρήστης σε αυτά τα κριτήρια
- Σε συναρτήσεις ωφέλειας (utility functions)

Το αποτέλεσμα είναι ένα **συνολικό score** για κάθε πλατφόρμα, βάσει του οποίου προκύπτει η τελική πολυκριτηριακή κατάταξη.

Διαθέσιμες λειτουργίες / REST API μονοπάτια

Οι κυριότερες λειτουργίες που προσφέρει η υπηρεσία διαχείρισης πλατφορμών, μαζί με τα αντίστοιχα API endpoints, είναι οι εξής:

- **Ανάκτηση όλων των πλατφορμών**
  - Μέθοδος: GET
  - Μονοπάτι: /api/platforms
  - Περιγραφή: Επιστρέφει λίστα πλατφορμών (με δυνατότητα σελιδοποίησης).
  - Ενδεικτική υπογραφή: List<PlatformDTO> getAllPlatforms(int page, int size) (όπου **page** είναι ο δείκτης της σελίδας που ζητείται και **size** ο αριθμός εγγραφών ανά σελίδα, στο πλαίσιο της σελιδοποιημένης ανάκτησης δεδομένων).
- **Ανάκτηση πλατφόρμας βάσει αναγνωριστικού**
  - Μέθοδος: GET
  - Μονοπάτι: /api/platforms/{id}
  - Περιγραφή: Επιστρέφει τα πλήρη στοιχεία μιας συγκεκριμένης πλατφόρμας με βάση το αναγνωριστικό της id (παράμετρος μονοπατιού).
  - Υπογραφή:
    PlatformDTO getPlatformById(UUID id)
- **Δημιουργία νέας πλατφόρμας**
  - Μέθοδος: POST
  - Μονοπάτι: /api/platforms

Formatted: Greek

Commented [KK199]: Δεν χρειάζεται να επαναλαμβάνετε πράγματα. Εφοσον δεν χρειάζεται κάποια λεπτομέρεια, μπορειτε να προχωράτε στην επόμενη λειτουργια ή αν δεν υπάρχει άλλη, στην παρουσιαση των σχετικών endpoints.

Στην περιπτωση της καταταξης όμως, δεν υπάρχει εξειδικευμένη κλάση υλοποίησης που αξίζει να αναφερθει;
Στην προηγ. περιπτωση της ταξινόμησης, υπάρχει ειδική κλάση; Ή η κλάση είναι η PlatformService;

Commented [AK200R199]: Η κλάση ειναι η platformService

Commented [KK201]: Ας πειτε τι σημαίνουν αυτές οι 2 παράμετροι

- Περιγραφή: Δημιουργεί νέα εγγραφή πλατφόρμας όπου η περιγραφή της περιλαμβάνει ορισμένα χαρακτηριστικά και το JSON πεδίο featuresJson.
- Υπογραφή:
  PlatformDTO createPlatform(PlatformCreateRequestDTO request) όπου το **PlatformCreateRequestDTO** είναι DTO εισόδου που περιλαμβάνει τα απαιτούμενα στοιχεία της νέας πλατφόρμας — όνομα, περιγραφή, κατηγορία και JSON χαρακτηριστικών

- **Ενημέρωση υπάρχουσας πλατφόρμας**
  - Μέθοδος: PUT
  - Μονοπάτι: /api/platforms/{id}
  - Περιγραφή: Ενημερώνει τα στοιχεία μιας υπάρχουσας πλατφόρμας με βάση το αναγνωριστικό της id (παράμετρος μονοπατιού).
  - Υπογραφή:
    PlatformDTO updatePlatform(UUID id, PlatformUpdateRequestDTO request) όπου το **PlatformUpdateRequestDTO** είναι DTO εισόδου, το οποίο περιλαμβάνει προαιρετικά πεδία για την ενημέρωση επιλεγμένων χαρακτηριστικών μιας ήδη υπάρχουσας πλατφόρμας.
- **Διαγραφή πλατφόρμας**
  - Μέθοδος: DELETE
  - Μονοπάτι: /api/platforms/{id}
  - Περιγραφή: Διαγράφει μια πλατφόρμα από το σύστημα με βάση το αναγνωριστικό της id (παράμετρος μονοπατιού).
  - Υπογραφή:
    void deletePlatform(UUID id)
- **Full-text αναζήτηση και φιλτράρισμα πλατφορμών**
  - Μέθοδος: POST
  - Μονοπάτι: /api/platforms/search
  - Περιγραφή: Δέχεται φίλτρα (keyword, κατηγορίες κ.λπ.) και εκτελεί full-text search στη βάση δεδομένων, συνδυάζοντας φιλτράρισμα και ταξινόμηση αποτελεσμάτων. Η ταξινόμηση των αποτελεσμάτων δεν αποτελεί ξεχωριστή παράμετρο επερώτησης, αλλά υλοποιείται μέσω των

Formatted: Greek

Commented [KK202]: Ας πειτε τι είναι αυτό - μπορει να γινει εδώ ή στο προηγούμενο bullet.

Commented [KK203]: Εξηγήστε τι είναι αυτό.

Commented [KK204]: Η ταξινόμηση πως γινεται ακριβώς με βαση τις πληροφοριες που παρέχει ο χρήστης στην τρέχουσα μέθοδο; Υπάρχουν παράμετροι επερώτησης προς αυτό τον σκοπό διότι δεν τις βλέπω στην υπογραφή της μεθόδου;
Το ρωτώ διότι το αντικειμενο FilterRequestDTO όπως λέει και το όνομά του χρησιμοποιειται για το φιλτραρισμα οπότε θα περιλαμβάνει τιμές για χαρακτηριστικά ή/και λέξεις-κλειδιά.
Συνεπώς, θα περιλαμβάνει τελικά και το όνομα κάποιου κριτηριου για την ταξινόμηση ή όχι;

πεδίων sortBy και sortDirection του ίδιου του αντικειμένου FilterRequestDTO. (Το FilterRequestDTO είναι ένα Data Transfer Object που συγκεντρώνει σε **ένα ενιαίο σώμα αιτήματος** όλες τις παραμέτρους που χρειάζονται για αναζήτηση πλατφορμών)

- Υπογραφή:
  Page<PlatformDTO> searchPlatforms(FilterRequestDTO filterRequest)

- **Κατάταξη πλατφορμών βάσει κριτηρίων (ranking)**
  - Μέθοδος: POST
  - Μονοπάτι: /api/platforms/rank
  - Περιγραφή: Υπολογίζει ένα συνολικό score για κάθε πλατφόρμα με βάση τεχνικά και ποιοτικά κριτήρια, εφαρμόζοντας αλγόριθμο σταθμισμένης κατάταξης. *Τα κριτήρια αξιολόγησης καθώς και τα αντίστοιχα βάρη τους παρέχονται από τον χρήστη μέσω του αντικειμένου RankingRequestDTO. Το αντικείμενο αυτό περιλαμβάνει λίστα από CriterionRequestDTO, όπου κάθε στοιχείο καθορίζει: (α) το όνομα του κριτηρίου, (β) τις τιμές ή σύνολα τιμών που σχετίζονται με το κριτήριο (ανάλογα με τον τύπο του), και (γ) το βάρος (weight) που χρησιμοποιείται στον υπολογισμό του συνολικού score.Η αξιολόγηση κάθε πλατφόρμας πραγματοποιείται με διαφορετικό τρόπο ανάλογα με τον τύπο του κριτηρίου:- Για αριθμητικά κριτήρια (π.χ. μνήμη, μέγιστος χρόνος εκτέλεσης), εφαρμόζεται κανονικοποίηση min-max με βάση τις τιμές όλων των πλατφορμών. Για κατηγορικά κριτήρια (π.χ. runtime, supported languages), εφαρμόζεται σύγκριση συνόλων (set matching), όπου το score προκύπτει από το ποσοστό των απαιτήσεων του χρήστη που ικανοποιεί η πλατφόρμα.Με τον τρόπο αυτό, το σύστημα δεν βασίζεται σε απόσταση από κάποια προκαθορισμένη τιμή-στόχο, αλλά στον βαθμό κάλυψης των δυνατοτήτων της πλατφόρμας σε σχέση με τις απαιτήσεις του χρήστη, παρέχοντας μια πιο ρεαλιστική και προσαρμοστική κατάταξη.*
  - Υπογραφή:
    List<RankedPlatformDTO> rankPlatforms(RankingRequestDTO rankingRequest)

- **Auto-complete / προτάσεις πλατφορμών**
  - Μέθοδος: GET
  - Μονοπάτι: /api/platforms/autocomplete?query=...

Formatted: Greek

Commented [KK205]: Δεν εξηγήσατε όμως τι είναι αυτό

Commented [KK206]: Τα οποία παρέχονται μέσω του αντικειμένου RankingRequestDTO με βάση και τα σχετικά τους βάρη;

Formatted: Greek

- Περιγραφή: Επιστρέφει προτάσεις **ονόματος πλατφόρμας** με βάση πρόθεμα (prefix), ώστε ο χρήστης —ιδιαίτερα μη τεχνικός— να μπορεί να εντοπίζει γρήγορα γνωστές πλατφόρμες (π.χ. “goo…” → “Google Cloud Functions”) στο search box χωρίς να πληκτρολογεί πλήρως τον όρο. Η επιλογή περιορισμού στο name αποτελεί **σχεδιαστική επιλογή MVP** για απλότητα/ταχύτητα. Σε μελλοντική επέκταση, ο μηχανισμός μπορεί να υποστηρίξει και auto-complete για **ονόματα κριτηρίων** (π.χ. memory, timeout) ή/και **επιτρεπτές τιμές** τους (π.χ. runtimes/regions), όπως προτείνεται στη βιβλιογραφία UX για συστήματα πολυκριτηριακής επιλογής.

- Υπογραφή:
  List<PlatformSuggestionDTO> autocomplete(String query)

Η συνεργασία του PlatformService με το PlatformRepository ενσωματώνει τόσο κλασικές CRUD λειτουργίες όσο και πιο σύνθετη λογική αναζήτησης και κατάταξης, αποτελώντας την βασική υλοποίηση του Υποσυστήματος Διαχείρισης Πλατφορμών της εφαρμογής.

Η Υπηρεσία Κατάταξης (RankingService) αποτελεί λειτουργικό υποσύνολο του συστατικού διαχείρισης πλατφορμών και είναι υπεύθυνη για τον υπολογισμό ενός συνολικού βαθμού (ranking score) για κάθε serverless πλατφόρμα, βάσει σταθμισμένων κριτηρίων που παρέχονται δυναμικά από τον χρήστη.

Η διαδικασία κατάταξης υπολογίζει ένα νέο παράγωγο μέγεθος (ranking score), το οποίο καθορίζει την τελική σειρά των αποτελεσμάτων. Ο χρήστης καθορίζει ποια κριτήρια θα χρησιμοποιηθούν και ορίζει τα αντίστοιχα βάρη (weights). Τα χαρακτηριστικά κάθε κριτηρίου (όπως τύπος, κατεύθυνση και επιτρεπτές τιμές) είναι προκαθορισμένα από το σύστημα (criterion definitions) και χρησιμοποιούνται για τη σωστή ερμηνεία των δεδομένων κατά τον υπολογισμό του score.Η υλοποίηση της κατάταξης πραγματοποιείται μέσω της κλάσης PlatformRankingService, με τη συνδρομή βοηθητικών κλάσεων όπως οι PlatformRankingUtil και CriterionRequestDTO. Για κάθε πλατφόρμα υπολογίζεται επιμέρους βαθμολογία ανά κριτήριο και το τελικό σκορ προκύπτει ως άθροισμα των επιμέρους βαθμολογιών, σταθμισμένων με βάση τα βάρη που έχει ορίσει ο χρήστης.

Η αξιολόγηση εξαρτάται από τον τύπο του κριτηρίου:

- Αριθμητικά κριτήρια: εφαρμόζεται κανονικοποίηση min–max με βάση τις τιμές όλων των πλατφορμών που συμμετέχουν στην κατάταξη. Σε περίπτωση αρνητικής κατεύθυνσης (π.χ. “όσο μικρότερο, τόσο καλύτερο”), η κλίμακα αντιστρέφεται.

- Κατηγορικά κριτήρια (single ή multiple): εφαρμόζεται σύγκριση συνόλων (set matching), όπου το score προκύπτει από το ποσοστό των απαιτήσεων του χρήστη που ικανοποιεί η πλατφόρμα.Με αυτόν τον τρόπο, η υπηρεσία παρέχει μια πλήρως παραμετροποιήσιμη, αντικειμενική και επεκτάσιμη μέθοδο πολυκριτηριακής αξιολόγησης serverless πλατφορμών, εναρμονισμένη με τον αλγόριθμο που ορίζεται στη φάση σχεδιασμού και υλοποιεί με ακρίβεια τη θεωρητική διαδικασία ranking στο backend.

4.3.5 Ενότητα Διάταξης (Deployment Module)

Το **Deployment Module** είναι ενότητα του backend που εστιάζει στην διάταξη συναρτήσεων και αποτελείται από τα παρακάτω βασικά συστατικά:

- **DeploymentController** (/api/deployments): εκθέτει τα HTTP endpoints προς το frontend και δρομολογεί τα αιτήματα διάταξης προς την υπηρεσία.
- **DeploymentService**: περιέχει την επιχειρησιακή λογική της διάταξης και ενορχηστρώνει τα βήματα δημιουργίας configuration, εκτέλεσης deployment και καταγραφής αποτελεσμάτων.
- **Deployment** (entity) και **DeploymentRepository** (Spring Data repository): υλοποιούν την επιμονή (persistence) των αποτελεσμάτων διάταξης στη βάση (status, logs, endpoint URL, timestamps).
- **PlatformRepository**: χρησιμοποιείται για ανάκτηση στοιχείων της επιλεγμένης πλατφόρμας (provider/runtime defaults κ.λπ.).
- **UserCredentialsService** + **EncryptionUtil**: αναλαμβάνουν την ανάκτηση και ασφαλή χρήση των credentials του χρήστη για τον συγκεκριμένο πάροχο, ώστε η διάταξη να εκτελείται εκ μέρους του χρήστη χωρίς να εκτίθενται μυστικά στο UI.

Deployment Engine (Serverless Framework)

Στο πλαίσιο της αρχιτεκτονικής, το **Deployment Engine** αντιστοιχεί στο **Serverless Framework** και χρησιμοποιείται ως εξωτερικό “deployment component” το οποίο εκτελείται από το backend μέσω CLI (άρα αποτελεί συστατικό/υπηρεσία υποδομής). Η υπηρεσία του backend (Deployment Service) δεν υλοποιεί χαμηλού επιπέδου κλήσεις σε

Formatted: Greek

Commented [KK207]: Εδώ θα πρεπει να δώσετε μόνο λεπτομέρειες υλοποίησης για την ενότητα Deployment Module. Δεν χρειάζεται να επαναλάβετε διαδικαστικά πράγματα που αναφέρατε προηγουμένως για την λειτουργια διάταξης.

Αλλά μπορειτε να πειτε για τις κλάσεις που υλοποιούν την ενότητα και για το γεγονός πως χρησιμοποιείται συγκεκριμένο συστατικό διάταξης που έχει υλοποιηθεί από το Serverless framework.

Στην αρχιτεκτονική, το συστατικό αυτό το ονομάζετε ορθώς Deployment Engine!

SDKs των παρόχων, αλλά αξιοποιεί το Serverless Framework ως ενιαίο μηχανισμό διάταξης, παρέχοντας:

- **παραγωγή** του serverless.yml (provider-specific παράμετροι, runtime, region, plugins, events),
- **εκτέλεση** εντολών CLI μέσω ProcessBuilder (με timeout & συλλογή output),
- **ανάλυση** του αποτελέσματος διάταξης (π.χ. endpoint URL από το output),
- **καταγραφή** status & logs σε DeploymentRecord.

Persisted Deployment State (Deployment)

Κάθε διάταξη καταγράφεται στη βάση μέσω Deployment, το οποίο διατηρεί:

- την κατάσταση διάταξης (**PENDING / SUCCESS / FAILED**),
- το παραγόμενο endpoint (όπου υπάρχει),
- τα logs της εκτέλεσης και στοιχεία χρονοσήμανσης, ώστε να υποστηρίζεται ιστορικότητα, debugging και παρακολούθηση αποτελεσμάτων.

Με αυτόν τον τρόπο, το Deployment Module παραμένει **loosely coupled** ως προς τον μηχανισμό διάταξης: εφόσον απαιτηθεί, το Deployment Engine μπορεί να αντικατασταθεί (π.χ. με AWS SDK / GCP APIs / Azure tools) χωρίς να αλλάξει το υπόλοιπο σύστημα, καθώς οι υπόλοιπες ενότητες αλληλεπιδρούν μόνο με το DeploymentService και τα σχετικά repositories.

4.3.6 Frontend Υλοποίηση (ReactJS)

Το frontend του συστήματος υλοποιήθηκε με ReactJS (Vite) και αποτελεί το βασικό επίπεδο αλληλεπίδρασης του χρήστη με το σύστημα.

