ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ

ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Αποτίμηση Ποιότητας Κώδικα σε Έργα Λογισμικού (Code Quality Assessment in Software Projects)

Λέντζας Αθανάσιος (icsd13103)

Επιβλέπων Καθηγητής: Κρητικός Κυριάκος (Αν. Καθηγητής)

Μέλη Επιτροπής Εξέτασης: Κοκολάκης Σπυρίδων (Καθηγητής), Παναγιώτης Συμεωνίδης (Αν. Καθηγητής)

Σάμος 2023

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Περίληψη

Η διαδικασία αξιολόγησης της ποιότητας και της ασφάλειας των αποθετηρίων ανοιχτού λογισμικού για να διασφαλιστεί ότι πληρούν συγκεκριμένες ανάγκες και πρότυπα είναι μια επίπονη διαδικασία. Η χειρωνακτική φύση της όχι μόνο την καθιστά χρονοβόρα αλλά αυξάνει επίσης την ευαισθησία σε ανθρώπινο λάθος. Είναι εμφανής η ανάγκη για μια πιο αποτελεσματική και αυτοματοποιημένη λύση για την αξιολόγηση και την κατάταξη αποθετηρίων ανοιχτού λογισμικού με βάση την ποιότητα και την ασφάλεια τους.

Η προτεινόμενη εφαρμογή ιστού είναι ένα εργαλείο αξιολόγησης και αποτίμησης ποιότητας και ασφάλειας κώδικα για αποθετήρια λογισμικού ανοιχτού κώδικα που βρίσκονται στο Github. Η εφαρμογή μπορεί να λάβει τις διευθύνσεις των αποθετηρίων από τους χρήστες ενώ οι χρήστες μέσω της διεπαφής μπορούν να ορίσουν περιορισμούς και προτιμήσεις σχετικά με την αποτίμηση ως προς χαρακτηριστικά και μετρικές ποιότητας και ασφάλειας κώδικα. Κύριο πλεονέκτημα είναι ο αυτοματισμός της διαδικασίας καθώς το μόνο που χρειάζεται είναι η αρχική υποβολή των αποθετηρίων, προτιμήσεων και περιορισμών από τον χρήστη. Η εφαρμογή ανιχνεύει τις γλώσσες προγραμματισμού και τα σχετικά εργαλεία αυτοματοποίησης κατασκευής του κάθε απαιτούμενου αποθετηρίου καθώς και πραγματοποιεί την ανάλυση με αυτοματοποιημένο τρόπο, απλοποιώντας την διαδικασία και μειώνοντας το περιθώριο σφάλματος. Βασικό πλεονέκτημα αποτελεί και η ευελιξία της εφαρμογής στο κομμάτι της αποτίμησης. Δίνοντας στον χρήστη την δυνατότητα να μπορεί να ορίσει την βαρύτητα που επιθυμεί σε συγκεκριμένα χαρακτηριστικά ποιότητας και ασφάλειας καθώς και περιορισμούς πάνω σε αυτά, προσαρμόζεται η αξιολόγηση στο να ευθυγραμμίζεται καλύτερα με τους στόχους του χρήστη. Τα αποτελέσματα της αποτίμησης δεν είναι στατικά, διότι δίνεται στον χρήστη η δυνατότητα για την on the fly αλλαγή περιορισμών και προτιμήσεων, καθώς και αναπαρέχονται τάχιστα διότι δεν χρειάζεται να γίνει εκ νέου η ανάλυση των σχετικών έργων (λογισμικού) παρά μόνο η ανανέωση των αποτελεσμάτων με βάση τις καινούργιες προτιμήσεις/περιορισμούς του χρήστη.

Λέξεις Κλειδιά: λογισμικό, κώδικας, αποθετήριο, ποιότητα, μετρική, ασφάλεια, αποτίμηση, αξιολόγηση, εφαρμογή ιστού, RESTful API

Summary

Software repository quality and security assessment is an intensive process as it has to guarantee that certain requirements and standards are met. Its manual nature not only makes it time-consuming but also increases the sensitivity in human errors. It is apparent that there is a need for a more effective and automated solution for the assessment and ranking of open-source software repositories based on their quality and security.

The web application proposed is a tool for assessing and ranking code quality and security for open-source software repositories that reside on Github. The application can receive the URLs of the repositories from the users while the latter, through its interface, can define constraints and preferences related to the assessment on the code quality and security characteristics and metrics. The main advantage is the automation of the assessment process as only the initial submission of repository URLs, user preferences and constraints is needed. The application detects the programming language and the related construction automation tools for every required repository while it realises the analysis through an automated manner, by simplifying the assessment process and reducing the error margin. A basic advantage is the flexibility of the application in the assessment. By giving to the user the ability to define the desired weight on certain quality and security characteristics as well as constraints on them, the assessment is adjusted to align with the user goals. The assessment results are not static, as user is given the possibility for on-the-fly changing his/her constraints and preferences, as they are rapidly resupplied due to the fact that there is no need to re-analyse the related software projects but only the assessment results need to be updated based on the user's modified constraints/preferences.

Keyword: software, code, repository, quality, metric, security, evaluation, assessment, wen application, RESTful API

1. Εισαγωγή

Στον συνεχώς αναπτυσσόμενο τομέα της ανάπτυξης λογισμικού, η αξιολόγηση της ποιότητας και της ασφάλειας του κώδικα παραμένει ένα κρίσιμο μέλημα. Η έλλειψη της μπορεί να οδηγήσει σε κρίσιμα τρωτά σημεία ασφαλείας και κακή απόδοση του λογισμικού, με αποτέλεσμα αυξημένο κόστος συντήρησης καθώς και νομικούς κινδύνους και κινδύνους συμμόρφωσης. Κύρια πρόκληση αποτελεί η πολύπλοκη, χρονοβόρα και χειροκίνητη διαδικασία αξιολόγησης συνόλου από κώδικες ανοιχτού λογισμικού που είναι επιρρεπής σε ανθρώπινα σφάλματα. Η αποτελεσματικότητα, η ακρίβεια και η ταχύτητα της αξιολόγησης που με τον εντοπισμό και τον μετριασμό πιθανών ελαττωμάτων και τρωτών σημείων διασφαλίζουν την αξιοπιστία και την ασφάλεια του λογισμικού αποτελούν θέματα συνεχούς ανησυχίας, τονίζοντας την επείγουσα ανάγκη για προοδευτικές λύσεις σε αυτόν τον τομέα.[1][2]

Η διπλωματική μέσω της ανάπτυξης διαδικτυακής εφαρμογής αντιμετωπίζει το συγκεκριμένο ζήτημα εισάγοντας έναν αυτοματοποιημένο και αποτελεσματικό μηχανισμό αξιολόγησης και κατάταξης αποθετηρίων ανοιχτού κώδικα. Βελτιώνει την διαδικασία διασφαλίζοντας ότι οι κώδικες πληρούν κατάλληλα κριτήρια ποιότητας και ασφάλειας. Οι χρήστες μπορούν να εισαγάγουν τις διευθύνσεις των αποθετηρίων καθώς και να ορίσουν προτιμήσεις και περιορισμούς σχετικά με την αξιολόγηση των χαρακτηριστικών και των μετρήσεων ποιότητας και ασφάλειας κώδικα. Η αυτοματοποιημένη διαδικασία, που ενισχύεται από την ευελιξία της, διασφαλίζει ότι οι αξιολογήσεις ευθυγραμμίζονται με τους συγκεκριμένους στόχους των χρηστών, μετριάζοντας τα σφάλματα και βελτιώνοντας την αποτελεσματικότητα και ακρίβεια. Τα αποτελέσματα είναι δυναμικά, επιτρέποντας προσαρμογές σε πραγματικό χρόνο στις προτιμήσεις και τους περιορισμούς, προωθώντας μια προσέγγιση με επίκεντρο τον χρήστη στην αξιολόγηση κώδικα.

Η υπόλοιπη αναφορά της διπλωματικής δομείται ως εξής. Στο 1ο Κεφάλαιο θα δούμε τα κύρια χαρακτηριστικά της τεχνολογίας λογισμικού, τους τομείς εφαρμογής της και τις βασικές πτυχές ποιότητας και ασφάλειας. Στο 2ο Κεφάλαιο γίνεται ανάλυση παρόμοιων εργασιών πάνω στην αποτίμηση ποιότητας λογισμικού καθώς και σύγκριση μεθοδολογιών με την δική μας. Στο 3ο Κεφάλαιο αποτυπώνεται ο προσδιορισμός όλων των απαιτήσεων της εφαρμογής και παρέχονται διαγράμματα αρχιτεκτονικής και σχεδίασης καθώς και προσδιορίζονται βασικές λεπτομέρειες της σχετικής υλοποίησης. Στο 4ο Κεφάλαιο έχουμε έναν πρακτικό οδηγό που περιγράφει λεπτομερώς τις διαδικασίες εγκατάστασης της εφαρμογής βήμα προς βήμα καθώς και σενάρια βασικής χρήσης της. Τέλος το τελευταίο κεφάλαιο παρουσιάζει μια σύνοψη της κύριας συνεισφοράς της διπλωματικής καθώς και μελλοντικές κατευθύνσεις επέκτασης και βελτίωσης της συνεισφοράς αυτής.

2. Υπόβαθρο

2.1. Λογισμικό

Το λογισμικό αναφέρεται στη συλλογή προγραμμάτων και πληροφοριών που κατευθύνουν έναν υπολογιστή σχετικά με τον τρόπο εκτέλεσης συγκεκριμένων εργασιών. Λειτουργεί ως μεσολαβητής μεταξύ του χρήστη και του υλικού του υπολογιστή. Ο όρος "λογισμικό" χρησιμοποιείται για τη διάκριση μεταξύ των φυσικών πτυχών ενός υπολογιστή (hardware) και των άυλων στοιχείων που περιλαμβάνουν προγράμματα και δεδομένα. Στην ουσία, το λογισμικό αποτελείται από κωδικοποιημένες οδηγίες που καθοδηγούν το υλικό για να ολοκληρώσει μια ποικιλία εργασιών, μετατρέποντας ένα πολύπλοκο κομμάτι μηχανήματος σε ένα λειτουργικό και διαδραστικό εργαλείο.

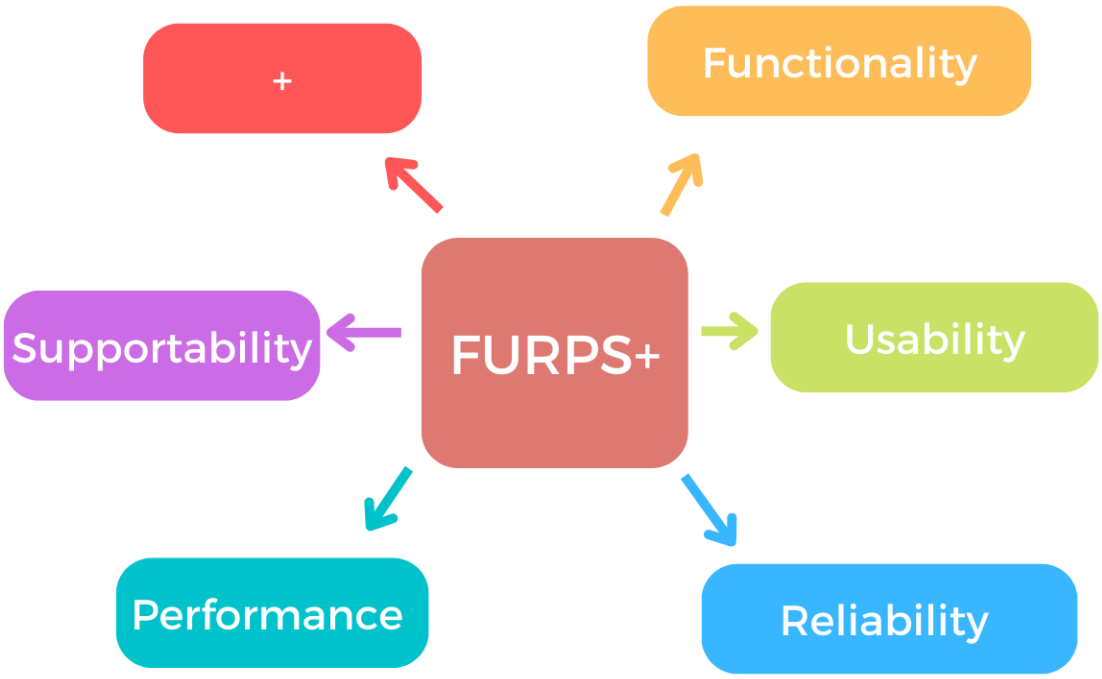
Το λογισμικό μπορεί να χωριστεί στις εξής κατηγορίες:

- **Λογισμικό εφαρμογών (Application software):** Επιτρέπει στους χρήστες να εκτελούν συγκεκριμένες εργασίες σε έναν υπολογιστή, επεξεργάζοντας δεδομένα σύμφωνα με τις εντολές που παρέχονται από τον χρήστη. Περιλαμβάνει μια ποικιλία προγραμμάτων, όπως επεξεργαστές κειμένου, υπολογιστικά φύλλα, και συστήματα διαχείρισης βάσεων δεδομένων μεταξύ άλλων. Κάθε εφαρμογή έχει σχεδιαστεί για να βοηθά τον χρήστη στην ολοκλήρωση συγκεκριμένων εργασιών, οδηγώντας σε μια πιο διαδραστική και παραγωγική εμπειρία με το σύστημα του υπολογιστή.
- **Λογισμικό συστήματος (System software):** Ελέγχει τις λειτουργίες υλικού και παρέχει βασικές λειτουργίες που χρειάζονται οι χρήστες και άλλο λογισμικό. Λειτουργεί ως θεμέλιο που υποστηρίζει και επιτρέπει τη λειτουργία του λογισμικού εφαρμογών. Ουσιαστικά, το λογισμικό συστήματος διασφαλίζει ότι το υλικό και οι εφαρμογές χρήστη λειτουργούν μαζί, διευκολύνοντας την αλληλεπίδραση με τον χρήστη και την εκτέλεση διαφόρων εργασιών. Ενδεικτικά περιλαμβάνει λειτουργικά συστήματα (operating systems), οδηγούς συσκευών (device drivers), προγράμματα ωφέλειας (utility programs) κα. [3]

2.1.1. Χαρακτηριστικά λογισμικού

Ο όρος για τα χαρακτηριστικά λογισμικού αναφέρεται συχνά ως "χαρακτηριστικά ποιότητας λογισμικού" (software quality attributes) ή "παράγοντες ποιότητας λογισμικού" (software quality factors). Αυτά τα χαρακτηριστικά είναι ουσιαστικά στοιχεία που συμβάλλουν στη συνολική ποιότητα, χρηστικότητα και απόδοση ενός προϊόντος λογισμικού. Κάθε χαρακτηριστικό αντιπροσωπεύει μια συγκεκριμένη πτυχή της συμπεριφοράς ή της απόδοσης του λογισμικού που μπορεί να μετρηθεί ή να αξιολογηθεί για να προσδιοριστεί το επίπεδο ποιότητας του λογισμικού.