Η επίτευξη της ευχρηστίας και του responsive σχεδιασμού βασίστηκε σε δύο βασικές αρχές:

1. **Απλότητα χρήσης (Usability-Driven Design)**
    - Η διεπαφή υλοποιήθηκε με React και επαναχρησιμοποιήσιμα components ώστε να υπάρχει ομοιομορφία και καθαρότητα στη διάταξη.

Formatted: Greek

Formatted: Greek

Formatted: Greek

- Τα χειριστήρια (φορτώσεις αρχείων, φόρμες, πίνακες αποτελεσμάτων) οργανώθηκαν έτσι ώστε ο χρήστης να μπορεί να ολοκληρώνει κάθε ενέργεια με ελάχιστα κλικ / ελάχιστες εισόδους.
- Χρησιμοποιήθηκαν ξεκάθαρες ενδείξεις κατάστασης (loading indicators, success/error messages), ώστε κάθε βήμα της διαδικασίας διάταξης να γίνεται άμεσα κατανοητό.

2. **Responsive σχεδιασμός (Responsive UI)**
   - Χρησιμοποιήθηκε **CSS Flexbox** και **CSS Grid**, μαζί με media queries, ώστε τα στοιχεία της σελίδας να επαναδιατάσσονται δυναμικά ανάλογα με το μέγεθος οθόνης.
   - Επιπλέον αξιοποιήθηκε το σύστημα responsive breakpoints του **React + Vite**, επιτρέποντας στο UI να προσαρμόζει το layout χωρίς επανεκκίνηση.
   - Τα tables, cards και forms έχουν σχεδιαστεί έτσι ώστε να μπορούν να καταπίπτουν (collapse) σε στοίχιση μίας στήλης σε μικρές συσκευές, εξασφαλίζοντας άνετη ανάγνωση και αλληλεπίδραση.

Η υλοποίηση οργανώνεται σε τέσσερα κύρια επίπεδα:

1. **UI Components**

   Βασικά δομικά στοιχεία, όπως:

   - **Input** *(πεδίο εισαγωγής δεδομένων για φόρμες και φίλτρα)*,
   - **Button** *(κουμπί ενεργειών για αποστολή φορμών, εκτέλεση αναζητήσεων, έναρξη διάταξης κ.λπ.)*,
   - **Card** *(δομικό πλαίσιο παρουσίασης περιεχομένου, π.χ. πληροφοριών πλατφόρμας ή αποτελεσμάτων)*,
   - **PlatformCard** *(εξειδικευμένη κάρτα που εμφανίζει τα πλήρη χαρακτηριστικά μιας serverless πλατφόρμας)*,
   - **Autocomplete** *(πεδίο με προτάσεις δυναμικής συμπλήρωσης για πλατφόρμες κατά την αναζήτηση)* τα οποία επαναχρησιμοποιούνται σεόλη την εφαρμογή.
     Αυτά εξασφαλίζουν ομοιομορφία και ευκολία συντήρησης.

2. **Pages / Views**

Formatted: Greek

Οθόνες που υλοποιούν τις βασικές λειτουργίες του συστήματος:

- Προβολή αποτελεσμάτων και endpoint URL μετά τη διάταξη συνάρτησης HomePage.jsx — αρχική σελίδα / entry point
- LoginPage.jsx — σύνδεση χρήστη
- SignupPage.jsx— εγγραφή χρήστη
- ProfilePage.jsx — προφίλ χρήστη
- PlatformSearch.jsx — αναζήτηση & φιλτράρισμα πλατφορμών (σελίδα λίστας με φίλτρα)
- PlatformsList.jsx — εμφάνιση λίστας πλατφορμών
- PlatformDetail.jsx — λεπτομέρειες πλατφόρμας
- PlatformRanking.jsx — σελίδα κατάταξης (ranking UI)
- DeployFunctionForm.jsx — φόρμα/wizard για διάταξη (deployment)
- CredentialsForm.jsx — φόρμα καταχώρησης credentials (ανά πλατφόρμα)
- PlatformForm.jsx — δημιουργία/επεξεργασία πλατφόρμας (admin)
- AdminUsersPage.jsx — διαχείριση χρηστών (admin)

3. **State Management & Validation**

Η εφαρμογή χρησιμοποιεί react-hook-form (βιβλιοθήκη διαχείρισης φορμών που μειώνει τα re-renders και προσφέρει αποτελεσματική διαχείριση πεδίων, κατάστασης και υποβολής) και yup (βιβλιοθήκη ορισμού schemas για validation τύπων και τιμών) για:

- έλεγχο εγκυρότητας εισόδου (μέσω yup),
- άμεση ανατροφοδότηση (inline validation μέσω react-hook-form + yup),
- και άρα αποφυγή λανθασμένων αιτημάτων προς το backend.

Αυτό ενισχύει τη χρηστικότητα και μειώνει τα σφάλματα που φτάνουν στο API.

4. **Axios Client & REST API Integration**

Commented [KK208]: Ας εξηγήσετε τι είναι αυτά σε παρενθέσεις. Εναλλακτικά αυτό μπορει να γινεται σε επόμενη προταση ή με τη μορφή footnote.

Commented [KK209]: Αυτό γινεται από το yup ενώ το επόμενο από το react-hook-form;

Commented [KK210]: Μήπως ο συνδυασμός των 2 bullets οδηγει στο τριτο;

Αν όχι, θα πρεπει να πειτε ποια από τα 2 (react-hook-form & yup) χρησιμοποιειται για την υποστήριξη του τριτου bullet

Ένας axios client (με interceptors) αναλαμβάνει όλες τις HTTP κλήσεις προς το backend.

Ο client προσθέτει αυτόματα το JWT token σε κάθε αίτημα και διαχειρίζεται σφάλματα και retries καθώς και redirects σε περίπτωση μη εξουσιοδοτημένης πρόσβασης.

Το συνολικό αποτέλεσμα είναι ένα ευέλικτο και επεκτάσιμο frontend, το οποίο ενσωματώνει μοντέρνες πρακτικές UI/UX και παρέχει μια ομαλή εμπειρία χρήσης.

4.3.7 Βάση Δεδομένων (PostgreSQL)

Η εφαρμογή χρησιμοποιεί PostgreSQL ως το κύριο σύστημα διαχείρισης βάσης δεδομένων (ΣΔΒΔ), λόγω της υψηλής συμβατότητας με Spring Boot, της υποστήριξης JSON πεδίων, της αξιοπιστίας σε παραγωγικά περιβάλλοντα και των ενσωματωμένων δυνατοτήτων full-text search που αξιοποιούνται από το υποσύστημα αναζήτησης πλατφορμών.

Η επιλογή της PostgreSQL αιτιολογείται από τα εξής:

- Υποστήριξη JSON & JSONB, επιτρέποντας την ευέλικτη αποθήκευση δομημένων χαρακτηριστικών πλατφορμών (featuresJson).
- Native full-text search (tsvector, tsquery, ts_rank), το οποίο αξιοποιήθηκε για την υλοποίηση της αναζήτησης.
- Βελτιώσεις απόδοσης σε σύνθετες επερωτήσεις (queries), χρήσιμες για filtering, sorting και ranking.
- Συμβατότητα με Spring Data JPA & Hibernate, επιτρέποντας γρήγορη ανάπτυξη.
- Αξιοπιστία και επεκτασιμότητα, κατάλληλες για μελλοντική υποστήριξη περισσότερων λειτουργιών.

Η βάση οργανώνεται σε πίνακες όπως:

- *platforms* (κύρια δεδομένα πλατφόρμας + JSON characteristics),
- *users*,
- *user_credentials*,
- *deployment_records*,

Formatted: Greek

Commented [KK211]: Απλώς να πω πως τα retries θα πρέπει να γινονται εφοσον υπάρχει κάποιο ζήτημα με το συστημα. Ενώ τα redirects αφορούν λογικά περιπτώσεις μη εξουσιοδοτημένης προσβασης.

Commented [AK212R211]: Συμφώνω ήθελα να γράψω κάτι για idempotency αλλα δεν μου βγηκε ας το αφησουμε ετσι.

Formatted: Greek

- καθώς και βοηθητικούς πίνακες για ιστορικό και logging διάταξης.

Τα δεδομένα αυτά διαχειρίζονται αποκλειστικά από το backend μέσω JPA repositories, εξασφαλίζοντας εγγυημένη ακεραιότητα και έλεγχο πρόσβασης.

4.3.8 Περίληψη Υλοποίησης

Το σύστημα συνδυάζει **Spring Boot**, **React** και **PostgreSQL**, παρέχοντας μια ολοκληρωμένη λύση εφαρμογής ιστού (web application) για το φιλτράρισμα και την αξιολόγηση serverless πλατφορμών καθώς και τη διάταξη εντός των πλατφορμών συναρτήσεων του χρήστη.

Η αρχιτεκτονική της λύσης αυτής είναι επεκτάσιμη, ασφαλής και πλήρως αυτοματοποιημένη, ενώ μπορεί να προσαρμοστεί για μελλοντική υποστήριξη πρόσθετων serverless πλατφορμών και εργαλείων παρακολούθησης.

Formatted: Greek

Commented [KK213]: Οκ, σε επίπεδο πληροφοριακό (class diagram), υπάρχει κάποια οντοτητα/κλάση που θα έπρεπε κι αυτή να μοντελοποιηθεί;

Commented [AK214R213]: Είναι η deploymentRecord

Commented [KK215]: Θα μπορούσατε να είχατε ξεχωριστή υπο-ενότητα της 4.3 για την βάση δεδομένων και την δικαιολόγηση της επιλογής της. Με την λογική πως θα πρέπει να αναλυθούν όλα τα συστατικά του υλοποιημένου συστήματος.

Formatted: Greek

5. Εγκατάσταση & Επίδειξη Συστήματος

Το παρόν κεφάλαιο οργανώνεται σε δύο βασικές ενότητες. Στην Ενότητα 5.1 παρουσιάζεται η διαδικασία εγκατάστασης του συστήματος, συμπεριλαμβανομένων των προαπαιτούμενων εργαλείων, της δημιουργίας και ρύθμισης της βάσης δεδομένων, καθώς και των βημάτων εκτέλεσης του backend και του frontend. Στην Ενότητα 5.2 ακολουθεί η επίδειξη της λειτουργικότητας του συστήματος μέσω αντιπροσωπευτικών σεναρίων χρήσης, τα οποία καλύπτουν τις βασικές λειτουργίες της εφαρμογής για απλούς χρήστες και διαχειριστές. Με τον τρόπο αυτό, το κεφάλαιο παρουσιάζει αφενός τον τρόπο εγκατάστασης του συστήματος και αφετέρου την πρακτική αξιοποίησή του μέσω συγκεκριμένων ροών χρήσης.

5.1 Εγκατάσταση

Το σύστημα *Serverless Platform Selector* που αναπτύξαμε μπορεί να εγκατασταθεί τοπικά ή σε περιβάλλον cloud. Προς αυτό τον σκοπό, η παρούσα ενότητα περιγράφει τα προαπαιτούμενα εργαλεία, τα βήματα δημιουργίας της βάσης δεδομένων και τη διαδικασία εκτέλεσης του backend και του frontend σε τοπικό σύστημα.

5.1.1 Προαπαιτούμενα

Για την εγκατάσταση και εκτέλεση του συστήματος απαιτούνται τα ακόλουθα εργαλεία:

- **Git** (για κλωνοποίηση των αποθετηρίων)[35]
- **Java 21 ή νεότερη**[36]
- **Maven 3.9+**[37]
- **Node.js 18+ και npm**[38]
- **PostgreSQL 14+**[39]
- **Προαιρετικά IDE:** IntelliJ IDEA ή Eclipse *(το IDE δεν είναι απαραίτητο, όλα μπορούν να γίνουν μέσω γραμμής εντολών)*

[35] Git - Installing Git
[36] Java Downloads | Oracle
[37] https://maven.apache.org/install.html
[38] Node.js — Download Node.js®
[39] PostgreSQL: Downloads

Formatted: Greek

Commented [KK216]: Ας εξηγήσετε με τη μορφή μιας μικρής παραγράφου τη δομή του κεφαλαιου αυτού.

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Normal, Justified

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: (Default) Times New Roman, 12 pt

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Font: (Default) Times New Roman, 12 pt, Greek

Formatted: Greek

Commented [KK217]: Οι συνδέσμοι στα footnotes θα πρεπει να είναι κανονικά URLs (όπως γινόταν και σε άλλα footnotes).

Commented [KK218]: Θα μπορούσατε να είχατε συνδέσμους προς οδηγίες εγκαταστασης των εργαλειων αυτών με την μορφή footnotes

5.1.2 Δημιουργία Βάσης Δεδομένων

Η βάση δεδομένων πρέπει να δημιουργηθεί πριν την εκκίνηση του backend.

Βήμα 1 — Σύνδεση στο PostgreSQL

Ο χρήστης μπορεί να συνδεθεί είτε μέσω:

- **psql CLI** (είναι διαθέσιμο μετά την εγκατάσταση από την γραμμή εντολών),
- είτε μέσω γραφικού περιβάλλοντος, όπως **pgAdmin**[40].

Βήμα 2 — Δημιουργία της βάσης

SQL εντολή:

```
CREATE DATABASE serverless_platform_db;
```

Οδηγία χρήσης

- *pgAdmin:* Create → Database → Name: serverless_platform_db → Save
- *psql:* `psql -U postgres` και εκτέλεση της παραπάνω εντολής.

5.1.3 Εγκατάσταση Backend

Βήμα 1 — Κλωνοποίηση αποθετηρίου

```
git clone https://github.com/KamperopoulosA/serverless_hub_backend_thesis.git
```

Βήμα 2 — Μετάβαση στον φάκελο του backend

```
cd serverless_hub_backend_thesis/project
```

Βήμα 3 — Ρύθμιση του αρχείου διασύνδεσης με τη βάση

Το αρχείο βρίσκεται στη διαδρομή:

project/src/main/resources/application.properties

Απαιτούμενες τιμές:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/serverless_platform_db
```

[40] Download

Formatted: Greek

Formatted: Greek

Commented [KK219]: Ας το κοιταξετε ...

Commented [KK220]: Ας δώσετε οδηγίες εγκατάστασης μέσω σχετικού συνδέσμου (πχ. με μορφη footnote)

Commented [KK221R220]: Δείτε προηγούμενο σχολιο

Commented [AK222R220]: Έχει μπει σχετικό footnote

Commented [KK223]: Ας εμφανίζετε τις εντολές με διαφορετική μορφοποίηση (πχ. Font) για να ξεχωριζουν από το κανονικό κείμενο.

Formatted: Font: (Default) Courier New

Formatted: Greek

Commented [KK224]: Ξεχνάτε συνεχώς να εφαρμόζετε στυλ (μορφοποίησης) για την κάθε κεφαλίδα.

Commented [KK225]: Ας πειτε πως οι τιμές αυτές εντός του αρχειου υπάρχουν εκ των προτέρων ως το προκαθορισμένο περιεχόμενό του. Οπότε, θα πρέπει να ελεγχθούν από τον χρηστη και να τροποποιηθούν εφοσον αυτό είναι απαραιτητο.
Μετά «κολλάτε» και αυτό για την βάση δεδομένων .... που είναι ο συχνότερος λόγος αλλαγής τιμών ...

spring.datasource.username=postgres

spring.datasource.password=admin

spring.jpa.hibernate.ddl-auto=update

(Οι τιμές του αρχείου αυτού είναι αρχικά προκαθορισμένες. Οι τιμές username/password εξαρτώνται από το λειτουργικό σύστημα, την μέθοδο εγκατάστασης και τις επιλογές που παρέχονται από τον χρήστη κατά την εγκατάσταση. Συνεπώς, συνίσταται να αλλάζουν για να ταιριάζουν με αυτές που έχουν τεθεί και να μην είναι προκαθορισμένες (αυτό ισχύει σε ορισμένες περιπτώσεις). Εφόσον η βάση είναι εγκατεστημένη σε άλλο μηχάνημα, το localhost στο spring.datasource.url θα πρέπει να αντικατασταθεί με το IP ή host name του μηχανήματος.)

Βήμα 4 — Εκτέλεση του backend

mvn spring-boot:run

Με επιτυχή εκτέλεση, το backend είναι διαθέσιμο στο: http://localhost:8080/api/

5.1.4 Εγκατάσταση Frontend

Βήμα 1 — Κλωνοποίηση αποθετηρίου

git clone https://github.com/KamperopoulosA/serverless_hub_frontend_thesis.git

Βήμα 2 — Μετάβαση στον φάκελο

cd serverless_hub_frontend_thesis

Βήμα 3 — Εγκατάσταση εξαρτήσεων

npm install

Βήμα 4 — Εκτέλεση του frontend

npm run dev

Το frontend θα είναι διαθέσιμο στη διεύθυνση: http://localhost:5173

5.2 Επίδειξη & Σενάρια Χρήσης

Το κεφάλαιο αυτό παρουσιάζει τη λειτουργικότητα του συστήματος *Serverless Platform Selector* μέσα από αντιπροσωπευτικά σενάρια χρήσης. Στόχος είναι να καταδειχθεί πώς

Formatted: Greek

Field Code Changed

Formatted: Greek

Field Code Changed

Formatted: Greek

Commented [KK226]: Τα σενάρια χρησης φαινονται απλοϊκά, στοχεύοντας στην κάλυψη μεμονωμένων περιπτώσεων χρησης. Θα πρότεινα αυτά να αλλάξουν ως εξής:
Σενάριο 1: Εγγραφή & Σύνδεση Χρήστη (σημ. αν ένας εγγεγραμμένος χρηστης είναι στην αρχή απενεργοποιημένος, θα πρεπει να δειξετε και τι κάνει ο διαχειριστής για να τον ενεργοποιήσει. Σε μια τέτοια περιπτωση, ορισμένα πράγματα από το σενάριο 5 μπορούν να μεταφερθούν εδώ)
Σενάριο 2: Δημιουργία & Ενημέρωση Πλατφόρμας
Σενάριο 3: Αναζητηση/Φιλτραρισμα & Κατάταξη Πλατφορμών
Σενάριο 4: Διάταξη συνάρτησης
Σενάριο 5: Διαχείριση χρηστών (ενημέρωση στοιχειων χρηστη, ανανέωση κωδικού, αναζήτηση χρηστη, διαγραφή χρηστη - ότι υποστηρίζεται από το σύστημά σας)

Formatted: Greek

ένας χρήστης (απλός ή διαχειριστής) αλληλεπιδρά με το σύστημα, ποιες οθόνες συναντά και πώς εξελίσσεται κάθε ροή λειτουργίας.

Κάθε σενάριο περιλαμβάνει:

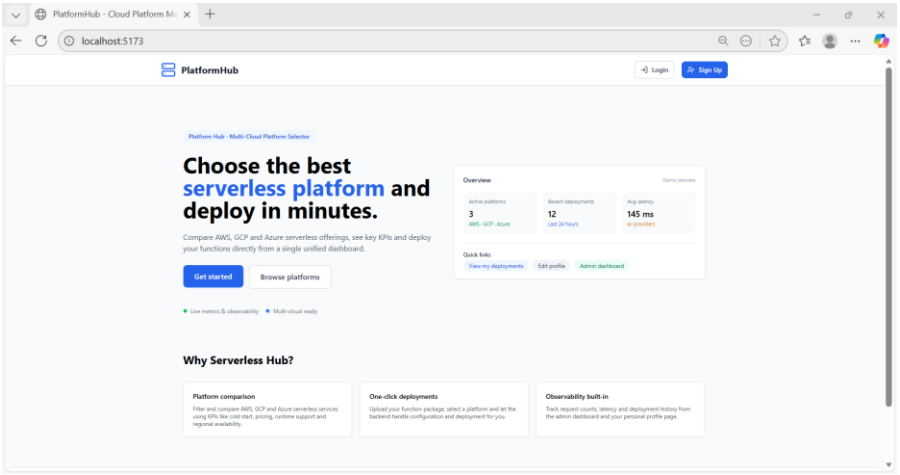
1. **Αφετηρία (από ποια σελίδα ξεκινά ο χρήστης)**
2. **Βήμα προς βήμα αλληλεπίδραση**
3. **Αντίστοιχα στιγμιότυπα πριν και μετά τη δράση**
4. **Λεξάντες εικόνων**

5.2.1 Σενάριο 1 — Εγγραφή & Σύνδεση Χρήστη

Commented [KK227]: Ας μπει κάθε σενάριο στη δική του υπο-ενότητα. Πχ. 5.2.1. Σενάριο 1 - ...

Αφετηρία

Ο επισκέπτης βρίσκεται αρχικά στην αρχική (landing) σελίδα της εφαρμογής και επιλέγει την επιλογή «**Login**», ώστε να μεταβεί στη σελίδα αυθεντικοποίησης.

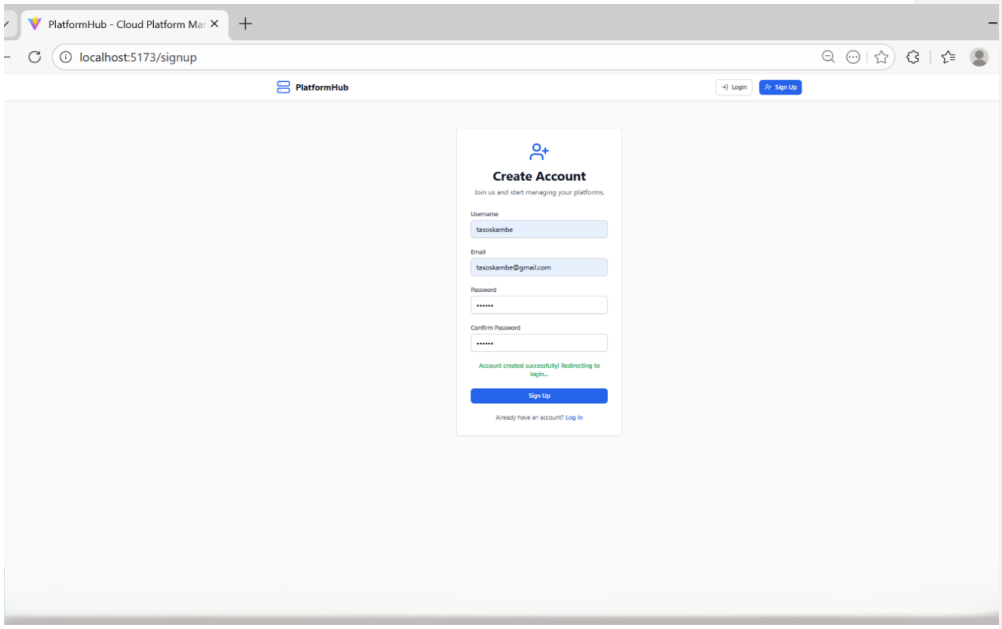


Εικόνα 1: Αρχική σελίδα(Landing Page)

Ενέργειες Χρήστη

Formatted: Greek

1. Εφόσον ο χρήστης αρχικά δεν υπάρχει, θα πρέπει να γίνει η εγγραφή του. Οπότε, θα επιλέγεται το δεύτερο κουμπί Sign up από το πάνω μενού ώστε να εμφανιστεί η φόρμα εγγραφής προς συμπλήρωση.



Εικόνα ~~22~~: Φόρμα Εγγραφής (Sign Up Form)

2. Συμπληρώνει τα απαιτούμενα πεδία (όνομα, email και κωδικό) και υποβάλλει τη φόρμα. Το σύστημα δημιουργεί τον λογαριασμό χρήστη εφόσον τα πεδία είναι έγκυρα και δεν υπάρχει άλλος χρήστης με το ίδιο email. Μετά την επιτυχή δημιουργία του λογαριασμού, το σύστημα εμφανίζει σχετικό μήνυμα επιβεβαίωσης στο γραφικό περιβάλλον χρήστη.(Δείτε εικόνα 2)

3. Ο χρήστης οδηγείται στη φόρμα σύνδεσης μετά την επιτυχή εγγραφή, εισάγει τα στοιχεία του και επιλέγει «**Login**». Σε περίπτωση επιτυχούς αυθεντικοποίησης, το

Commented [KK228]: Ο κωδικός θα έπρεπε να παρέχεται εις διπλούν!

Commented [KK229]: Εφόσον όλα πάνε καλά και δημιουργειται ο λογαριασμός, δεν ενημερώνεται ο χρηστης με κάποιο τροπο από το UI. Αυτό ούτε το λέτε στο κείμενο ούτε το δειχνετε με κάποιο screenshot.

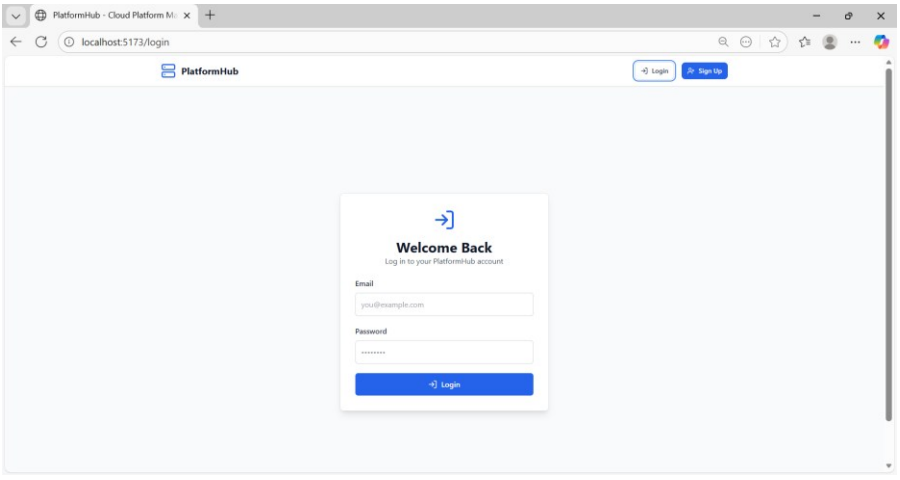
Commented [KK230]: Μήπως σε αυτή την σελίδα οδηγειται όταν εγγραφει επιτυχώς στο συστημα;

Commented [AK231R230]: Ναι σωστα

σύστημα εκδίδει JWT και ολοκληρώνει τη σύνδεση **(Εικόνα 3)**. Σημειώνεται πως μετά την εγγραφή ο χρήστης είναι ήδη ενεργοποιημένος.

4. Εφόσον ο χρήστης ξεχάσει τον κωδικό του και προχωρήσει σε 3 διαδοχικές αποτυχημένες προσπάθειες αυθεντικοποίησής του στο σύστημα, ο λογαριασμός του απενεργοποιείται (Δείτε Εικόνα 4). Σε αυτή την περίπτωση, μπορεί να επικοινωνήσει με τον διαχειριστή ώστε να του ενεργοποιήσει τον λογαριασμό. Τα βήματα που πρέπει να πραγματοποιηθούν από τον διαχειριστή για αυτή την ενεργοποίηση είναι τα ακόλουθα:

   5.1 Ο διαχειριστής συνδέεται στο σύστημα (ADMIN) (δείτε Εικόνα 4).
   5.2 Από το μενού επιλέγει **Admin DashBoard** (δείτε Εικόνα 5)
   5.3 Εντοπίζει τον νέο χρήστη και επιλέγει **Activate/Enable** (ή αντίστοιχη ενέργεια). (δείτε Εικόνα 6)
   5.4 Το σύστημα ενημερώνει την κατάσταση του λογαριασμού και εμφανίζει επιβεβαίωση. (δείτε Εικόνα 7)



Εικόνα 33: Φόρμα Σύνδεσης (Login Form)

Formatted: Greek

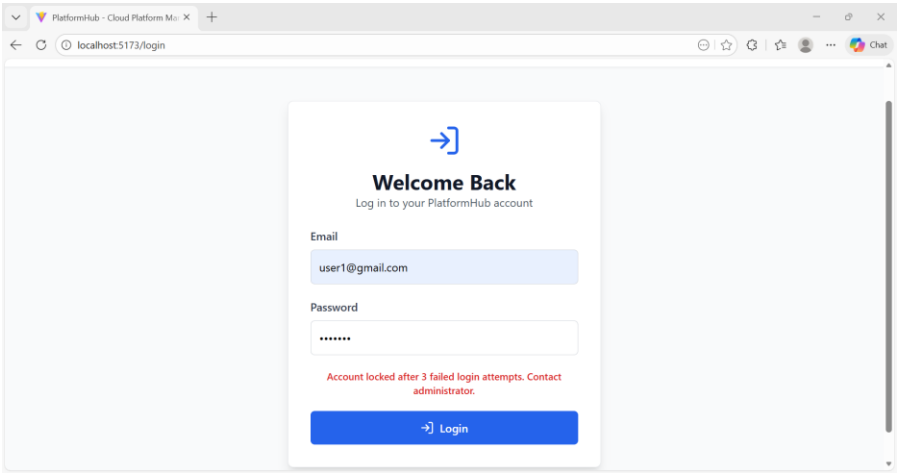
Commented [KK232]: Προχώρησα σε αυτό το σενάριο. Δειτε προηγ. σχόλιο αν τελικά το σενάριο αυτό δεν ισχυει.

Commented [KK233]: Εδώ λείπουν οι αντιστοιχες εικόνες

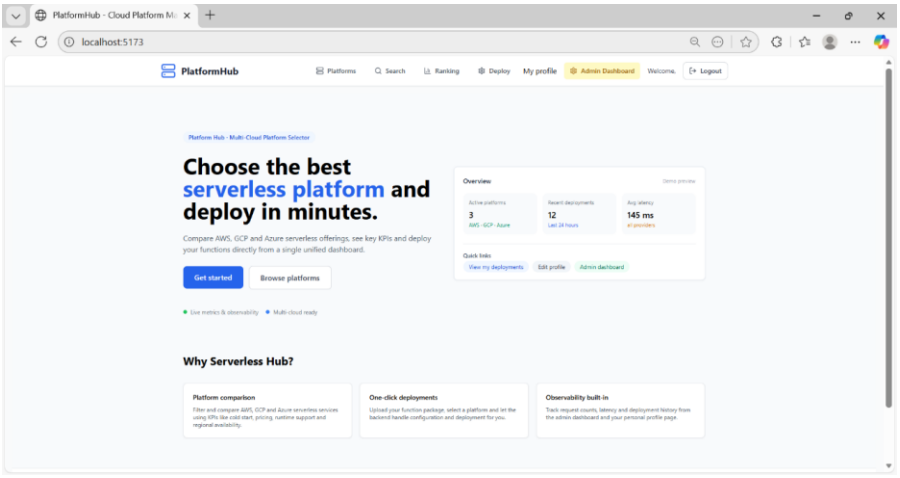
Commented [KK234]: Οι εικόνες 3, 4, 6 & 7 δείχνουν έναν διαφορετικό χρηστη από τον αρχικό που έκανε την εγγραφή. Επειδή όμως έχουμε να κάνουμε με ένα ενιαίο σενάριο, καλό θα ήταν να εμφανίζονταν τα στοιχεια για τον ιδιο χρηστη. Οπότε, προτείνω την ενημέρωση των εικόνων αυτών.