Χωρίζονται ευρέως σε έξι κύριες κατηγορίες, οι οποίες αναφέρονται συχνά στο πλαίσιο του μοντέλου "FURPS+"



Εικόνα 1. FURPS+ μοντέλο

1. **Λειτουργικότητα (Functionality):** η ικανότητα του λογισμικού να εκτελεί τις εργασίες ή τις λειτουργίες για τις οποίες έχει σχεδιαστεί. Περιλαμβάνει τις δυνατότητες και τις συμπεριφορές που επιτρέπουν στους χρήστες να επιτύχουν τους στόχους τους χρησιμοποιώντας το λογισμικό. Το λογισμικό θα πρέπει να πληροί τις καθορισμένες απαιτήσεις και να παρέχει την επιθυμητή λειτουργικότητα για την αποτελεσματική αντιμετώπιση των αναγκών των χρηστών.
2. **Χρηστικότητα (Usability):** καθορίζει την αισθητική και την προσβασιμότητα καθώς και τον τρόπο πρόσβασης και διαμόρφωσης της διεπαφής ανθρώπου-μηχανής (user-machine interface) του λογισμικού. Περιλαμβάνει ιδιότητες όπως:
    - Αποδοτικότητα χρήσης (Efficiency of use): πόσο γρήγορα ένας έμπειρος χρήστης μπορεί να ολοκληρώσει εργασίες.
    - Κατανόηση (Understandability): αβίαστη κατανόηση της αρχιτεκτονικής και της πλοήγησης του ιστότοπου / εφαρμογής / λογισμικού.
    - Δυνατότητα εκμάθησης (Learnability): πόσο γρήγορα ένας χρήστης που δεν έχει ξαναδεί τη διεπαφή (του λογισμικού) μπορεί να ολοκληρώσει βασικές εργασίες.
3. **Αξιοπιστία (Reliability):** διασφαλίζει την ικανότητα του λογισμικού να λειτουργεί με συνέπεια και ακρίβεια κάτω από διάφορες συνθήκες. Το αξιόπιστο λογισμικό ελαχιστοποιεί την εμφάνιση σφαλμάτων, λαθών ή απροσδόκητης συμπεριφοράς. Οι χρήστες βασίζονται στο λογισμικό για να λειτουργήσει όπως προβλέπεται ενώ τυχόν απροσδόκητος χρόνος διακοπής λειτουργίας ή απώλεια δεδομένων μπορεί να έχει σοβαρές συνέπειες.
4. **Απόδοση (Performance):** περιλαμβάνει τις εξής ιδιότητες:

- Ταχύτητα (Speed): ανταπόκριση του λογισμικού, υποδεικνύοντας πόσο γρήγορα μπορεί να ολοκληρώσει εργασίες ή να επεξεργαστεί δεδομένα. Επηρεάζει άμεσα την ικανοποίηση των χρηστών, καθώς οι χρήστες προτιμούν συνήθως λογισμικό που παρέχει γρήγορα και έγκαιρα αποτελέσματα.
- Αποδοτικότητα (Efficiency): βέλτιστη χρήση των πόρων του συστήματος, συμπεριλαμβανομένης της μνήμης, της CPU και της αποθήκευσης, διασφαλίζοντας ότι το λογισμικό παρέχει τη μέγιστη απόδοση με ελάχιστη χρήση πόρων

5. **Δυνατότητα υποστήριξης (Supportability):** περιλαμβάνει τις εξής ιδιότητες:
    - Συντηρησιμότητα (Maintainability): η ευκολία με την οποία το λογισμικό μπορεί να τροποποιηθεί για να διορθώσει ελαττώματα, να βελτιώσει την απόδοση ή να προσαρμοστεί σε ένα τροποποιημένο περιβάλλον. Διασφαλίζει ότι το λογισμικό μπορεί να προσαρμοστεί στις μεταβαλλόμενες ανάγκες και περιβάλλοντα καθώς και να παρατείνει τη διάρκεια ζωής του.
    - Επεκτασιμότητα (Extensibility): η δυνατότητα του λογισμικού να προσθέτει λειτουργικότητα χωρίς να βλάπτει τη δομή του συστήματος. Το λογισμικό μπορεί να εξελιχθεί και να προσαρμοστεί στις αναδυόμενες ανάγκες και ευκαιρίες, ενισχύοντας τη μακροπρόθεσμη αξία και τη χρησιμότητά του για τους χρήστες. [4][5]

2.2. Ποιότητα λογισμικού

Σε αυτό το πλαίσιο, η ποιότητα του λογισμικού αναδεικνύεται ως βασική πτυχή. Η κακή ποιότητα λογισμικού μπορεί να έχει εκτεταμένες συνέπειες, επηρεάζοντας κάθε πτυχή μιας επιχείρησης, από την ικανοποίηση των χρηστών και τη αποτελεσματικότητα έως την οικονομική απόδοση και τη στρατηγική τοποθέτηση. Η προληπτική αντιμετώπιση της ποιότητας λογισμικού είναι κρίσιμη για τον μετριασμό αυτών των δυσμενών επιπτώσεων και τη διασφάλιση της παράδοσης αξιόπιστων, αποτελεσματικών και ασφαλών προϊόντων λογισμικού. [6]
Στην ανάπτυξη λογισμικού, η ποιότητα αναφέρεται σε δύο σχετικές αλλά διακριτές έννοιες:

- Λειτουργική ποιότητα λογισμικού: υποδεικνύει τον βαθμό στον οποίο το λογισμικό συμμορφώνεται με τον καθορισμένο σχεδιασμό του και πληροί τις περιγραφόμενες λειτουργικές απαιτήσεις ή προδιαγραφές. Μπορεί να εξισωθεί με την αποτελεσματικότητα του λογισμικού στην εκπλήρωση του επιδιωκόμενου σκοπού του ή την ανταγωνιστικότητά του σε σύγκριση με παρόμοια προϊόντα στην αγορά. Ουσιαστικά, η λειτουργική ποιότητα αξιολογεί εάν το αναπτυγμένο λογισμικό ευθυγραμμίζεται με ακρίβεια με τις καθορισμένες απαιτήσεις και παρέχει τα αναμενόμενα αποτελέσματα.[7]
- Δομική ποιότητα λογισμικού: σχετίζεται με την εκπλήρωση μη λειτουργικών απαιτήσεων που ενισχύουν την παροχή λειτουργικών αναγκών, συμπεριλαμβανομένων πτυχών όπως η ανθεκτικότητα και η συντηρησιμότητα. Ουσιαστικά αξιολογεί τον βαθμό στον οποίο το λογισμικό λειτουργεί αποτελεσματικά και πληροί τα αναμενόμενα πρότυπα απόδοσης, διασφαλίζοντας όχι μόνο ότι το λογισμικό κάνει αυτό που υποτίθεται ότι πρέπει να κάνει, αλλά ότι το κάνει αξιόπιστα και βιώσιμα.

2.2.1. Αξιολόγηση ποιότητας λογισμικού

Οι δομικές ιδιότητες, όπως η χρηστικότητα, μπορούν να αξιολογηθούν μόνο δυναμικά (πχ. με τους χρήστες αλληλεπιδρούν με το λογισμικό). Άλλες πτυχές, όπως η αξιοπιστία, ενδέχεται να περιλαμβάνουν όχι μόνο το λογισμικό αλλά και το υποκείμενο υλικό, επομένως, μπορούν να αξιολογηθούν τόσο στατικά όσο και δυναμικά (stress test). Η λειτουργική ποιότητα συνήθως αξιολογείται δυναμικά, αλλά είναι επίσης δυνατή η χρήση επιθεωρήσεων λογισμικού (software reviews).
Ιστορικά, η δομή, η ταξινόμηση και η ορολογία των χαρακτηριστικών και των μετρήσεων που ισχύουν για τη διαχείριση ποιότητας λογισμικού έχουν προέλθει από το πρότυπο ISO 9126 [8] και το μεταγενέστερο πρότυπο ISO/IEC 25000 [9]. Με βάση αυτά τα μοντέλα το Consortium for IT Software Quality (CISQ) [10] έχει ορίσει πέντε βασικά επιθυμητά δομικά χαρακτηριστικά που απαιτούνται για ένα κομμάτι λογισμικού για την παροχή επιχειρηματικής αξίας:

- Αξιοπιστία
- Αποδοτικότητα
- Ασφάλεια
- Συντηρησιμότητα
- Μέγεθος

Εικόνα 2. ISO 9126

Η μέτρηση ποιότητας λογισμικού πραγματοποιείται ουσιαστικά μέσω δοκιμών, ωστόσο, οι δοκιμές δεν αρκούν. Σύμφωνα με μια μελέτη, οι προγραμματιστές είναι λιγότερο από 50%

αποτελεσματικοί στην εύρεση σφαλμάτων στο δικό τους λογισμικό. Επιπλέον, οι περισσότερες μορφές δοκιμών είναι μόνο 35% αποτελεσματικές. Αυτό καθιστά δύσκολο τον προσδιορισμό της ποιότητας. [11]

Η μέτρηση της ποιότητας λογισμικού αφορά τον ποσοτικό προσδιορισμό σε ποιο βαθμό ένα σύστημα ή λογισμικό διαθέτει επιθυμητά χαρακτηριστικά. Αυτό μπορεί να πραγματοποιηθεί με ποιοτικά ή ποσοτικά μέσα ή με συνδυασμό και των δύο. Και στις δύο περιπτώσεις, για κάθε επιθυμητό χαρακτηριστικό, υπάρχει ένα σύνολο μετρήσιμων χαρακτηριστικών, δηλαδή μετρικών (metrics), η ύπαρξη των οποίων σε ένα κομμάτι λογισμικού ή συστήματος τείνει να συσχετίζεται με αυτό το χαρακτηριστικό.

Αξιοπιστία (Reliability): Οι κυριότερες αιτίες της χαμηλής αξιοπιστίας εντοπίζονται σε συνδυασμό μη συμμόρφωσης με καλές αρχιτεκτονικές πρακτικές. Αυτή η μη συμμόρφωση μπορεί να εντοπιστεί με τη μέτρηση των χαρακτηριστικών στατικής ποιότητας μιας εφαρμογής. Η αξιολόγηση των στατικών χαρακτηριστικών στα οποία βασίζεται η αξιοπιστία μιας εφαρμογής παρέχει μια εκτίμηση του επιπέδου επιχειρηματικού κινδύνου και της πιθανότητας πιθανών αποτυχιών και ελαττωμάτων που θα αντιμετωπίσει η εφαρμογή όταν τεθεί σε λειτουργία. Τα κυριότερα χαρακτηριστικά που θα πρέπει να γίνουν έλεγχοι είναι:

- Πρακτικές αρχιτεκτονικής
- Πολυπλοκότητα αλγορίθμων
- Πολυπλοκότητα προγραμματιστικών πρακτικών
- Αναλογία επαναχρησιμοποίησης μοτίβων
- Διαχείριση σφαλμάτων και εξαιρέσεων
- Διαχείριση πόρων
- Αποφυγή μοτίβων που θα οδηγήσουν σε απροσδόκητες συμπεριφορές

Αποδοτικότητα (Efficiency): Όπως και με την αξιοπιστία, οι αιτίες της αναποτελεσματικότητας της απόδοσης εντοπίζονται συχνά σε παραβιάσεις της ορθής αρχιτεκτονικής και βέλτιστων πρακτικών προγραμματισμού (coding practices), οι οποίες μπορούν να εντοπιστούν με τη μέτρηση των χαρακτηριστικών στατικής ποιότητας μιας εφαρμογής. Αυτά τα στατικά χαρακτηριστικά προβλέπουν πιθανά σημεία συμφόρησης λειτουργικών επιδόσεων και μελλοντικά προβλήματα επεκτασιμότητας, ειδικά για εφαρμογές που απαιτούν υψηλή ταχύτητα εκτέλεσης για το χειρισμό πολύπλοκων αλγορίθμων ή τεράστιους όγκους δεδομένων. Η αξιολόγηση της απόδοσης απαιτεί τον έλεγχο τουλάχιστον των ακόλουθων χαρακτηριστικών:

- Πρακτικές Αρχιτεκτονικής
- Κατάλληλες αλληλεπιδράσεις με ακριβούς και/ή απομακρυσμένους πόρους
- Απόδοση πρόσβασης δεδομένων και διαχείριση δεδομένων
- Διαχείριση μνήμης, δικτύου και χώρου στο δίσκο
- Συμμόρφωση με καλές προγραμματιστικές πρακτικές [12]

Ασφάλεια (Security): Η ποιότητα του λογισμικού περιλαμβάνει την ασφάλεια του λογισμικού. Η ελλιπής ασφάλεια λογισμικού μπορεί να προκύψει από έναν συνδυασμό παραγόντων, συμπεριλαμβανομένου του ανεπαρκούς σχεδιασμού, των ανεπαρκών δοκιμών και της έλλειψης ενημέρωσης σχετικά με πιθανές απειλές ασφαλείας. Ένα σύστημα λογισμικού είναι επιρρεπές σε ευπάθειες που μπορούν να χρησιμοποιηθούν από κακόβουλες οντότητες, οδηγώντας σε μη εξουσιοδοτημένη πρόσβαση, παραβιάσεις δεδομένων και άλλα συμβάντα ασφαλείας. Αυτή η

επισφαλής κατάσταση ασφάλειας δεν περιορίζεται στα εγγενή τρωτά σημεία του κώδικα, αλλά επεκτείνεται σε σφάλματα διαμόρφωσης, ξεπερασμένα (deprecated) στοιχεία λογισμικού και αδυναμίες στο περιβάλλον ανάπτυξης. Η αξιολόγηση της ασφάλειας απαιτεί τον έλεγχο τουλάχιστον των ακόλουθων χαρακτηριστικών:

- Διαχείριση μιας διαδικασίας ανάπτυξης με επίγνωση της ασφάλειας
- Πρακτικές Ασφαλούς Αρχιτεκτονικής Εφαρμογών
- Βέλτιστες πρακτικές ασφάλειας
- Ασφαλείς και καλές πρακτικές προγραμματισμού [13]
- Χειρισμός σφαλμάτων & εξαιρέσεων

Συντηρησιμότητα (Maintainability): Τα προβλήματα συντηρησιμότητας συνήθως εντοπίζονται στη φάση ανάπτυξης, όπου ορισμένες αποφάσεις και πρακτικές θέτουν τις βάσεις για μελλοντικές προκλήσεις, όσον αφορά την ενημέρωση, την επιδιόρθωση και τη βελτίωση του λογισμικού. Ενσωματώνει τις δυσκολίες και τις πολυπλοκότητες που συναντώνται κατά την προσπάθεια τροποποίησης του λογισμικού για τη διόρθωση σφαλμάτων, τη βελτίωση της απόδοσης ή την προσαρμογή σε αλλαγές στο περιβάλλον και τις απαιτήσεις.[14]

Η αξιολόγηση της συντηρησιμότητας απαιτεί τον έλεγχο τουλάχιστον των ακόλουθων χαρακτηριστικών:

- Πρακτικές Αρχιτεκτονικής Εφαρμογών
- Αναγνωσιμότητα κώδικα
- Code Smells
- Επίπεδο πολυπλοκότητας συναλλαγών
- Πολυπλοκότητα αλγορίθμων
- Πολυπλοκότητα προγραμματιστικών πρακτικών
- Αναλογία επαναχρησιμοποίησης μοτίβων
- Ελεγχόμενο επίπεδο δυναμικής κωδικοποίησης
- Ποσοστό σύζευξης (Coupling ratio)
- Τεκμηρίωση
- Υλικό, λειτουργικό σύστημα, ενδιάμεσο λογισμικό, στοιχεία λογισμικού και ανεξαρτησία βάσης δεδομένων
- Φορητότητα (Portability)
- Αναπαραγωγή σε διπλότυπα (Duplication)

Μέγεθος (Size): Η μέτρηση του μεγέθους λογισμικού απαιτεί τη σωστή συλλογή ολόκληρου του πηγαίου κώδικα, συμπεριλαμβανομένων σεναρίων δομής βάσης δεδομένων, πηγαίο κώδικα χειρισμού δεδομένων, κεφαλίδων στοιχείων, αρχείων διαμόρφωσης κ.λπ. Υπάρχουν ουσιαστικά δύο τύποι μεγεθών λογισμικού που πρέπει να μετρηθούν, το τεχνικό μέγεθος (footprint) και το λειτουργικό μέγεθος:

- Η πιο συνηθισμένη τεχνική προσδιορισμού μεγέθους είναι ο Αριθμός Γραμμών Κώδικα (Lines of Code Number) (#LOC) ανά τεχνολογία, αριθμό αρχείων, συναρτήσεων, κλάσεων, πινάκων κ.λπ., από τους οποίους μπορούν να υπολογιστούν τα σημεία συνάρτησης (function points) με αντίστροφο αποτέλεσμα.

- Η πιο κοινή μέθοδος για τη μέτρηση του λειτουργικού μεγέθους είναι η ανάλυση σημείων συνάρτησης (function points) [15]. Εστιάζοντας στις δυνατότητες και τις λειτουργίες στις οποίες μπορεί να έχει πρόσβαση και να χρησιμοποιήσει ένας χρήστης, αυτή η μέτρηση μπορεί να προσδιορίσει με ακρίβεια την πολυπλοκότητα μιας εφαρμογής. Είναι ιδιαίτερα χρήσιμη όταν συγκρίνετε έργα διαφορετικών μεγεθών. Παρέχει μια συνεπή μέθοδο για την αξιολόγηση και τη μέτρηση του εύρους του έργου, αντί να αιτείται από τους προγραμματιστές να εκτιμήσουν τον αριθμό των γραμμών κώδικα που απαιτεί κάθε έργο. Επιπλέον, επειδή λαμβάνουν υπόψη τις διαφορές μεταξύ των γλωσσών προγραμματισμού, τα σημεία συνάρτησης είναι ιδιαίτερα ωφέλιμα κατά την ανάπτυξη ή τις αλλαγές σχεδιασμού. Όχι μόνο επιτρέπουν συγκρίσεις μεταξύ συστημάτων διαφορετικών μεγεθών και δομών, αλλά παρέχουν επίσης ένα ευρύ φάσμα άλλων πλεονεκτημάτων, όπως ο εντοπισμός σημείων συμφόρησης στην ανάπτυξη ή η επισήμανση διαφορών στις προδιαγραφές των πελατών. [16]

2.3. Στατική ανάλυση κώδικα

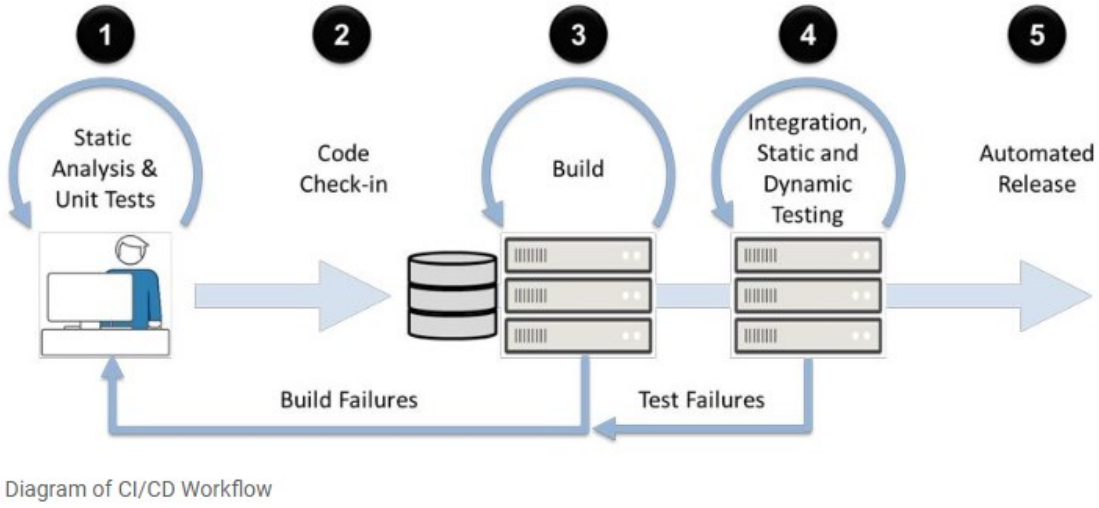
Η στατική ανάλυση κώδικα είναι μια ζωτική πτυχή της σύγχρονης ανάπτυξης λογισμικού, που χαρακτηρίζεται από την ικανότητά να αξιολογεί και να ελέγχει με αυτοματοποιημένο τρόπο την ποιότητα του κώδικα χωρίς την ανάγκη εκτέλεσης. Είναι παρόμοια με μια σχολαστική διαδικασία διόρθωσης, όπου ο κώδικας εξετάζεται συστηματικά για να βρεθούν σφάλματα, παραβιάσεις ασφάλειας και δομικά ζητήματα, μεταξύ άλλων, πολύ πριν από την εκτέλεση της εφαρμογής (runtime). Αυτή η ανάλυση είναι καθοριστικής σημασίας όχι μόνο για τον εντοπισμό των προβλημάτων, αλλά και για την προσφορά αξιόπιστων πληροφοριών για την επίλυσή τους, βελτιώνοντας έτσι τη συνολική ποιότητα, ασφάλεια και απόδοση του κώδικα.

Στον πυρήνα της, η στατική ανάλυση στοχεύει στον εντοπισμό μιας πληθώρας ζητημάτων, συμπεριλαμβανομένων, ενδεικτικά, των σφαλμάτων προγραμματισμού, των τρωτών σημείων ασφαλείας και της τήρησης των βέλτιστων πρακτικών προγραμματισμού. Στο πλαίσιο των σφαλμάτων προγραμματισμού, αποκαλύπτει πολύπλοκα σφάλματα που θα μπορούσαν ενδεχομένως να οδηγήσουν σε σφάλματα συστήματος ή απρόσμενη συμπεριφορά, δίνοντας στους προγραμματιστές την ευκαιρία να τα αντιμετωπίσουν. Επιπλέον, η επιβολή των βέλτιστων πρακτικών προγραμματισμού διασφαλίζει ότι ο κώδικας συμμορφώνεται με ένα προκαθορισμένο σύνολο προτύπων ενώ ενισχύει ένα περιβάλλον συνέπειας, αναγνωσιμότητας και δυνατότητας συντήρησης. Αυτό όχι μόνο διευκολύνει ένα περιβάλλον συνεργατικής ανάπτυξης, αλλά περιορίζει επίσης σημαντικά την εισαγωγή σφαλμάτων. [17]

Η ενσωμάτωση στατικής ανάλυσης κώδικα σε αγωγούς CI/CD[1] (CI/CD pipelines) ενισχύει την αποτελεσματικότητά του. Κάθε commit ή pull request ενεργοποιεί μια αυτοματοποιημένη στατική ανάλυση, διασφαλίζοντας ότι ο κώδικας ελέγχεται και επικυρώνεται με βάση προκαθορισμένα σημεία αναφοράς ποιότητας προτού προχωρήσει περαιτέρω στη γραμμή ανάπτυξης. Αυτή η συνεχής και αυτοματοποιημένη επικύρωση διασφαλίζει ότι ο κώδικας δεν είναι απλώς λειτουργικά σωστός, αλλά είναι επίσης βελτιστοποιημένος, ασφαλής και συμμορφώνεται με τα σωστά πρότυπα.

[1] CI/CD σημαίνει Continuous Integration/Continuous Delivery δηλαδή Συνεχής Ενοποίηση / Συνεχής Παράδοση

Οι πύλες ποιότητας (quality gates), ένα βασικό στοιχείο της ενοποίησης CI/CD, διασφαλίζουν ότι ο κώδικας πληροί ένα συγκεκριμένο σύνολο κριτηρίων προτού κριθεί κατάλληλος για παραγωγή. Αυτά τα κριτήρια, που συχνά περιλαμβάνουν την ασφάλεια, τις επιδόσεις και τα δομικά σημεία αναφοράς, λειτουργούν ως μια γραμμή άμυνας, διασφαλίζοντας ότι μόνο ποιοτικός κώδικας οδηγείται στην παραγωγή.



Εικόνα 3. Στατική ανάλυση κώδικα σε αγωγούς CI/CD

Η στατική ανάλυση κώδικα δεν είναι χωρίς περιορισμούς, ιδιαίτερα στον τομέα της ασφάλειας. Συχνά δυσκολεύεται να εντοπίσει περίπλοκα τρωτά σημεία, όπως ζητήματα ελέγχου ταυτότητας (authentication), ελέγχου πρόσβασης (access control) και αδύναμης κρυπτογραφίας (weak cryptography). Η πολυπλοκότητα αυτών των πτυχών ασφαλείας καθιστά ένα σημαντικό μέρος αόριστο στην αυτοματοποιημένη ανίχνευση. Επομένως, αυτά τα εργαλεία αντιμετωπίζουν προκλήσεις από υψηλά περιστατικά ψευδών θετικών στοιχείων, θολώνοντας την ανάλυση και οδηγούν σε απλή επίβλεψη κρίσιμων ζητημάτων ή περιττή εστίαση σε μη ζητήματα. Τα ζητήματα διαμόρφωσης (configuration issues), επίσης, δεν καλύπτονται καθώς συχνά δεν ενσωματώνονται στον κώδικα και, ως εκ τούτου, παραμένουν απαρατήρητα. Τα εργαλεία μπορούν να εντοπίσουν πιθανές ευπάθειες, αλλά η επικύρωσή τους ως πραγματικές απειλές απαιτεί πρόσθετα, συχνά χειροκίνητα, βήματα. Επιπλέον, η αποτελεσματικότητα της ανάλυσης στατικού κώδικα περιορίζεται σημαντικά όταν ασχολούμαστε με κώδικα που δεν μπορεί να μεταγλωττιστεί. Οι προγραμματιστές συχνά συναντούν εμπόδια, όπως την απουσία βασικών βιβλιοθηκών, ολοκληρωμένων οδηγιών μεταγλώττισης ή πλήρους κώδικα, εμποδίζοντας μια ενδελεχή ανάλυση και δυνητικά αφήνοντας τα τρωτά σημεία απαρατήρητα. [18]

Συμπερασματικά, η στατική ανάλυση κώδικα αναδεικνύεται ως ένας απαραίτητος σύμμαχος στην αναζήτηση άψογης ποιότητας κώδικα. Προσφέροντας έναν σχολαστικό, συστηματικό και αυτοματοποιημένο μηχανισμό για τον εντοπισμό και τη διόρθωση ζητημάτων στα πρώτα στάδια ανάπτυξης, μειώνει σημαντικά τον κίνδυνο, το κόστος και την προσπάθεια που σχετίζονται με σφάλματα και τρωτά σημεία. Σε μια εποχή όπου η ασφάλεια, η απόδοση και η ποιότητα είναι καθοριστικής σημασίας, η ανάλυση στατικού κώδικα αποτελεί προπύργιο, διασφαλίζοντας ότι αυτά τα κρίσιμα χαρακτηριστικά είναι ενσωματωμένα στον κώδικα από την αρχή έως την ανάπτυξη.