Commented [AK235R234]: Στο συγκεκριμένο σενάριο χρησιμοποιείται ο ίδιος χρήστης σε όλα τα βήματα (εγγραφή, αποτυχημένες προσπάθειες σύνδεσης και επανενεργοποίηση από τον διαχειριστή),

Formatted: Greek



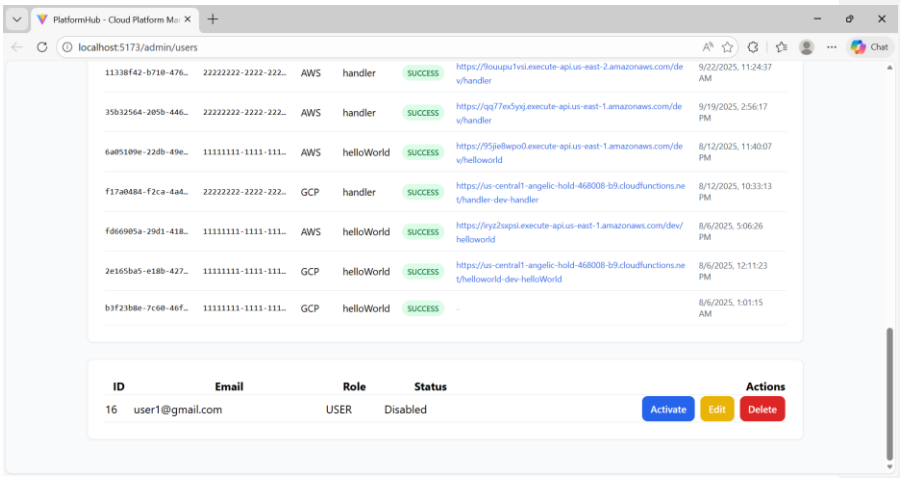
Εικόνα 44: Κλείδωμα λογαριασμού μετά από 3 αποτυχημένες προσπάθειες σύνδεσης



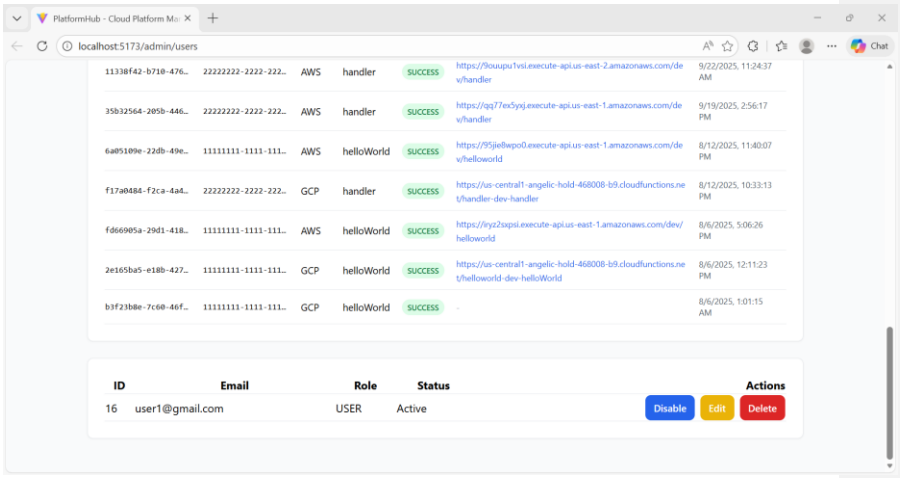
Εικόνα 5: Επιλογή Admin Dashboard

Commented [KK236]: Δεν είναι ωραία τα failures που φαινονται στη τρεχουσα σελίδα. Θα προτιμούσα να ειχαν αφαιρεθεί.

Formatted: Greek



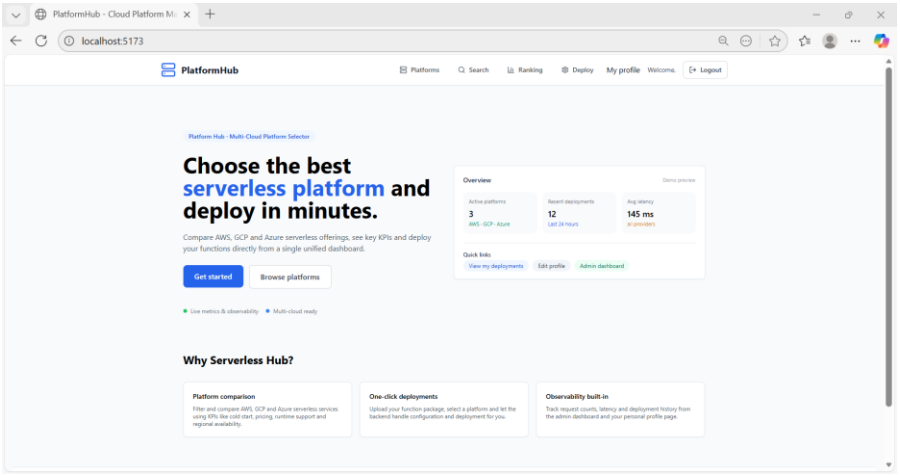
Εικόνα 66: Εμφάνιση νέου χρήστη απενεργοποιημένου στη λίστα χρηστών (Admin View)



Εικόνα 77: Εμφάνιση νέου, πλέον ενεργοποιημένου χρήστη στη λίστα χρηστών (Admin View)

Αναμενόμενο Αποτέλεσμα

Μετά την επιτυχή σύνδεση, ο χρήστης μεταφέρεται στην **αρχική σελίδα.** Όπως μπορεί να φανεί (δείτε Εικόνα 8), στην μπάρα πλοήγησης στο πάνω μέρος της ιστοσελίδας πλέον φαίνεται το ολοκληρωμένο μενού της εφαρμογής καθώς και η επιλογή της αποσύνδεσης για τον συνδεδεμένο χρήστη.



Εικόνα 88: Αρχική Σελίδα (Landing Page) μετά από επιτυχή αυθεντικοποίηση

5.2.2 Σενάριο 2 — Δημιουργία & Ενημέρωση Πλατφόρμας (Διαχειριστής)

Α. Δημιουργία νέας πλατφόρμας

Αφετηρία

Ο διαχειριστής έχει **αυθεντικοποιηθεί επιτυχώς** στο σύστημα και έχει μεταβεί από το βασικό μενού/σελίδα διαχείρισης στην ενότητα **Platforms** (σελίδα λίστας πλατφορμών) (δείτε Εικόνα 9).

Ενέργειες

Formatted: Greek

Commented [KK237]: Θα έπρεπε να ειχαμε Α. Δημιουργια Πλατφορμας & Β. Ενημέρωση Πλατφορμας. Μόνο το Β. υπάρχει εντός της 5.2.2

Commented [KK238]: Να διορθωθούν και όλες οι παραπομπές στις εικόνες στο κεφάλαιο αυτό. Ας χρησιμοποιειτε cross-references ώστε η ενημέρωση του αριθμού της παραπομπής να γινεται αυτοματα.

1. Επιλέγει “New Platform” για δημιουργία νέας πλατφόρμας.
2. Συμπληρώνει τα στοιχεία της πλατφόρμας, τα οποία αντιστοιχούν στο μοντέλο δεδομένων του backend:
    - **name** (όνομα πλατφόρμας),
    - **description** (περιγραφή),
    - **category** (κατηγορία/τύπος πλατφόρμας, ως ελεύθερο κείμενο),
    - **featuresJson** (τεχνικά χαρακτηριστικά σε μορφή JSON, π.χ. runtime, limits, regions, triggers/events κ.λπ.).
3. Υποβάλλει τη φόρμα (δείτε Εικόνα 10).
4. Εφόσον η καταχώριση είναι επιτυχής, η νέα πλατφόρμα προστίθεται στη λίστα πλατφορμών (δείτε Εικόνα 11 όπου έχει προστεθεί η νέα πλατφόρμα στο κάτω μέρος της λίστας).
5. Από τη λίστα πλατφορμών ο διαχειριστής μπορεί επιπλέον να:
    - επιλέξει **Edit** σε μία υπάρχουσα πλατφόρμα, να τροποποιήσει πεδία (π.χ. description / category / featuresJson) και να αποθηκεύσει τις αλλαγές,
    - επιλέξει **Delete** για οριστική διαγραφή μίας πλατφόρμας (και αφαίρεσή της από τη λίστα).

    Αυτό είναι δυνατό μέσω κουμπιών διαχείρισης που φαίνονται οπτικά εντός του πλαισίου της εκάστοτε πλατφόρμας που εμφανίζεται στην λίστα.

Αναμενόμενο Αποτέλεσμα

Η νέα πλατφόρμα εμφανίζεται στη λίστα, διαθέσιμη σε χρήστες προς αναζήτηση, αξιολόγηση και κατάταξη.

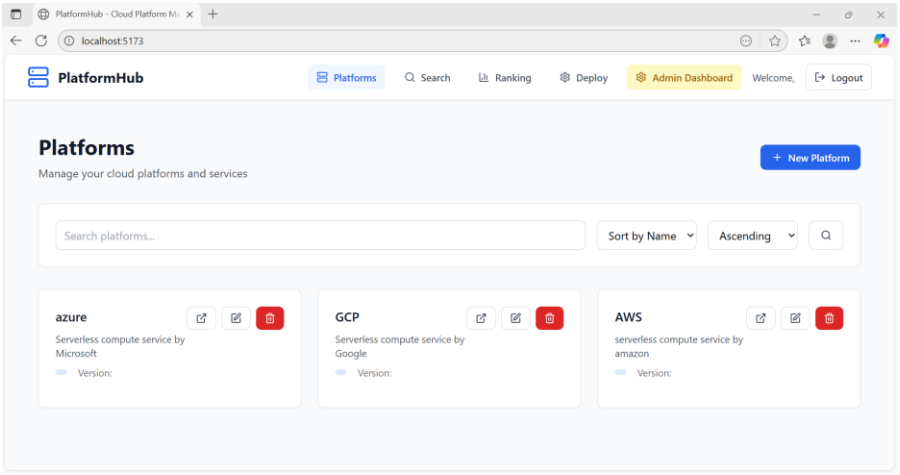
Formatted: Greek

Commented [KK239]: Προσέξτε διότι το font άλλαξε. Θα πρέπει να είναι σταθερό κατά μήκος όλης της αναφοράς.

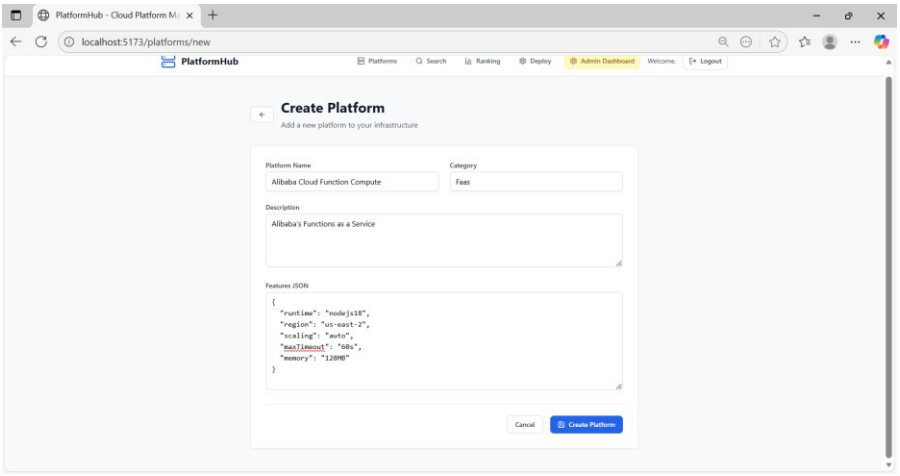
Commented [KK240]: Το σενάριο υπόσχεται και την ανανέωση της πλατφορμας. Συνεπώς, θα πρέπει να δειξετε και τα σχετικά βηματα για την ανανέωση ως επόμενο μέρος του σεναριου.

Commented [KK241R240]: Θα πρεπει να έχετε αναμενόμενο αποτέλεσμα ανά υπο-σενάριο (Α. & Β.) για το τρεχων σενάριο χρησης

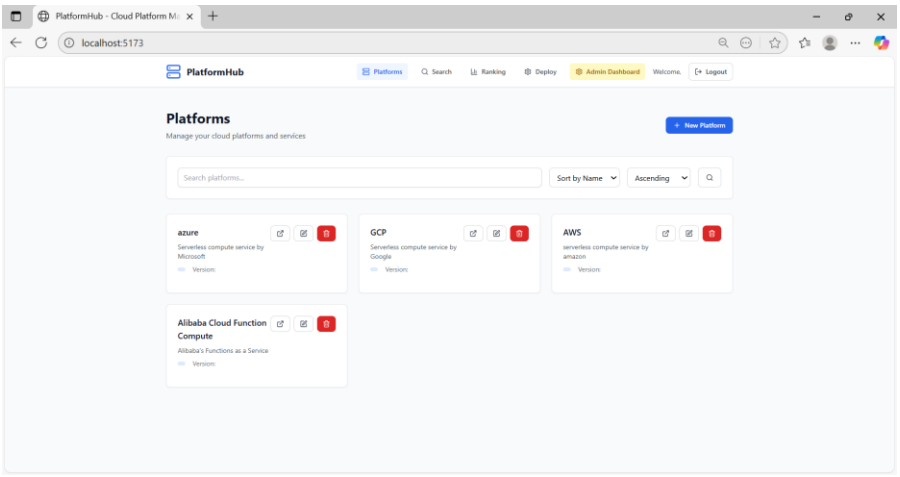
Formatted: Greek



Εικόνα 99: Λίστα Πλατφορμών(Admin View)



Εικόνα 10: Φόρμα Νέας Πλατφόρμας



Εικόνα ~~1111~~: Λίστα Πλατφορμών(Admin View)

Β. Ενημέρωση υπάρχουσας πλατφόρμας

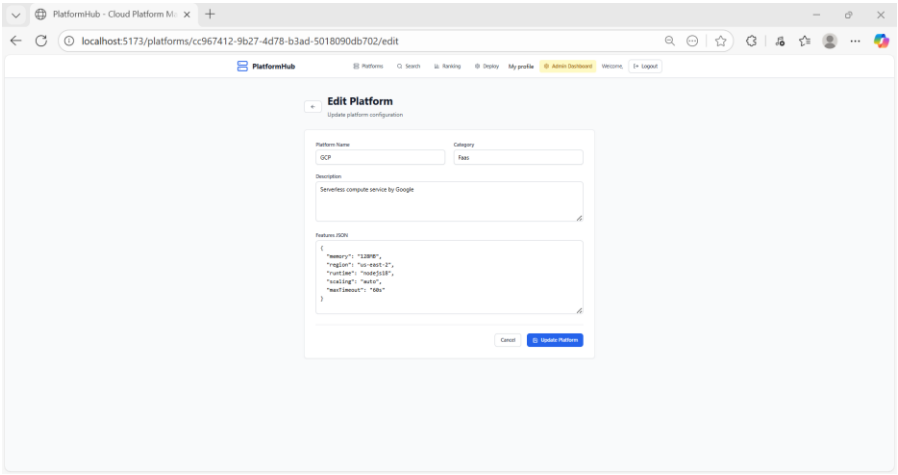
Ενέργειες

1. Ο διαχειριστής βρίσκεται στη σελίδα λίστας πλατφορμών Platforms. *(Εικόνα 11)*
2. Εντοπίζει την πλατφόρμα που θέλει να τροποποιήσει και επιλέγει Edit. *(Εικόνα 12)*
3. Εμφανίζεται η φόρμα επεξεργασίας με προσυμπληρωμένα τα υπάρχοντα δεδομένα της πλατφόρμας.
4. Τροποποιεί ένα ή περισσότερα πεδία (ενδεικτικά: description, category, featuresJson). *(Εικόνα 13 — μετά την αλλαγή και πριν την υποβολή)*
5. Επιλέγει Save/Update για αποθήκευση των αλλαγών.
6. Το UI αποστέλλει αίτημα προς το backend μέσω REST API για ενημέρωση της πλατφόρμας.
7. Εφόσον η ενημέρωση ολοκληρωθεί επιτυχώς, το σύστημα εμφανίζει ενημέρωση επιτυχίας και η λίστα ανανεώνεται με τα νέα στοιχεία. *(Εικόνα 14)*

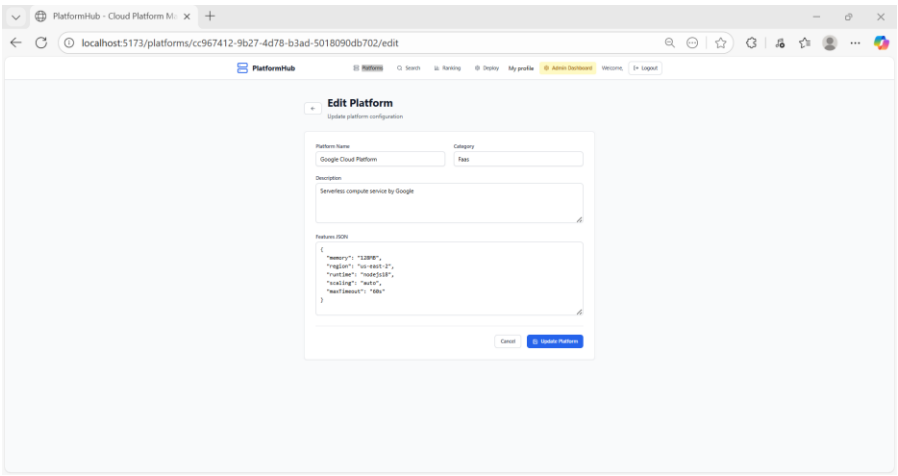
Formatted: Greek

Commented [KK242]: Κι εδώ προσέξτε την αριθμηση των παραπομπών και των εικόνων (στις λεζάντες)

Formatted: Greek

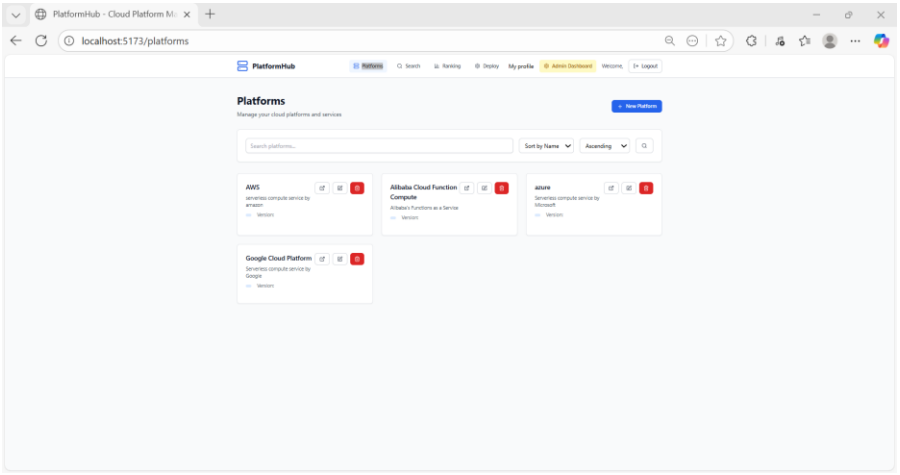


Εικόνα 1212: Φόρμα επεξεργασίας πλατφόρμας (Edit Platform Form).



Εικόνα 1313: Φόρμα επεξεργασίας πλατφόρμας, μετά την αλλαγή (Edit Platform Form).

Formatted: Greek



Εικόνα 1414: Ενημερωμένη λίστα πλατφορμών με τα νέα στοιχεία της τροποποιημένης πλατφόρμας.

Αναμενόμενο Αποτέλεσμα

- Η επιλεγμένη πλατφόρμα ενημερώνεται στη βάση δεδομένων με τα νέα στοιχεία.
- Η λίστα Platforms εμφανίζει τα τροποποιημένα δεδομένα (π.χ. νέα περιγραφή/κατηγορία/features).
- Το σύστημα εμφανίζει ένδειξη επιτυχούς ενημέρωσης (οπτική επιβεβαίωση μέσω refreshed list).

Προϋποθέσεις & Ρόλοι

Στο συγκεκριμένο σενάριο εμπλέκεται **διαχειριστής (ADMIN)**, καθώς μόνο ο ρόλος αυτός επιτρέπεται να δημιουργεί/ενημερώνει/διαγράφει πλατφόρμες. Τα σενάρια που αφορούν αναζήτηση, κατάταξη και διάταξη συναρτήσεων μπορούν να εκτελούνται και από **απλό χρήστη (USER)** που έχει προηγουμένως εγγραφεί και συνδεθεί επιτυχώς στο σύστημα/εφαρμογή μας.

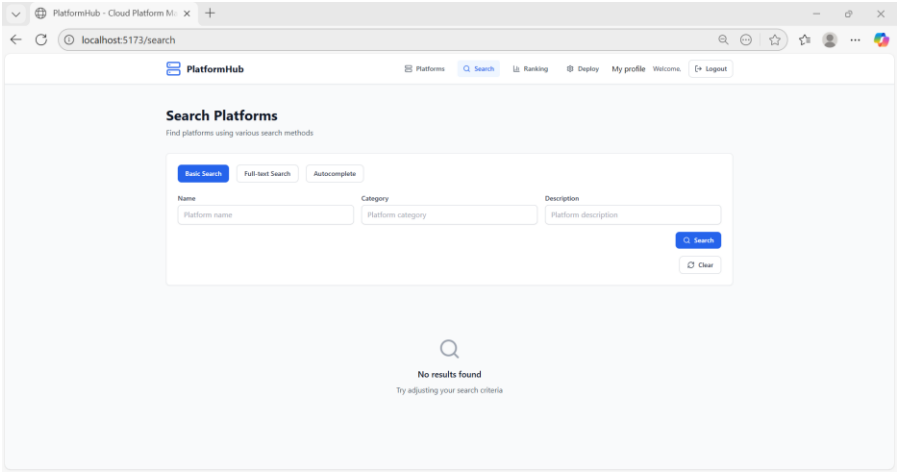
Formatted: Greek

5.2.3 Σενάριο 3 — Αναζήτηση Πλατφορμών (User)

Αφετηρία

Ο χρήστης έχει ήδη συνδεθεί επιτυχώς στο σύστημα και από το επάνω μενού πλοήγησης επιλέγει Search, ώστε να μεταβεί στη σελίδα αναζήτησης πλατφορμών.

Commented [KK243]: Θα πρέπει να αποφεύγετε την συχνή χρηση bold ακόμα και εκει που δεν χρειάζεται ...



Εικόνα ~~15~~15: Σελίδα Αναζήτησης Πλατφορμών (Search Platforms Page)

Ενέργειες Χρήστη

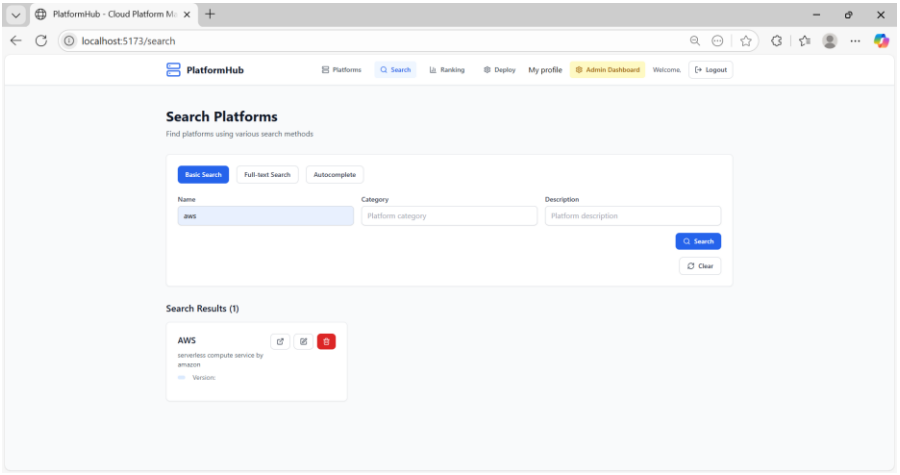
Η σελίδα Search Platforms προσφέρει τρεις εναλλακτικούς τρόπους αναζήτησης μέσω καρτελών (tabs). Ο χρήστης επιλέγει την επιθυμητή μέθοδο και εκτελεί αναζήτηση ως εξής:

(A) Basic Search (αναζήτηση με φίλτρα πεδίων πάνω στα μεταδεδομένα των πλατφορμών)

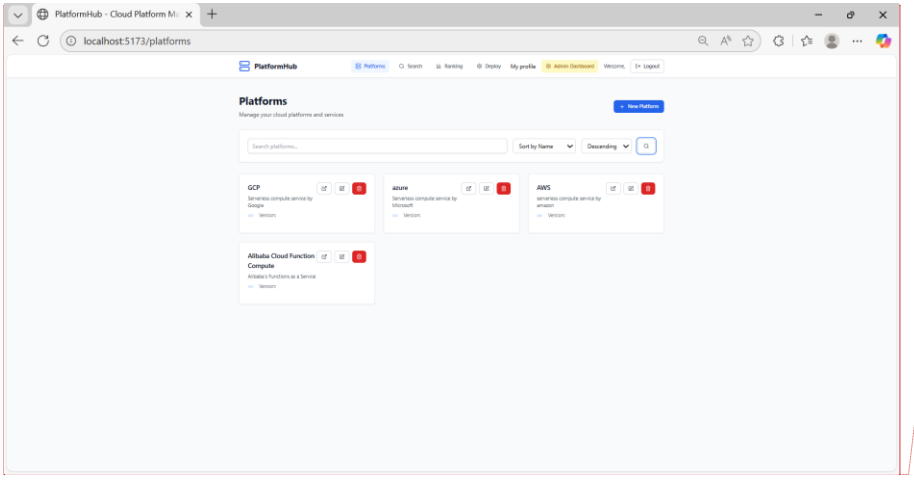
1. Ο χρήστης επιλέγει την καρτέλα Basic Search (δείτε Εικόνα 15).
2. Συμπληρώνει προαιρετικά ένα ή περισσότερα φίλτρα πεδίων, όπως:
    - Name (όνομα πλατφόρμας)
    - Category (κατηγορία)

- Description (περιγραφή)

3. Πατά το κουμπί Search για εκτέλεση του ερωτήματος.
4. Μετά την εμφάνιση των αποτελεσμάτων (δείτε Εικόνα 16), ο χρήστης μπορεί να επιλέξει ταξινόμηση (Sort by) και κατεύθυνση (Ascending/Descending) για την αναδιάταξη της λίστας (δείτε Εικόνα 17).



Εικόνα ~~16~~17: Basic Search - Αναζήτηση με φίλτρα



Εικόνα 17: Basic Search - Ταξινόμηση με φίλτρα (Name Desc)

Formatted: Greek

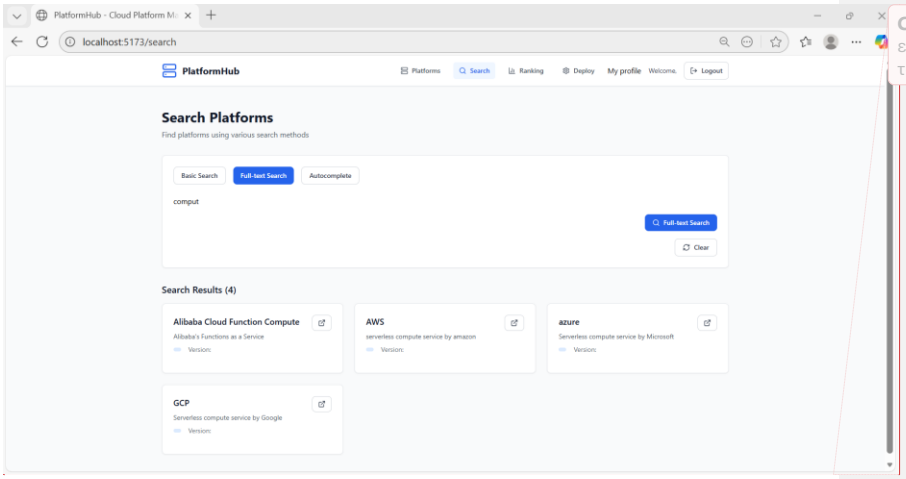
Commented [KK244]: Θα πρεπει να υποστηριχθει με ενιαίο τροπο το σενάριο. Οπότε, αυτό σημαινει πως θα έπρεπε να συνεχισουμε από την εικόνα 16 και να επιλέξουμε την αλλαγη της ταξινόνησης.
Αλλά για να έχει νόημα αυτό, θα έπρεπε η αναζητηση να επιστρέφει πολλά αποτελέσματα στην Εικόνα 16 και όχι μόνο ένα.
Άρα, θα πρεπει να αλλάξετε το περιεχομενο της Εικόνα 16.

Formatted: Greek

(B) Full-text Search (αναζήτηση με λέξη-κλειδί)

1. Ο χρήστης επιλέγει την καρτέλα Full-text Search (Δείτε Εικόνα 18).
2. Πληκτρολογεί ένα keyword στο αντίστοιχο πεδίο (π.χ. όρο σχετικό με όνομα/περιγραφή/χαρακτηριστικά).
3. Πατά Full-text Search για εκτέλεση αναζήτησης στο σύνολο των διαθέσιμων δεδομένων πλατφορμών. Οπότε, του επιστρέφονται τα αντίστοιχα αποτελέσματα, όπως φαίνεται και από την Εικόνα 18. Σημειώνεται πως στο τρέχων σενάριο απεικονίζεται και η δυνατότητα fuzzy matching διότι ο χρήστης δεν έχει παραδόσει μια πλήρης λέξη-κλειδί για την αναζήτηση.

Commented [KK245]: Κι εδώ λείπουν οι παραπομπές στις εικόνες



Commented [KK246]: Δεν χρειάζεται να έχετε και τις 2 εικόνες εδώ. Ας κρατήσετε την δευτερη που εμφανίζει και το fuzzy matching.

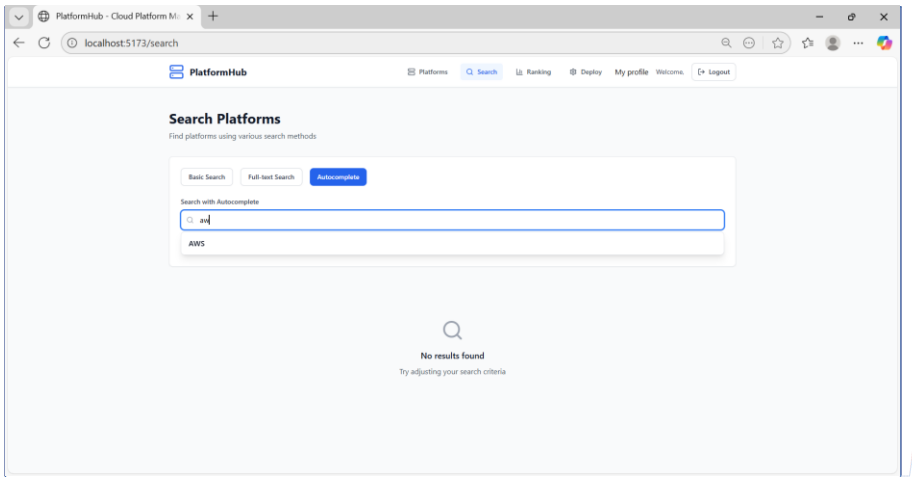
Εικόνα 18: Full-text Search — Αναζήτηση με keyword(fuzzy matching)

(C) Autocomplete (αναζήτηση με προτάσεις καθώς πληκτρολογεί)

1. Ο χρήστης επιλέγει την καρτέλα Autocomplete (Δείτε Εικόνα 19).

Commented [KK247]: Κι εδώ λειπουν οι παραπομπές του κειμένου προς τις εικόνες

2. Πληκτρολογεί στο πεδίο αναζήτησης (π.χ. μερικούς χαρακτήρες από όνομα πλατφόρμας).
3. Το σύστημα εμφανίζει προτάσεις καθώς ο χρήστης γράφει.
4. Ο χρήστης επιλέγει μία πρόταση (ή ολοκληρώνει το κείμενο) και η αναζήτηση/φόρτωση αποτελεσμάτων πραγματοποιείται βάσει της επιλογής του (Δείτε Εικόνα 20).

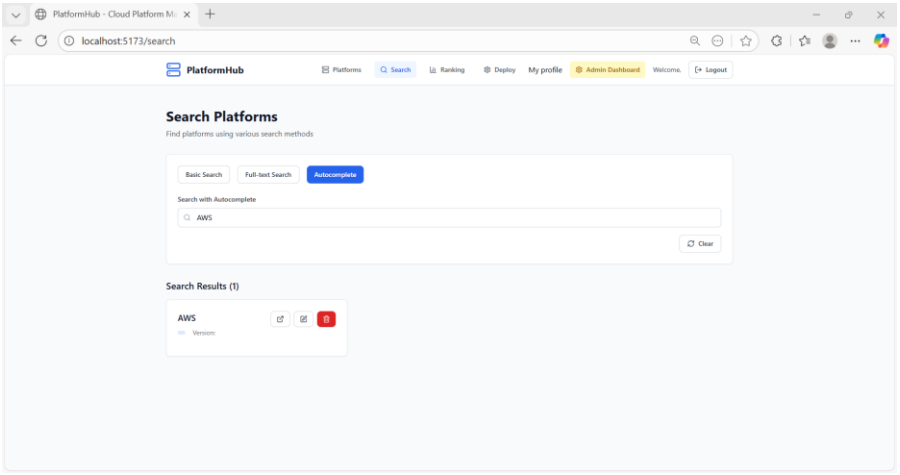


Εικόνα ~~19~~19: Autocomplete - Αναζήτηση με προτάσεις

Formatted: Greek

Commented [KK248]: Επειδή εδώ έχουμε 2 βηματα για την ολοκλήρωση της αναζητησης (πληκτρολόγηση & επιλογή suggestion), θα προτεινα να εισάγετε και μια δευτερη εικόνα ώστε να φαινεται πιο είναι το αποτέλεσμα από την επιλογή ενός suggestion.

Commented [AK249R248]: Προστέθηκέ **Εικόνα 5: Autocomplete — Αναζήτηση με προτάσεις(Αποτέλεσμα επιλογής suggestion)**



Εικόνα 2020: Autocomplete - Αναζήτηση με προτάσεις (Αποτέλεσμα επιλογής suggestion)

Αναμενόμενο Αποτέλεσμα

Το σύστημα εμφανίζει τα αποτελέσματα της αναζήτησης στην ίδια σελίδα (δείτε Εικόνες 16, 18 & 20), κάτω από τη φόρμα, με βάση τη μέθοδο που επέλεξε ο χρήστης (Basic / Full-text / Autocomplete). Αν δεν βρεθούν αντιστοιχίες, εμφανίζεται σχετικό μήνυμα (π.χ. "No results found" (δείτε Εικόνα 19). Ο χρήστης μπορεί να τροποποιήσει τα κριτήρια και να επαναλάβει την αναζήτηση ή να χρησιμοποιήσει το Clear (δείτε Εικόνα 20) για να ξεκινήσει από την αρχή.

5.2.4 Σενάριο 4 — Κατάταξη Πλατφορμών (Ranking)

Αφετηρία

Ο χρήστης έχει ήδη αυθεντικοποιηθεί επιτυχώς και από το κεντρικό μενού επιλέγει **Ranking**, μεταβαίνοντας στη σελίδα κατάταξης πλατφορμών. Θα του εμφανιστούν όλες οι πλατφόρμες που υποστηρίζονται για διάταξη συναρτήσεων από το σύστημα ταξινομημένες με προκαθορισμένο τρόπο (δείτε Εικόνα 21).

Formatted: Greek

Commented [KK250]: Υποτίθεται πως υποστηριζεται και την ταξινόμηση των αποτελεσμάτων. Αλλά δεν βλέπουμε μια τέτοια επιλογή να είναι διαθέσιμη όταν εμφανίζονται τα αποτελέσματα. Οπότε, αυτή η λειτουργια έχει υλοποιηθεί στο backend αλλά όχι στο frontend ή καθόλου; Στην δευτερη περιπτωση, θα πρέπει να την αφαιρέσετε από την Ενότητα 4.2 & 4.3.

Commented [KK251]: Ας παραπέμψετε στις προηγ. εικόνες όπου εμφανίζονται αποτελέσματα.

Commented [KK252]: Ας παραπέμψετε στην πάνω εικόνα όπου όντως φαινεται το μήνυμα αυτό.

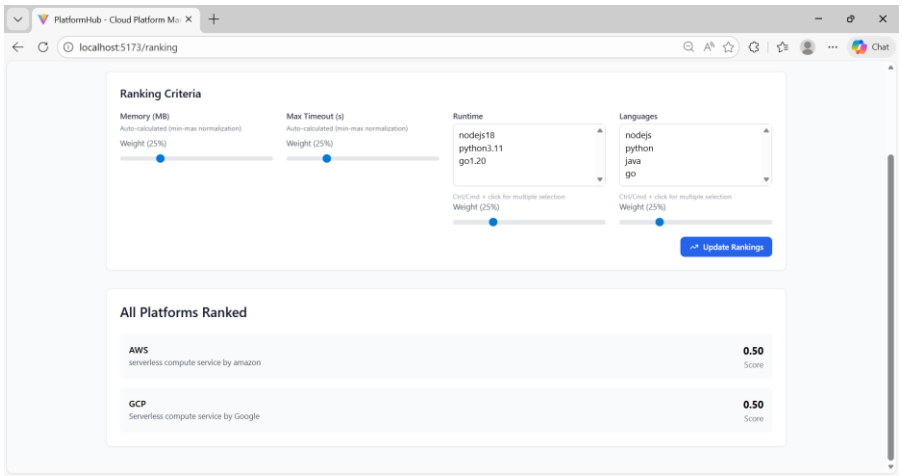
Commented [KK253]: Ας παραπέμψετε σε μια εικόνα όπου φαινεται αυτή η επιλογή/κουμπί

Commented [KK254]: Εδώ θέλουμε παραπομπή προς την σχετική εικόνα

Commented [KK255]: Θα πρότεινα τα βάρη να είναι ισομερώς κατανεμημένα στα 4 κριτήρια (δηλ. 0.25 έκαστο).

Επίσης, για ποιο λόγο βλέπουμε μόνο 2 πλατφορμες ενώ πριν ειχατε δείξει τουλάχιστον 4; Οι πλατφορμες αυτές κανονικά δεν θα πρέπει να ειχαν φιλτραριστει μετά από κάποια αναζητηση του χρηστη; Εδώ όμως βλέπουμε πως λογικά όλες λογαριάζονται για την κατάταξη.

Ένα τελευταιο: ενώ θα μπορούσε να γινεται η καταταξη με οποιοδήποτε τεχνικό κριτήριο, βλέπουμε στην εικόνα πως υποστηριζονται μόνο 4.



Εικόνα 2124: Σελίδα Ranking (φόρμα κριτηρίων + λίστα αποτελεσμάτων)

Ενέργειες Χρήστη

1. **Ορίζει τα κριτήρια κατάταξης** (κριτήρια προτίμησης), επιλέγοντας τιμές για τα διαθέσιμα πεδία:
    - **Memory (MB)** (αριθμητικό κριτήριο)
    - **Max Timeout (s)** (αριθμητικό κριτήριο)
    - **Runtime** (κατηγορικό κριτήριο – επιλογή από λίστα)
    - **Region** (κατηγορικό κριτήριο – επιλογή από λίστα)
2. **Ορίζει βάρη (weights) ανά κριτήριο**, μέσω sliders, τα οποία εμφανίζονται και αριθμητικά ως ποσοστά (π.χ. *Weight 25%*). Τα βάρη **κανονικοποιούνται** ώστε το άθροισμά τους να είναι **100%**, εξασφαλίζοντας ότι η τελική βαθμολογία είναι εντός συγκεκριμένου εύρους και σωστά υπολογισμένη.
3. Επιλέγει **Update Rankings** για να εκτελεστεί ο υπολογισμός κατάταξης.

Formatted: Greek

Commented [AK256R255]: Έχω για την κατάταξη τις πλατφόρμες που υποστηριζουν και διαταξη.Για τα κριτήρια Το UI δείχνει 4 κριτήρια γιατί στο MVP έγιναν τα πιο αντιπροσωπευτικά συχνά. Το backend όμως είναι επεκτάσιμο.

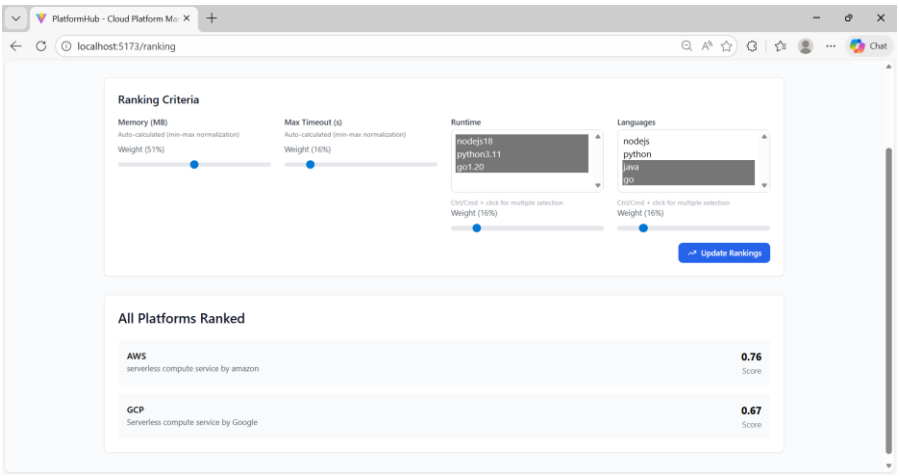
Commented [KK257R255]: Αυτό θα πρεπει να διευκρινιστεί εντος του κειμένου.

Commented [KK258]: Αν και πλέον είναι αργά, είναι λίγο «κουφό» να παρέχεται μια τιμή διότι έτσι είναι σαν να έχουμε φιλτράρισμα των πλατφορμών. Αν βέβαια κάτι τέτοιο ισχύει, δεν είναι κακό διότι ενδέχεται ο χρήστης να επιθυμεί κάτι τέτοιο. Απλώς, σε μια τέτοια περιπτωση, προτεινω η λειτουργια του συστήματος να μην λέγεται κατάταξη αλλά «φιλτραρισμένη κατάταξη» ώστε να φαινεται πως γίνονται και τα δύο. Η αλλαγη θα πρέπει να γινει κι εδώ αλλά και στο Κεφάλαιο 4!!!

Αναμενόμενο Αποτέλεσμα