2.4. Δυναμική ανάλυση κώδικα

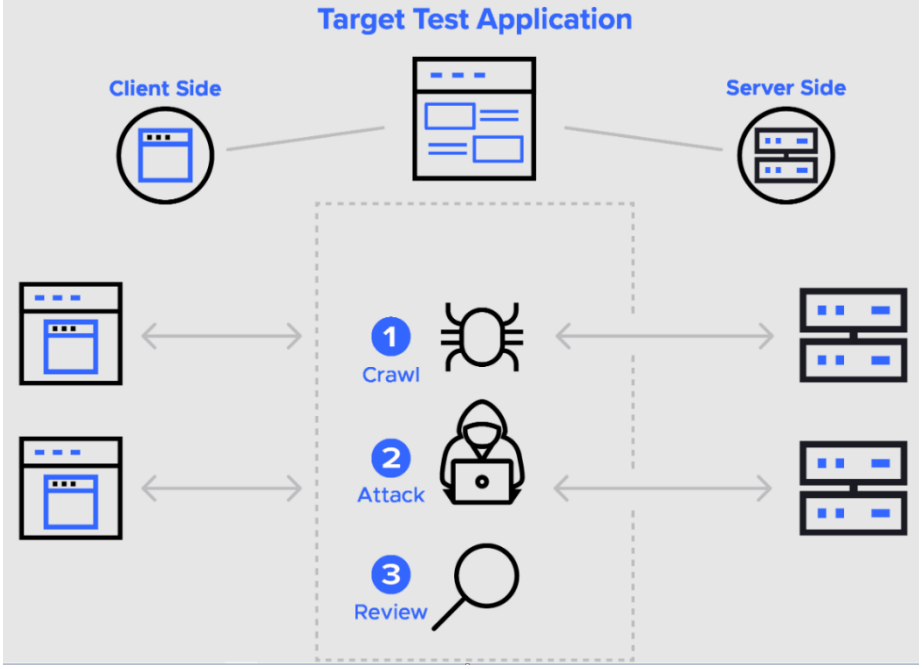
Η δυναμική ανάλυση κώδικα αποτελεί αναπόσπαστο μέρος της δοκιμής λογισμικού και της διασφάλισης ποιότητας, που διακρίνεται από την προσέγγισή της στην ανάλυση και αξιολόγηση του λογισμικού κατά την εκτέλεσή του. Το λογισμικό, αντί να είναι μια στατική οντότητα, εκτελείται ενεργά, προσφέροντας πληροφορίες για τη συμπεριφορά, την απόδοσή του και τα πιθανά ζητήματα σε πραγματικό χρόνο. Αυτή η μέθοδος είναι αποφασιστικής σημασίας για τον εντοπισμό προβλημάτων που δεν είναι εμφανή κατά τη στατική ανάλυση, όπως σφάλματα χρόνου εκτέλεσης, διαρροές μνήμης και ζητήματα που σχετίζονται με την αλληλεπίδραση με τον χρήστη.

Στον τομέα της δυναμικής ανάλυσης κώδικα, το λογισμικό υποβάλλεται σε ποικίλες δοκιμές και αξιολογήσεις για την κατανόηση της συμπεριφοράς του υπό διαφορετικές συνθήκες και εισόδους. Αυτή η πρακτική προσέγγιση είναι ιδιαίτερα ικανή στην αποκάλυψη ζητημάτων που εκδηλώνονται μόνο κατά την εκτέλεση, προσφέροντας μια ολοκληρωμένη εικόνα της απόδοσης και της σταθερότητας του λογισμικού. Εμβαθύνει σε πτυχές όπως η απόκριση του λογισμικού σε διάφορες εισροές, η συμπεριφορά του υπό πίεση και η διαχείριση των πόρων του, προσφέροντας έτσι μια πολυδιάστατη άποψη της ποιότητας και της αξιοπιστίας του. [19]

Η δοκιμή απόδοσης και φορτίου (performance and load testing) αξιολογεί τη συμπεριφορά του λογισμικού κάτω από διαφορετικά επίπεδα φορτίου και καταπόνησης. Βοηθά στον εντοπισμό των σημείων συμφόρησης, της υποβάθμισης της απόδοσης και των ορίων του λογισμικού, επιτρέποντας έτσι βελτιστοποιήσεις στην επεκτασιμότητα και την απόδοσή του. Αυτό είναι ζωτικής σημασίας για τη διασφάλιση ότι το λογισμικό μπορεί να χειριστεί αποτελεσματικά τις απαιτήσεις των χρηστών και να προσφέρει σταθερή απόδοση.

Η ενσωμάτωση της δοκιμής απόδοσης και φορτίου στον κύκλο ζωής ανάπτυξης του λογισμικού ενισχύει την αποτελεσματικότητα της δυναμικής ανάλυσης κώδικα. Υλοποιείται έτσι μια συνεχής διαδικασία όπου κάθε επανάληψη του λογισμικού δοκιμάζεται και αναλύεται, διασφαλίζοντας ότι οι αλλαγές που εισάγονται δεν επηρεάζουν αρνητικά τη συμπεριφορά και την απόδοσή του. Τα αυτοματοποιημένα εργαλεία δοκιμών και τα πλαίσια διαδραματίζουν κεντρικό ρόλο σε αυτήν την ενοποίηση, προσφέροντας αυτοματοποιημένες, επαναλαμβανόμενες και συνεπείς δοκιμές που επικυρώνουν τη συμπεριφορά και την απόδοση του λογισμικού σε σχέση με προκαθορισμένα σημεία αναφοράς. [20]

Οι (δυναμικές) δοκιμές ασφαλείας είναι μια από τις βασικές πτυχές της δυναμικής ανάλυσης κώδικα. Προσομοιώνοντας επιθέσεις και κακόβουλες συμπεριφορές, οι αναλυτές μπορούν να αξιολογήσουν τους αμυντικούς μηχανισμούς του λογισμικού, να εντοπίσουν τρωτά σημεία και να κατανοήσουν τον πιθανό αντίκτυπο των παραβιάσεων της ασφάλειας. Αυτό το σενάριο ζωντανής δοκιμής προσφέρει πληροφορίες όχι μόνο για την ύπαρξη τρωτών σημείων, αλλά και για την εκμεταλλευσιμότητα και τη σοβαρότητά τους, επιτρέποντας στοχευμένη και ιεραρχημένη αποκατάσταση. [21]



Εικόνα 4. Δυναμική δοκιμή ασφάλειας εφαρμογών

Η δυναμική ανάλυση κώδικα αναδεικνύεται ως σύμμαχος της στατικής ανάλυσης κώδικα και τη συμπληρώνει. Ενώ η τελευταία προσφέρει πρώιμες πληροφορίες για την ποιότητα του κώδικα και τα πιθανά ζητήματα, η δυναμική ανάλυση επικυρώνει και δοκιμάζει το λογισμικό σε ένα ζωντανό περιβάλλον, προσφέροντας πληροφορίες για τη συμπεριφορά, την απόδοση και την ασφάλειά του. Μαζί, προσφέρουν μια ολοκληρωμένη προσέγγιση για τη διασφάλιση ποιότητας λογισμικού, εξασφαλίζοντας ότι οι εφαρμογές δεν χρησιμοποιούν βέλτιστες πρακτικές προγραμματισμού, αλλά είναι επίσης αξιόπιστες, ασφαλείς και αποτελεσματικές κατά την εκτέλεση τους.

3. Σχετικές Εργασίες

3.1. Ερευνητικές Εργασίες

3.1.1. Squale Quality Model

Το Squale Quality Model είναι μια συστηματική προσέγγιση που βασίζεται σε καθιερωμένες πρακτικές για την αξιολόγηση της ποιότητας του λογισμικού. Χαρακτηρίζεται από τη δομημένη μεθοδολογία του, που εστιάζει σε συγκεκριμένα χαρακτηριστικά ποιότητας και προσφέρει αξιολογήσεις με βάση ένα σύνολο προκαθορισμένων κριτηρίων και μετρικών. Σε αυτό το μοντέλο είναι ενσωματωμένο ένα σύστημα αξιολόγησης, «mark», που ποσοτικοποιεί και κατηγοριοποιεί την ποιότητα του λογισμικού, παρέχοντας μια συνεπή αξιολόγηση. Αν και ισχυρό, το συγκεκριμένο μοντέλο ενδέχεται να μην έχει την ευελιξία προσαρμογής στις ποικίλες και δυναμικές ανάγκες διαφόρων έργων λογισμικού και περιβαλλόντων ανάπτυξης. [22]

Ενώ το Squale Quality Model προσφέρει δομημένες αξιολογήσεις, η προτεινόμενη εφαρμογή της τρέχουσας διπλωματικής εργασίας επιτρέπει στους χρήστες να προσαρμόζουν τις αξιολογήσεις με βάση τις συγκεκριμένες ανάγκες και προτιμήσεις τους. Οι χρήστες μπορούν να ορίσουν μοναδικούς περιορισμούς και να εκχωρήσουν διαφορετικά βάρη σε διάφορα χαρακτηριστικά ποιότητας και ασφάλειας κώδικα, διασφαλίζοντας μια προσαρμοσμένη αξιολόγηση που ευθυγραμμίζεται με τις συγκεκριμένες απαιτήσεις τους.

3.1.2. Free/Libre/Open Source Metrics (FLOSS)

Το FLOSS προσφέρει μια προοπτική βάσει δεδομένων για την αξιολόγηση του Ελεύθερου Λογισμικού Ανοικτού Κώδικα. Δίνει έμφαση στις ποσοτικές μετρήσεις και τα δεδομένα που συγκεντρώνονται σε μια βάση δεδομένων για την αξιολόγηση της ποιότητας και της απόδοσης των έργων. Αυτή η προσέγγιση είναι ολοκληρωμένη και βασίζεται σε δομημένες μετρήσεις, προσφέροντας πολύτιμες πληροφορίες για την ποιότητα και την αποτελεσματικότητα του λογισμικού ανοιχτού κώδικα. Ωστόσο, το FLOSS μπορεί να εκληφθεί ως στατικό, με την εστίασή του σε δεδομένα και μετρήσεις που ενδεχομένως δεν διαθέτουν τα δυναμικά στοιχεία που καλύπτουν τις εξελισσόμενες και συγκεκριμένες ανάγκες μεμονωμένων χρηστών και έργων. Η ανάλυση είναι περιεκτική, αλλά δεν διαθέτει τα διαδραστικά στοιχεία που επιτρέπουν την ανάδραση και την προσαρμογή σε πραγματικό χρόνο. [23]

Η προτεινόμενη εφαρμογή της διπλωματικής εργασίας έχει σχεδιαστεί για να είναι διαδραστική και προσαρμόσιμη. Οι χρήστες μπορούν να αλληλεπιδράσουν με το σύστημα σε πραγματικό χρόνο, προσαρμόζοντας τα κριτήριά τους και βλέποντας αμέσως τα ενημερωμένα αποτελέσματα αξιολόγησης. Αυτή η δυνατότητα είναι ιδιαίτερα ευεργετική για την προσαρμογή στις εξελισσόμενες ανάγκες των έργων λογισμικού, διασφαλίζοντας ότι η αξιολόγηση παραμένει σχετική και ενεργή εν μέσω μεταβαλλόμενων απαιτήσεων και περιβαλλόντων του έργου.

Τέλος, συγκριτικά και με τα δύο αυτά σχετικά έργα, η προτεινόμενη εφαρμογή υπερτερεί στο κομμάτι του αυτοματισμού απλοποιώντας κατά πολύ την όλη διαδικασία αξιολόγησης. Οι χρήστες απλώς υποβάλλουν τα αποθετήρια τους και το σύστημα ανιχνεύει τις γλώσσες προγραμματισμού και τα σχετικά εργαλεία αυτοματοποίησης κατασκευής του κάθε απαιτούμενου αποθετηρίου. Αυτό όχι μόνο απλοποιεί τη διαδικασία, αλλά μειώνει το περιθώριο για λάθη ενώ εξασφαλίζει ακριβή και προσαρμοσμένα αποτελέσματα. Αυτό το επίπεδο αυτοματισμού, ακρίβειας και διαδραστικότητας προσφέρει στους χρήστες μια πιο αποτελεσματική, ακριβή και φιλική προς τον χρήστη εμπειρία.

3.2. Εργαλεία ανάλυσης κώδικα/λογισμικού

3.2.1. Εργαλεία στατικής ανάλυσης

SonarQube

Το SonarQube[2] είναι ένα εργαλείο για συνεχή έλεγχο της ποιότητας κώδικα, που προσφέρει στους προγραμματιστές πολύτιμες πληροφορίες για τη σύνταξη καθαρότερου και ασφαλέστερου κώδικα. Σαρώνει τον πηγαίο κώδικα για σφάλματα, τρωτά σημεία και μυρωδιές κώδικα (code smells), παρέχοντας λεπτομερείς αναφορές και ανατροφοδότηση με δυνατότητα δράσης για την αντιμετώπιση εντοπισμένων προβλημάτων. Ενσωματώνεται σε CI/CD αγωγούς, όπου αναλύει αυτόματα τον κώδικα σε πραγματικό χρόνο κατά τη διαδικασία ανάπτυξης. Αυτό διασφαλίζει ότι κάθε κομμάτι κώδικα ελέγχεται εξονυχιστικά για ποιότητα και ασφάλεια προτού προχωρήσει περαιτέρω στη γραμμή ανάπτυξης, ενισχύοντας τη συνολική ακεραιότητα και αξιοπιστία των εφαρμογών λογισμικού.

Το SonarLint[3], αναπόσπαστο μέρος του οικοσυστήματος SonarQube, είναι ένας βοηθός ποιότητας κώδικα που οι προγραμματιστές μπορούν να χρησιμοποιήσουν απευθείας στα IDE τους. Παρέχει άμεση ανατροφοδότηση σχετικά με την ποιότητα του κώδικα καθώς οι προγραμματιστές γράφουν κώδικα, εντοπίζοντας και επισημαίνοντας προβλήματα σε πραγματικό χρόνο. Αυτή η άμεση ανατροφοδότηση επιτρέπει στους προγραμματιστές να αντιμετωπίσουν πιθανά προβλήματα βελτιώνοντας την ποιότητα του κώδικα από τα αρχικά στάδια ανάπτυξης.

Το SonarCloud[4], από την άλλη πλευρά, επεκτείνει τις δυνατότητες του SonarQube στο υπολογιστικό νέφος (cloud). Προσφέρει μια υπηρεσία που βασίζεται στο cloud που επικεντρώνεται στην αυτοματοποιημένη ανάλυση κώδικα, καθιστώντας την προσβάσιμη και βολική για τις ομάδες ανάπτυξης. Με το SonarCloud, οι ομάδες μπορούν να απολαμβάνουν τις ισχυρές δυνατότητες ανάλυσης κώδικα του SonarQube χωρίς την ανάγκη διαχείρισης υποδομής διακομιστή, καθιστώντας το μια προτιμώμενη επιλογή για ομάδες που αναζητούν ευελιξία και επεκτασιμότητα.

Μαζί, τα SonarQube, SonarLint και SonarCloud δημιουργούν μια τριάδα εργαλείων που αντιμετωπίζουν ολοκληρωμένα την ποιότητα και την ασφάλεια του κώδικα σε ολόκληρο τον κύκλο ζωής της ανάπτυξης. Από την άμεση ανατροφοδότηση σε πραγματικό χρόνο που παρέχεται από το SonarLint στο IDE, τις διεξοδικές και αυτοματοποιημένες επιθεωρήσεις κώδικα από το SonarQube στους CI/CD αγωγούς, μέχρι την ευελιξία και την επεκτασιμότητα που προσφέρει το SonarCloud για έργα που βασίζονται σε σύννεφο, αυτά τα εργαλεία διασφαλίζουν την ποιότητα του κώδικα και η ασφάλεια δεν ελέγχονται απλώς, αλλά ενσωματώνονται στον ιστό της διαδικασίας ανάπτυξης λογισμικού. Κάθε εργαλείο παίζει καθοριστικό ρόλο στη διασφάλιση ότι ο κώδικας δεν είναι μόνο λειτουργικός, αλλά πληροί τα υψηλότερα πρότυπα ποιότητας, ασφάλειας και συντηρησιμότητας.[24]

---

[2] https://www.sonarsource.com/products/sonarqube/
[3] https://www.sonarsource.com/products/sonarlint/
[4] https://www.sonarsource.com/products/sonarcloud/

Codacy

Το Codacy[5] είναι ένα εργαλείο ελέγχου κώδικα που αυτοματοποιεί τη διαδικασία ανάλυσης, προσφέροντας πληροφορίες σε πραγματικό χρόνο για τη βελτίωση της ποιότητας. Εντοπίζει προβλήματα και παρέχει συστάσεις για επιδιορθώσεις, βοηθώντας στη βελτίωση της ποιότητας και της ασφάλειας του κώδικα. Το εργαλείο υποστηρίζει πολλές γλώσσες προγραμματισμού και ενσωματώνεται με διάφορα περιβάλλοντα ανάπτυξης, προωθώντας την ευελιξία. Επιτρέπει στα μέλη της ομάδας να βλέπουν, να συζητούν και να αντιμετωπίζουν θέματα κώδικα συλλογικά, ενισχύοντας μια κουλτούρα κοινής ευθύνης και συνεχούς βελτίωσης. Οι οπτικές αναφορές προσφέρουν με μια ματιά την ποιότητα του κώδικα και την πρόοδο της ομάδας με την πάροδο του χρόνου. Η ανάλυση ασφαλείας είναι ενσωματωμένη στη λειτουργικότητα του Codacy, διασφαλίζοντας ότι τα τρωτά σημεία εντοπίζονται και αντιμετωπίζονται νωρίς στον κύκλο ζωής της ανάπτυξης.

Το Codacy ενσωματώνεται σε CI/CD αγωγούς, ενισχύοντας τις διαδικασίες συνεχούς ενοποίησης και ανάπτυξης και προσθέτοντας ένα αυτοματοποιημένο επίπεδο ποιότητας κώδικα και ελέγχων ασφαλείας. Το Codacy εκτελεί αυτόματα λεπτομερείς αναλύσεις, διασφαλίζοντας ότι ο κώδικας είναι ποιοτικός και ασφαλής. Αυτή η ενοποίηση διασφαλίζει ότι οι έλεγχοι κώδικα και οι έλεγχοι ασφαλείας δεν είναι μεμονωμένα συμβάντα, αλλά αναπόσπαστα μέρη ολόκληρου του κύκλου ζωής ανάπτυξης.

Σε πραγματικό χρόνο, το Codacy αξιολογεί τον κώδικα για πιθανά ζητήματα και τρωτά σημεία και παρέχει ανατροφοδότηση απευθείας εντός του CI/CD αγωγού. Αυτός ο άμεσος βρόχος ανάδρασης διασφαλίζει ότι τυχόν προβλήματα αντιμετωπίζονται έγκαιρα, μειώνοντας τον κίνδυνο ανάπτυξης κώδικα κακής ποιότητας ή χαμηλού επιπέδου ασφάλειας. Η συμπερίληψη του Codacy στον CI/CD αγωγό αναβαθμίζει τα πρότυπα ποιότητας και ασφάλειας κώδικα, διασφαλίζοντας ότι διατηρούνται με συνέπεια καθόλη την ανάπτυξη του λογισμικού. Αυτή η βελτιωμένη ενοποίηση βοηθά στην επίτευξη πιο αξιόπιστης, αποτελεσματικής και ασφαλούς παράδοσης προϊόντων λογισμικού. [25]

3.2.2. Εργαλεία δυναμικής ανάλυσης

OWASP ZAP

Το OWASP ZAP (Zed Attack Proxy)[6] αποτελεί ένα εργαλείο στον τομέα της ασφάλειας, ειδικά προσαρμοσμένο για τη δοκιμή της ασφάλειας εφαρμογών ιστού (web applications). Αυτό το εργαλείο ανοιχτού κώδικα είναι εξοπλισμένο με λειτουργίες που είναι ολοκληρωμένες για τους ειδικούς και προσβάσιμες για αρχάριους, καθιστώντας το μια ευέλικτη επιλογή για μια ποικιλία χρηστών.

Η διαδικασία ξεκινά όταν ένας χρήστης εισάγει μια διεύθυνση URL στο εργαλείο ZAP. Αυτό έπειτα εκκινεί, πραγματοποιώντας αυτοματοποιημένες σαρώσεις για τον εντοπισμό τρωτών σημείων ασφαλείας εντός της εφαρμογής ιστού. Αλλά η ικανότητά του δεν σταματά εκεί. δεν είναι μόνο ο εντοπισμός προβλημάτων. Το ZAP παρέχει λεπτομερείς αναφορές, για την φύση των τρωτών σημείων, τον πιθανό αντίκτυπό τους και προσφέροντας πληροφορίες για την αποκατάσταση για την ασφάλεια της εφαρμογής.

Στον κόσμο της συνεχούς ενοποίησης και ανάπτυξης (CI/CD), το ZAP αποδεικνύεται ένας τρομερός σύμμαχος. Ενσωματώνεται απρόσκοπτα σε CI/CD αγωγούς, διασφαλίζοντας ότι κάθε

[5] https://www.codacy.com/
[6] https://www.zaproxy.org/

κομμάτι κώδικα ελέγχεται για ευπάθειες ασφαλείας πριν προχωρήσει στη διαδικασία ανάπτυξης. Η ασφάλεια, στη σφαίρα του ZAP, δεν είναι ένα τελικό βήμα, αλλά μια συνεχής διαδικασία, υφασμένη στον ίδιο τον ιστό της ανάπτυξης.

Το OWASP ZAP δεν είναι απλώς ένα αυτόνομο εργαλείο, είναι μέρος ενός οικοσυστήματος. Με το API του, μπορεί να ενσωματωθεί ώστε να λειτουργεί αρμονικά με άλλα εργαλεία ασφάλειας και ανάπτυξης, βελτιώνοντας τη λειτουργικότητα και τη δυνατότητα εφαρμογής του. Προωθεί ένα περιβάλλον συνεργασίας όπου η ασφάλεια δεν είναι ευθύνη λίγων αλλά συλλογικός στόχος. Προσφέρει ένα περιβάλλον όπου οι δοκιμές ασφαλείας δεν είναι μια περίπλοκη, εσωτερική διαδικασία, αλλά ένα προσβάσιμο, περιεκτικό και αναπόσπαστο μέρος της ανάπτυξης εφαρμογών Ιστού. Κάθε δυνατότητα, από αυτοματοποιημένους σαρωτές έως λεπτομερείς αναφορές ευπάθειας, είναι προσαρμοσμένη ώστε να δίνει τη δυνατότητα στους χρήστες όχι απλώς να εντοπίζουν, αλλά να κατανοούν και να αντιμετωπίζουν ζητήματα ασφάλειας, καθιστώντας τις εφαρμογές Ιστού όχι μόνο λειτουργικές, αλλά και ασφαλείς. [26]

4. Ανάπτυξη Εφαρμογής

Η ανάπτυξη της εφαρμογής ακολούθησε το μοντέλο το καταρράκτη (waterfall model), όπου όλες οι δραστηριότητες ανάπτυξης εκτελούνται σειριακά, η μία μετά την άλλη. Στις επόμενες ενότητες του τρέχοντος κεφαλαίου θα αναλύσουμε την εργασία που επιτελέστηκε στις δραστηριότητες αυτές ενώ θα παρέχουμε στο τέλος έναν πίνακα που θα προσδιορίζει το βαθμό ικανοποίησης των απαιτήσεων, που εντοπίστηκαν και αναλύθηκαν, από την προτεινόμενη εφαρμογή ιστού.

4.1. Ανάλυση απαιτήσεων

4.1.1. Λειτουργικές Απαιτήσεις

- Το σύστημα θα πρέπει να επιτρέπει στον χρήστη να παρέχει μια λίστα από αποθετήρια προς υλοποίηση και προαιρετικά περιορισμούς στην ποιότητα του κώδικα και προτιμήσεις ως προς τα χαρακτηριστικά ποιότητας
    - Οι περιορισμοί θα εκφράζονται με την μορφή ανισοτήτων που συσχετίζουν ένα χαρακτηριστικό ή μετρική ποιότητας με ένα συγκεκριμένο άνω ή κάτω κατώφλι. Οι τελεστές που μπορούν να χρησιμοποιηθούν είναι οι: >, >=, <, <=, ==, <>. Τα κατώφλια θα πρέπει να εμπεριέχονται στα εύρη πεδίου τιμών των χαρακτηριστικών / μετρικών ποιότητας. Αν δεν δοθεί κάποιος περιορισμός, τότε η αρχική λίστα των αποθετηρίων δεν φιλτράρεται. Διαφορετικά, θα προκύπτει φιλτράρισμα που θα οδηγεί σε κατηγοριοποίηση των αποθετηρίων κώδικα σε 2 κατηγορίες: (α) αποθετήρια που ικανοποιούν πλήρως όλους τους περιορισμούς και (β) αποθετήρια που δεν ικανοποιούν τουλάχιστον έναν περιορισμό.
    - Οι προτιμήσεις μπορεί να εκφράζονται με ιεραρχικό τρόπο με βάση ένα δένδρο από χαρακτηριστικά/μετρικές ποιότητα και ασφάλειας κώδικα. Σε αυτή την περίπτωση, ο χρήστης θα πρέπει να δίνει σχετικά βάρη για κάθε κόμβο του δένδρου. Αν δεν δοθεί κάποια προτίμηση, τότε θεωρείται πως τα σχετικά βάρη είναι τα ίδια για όλα τα παιδιά-κόμβους σε ένα κόμβο-πατέρα. Επίσης, είναι καλό να ισχύει ο περιορισμός πως τα βάρη των παιδιών-κόμβων θα πρέπει να έχουν άθροισμα ίσο με 1.0 ενώ τα βάρη θα παίρνουν τιμές από το ακόλουθο εύρος πραγματικών αριθμών: [0.0, 1.0].
- Για την αποτίμηση του κάθε αποθετηρίου, θα πρέπει να εκφορτώνεται ο κώδικας του αποθετηρίου αυτού τοπικά στο σύστημα και έπειτα να εφαρμόζονται πάνω στον κώδικα ένα ή περισσότερα εργαλεία στατικής ανάλυσης ποιότητας και ασφάλειας κώδικα. Τα εργαλεία θα πρέπει να επιλεγούν με κατάλληλο τρόπο ώστε να παρέχουν συμπληρωματική γνώση αποτίμησης με την μορφή τιμών αποτίμησης για όλα τα χαρακτηριστικά/μετρικές ποιότητας & ασφάλειας που περιλαμβάνονται στο προαναφερόμενο δένδρο. Η γνώση αυτή θα πρέπει να συλλέγεται από το κάθε εργαλείο, να μορφοποιείται σε κατάλληλο μορφή και έπειτα θα χρησιμοποιείται για το φιλτράρισμα και την ταξινόμηση των αποθετηρίων.
    - Το φιλτράρισμα θα πραγματοποιείται ελέγχοντας τα αποτελέσματα αποτίμησης με βάση τους περιορισμούς που έχουν τεθεί. Αν μια τιμή αποτίμησης για ένα χαρακτηριστικό ή μετρική ποιότητας / ασφάλειας δεν ικανοποιεί έναν περιορισμό (π.χ. Είναι μικρότερη από ένα κάτω κατώφλι ή μεγαλύτερη από ένα άνω κατώφλι), τότε το αντίστοιχο αποθετήριο θα εντάσσεται στην κατηγορία των αποθετηρίων που δεν ικανοποιούν τις απαιτήσεις του χρήστη. Αν όλες οι τιμές αποτίμησης ικανοποιούν όλους τους περιορισμούς, τότε το αντίστοιχο

αποθετήριο θα εντάσσεται στην κατηγορία των αποθετηρίων που ικανοποιούν τις απαιτήσεις του χρήστη

- Η ταξινόμηση θα βασίζεται σε μια συνολική τιμή ποιότητας/ασφάλειας για το κάθε αποθετήριο που θα υπολογίζεται με ιεραρχικό τρόπο από κάτω προς τα πάνω, αρχικά με βάση τις αποτιμήσεις για τα φύλλα του δένδρου και εν τέλει με την παραγωγή μιας τιμής αποτίμησης για την ρίζα του δένδρου. Η τιμή αποτίμησης για έναν πατρικό κόμβο θα υπολογίζεται ως εξής: άθροισμα με βάρη των τιμών αποτίμησης για κάθε κόμβο παιδί του πατρικού κόμβου. Η τιμή αποτίμησης για ένα κόμβο φύλλο θα υπολογίζεται με βάση την επιθυμητή κατεύθυνση τιμών του αντίστοιχου χαρακτηριστικού ή μετρικής ποιότητας / ασφάλειας.
  - Αν η κατεύθυνση είναι θετική, τότε ο υπολογισμός θα γίνεται με βάση την εξής φόρμουλα: (x-min) / (max-min) όπου x είναι η τιμή αποτίμησης για το τρέχων αποθετήριο ενώ max & min είναι οι μέγιστες και ελάχιστες τιμές αποτίμησης του χαρακτηριστικού/μετρικής κατά μήκος όλων των αποθετηρίων. Η κατεύθυνση είναι θετική όταν προτιμάμε μεγαλύτερες τιμές για ένα χαρακτηριστικό/μετρική σε σχέση με τις χαμηλές
  - Αν η κατεύθυνση είναι αρνητική, τότε ο υπολογισμός θα γίνεται με βάση της εξής φόρμουλα: (max-x) / (max-min). Η κατεύθυνση είναι αρνητική όταν προτιμάμε χαμηλότερες τιμές για ένα χαρακτηριστικό/μετρική σε σχέση με τις υψηλές.