Το σύστημα εμφανίζει στην ίδια σελίδα τη λίστα **All Platforms Ranked** (δείτε Εικόνα 22), όπου για κάθε πλατφόρμα παρουσιάζονται:

- **ranking score** (0–1),
- βασικές πληροφορίες (όνομα/περιγραφή),
- η σειρά κατάταξης (rank order).



Εικόνα ~~2222~~: Αποτελέσματα κατάταξης (All Platforms Ranked + scores)

5.2.5 Σενάριο 5 — Διάταξη Συνάρτησης (Deployment)

Αφετηρία

Ο χρήστης ανοίγει τη σελίδα *Deploy Function* από το κύριο μενού (δείτε Εικόνα 23~~1~~).

Ενέργειες

1. Επιλέγει πλατφόρμα (AWS ή GCP διότι αυτές υποστηρίζονται μέχρι στιγμής από το σύστημά μας).
2. Εισάγει όνομα συνάρτησης.

Formatted: Greek

Commented [KK259]: Εφόσον έχουμε και φιλτραρισμα πριν την κατάταξη, θα πρότεινα να δειχνετε αρχικά 3 πλατφόρμες κι έπειτα να εμφανιζονται ταξινομημένες 2 διότι η 3η θα έχει φιλτραριστεί με βάση τις τιμές των κριτηριων που παρέχονται.

Commented [KK260]: Η Εικόνα 1 αφορά την αρχική σελίδα για επισκέπτη!
Οπότε, ας κάνετε τη σχετική διόρθωση.

Commented [KK261]: Οπότε μόνο αυτές υποστηρίζονται μέχρι τώρα. Ανεξάρτητα του ποιες μπορει να εισάγει ο διαχειριστής, σωστά;

Commented [AK262R261]: Ναι σωστά

3. Επιλέγει από το τοπικό του σύστημα ένα **ZIP αρχείο** που περιέχει τον κώδικα της συνάρτησης (*Upload .zip*) ή επικολλά το πακέτο κώδικα σε μορφή **Base64** *(οπότε η εφαρμογή δεν ζητά zip file απευθείας — ο χρήστης παρέχει string Base64)*.
4. Συμπληρώνει runtime, handler και optional region.
5. Υποβάλλει τη φόρμα (δείτε Εικόνα 23).
6. Αυτόματη μετάβαση σε νέα οθόνη κατάστασης (δείτε Εικόνα 24)

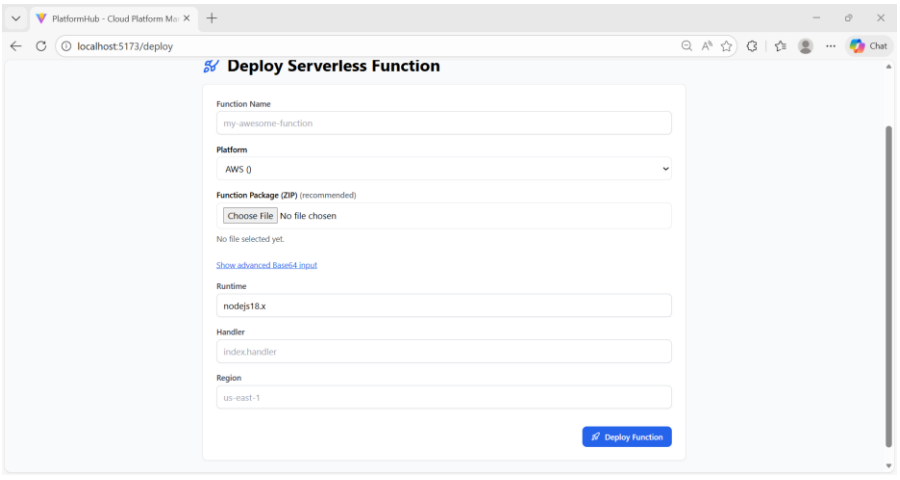
Αναμενόμενο Αποτέλεσμα

Ο χρήστης βλέπει τα εξής:

- Η διάταξη εκκινεί επιτυχώς (request accepted) και το σύστημα ολοκληρώνει τη διαδικασία deployment στο backend.
- Ο χρήστης **δεν παραμένει στη φόρμα** μετά το αποτέλεσμα, αλλά μεταφέρεται σε οθόνη κατάστασης.

Ο χρήστης βλέπει στη σελίδα **Profile**:

- **την κατάσταση διάταξης** (success/failure),
- **το τελικό endpoint URL** (μόνο σε success με clickable link),
- **οργανωμένα logs εκτέλεσης** (είτε εμφανιζόμενα είτε προσβάσιμα από εκεί μέσω συνδέσμου – δείτε Εικόνα 25).

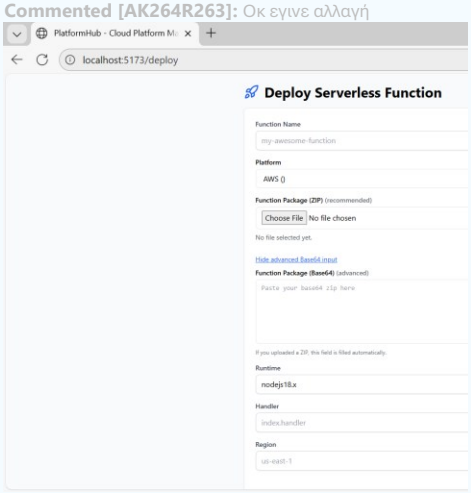


Εικόνα 23: Φόρμα Διατάξης Συνάρτησης

Formatted: Greek

Commented [KK263]: Αυτό δεν μου μοιάζει να είναι φιλικό ως προς τον χρηστη. Δηλαδή ο χρήστης παρέχει εδώ το περιεχομενο του zip αρχείου; Και πως θα το αντιγράψει και θα το μετατρέψει σε μορφή Base64; Θα ήταν προτιμότερη η επιλογή του ZIP αρχειου από το τοπικό σύστημα του χρηστη και η αποστολή του. Αλλιώς δεν είναι ευχρηστος ο τρεχων τροπος.

Commented [AK264R263]: Οκ εγινε αλλαγή



Commented [KK265R263]: Δεν βλέπουμε την εικόνα που έχετε στο σχολιο εντός της αναφοράς. Μάλιστα είναι πολύ μικρή και δεν μπορώ να δω και να ελέγξω το περιεχόμενό της.

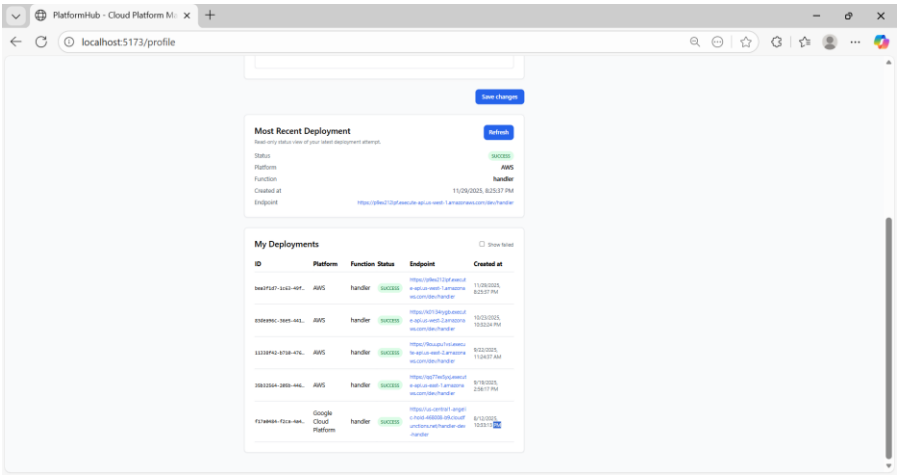
Commented [AK266R263]: Προστεθηκε η εικόνα στο σενάριο

Commented [KK267R263]: Οκ αλλά όπως ανέφερα, δεν είναι βολικό για τον χρηστη να επικολλά Base64 κωδικοποίηση ενός πράγματος. Θα έπρεπε να υπάρχει κουμπί ώστε να επιλέγει zip αρχειο προς μεταφορτωση. Αυτό όμως δεν το βλέπω στην εικόνα.

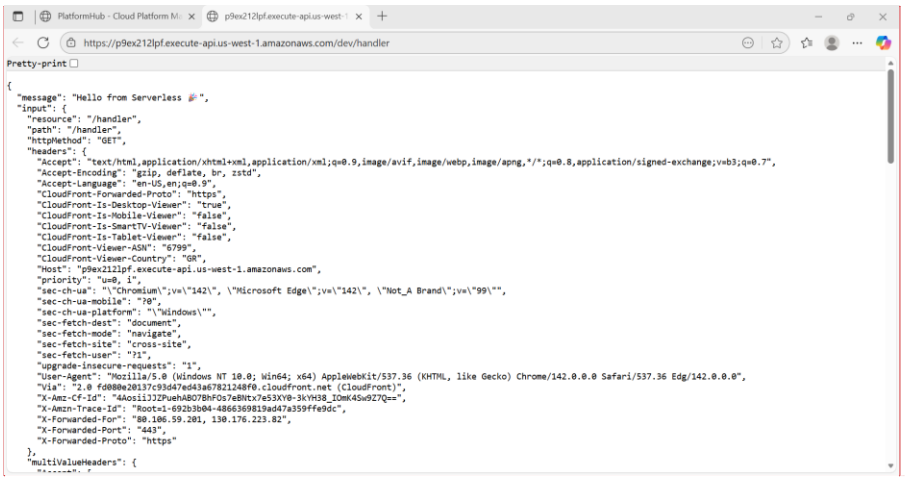
Commented [KK268]: Τα credentials του χρήστη τα έχετε λάβει ήδη; Και αν ναι, πότε; Διότι χωρις αυτά δεν μπορει να επιτύχει η διάταξη.

Commented [AK269R268]: Τα credentials δεν εισάγονται στη φόρμα διάταξης. Ο χρήστης τα έχει καταχωρήσει προηγουμένως στο σύστημα και αποθηκεύονται στη βάση ως UserCredentials

Formatted: Greek



Εικόνα 2424: Αποτέλεσμα Επιτυχούς Διάταξης (Endpoint View)



Εικόνα 25: Περιεχόμενο διάταξης

Commented [KK270]: Αυτή η εικόνα δεν έχει λεζάντα. Να θεωρήσουμε πως προκύπτει πατώντας τον συνδεσμο στο κάτω μέρος της προηγουμενης εικόνας

Formatted: Greek

5.2.6 Σενάριο 6 — Διαχείριση Χρηστών (Admin – Απενεργοποίηση χρήστη)

Αφετηρία

Ο διαχειριστής έχει αυθεντικοποιηθεί επιτυχώς και από το κεντρικό μενού μεταβαίνει στην ενότητα Admin → Admin Dashboard, όπου εμφανίζεται η λίστα χρηστών.

Ενέργειες

1. Επιλογή Admin Dashboard (δείτε Εικόνα 26) – οπότε προβάλλεται από το σύστημα η λίστα των χρηστών του (δείτε Εικόνα 27)
2. Ο διαχειριστής επιλέγει Ενεργοποίηση / Απενεργοποίηση λογαριασμού για συγκεκριμένο χρήστη οπότε το σύστημα εφαρμόζει το αίτημα αυτό και ανανεώνει αναλόγως την λίστα χρηστών (δείτε Εικόνα 28).

Αναμενόμενο Αποτέλεσμα

Ο διαχειριστής έχει ρυθμίσει τη προσβασιμότητα ενός χρήστη, απενεργοποιώντας τον λογαριασμό του όταν αυτό κρίνεται απαραίτητο (π.χ. σε περίπτωση ύποπτης ή κακόβουλης δραστηριότητας).

Commented [KK271]: Αν στο σύστημά σας δεν απενεργοποιειται αυτόματα ο λογαριασμός χρηστη μετά από συνεχομενες αποτυχημένες προσπάθειες και αν ένας νέος λογαριασμός είναι προκαθορισμένα ενεργοποιημένος, τοτε το τελευταιο μέρος του Σεναρίου 1 θα πρέπει να διαγραφει (ενεργοποίηση χρηστη). Διότι ουσιαστικά θεωρούμε πως δεν έχει νοημα. Έχει νόημα το σενάριο πως ο διαχειριστής απενεργοποιει τον λογαριασμό ενός πιθανώς κακόβουλου χρηστη.
Θα το παρουσιαζα μάλιστα εδώ με αυτό τον τροπο.

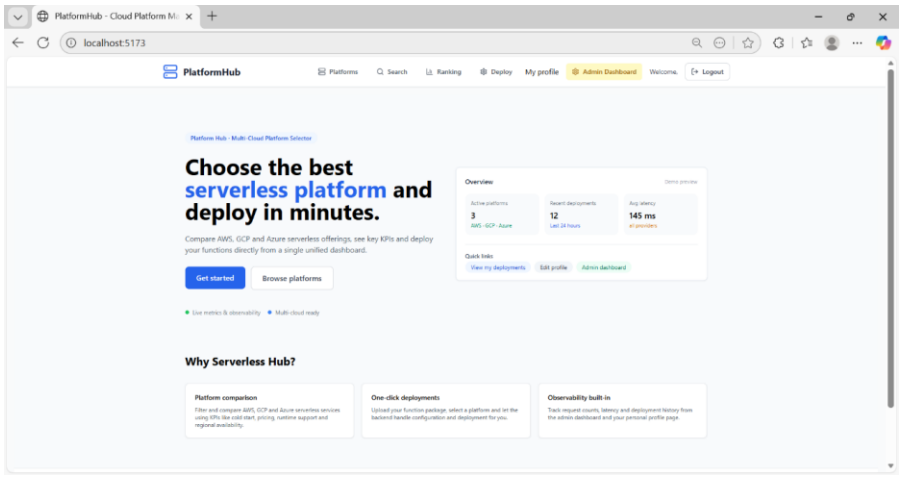
Commented [AK272R271]: Έχω προσθέσει σχετικές εικόνες στο σενάριο 1 για την απενεργοποίηση χρήστη μετά από 3 λανθασμένες προσπάθειες σύνδεσης.

Commented [KK273]: Έχετε ξεχάσει το περιεχομενο της προηγ. έκδοσης του σεναριου. Οπότε, κοιτάξτε μόνο τα σχόλια που έχω για την νέα έκδοση. Η παλαιά έκδοση θα πρεπει να διαγραφει.

Commented [KK274]: Η πρώτη ενέργεια είναι αυτό που εμφανίζεται στον χρηστη αν επιλέξει User Management. Οπότε, είναι ενέργεια του χρηστη;

Επίσης, όπως σας ειχα αναφέρει, η ενεργοποίηση, απενεργοποίηση και διαγραφή χρηστη είναι υπο-σενάρια της διαχειρισης χρηστων που θα πρεπει να καλυφθούν από το τρέχων σενάριο. Εσεις τα δειχνετε ως απλές ενέργειες, πραγμα που δεν είναι σωστο.

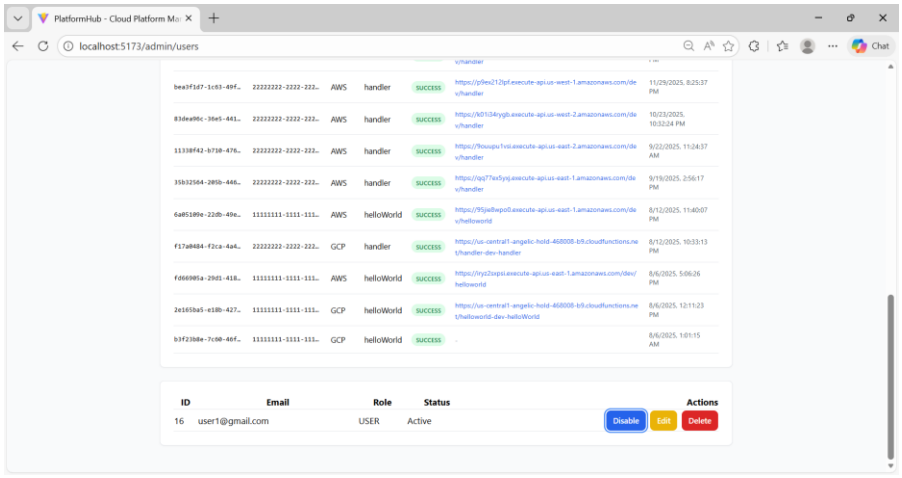
Formatted: Greek



Εικόνα 26: Επιλογή Admin Dashboard

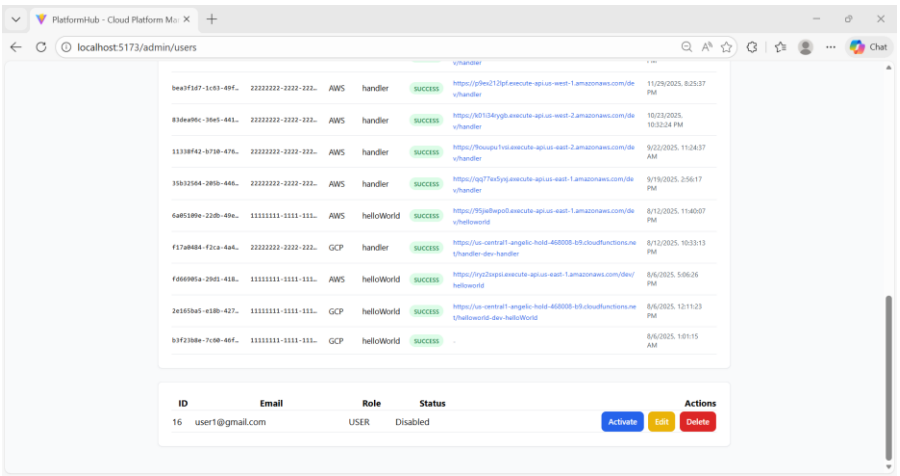
Commented [KK275]: Κανονικά ο admin δεν θα πρέπει να μπορει να απενεργοποιει τον ευατο του ή να τον διαγράφει.
Μόνο τους άλλους χρηστες. Άρα, το τρέχων περιεχομενο της εικόνας θα πρέπει να αλλάξει.

Επίσης, να υποθέσω πως τα στοιχεια ενός χρηστη δεν αλλάζουν από τον admin. Διότι δεν υπάρχει το κουμπί Edit.



Εικόνα ~~27~~27: Πίνακας Διαχείρισης Χρηστών (Admin View)

Formatted: Greek

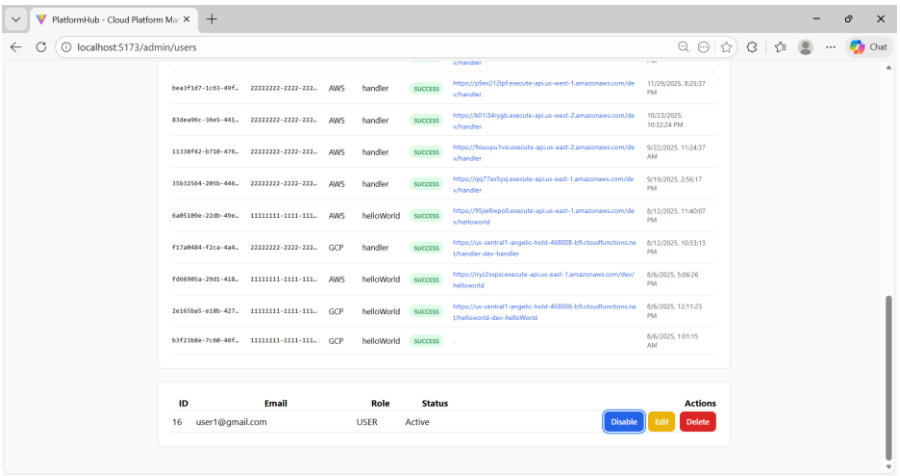


Εικόνα ~~2828~~: Απενεργοποίηση Χρήστη

5.2.7 Σενάριο 7 — Διαχείριση Χρηστών (Admin: Ενημέρωση στοιχείων/Διαγραφή Χρήστη)

Αφετηρία

*Ο διαχειριστής (**ADMIN**) έχει αυθεντικοποιηθεί επιτυχώς και από το κεντρικό μενού μεταβαίνει στην ενότητα **Admin → Admin Dashboard**, όπου εμφανίζεται η λίστα χρηστών (δείτε Εικόνα 29).*



Εικόνα 29: User Management — Λίστα χρηστών (Admin View)

Ενέργειες

Α. Ενημέρωση στοιχείων χρήστη

1. *Ο διαχειριστής επιλέγει τον χρήστη στη λίστα και πατάει το κουμπί **Edit**.*
2. *Το σύστημα εμφανίζει φόρμα/διάλογο με τα στοιχεία του χρήστη. ~~(δείτε Εικόνα 30).~~*
3. *Ο διαχειριστής τροποποιεί επιτρεπόμενα πεδία (π.χ. **email**, **username/name**) (δείτε Εικόνα 30).*
4. *Επιλέγει **Save** για αποθήκευση.*
5. *Το σύστημα εμφανίζει μήνυμα επιτυχίας και ενημερώνει τη λίστα με τα νέα στοιχεία (δείτε Εικόνα 31).*

Formatted: Greek

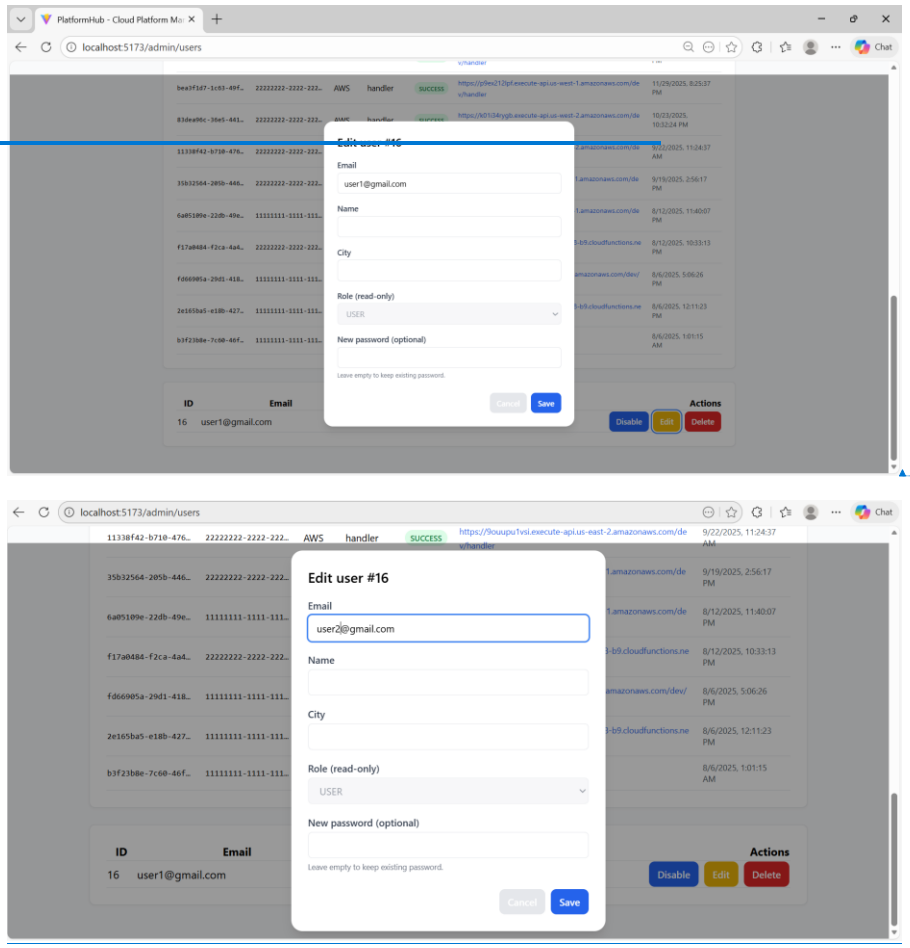
Commented [KK276]: Η Εικόνα 30 θα έπρεπε να εμφανίζει τα νέα στοιχεια του χρηστη προς εφαρμογή (φαντάζομαι πως έγινε αλλαγη του email οπότε θα έπρεπε να εμφανίσετε την νέα email διεύθυνση).
Εφόσον κάνετε την αλλαγή, μεταφέρετε την τρεχουσα παραπομπή στο βημα 3 (από το βημα 2)

Formatted: Greek

Commented [KK277]: Η Εικόνα 30 θα έπρεπε να εμφανίζει τα νέα στοιχεια του χρηστη προς εφαρμογή (φαντάζομαι πως έγινε αλλαγη του email οπότε θα έπρεπε να εμφανίσετε την νέα email διεύθυνση).
Εφόσον κάνετε την αλλαγή, μεταφέρετε την τρεχουσα παραπομπή στο βημα 3 (από το βημα 2)

Commented [KK278]: Καλό θα ήταν ο ρολος να φαινεται πως είναι fixed και δεν μπορει να αλλάξει.

Επίσης, ας δείξετε πως αλλάζετε το email ώστε να υπάρχει διασύνδεση μεταξύ της τρέχουσας εικόνας και της επόμενης που δειχνει το αναμενόμενο αποτελεσμα.



Εικόνα ~~3038~~: Φόρμα Επεξεργασίας Χρήστη (Edit User)

Formatted: Greek

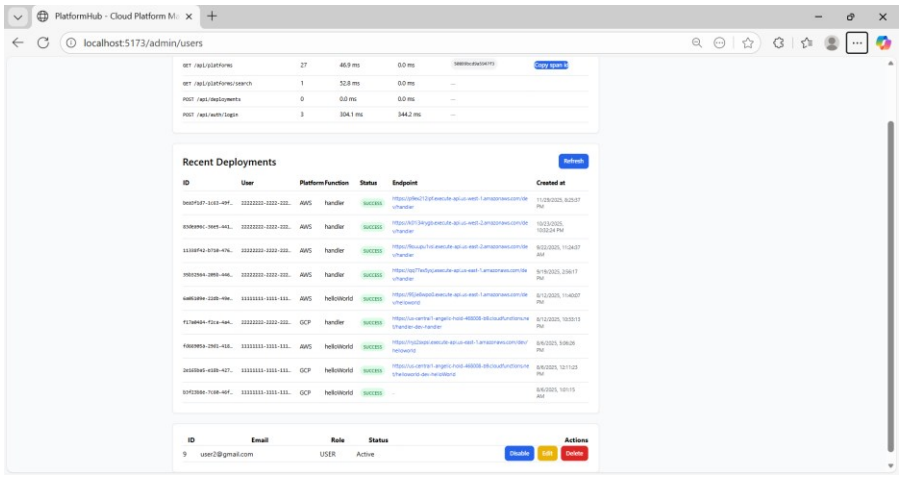
Commented [AK279R278]: Αλλαξε σε disabled το πεδιο

Commented [KK280R278]: Οκ αλλά εφοσον είναι disabled, θα έπρεπε να μην εμφανίζεται στην φορμα ή να εμφανιζοταν με εμφανή τροπο πως δεν μπορει να αλλάξει.

Commented [AK281R278]: έχω προχωρήσει σε αλλαγή με περιγραφη (readonly)

Formatted: Greek

Formatted: Greek

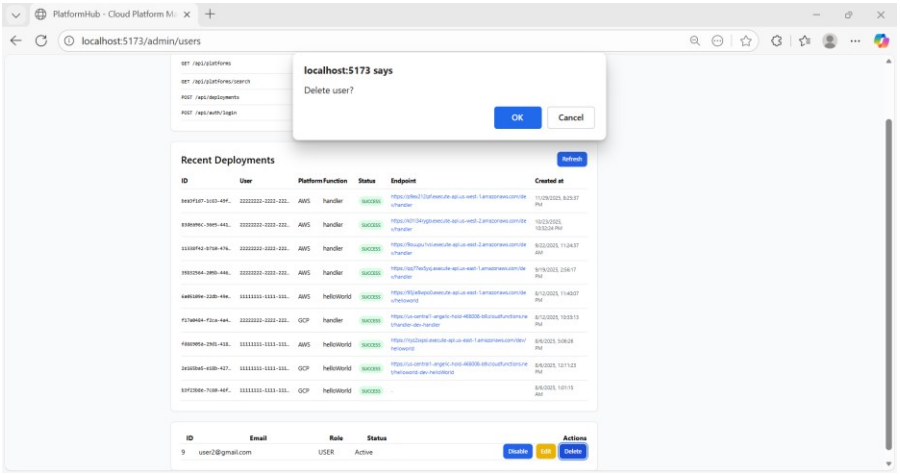


Εικόνα ~~3131~~: Λίστα χρηστών μετά την ενημέρωση (Admin View)

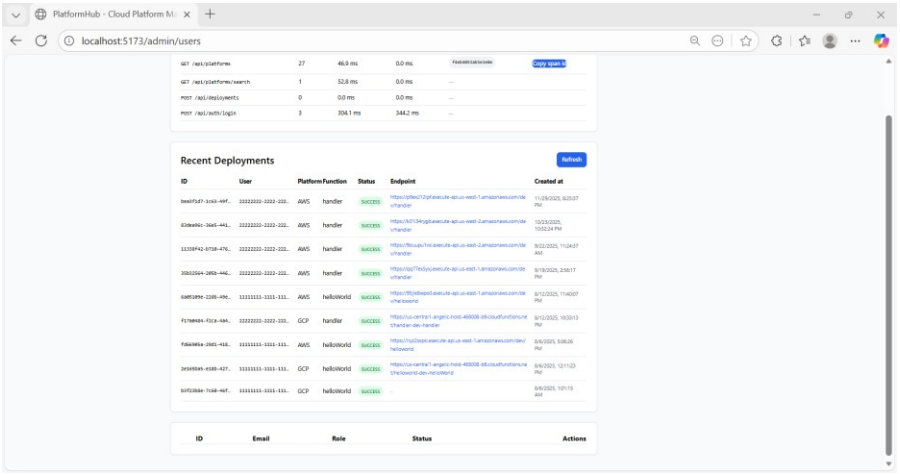
Β. Διαγραφή χρήστη

1. *Ο διαχειριστής εντοπίζει τον χρήστη στη λίστα και επιλέγει **Delete**.*
2. *Το σύστημα εμφανίζει μήνυμα επιβεβαίωσης (confirmation) (δείτε Εικόνα 32).*
3. *Ο διαχειριστής επιβεβαιώνει τη διαγραφή.*
4. *Το σύστημα αφαιρεί τον χρήστη από τη λίστα και εμφανίζει μήνυμα ολοκλήρωσης (δείτε Εικόνα 33).*

Formatted: Greek



Εικόνα 3232: Επιβεβαίωση Διαγραφής Χρήστη (Delete Confirmation)



Εικόνα 3333: Λίστα χρηστών μετά τη διαγραφή (Admin View)

Αναμενόμενο Αποτέλεσμα

*Ο διαχειριστής μπορεί να **ενημερώνει** τα στοιχεία ενός χρήστη και να **διαγράφει** χρήστες από το σύστημα, με άμεση ενημέρωση της λίστας χρηστών και αντίστοιχα μηνύματα επιβεβαίωσης.*

Σημείωση διαχείρισης admin λογαριασμού: Οι απλοί χρήστες μπορούν να εγγραφούν από το UI. Ο λογαριασμός διαχειριστή δημιουργείται αρχικά από τον διαχειριστή της εγκατάστασης με προκαθορισμένα στοιχεία

Formatted: Greek

6. Συμπεράσματα & Μελλοντική Εργασία

6.1 Συμπεράσματα

Η παρούσα εργασία παρουσίασε το πληροφοριακό σύστημα **Serverless Platform Selector**, μια ολοκληρωμένη εφαρμογή που επιτρέπει την **αναζήτηση** και **συγκριτική αξιολόγηση serverless πλατφορμών καθώς και την εναλλακτική διάταξη serverless συναρτήσεων** σε πολλαπλές serverless πλατφόρμες.
Η καινοτομία της υλοποίησης έγκειται:

- στην **ενιαία πολυνεφική/πολυπλατφορμική προσέγγιση (multi-cloud/multi-platform)**,
- στην καθοδηγούμενη αναζήτηση και ταξινόμηση των πλατφορμών με βάση τους περιορισμούς και προτιμήσεις του χρήστη
- στην **αυτόματη κατάταξη πλατφορμών βάσει σταθμισμένων κριτηρίων**,
- και στην **αυτόνομη διάταξη serverless συναρτήσεων** σε διαφορετικές serverless πλατφόρμες μέσω του Serverless Framework.

Η αρχιτεκτονική βασισμένη στο πρότυπο **Controller–Service–Repository (CSR)** ώστε να εξασφαλίσει υψηλή επεκτασιμότητα και καθαρό διαχωρισμό ρόλων. Επιπλέον, η χρήση τεχνολογιών όπως **Spring Boot, React και PostgreSQL** συνέβαλε στη δημιουργία ενός πλήρως λειτουργικού, χρηστικού και ευέλικτου συστήματος.

Το σύστημα ανταποκρίνεται σε πραγματικές ανάγκες προγραμματιστών και ερευνητών για **αντικειμενική αξιολόγηση πλατφορμών serverless** και **αυτοματοποιημένη ανάπτυξη εφαρμογών νέφους**.

6.2 Μελλοντική Εργασία

Η παρούσα εργασία μπορεί να επεκταθεί σε πολλαπλές κατευθύνσεις, οι οποίες μπορούν να ενισχύσουν τόσο τη λειτουργικότητα του συστήματος όσο και την ολοκληρωμένη εμπειρία των χρηστών. Ακολουθούν οι σημαντικότερες από αυτές:

(α) Υποστήριξη επιπλέον serverless πλατφορμών

Ένα σημαντικό επόμενο βήμα είναι η ενσωμάτωση περισσότερων serverless platform providers, όπως:

- **IBM Cloud Functions (OpenWhisk)**

Formatted: Greek

Commented [KK282]: Ας το αλλάξετε όπως επιθυμειτε - έβαλα καθοδηγούμενη με την λογική πως η εφαρμογή διευκολύνει το χρήστη να δώσει τους περιορισμούς του - αλλά αυτό μένει να αποδειχθεί πως όντως υπάρχει στην εφαρμογή σας

- **Alibaba Cloud Function Compute**
- **OpenFaaS (self-hosted serverless)**
- **Oracle Functions**
- **Cloudflare Workers**

Η προσθήκη αυτών των παρόχων θα επεκτείνει την πολυ-νεφική δυναμική του συστήματος, επιτρέποντας στον χρήστη να συγκρίνει μεγαλύτερο εύρος τεχνολογιών και να επιλέγει πλατφόρμες με βάση εξειδικευμένες ανάγκες (edge execution, enterprise-grade SLAs, hybrid deployment).

(β) Ενσωμάτωση Μονάδων benchmarking και Απομακρυσμένης Διαχείρισης Συναρτήσεων

Μια σημαντική κατεύθυνση είναι η προσθήκη ενός **ενσωματωμένου benchmark engine**. Ο σκοπός της ενότητας αυτής δεν είναι απλώς η προβολή εξωτερικών μετρήσεων, αλλά η **εκτέλεση δοκιμών πάνω στα deployed functions του χρήστη**.

Ενδεικτικά benchmarks:

- **Latency** (cold start vs warm start)
- **Throughput / concurrency tests**
- **Cost-efficiency metrics**
- **Invocation failure rate**
- **Execution duration variability**

Η διαδικασία θα περιλαμβάνει:

1. Αυτόματη δημιουργία ενός test payload.
2. Εκτέλεση ενός σετ μετρήσεων πάνω στο endpoint της συνάρτησης του χρήστη.
3. Αποθήκευση των μετρήσεων σε βάση δεδομένων.
4. Οπτικοποίηση των αποτελεσμάτων στο frontend.

Αυτό προσφέρει ουσιαστική αξιολόγηση των πραγματικών επιδόσεων της συνάρτησης μέσα στο περιβάλλον της επιλεγμένης πλατφόρμας.

Σε αυτή τη λογική, το σύστημα δεν περιορίζεται απλώς στη διάταξη σε πολυνεφικά περιβάλλοντα (AWS, Azure, GCP), αλλά επεκτείνεται και στην **απομακρυσμένη**

Formatted: Greek

Commented [KK283]: Θα προτεινα να επιτρέπεται η (απομακρυσμένη) διαχείριση των συναρτήσεων σε πολυνεφικά περιβάλλοντα και όχι μόνο η διάταξή τους. Στο πλαισιο αυτό μπορει να ενταχθεί και το benchmarking για τον έλεγχο της απόδοσης της συνάρτησης.