  Τονίζουμε πως και οι δύο φόρμουλες παράγουν τιμές αποτίμησης που είναι στο εύρος των πραγματικών αριθμών από [0.0, 1.0]. Επίσης, τονίζουμε πως εφαρμόζουμε αυτές τις φόρμουλες για κόμβους-φύλλα. Για τα υπόλοιπα είδη κόμβων, ο υπολογισμός γίνεται με βάση το άθροισμα με βάρη που αναφέρθηκε πριν.

- Μετά το φιλτράρισμα και την αποτίμηση των αποθετηρίων, αυτά θα πρέπει να εμφανίζονται σε ταξινομημένη μορφή με βάση τις 2 προαναφερόμενες κατηγορίες στον χρήστη.
- Τα αποτελέσματα που εμφανίζονται στον χρήστη δεν θα είναι στατικά. Ο χρήστης μπορεί να αλλάζει on-the-fly τους περιορισμούς ή τις απαιτήσεις του. Εφόσον γίνεται αυτό, τα αποτελέσματα θα ανανεώνονται για να ικανοποιήσουν τα νέα δεδομένα που παρέχονται από τον χρήστη. Προφανώς, η ανανέωση των αποτελεσμάτων θα πρέπει να γίνεται με γρήγορο τρόπο εφόσον η αποτίμηση της ποιότητας και ασφάλειας του κώδικα των αποθετηρίων έχει ήδη πραγματοποιηθεί.
- Διαχείριση χρηστών: Το σύστημα θα πρέπει να επιτρέπει την διαχείριση των χρηστών διότι μόνο ταυτοποιημένοι χρήστες θα μπορούν να το χρησιμοποιήσουν. Η διαχείριση των χρηστών περιλαμβάνει τις εξής λειτουργίες:
  - Εγγραφή χρήστη: ένας επισκέπτης θα μπορεί να αιτείται την εγγραφή του στο σύστημα/εφαρμογή ώστε να μπορεί να εκμεταλλευτεί την λειτουργικότητά της. Το σύστημα θα πρέπει να ελέγχει τα δεδομένα που δίνει ο χρήστης ως προς την εγκυρότητά τους. Ο κωδικός θα πρέπει να παρέχεται εις διπλούν για την αποφυγή λαθών ενώ καλό είναι να υπακούει σε κατάλληλους κανόνες εγκυρότητας (πχ. Μέγεθος > 8, χρήση γραμμάτων, ψηφίων και ειδικών χαρακτήρων, κλπ.).
  - Διαγραφή χρήστη: ο χρήστης θα μπορεί να αιτείται την διαγραφή του από την εφαρμογή. Αυτό σημαίνει πως όλα τα δεδομένα του χρήστη θα πρέπει να διαγράφονται κι αυτά.

    - Τροποποίηση στοιχείων χρήστη: ο χρήστης θα μπορεί να αλλάζει τα στοιχεία του εκτός από τον κωδικό.
    - Τροποποίηση κωδικού χρήστη: ο χρήστης θα μπορεί να αλλάζει τον κωδικό του. Σε αυτή την περίπτωση, θα πρέπει να παρέχει τον σωστό, παλαιό κωδικό και έναν έγκυρο, νέο κωδικό εις διπλούν.
    - Επαναφορά κωδικού χρήστη: ο χρήστης θα μπορεί να αιτείται την επαναφορά του κωδικού του, εφόσον τον έχει ξεχάσει. Σε αυτή την περίπτωση, το σύστημα θα στέλνει στον χρήστη έναν σύνδεσμο στο email που θα πρέπει να ακολουθήσει για να προχωρήσει στην αλλαγή του κωδικού του σε εύλογο χρονικό διάστημα.
    - Ταυτοποίηση χρήστη: το σύστημα θα πρέπει να επιτρέπει την ταυτοποίηση των χρηστών με βάση τα διαπιστευτήριά τους. Μετά την ταυτοποίηση, θα παράγεται ένα token, το οποίο ο χρήστης θα πρέπει να περιλαμβάνει στις αιτήσεις που κάνει στο σύστημα
- Διαχείριση ιστορικού ανάλυσης χρήστη:
    - Εμφάνιση αναλύσεων χρήστη: Το σύστημα θα πρέπει να εμφανίζει τις αναλύσεις που έχει κάνει ο χρήστης σε φθίνουσα χρονολογική σειρά.
    - Αναζήτηση αναλύσεων χρήστη: Ο χρήστης θα αιτείται την εμφάνιση μόνο εκείνων των αναλύσεων που βρίσκονται σε συγκεκριμένη χρονική περίοδο (δίνοντας τα δύο άκρα/όρια της χρονικής περιόδου)
    - Διαγραφή ανάλυσης χρήστη: Ο χρήστης μπορεί να αιτείται να διαγραφούν τα δεδομένα για μια ανάλυση (και άρα την ίδια την ανάλυση) που έχει επιτελέσει στο παρελθόν.
- Αντιγραφή ανάλυσης χρήστη: Ο χρήστης μπορεί να αιτείται την αντιγραφή της διαμόρφωσης μιας ανάλυσης για την δημιουργία μιας νέας.

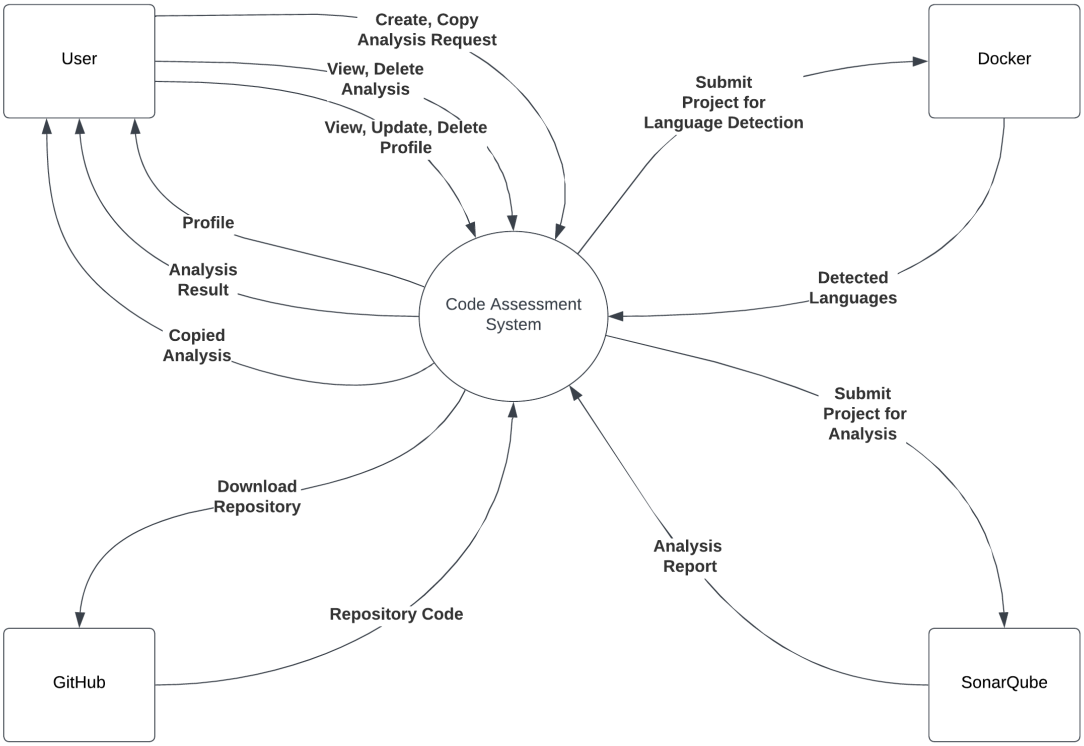
4.1.2. Μη Λειτουργικές Απαιτήσεις

- Η πρώτη παραγωγή αποτελεσμάτων ταξινόμησης δεν πρέπει να παίρνει πάνω από 30 δευτερόλεπτα. Οι επόμενες παραγωγές αποτελεσμάτων λόγω της ανανέωσης των περιορισμών/προτιμήσεων του χρήστη για την ίδια λίστα αποθετηρίων δεν θα πρέπει να ξεπερνά τα 10 δευτερόλεπτα
- Θα πρέπει να υποστηρίζονται τουλάχιστον 20 ταυτόχρονοι χρήστες
- Η διεπαφή χρήσης πρέπει να είναι χρηστική και αποκρίσιμη κατά μήκος διαφορετικών συσκευών
- Η διεπαφή χρήσης πρέπει να έχει ενιαίο look-n-feel
- Διαχωρισμένο API (Back-End) και Front-End
    - Προαιρετικό: containerisation της εφαρμογής (λογικά 3 συστατικά: front-end, back-end & βάση δεδομένων θα πρέπει να αντιστοιχιστούν σε εικόνες δοχείων) (COULD HAVE)
- Υλοποίηση backend σε REST με επίπεδο ωριμότητας 3
- Κατάλληλος χειρισμός λαθών/εξαιρέσεων με παροχή αυτο-επεξηγούμενων μηνυμάτων (και άλλων μέσων οπτικοποίησης)
- Κατάλληλη επικύρωση φορμών/δεδομένων τόσο στην πλευρά του πελάτη όσο και του εξυπηρετητή
- Υποστήριξη TLS & προστασία CSRF όσον αφορά την ασφάλεια της εφαρμογής ιστού

4.2. Σχεδίαση

Κατά την σχεδίαση του συστήματος, παράχθηκαν ορισμένα μοντέλα/διαγράμματα σχεδίασης, τα οποία προσδιορίζουν διάφορες πτυχές της δομής και συμπεριφοράς της εφαρμογής υπό ανάπτυξη καθώς και οδήγησαν την υλοποίησή της.

4.2.1. Διάγραμμα Περιβάλλοντος



Εικόνα 5. Το διάγραμμα περιβάλλοντος του συστήματος

Ένα διάγραμμα περιβάλλοντος (context diagram) ορίζει το όριο μεταξύ του συστήματος υπό ανάπτυξη ή μέρους του συστήματος αυτού και του περιβάλλοντός του, δείχνοντας τις οντότητες που αλληλεπιδρούν με αυτό. Το σύστημα υπό εξέταση βρίσκεται στο κέντρο του διαγράμματος.

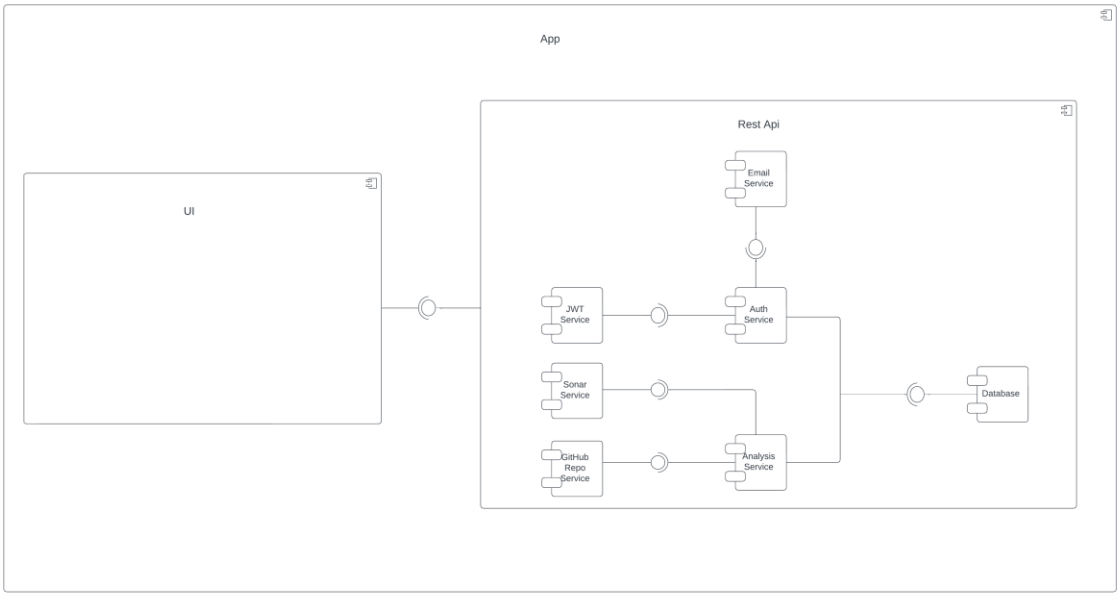
Εξωτερικές οντότητες για το προτεινόμενο σύστημα:

- **User:** Ένας ταυτοποιημένος χρήστης μπορεί να αιτηθεί την ανάλυση έργων, διαγραφή προηγούμενων αναλύσεων, αντιγραφή αρχικού αιτήματος ανάλυσης αλλά και την θέαση, ενημέρωση και διαγραφή του προφίλ του.
- **GitHub:** Για την ανάκτηση των αποθετηρίων που έχει υποβάλλει ο χρήστης
- **Docker:** Η ανίχνευση των γλωσσών προγραμματισμού γίνεται με ένα εργαλείο το όποιο τρέχει σε ένα δοχείο (container) docker. Το εργαλείο αυτό ονομάζεται GitHub Linguist[7]