διαχείριση των συναρτήσεων που έχουν εγκατασταθεί σε αυτούς τους παρόχους. Η απομακρυσμένη διαχείριση περιλαμβάνει ενέργειες όπως:

- ενεργοποίηση/απενεργοποίηση χειροκίνητης κλιμάκωσης,
- ανάκτηση ρυθμίσεων και μεταδεδομένων της συνάρτησης,
- λήψη ιστορικού invocation logs,
- εκτέλεση benchmarking,
- ενημέρωση παραμέτρων διάταξης (π.χ. μνήμη, timeout),
- μελλοντική υποστήριξη redeploy / rollback.

Με τον συνδυασμό αυτών των λειτουργιών, το σύστημα μετατρέπεται από ένα εργαλείο επιλογής πλατφόρμας σε ένα **ενιαίο πολυνεφικό περιβάλλον διαχείρισης και αξιολόγησης serverless συναρτήσεων**, επιτρέποντας:

- συνεχή παρακολούθηση της απόδοσης,
- συγκρισιμότητα μεταξύ παρόχων με πραγματικά δεδομένα,
- εντοπισμό προβλημάτων cold start ή scaling,
- καλύτερη εκτίμηση κόστους χρήσης για κάθε πλατφόρμα,
- υποστήριξη κύκλων optimization και fine-tuning.

Με αυτόν τον τρόπο, το benchmarking δεν αποτελεί μεμονωμένη λειτουργία, αλλά εντάσσεται σε μια **ευρύτερη μονάδα απομακρυσμένης, πολυνεφικής λειτουργικής διαχείρισης**, η οποία ενισχύει δραστικά τη χρησιμότητα και την πρακτική αξία του συστήματος.

(γ) Προηγμένος μηχανισμός ranking με χρήση machine learning

Η υπάρχουσα διαδικασία κατάταξης (ranking) βασίζεται σε στατικά κριτήρια και βάρη που ορίζει ο χρήστης βάσει των προτιμήσεών του. Μελλοντικά, μπορεί να ενσωματωθεί ένας δυναμικός αλγόριθμος μάθησης προτιμήσεων (preference learning), ο οποίος:

- θα μαθαίνει **τα βάρη** που θεωρεί σημαντικά ο χρήστης (π.χ. latency > cost)

- θα μαθαίνει **τα κριτήρια** που ο χρήστης λαμβάνει υπόψη συστηματικά
- θα αναπροσαρμόζει το ranking σε πραγματικό χρόνο
- θα προσαρμόζεται στο ιστορικό επιλογών του χρήστη. Σημαντικό είναι ότι **δεν φιλτράρονται πλατφόρμες** ούτε αποκλείονται νέες επιλογές. Όλες οι πλατφόρμες παραμένουν διαθέσιμες· απλώς η κατάταξη εμπλουτίζεται με ήπιες ενδείξεις από τη συμπεριφορά του χρήστη. Έτσι, αποφεύγεται το πρόβλημα του περιορισμού (filter bubble) και διατηρείται πλήρως η ιδιότητα του *serendipity*, επιτρέποντας την ανακάλυψη νέων ή απρόσμενων επιλογών.

Εκτός από το adaptive ranking, μπορούν να ενσωματωθούν:

- **συστήματα προτάσεων** πλατφορμών με collaborative filtering. Ο μηχανισμός collaborative filtering, σε συνδυασμό με τα υπόλοιπα ML στοιχεία, αποτελεί τη βάση ενός **πλήρως ολοκληρωμένου συστήματος συστάσεων (multicloud serverless recommendation engine)**. Είναι η φυσική εξέλιξη της πλατφόρμας και μπορεί να ενσωματώσει:
    - προτιμήσεις χρηστών,
    - ιστορικά δεδομένα διατάξεων,
    - benchmarking αποτελέσματα,
    - runtime performance,
    - κόστη,
    - patterns χρήσης συναρτήσεων.

- **weight optimization models** με Bayesian learning
- **explainable AI** ώστε να δικαιολογείται γιατί μια πλατφόρμα κατατάχθηκε ψηλότερα σε σχέση με τις άλλες – αυτό θα αυξήσει την εμπιστοσύνη των χρηστών στο σύστημα

(δ) Ενοποίηση με observability APIs και dashboards παρακολούθησης

Σημαντική κατεύθυνση αποτελεί η σύνδεση του συστήματος με APIs παρακολούθησης όπως:

- **Prometheus**

Formatted: Greek

Commented [KK284]: Αυτό σημαινει δηλαδή πως θα φιλτράρει τις πλατφορμες για να ταξινομεί αυτές που διαχρονικά προτιμά ο χρήστης;
Αυτό φαινεται ενδιαφέρον αλλά αντικρούσει στην ιδιότητα του serendipity (παροχή και νέων επιλογών που ενδέχεται να εντυπωσιάσουν τον χρηστη).

Commented [KK285]: Ουσιαστικό αυτό είναι το όλο vision: ολοκληρωμένο συστημα συστάσεων που να περιλαμβάνει όλες τις υπόλοιπες ωραίες ιδέες που έχετε.

Commented [AK286R285]: Ευχαριτώ κύριε Κρητικέ

- **Datadog**
- **AWS CloudWatch**
- **GCP Cloud Monitoring**
- **New Relic**

Η πληροφορία που μπορεί να συλλεχθεί περιλαμβάνει:

- *invocation count*
- *average execution time*
- *error rate*
- *memory usage*
- *cold start trends*
- *Custom metrics (προσωποποιημένες μετρικές)*: Για ακόμη μεγαλύτερη ευελιξία, το σύστημα μπορεί να επιτρέπει και την ενσωμάτωση *custom metrics* που ορίζει ο ίδιος ο χρήστης. Αυτό είναι σημαντικό γιατί κάθε εφαρμογή ή domain μπορεί να αξιολογεί διαφορετικού τύπου μετρικές ως κρίσιμες (π.χ. SLA violations, domain-specific KPIs, application-level latency buckets, success ratios για συγκεκριμένες λειτουργίες). Οι χρήστες θα μπορούν να ορίζουν τις δικές τους μετρικές μέσω configuration ή metrics endpoints ενώ θα μπορούν να καταχωρούν thresholds για alerting και να παρακολουθούν custom dashboards στο UI της εφαρμογής. Επιπλέον, είναι δυνατός ο συνδυασμός custom μετρικών με προεπιλεγμένες για πιο ακριβή αξιολόγηση πλατφορμών και επιλογών deployment.

Η ενσωμάτωση αυτή μπορεί να λειτουργήσει με δύο τρόπους:

1. **Pull-based** **μοντέλο**
   Το σύστημα (Monitoring Integration) ανακτά τα metrics μέσω API calls (από τις serverless πλατφόρμες) και τα αποθηκεύει στη βάση.
2. **Push-based** **μοντέλο**
   Οι serverless πλατφόρμες στέλνουν metrics προς το σύστημα μέσω webhooks.

Κατόπιν, μέσω του frontend μπορούν να δημιουργηθούν **interactive dashboards**, τα οποία θα παρέχουν:

- real-time γραφήματα
- historical analytics

Formatted: Greek

Commented [KK287]: + custom μετρικές του χρήστη; Αυτό θα το κάνει πιο ενδιαφέρον. Διότι σε διαφορετικά πεδια εφαρμογών διαφορετικές μετρικές μπορει να είναι σημαντικές και θα πρεπει να μετρηθούν/παρακολουθηθούν

- ειδοποιήσεις (alerts) βάσει thresholds

Έτσι, ο χρήστης αποκτά ένα πλήρες περιβάλλον παρακολούθησης και παρατηρησιμότητας χωρίς να χρειάζεται να μεταβαίνει σε εξωτερικά εργαλεία.

Formatted: Greek

Formatted: Greek

Βιβλιογραφία

[1] Amazon Web Services (AWS). (n.d.). *What is Serverless Computing?* Available at: https://aws.amazon.com/serverless/ (Accessed: 18 February 2025).

[2] Red Hat. (n.d.). *What is Serverless?* Available at: https://www.redhat.com/en/topics/serverless (Accessed: 18 February 2025).

[3] Lumigo. (2023). *AWS Lambda vs. Google Cloud Functions – A Comprehensive Comparison.* Available at: https://lumigo.io/blog/aws-lambda-vs-google-cloud-functions/ (Accessed: 18 February 2025).

[4] National Institute of Standards and Technology (NIST). (2011). *NIST SP 800-145: The NIST Definition of Cloud Computing.* Available at: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[5] P. V. R. Prasad & S. K. Chaitanya. (2021). *Serverless Through Cloud Native Architecture, International Journal of Engineering Research & Technology (IJERT)*, Vol. 10 Issue 07. Available at: https://www.ijert.org/serverless-through-cloud-native-architecture

[6] Mell, P. & Grance, T. (2011). *The NIST Definition of Cloud Computing,* NIST Special Publication 800-145. Gaithersburg, MD: National Institute of Standards and Technology. DOI: https://doi.org/10.6028/NIST.SP.800-145