[7] https://github.com/github-linguist/linguist

- **SonarQube:** Υπεύθυνο για την στατική ανάλυση των αποθετηρίων

4.2.2. Διάγραμμα Συστατικών / Αρχιτεκτονικής



Εικόνα 6. Το διάγραμμα συστατικών του συστήματος

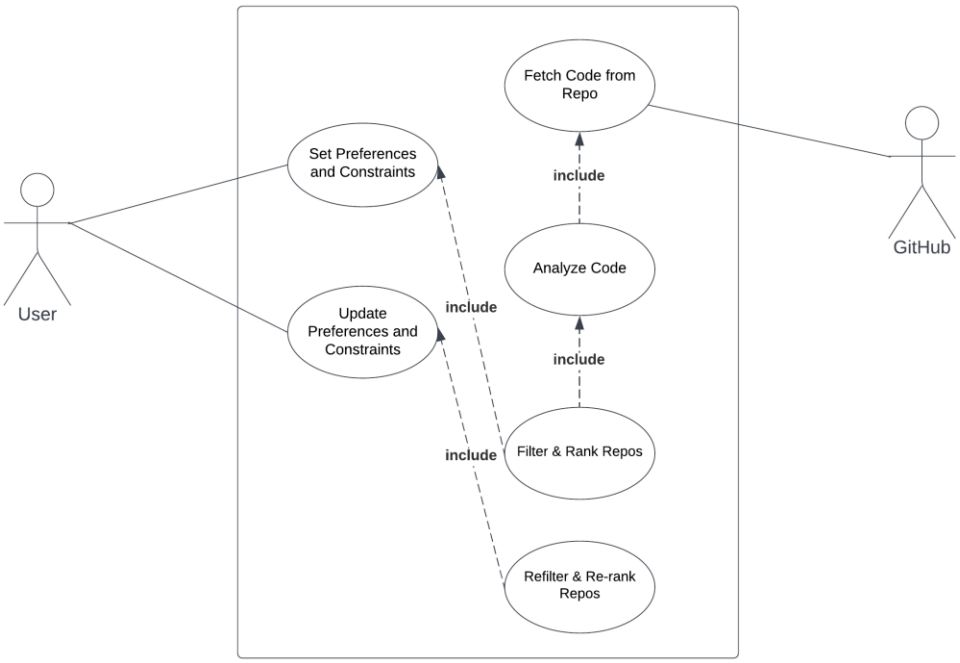
Ένα διάγραμμα συστατικών (component diagram) βοηθά στη μοντελοποίηση των συστατικών ενός συστήματος. Ειδικότερα, οπτικοποιεί την οργάνωση και την σύνδεση των συστατικών στο σύστημα. Κάθε συστατικό είναι ενθυλακωμένο και αλληλεπιδρά με άλλα συστατικά μέσω διεπαφών, διασφαλίζοντας έτσι ότι κάθε συστατικό είναι ένα μέρος του συστήματος.

Συστατικά:

- **App:** Η κύρια εφαρμογή που αποτελείται από 2 βασικά συστατικά, το UI και το API
- **UI:** Η διεπαφή μέσω της οποίας αλληλεπιδρά ο χρήστης με το σύστημα
- **Rest Api:** Λειτουργεί ως μια απομακρυσμένη προγραμματιστική διεπαφή που επιτρέπει άλλα συστατικά της εφαρμογής και εξωτερικές οντότητες να αλληλεπιδρούν με τις υποκείμενες υπηρεσίες της. Αποτελείται από 3 βασικά components:
  - **Auth Service:** Διαχειρίζεται την ταυτοποίηση των χρηστών
  - **User Service:** Διαχειρίζεται όλες τις ενέργειες διαχείρισης του προφίλ των χρηστών
  - **Analysis Service:** Διαχειρίζεται τη διαδικασία ανάλυσης κώδικα
- **Email Service:** Διαχειρίζεται λειτουργίες που σχετίζονται με email, όπως αποστολή ειδοποιήσεων και email επαλήθευσης σε χρήστες
- **Jwt Service:** Διαχειρίζεται τη δημιουργία και την επικύρωση JSON Web Tokens (JWT) για τη ταυτοποίηση χρηστών
- **GitHub Service:** Διαχειρίζεται τις αλληλεπιδράσεις με τα αποθετήρια του GitHub

- Sonar Service: Αλληλεπιδρά με το SonarQube για την ανάκτηση των αποτελεσμάτων της ανάλυσης

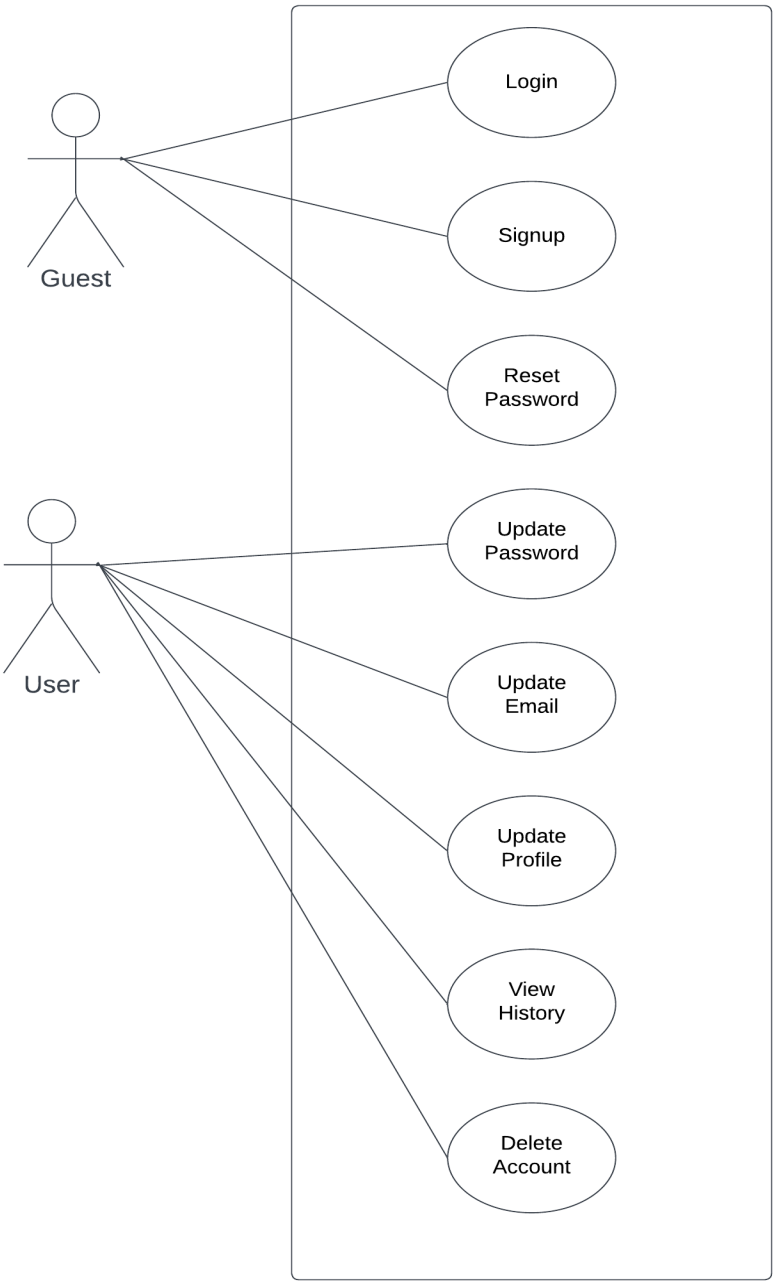
4.2.3. Διαγράμματα Περιπτώσεων Χρήσης



Εικόνα 7. Η περίπτωση χρήσης της ανάλυσης κώδικα

Ένας ταυτοποιημένος χρήστης μπορεί να αιτηθεί την ανάλυση αποθετηρίων υποβάλλοντας τα σχετικά GitHub links και προαιρετικά περιορισμούς και προτιμήσεις. Στην συνέχεια η εφαρμογή αναλαμβάνει να κατεβάσει τα αποθετήρια από το GitHub και, αφού αναλυθούν και ταξινομηθούν, επιστρέφονται τα αποτελέσματα στον χρήστη.

Επίσης, ο ταυτοποιημένος χρήστης μπορεί να ενημερώσει τους περιορισμούς και τις προτιμήσεις του για την ανανέωση των αποτελεσμάτων. Έπειτα, γίνεται εκ νέου ταξινόμηση των αποθετηρίων από το σύστημα και επιστρέφονται τα νέα αποτελέσματα στον χρήστη.



Εικόνα 8. Η περίπτωση χρήσης διαχείρισης χρηστών

Ένας επισκέπτης μπορεί να αιτηθεί:

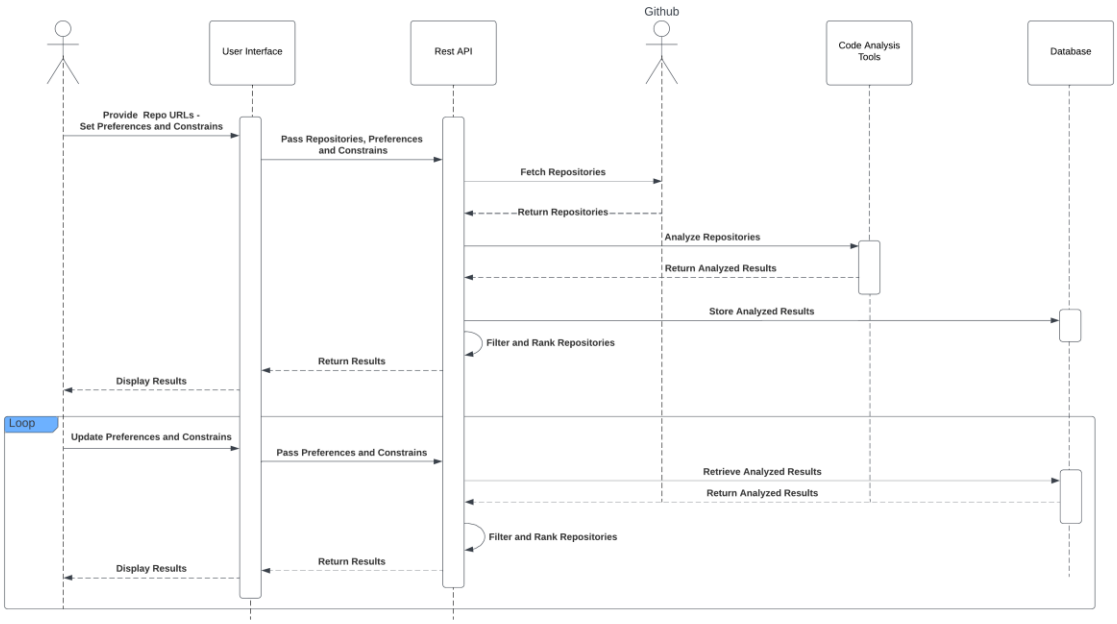
- Εγγραφή στο σύστημα
- Ταυτοποίηση στο σύστημα
- Επαναφορά κωδικού

Επιπλέον, ένας ταυτοποιημένος χρήστης μπορεί να αιτηθεί:

- Ενημέρωση κωδικού πρόσβασης
- Ενημέρωση διεύθυνσης email
- Ενημέρωση προφίλ
- Διαγραφή λογαριασμού

4.2.4. Διαγράμματα Ακολουθίας

Ένα διάγραμμα ακολουθίας (sequence diagram) εμφανίζει τις αλληλεπιδράσεις μεταξύ των χρηστών και του συστήματος καθώς και μεταξύ των συστατικών του συστήματος.

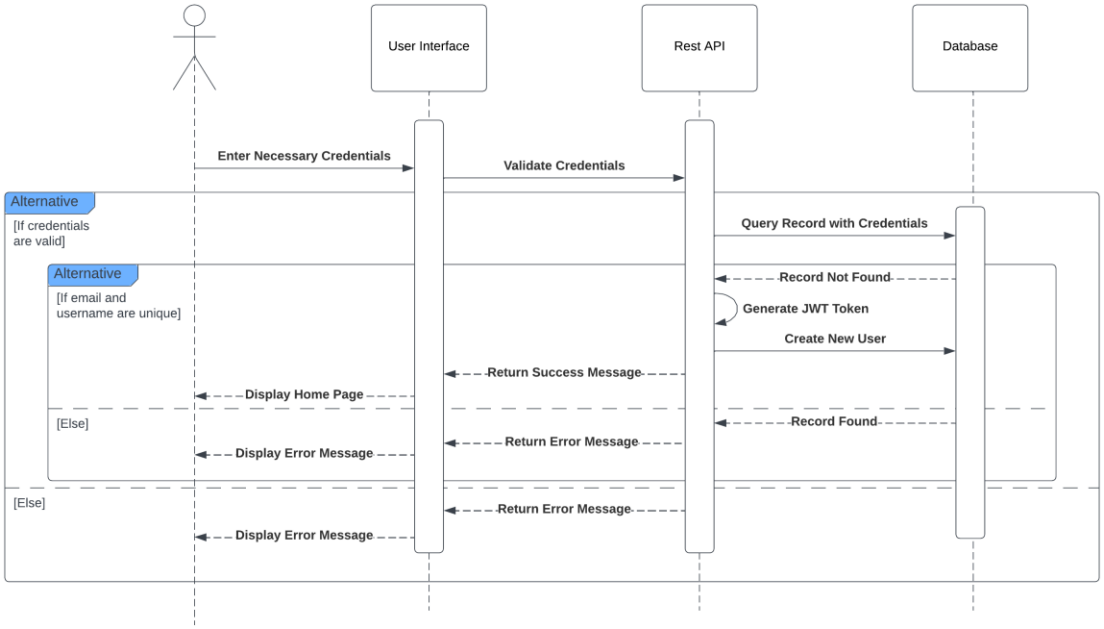


Εικόνα 9. Το διάγραμμα ακολουθίας για την ανάλυση κώδικα

Παρατηρούμε πως στο παραπάνω διάγραμμα για την ανάλυση του κώδικα αποθετηρίων λογισμικού οπτικοποιούνται οι εξής αλληλεπιδράσεις:

1) Ο χρήστης υποβάλλει τα σχετικά links των αποθετηρίων, περιορισμούς και προτιμήσεις για την ανάλυση μέσω της διεπαφής
2) Η διεπαφή επικοινωνεί και ενημερώνει το Rest Api για την σχετική αίτηση
3) Γίνεται ανάκτηση των σχετικών αποθετηρίων από το GitHub
4) Αποστολή των αποθετηρίων για ανάλυση στο εργαλείο ανάλυσης κώδικα και επιστροφή αποτελεσμάτων
5) Αποθήκευση αποτελεσμάτων στην βάση
6) Εφαρμογή των περιορισμών και προτιμήσεων για το φιλτράρισμα και την ταξινόμηση των αποτελεσμάτων
7) Επιστροφή αποτελεσμάτων στον χρήστη
8) Θέαση αποτελεσμάτων
9) Ανανέωση περιορισμών και προτιμήσεων από τον χρήστη μέσω της διεπαφής
10) Η διεπαφή επικοινωνεί και ενημερώνει το Rest Api για την σχετική αίτηση

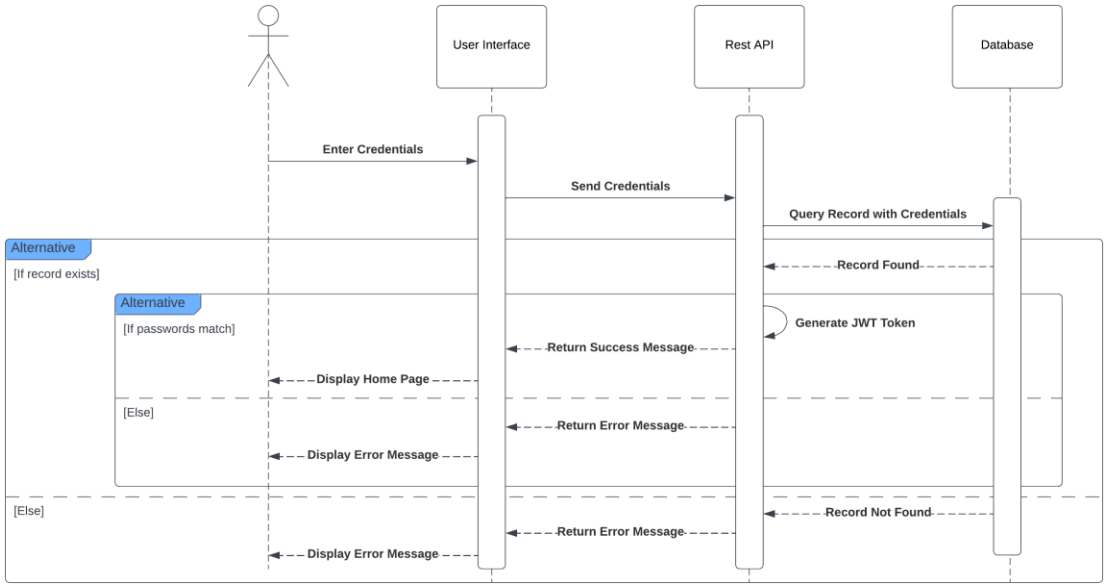
11) Ανάκτηση αποτελεσμάτων ανάλυσης από την βάση
12) Εφαρμογή των περιορισμών και προτιμήσεων για το φιλτράρισμα και την ταξινόμηση των αποτελεσμάτων
13) Επιστροφή αποτελεσμάτων στον χρήστη
14) Θέαση αποτελεσμάτων



Εικόνα 10. Το διάγραμμα ακολουθίας για την εγγραφή χρηστών

Το παραπάνω διάγραμμα εμφανίζει τις αλληλεπιδράσεις για την εγγραφή ενός χρήστη στο σύστημα, οι οποίες είναι οι εξής:

1) Υποβολή απαραίτητων στοιχείων από τον χρήστη για την εγγραφή στο σύστημα μέσω της διεπαφής
2) Επικύρωση στοιχείων
    a. Εάν τα στοιχεία είναι έγκυρα, αναζήτηση στην βάση
        1. Εάν δεν βρεθεί εγγραφή με το ίδιο username ή email
            a. Δημιουργείται ένα JWT
            b. Επιστροφή μηνύματος επιτυχούς εγγραφής
            c. Θέαση μηνύματος
        2. Εάν βρεθεί εγγραφή με το ίδιο username ή email
            a. Επιστρέφεται σχετικό μήνυμα λάθους
    b. Εάν τα στοιχεία δεν είναι έγκυρα, επιστρέφεται μηνύματος λάθους



Εικόνα 11. Το διάγραμμα ακολουθίας για την ταυτοποίηση χρήστη

Το παραπάνω διάγραμμα ακολουθίας περιλαμβάνει τις εξής αλληλεπιδράσεις στα πλαίσια της ταυτοποίησης ενός χρήστη στο σύστημα:

1) Υποβολή απαραίτητων στοιχείων από τον χρήστη για την ταυτοποίηση στο σύστημα μέσω της διεπαφής
2) Η διεπαφή επικοινωνεί και ενημερώνει το Rest Api για την σχετική αίτηση
3) Αναζήτηση στην βάση
    a. Εάν βρεθεί εγγραφή
        i. Εάν ο κωδικός που έδωσε ο χρήστης και ο κωδικός της εγγραφής ταιριάζουνε
            1. Δημιουργείται ένα JWT
            2. Επιστροφή μηνύματος επιτυχούς ταυτοποίησης
            3. Πλοήγηση στην αρχική σελίδα
        ii. Εάν οι κωδικοί δεν ταιριάζουνε, επιστρέφεται μηνύματος λάθους
    b. Εάν δεν βρεθεί η εγγραφή, επιστρέφεται μηνύματος λάθους

Ακολουθείται η κατάλληλη σειρά ενεργειών για την προστασία του συστήματος από επιθέσεις τύπου rainbow table.

Στο συγκεκριμένο τύπο επίθεσης οι εισβολείς χρησιμοποιούν έναν πίνακα προϋπολογισμένων τιμών κατακερματισμού για εκατομμύρια πιθανούς συνδυασμούς κωδικών πρόσβασης για να βρουν γρήγορα έναν κωδικό πρόσβασης που ταιριάζει με έναν δεδομένο κατακερματισμό. Επιτρέπει στους εισβολείς να αναθεωρήσουν έναν κατακερματισμό κωδικού πρόσβασης για να ανακαλύψουν τον αρχικό κωδικό πρόσβασης, αποκτώντας έτσι μη εξουσιοδοτημένη πρόσβαση σε ένα σύστημα.

Ένας απλός κατακερματισμός (hashing) στον κωδικό (πρόσβασης) που υποβάλλει ο χρήστης και η σύγκριση της τιμής κατακερματισμού με την αντίστοιχη τιμή της εγγραφής που βρέθηκε θα μας έκαναν ευάλωτους σε αυτή την επίθεση. Για να προστατευτούμε από αυτήν την επίθεση,

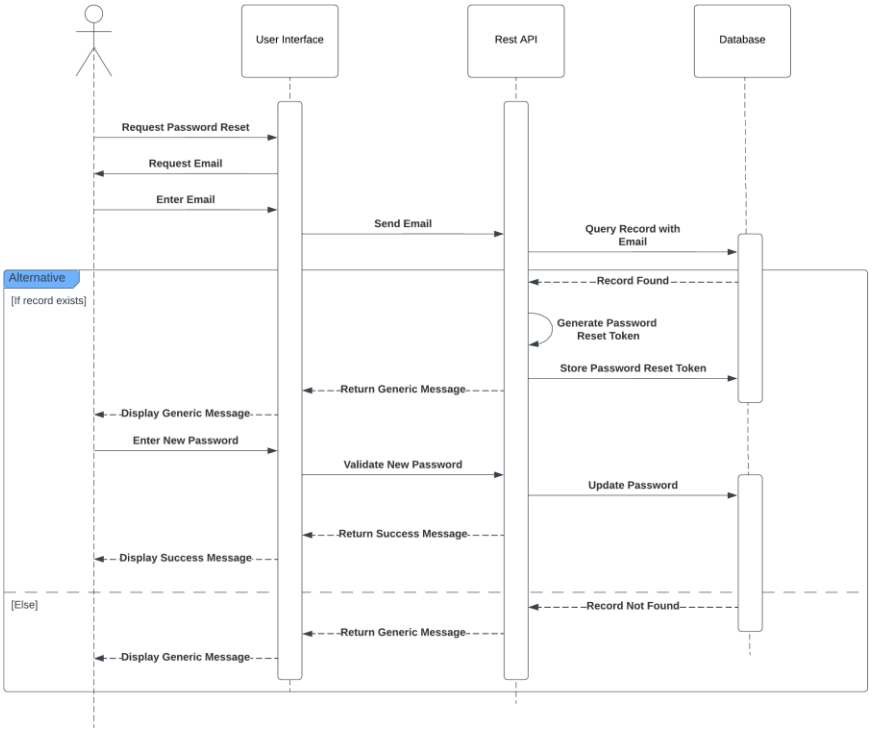
χρησιμοποιούμε έναν αλγόριθμο, ο όποιος προσθέτει “salt” στον κωδικό που υποβάλει ο χρήστης. Salt είναι μια τυχαία σειρά δεδομένων που προσαρτάται σε έναν κωδικό πρόσβασης πριν κατακερματιστεί. Επειδή το salt δημιουργεί μοναδικούς κατακερματισμούς για κάθε συνδυασμό χρήστη/κωδικού πρόσβασης, ένας εισβολέας θα πρέπει να δημιουργήσει έναν ξεχωριστό rainbow table για κάθε μοναδικό salt. Δεδομένου του σχεδόν άπειρου αριθμού πιθανών salt, η δημιουργία αυτών των πινάκων γίνεται υπολογιστικά απαγορευτική.

User	Password
Stephen	auhsoJ
Lisa	hsifdrowS
James	1010NO1Z
Harry	sinocarD tupaC
Sarah	auhsoJ

User	Password Hash
Stephen	39e717cd3f5c4be78d97090c69f4e655
Lisa	f567c40623df407ba980bfad6dff5982
James	711f1f88006a48859616c3a5cbcc0377
Harry	fb74376102a049b9a7c5529784763c53
Sarah	39e717cd3f5c4be78d97090c69f4e655

User	Random Salt	Password Hash
Stephen	06917d7ed65c466fa180a6fb62313ab9	b65578786e544b6da70c3a9856cdb750
Lisa	51f2e43105164729bb46e7f20091adf8	2964e639aa7d457c8ec0358756cbffd9
James	fea659115b7541479c1f956a59f7ad2f	dd9e4cd20f134dda87f6ac771c48616f
Harry	30ebf72072134f1bb40faa8949db6e85	204767673a8d4fa9a7542ebc3eceb3a2
Sarah	711f51082ea84d949f6e3efecf29f270	e3afb27d59a34782b6b4baa0c37e2958

Εικόνα 12. Η χρήση Salt για την αντιμετώπιση επιθέσεων τύπου rainbow table



Εικόνα 13. Το διάγραμμα ακολουθίας για την επαναφορά κωδικού

Το παραπάνω διάγραμμα ακολουθίας περιλαμβάνει τις εξής αλληλεπιδράσεις στο πλαίσιο της λειτουργίας επαναφοράς κωδικού:

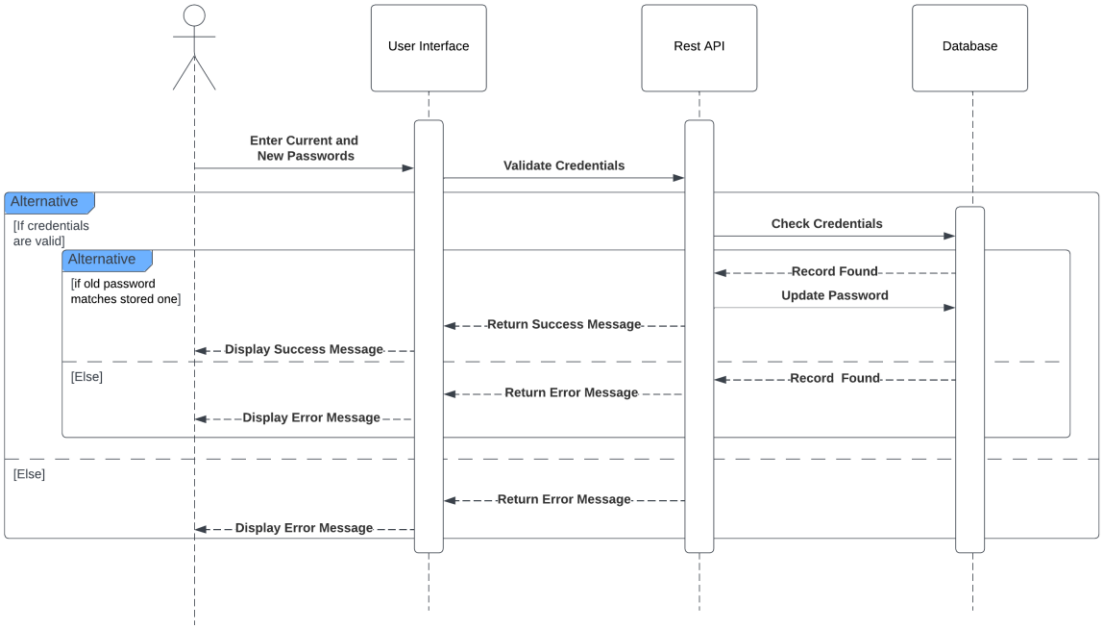
1) Αίτηση επαναφοράς κωδικού από τον χρήστη
2) Αίτηση και υποβολή σχετικής διεύθυνσης email από τον χρήστη
3) Η διεπαφή επικοινωνεί και ενημερώνει το Rest Api για την σχετική αίτηση
4) Αναζήτηση στην βάση με τη σχετική email διεύθυνση
    a. Εάν βρεθεί εγγραφή με τη συγκεκριμένη email διεύθυνση
        i. Δημιουργείται μοναδικό token
        ii. Αποθηκεύεται στην βάση
        iii. Επιστρέφεται μήνυμα με γενικό περιεχόμενο
        iv. Υποβάλλεται νέος κωδικός
        v. Επιβεβαιώνεται ότι ο νέος κωδικός πληροί τις απαραίτητες προϋποθέσεις
        vi. Ενημερώνεται ο κωδικός στην βάση
        vii. Επιστρέφεται μήνυμα επιτυχίας
        viii. Οπτικοποιείται το μήνυμα επιτυχίας
    b. Εάν δεν βρεθεί εγγραφή με το συγκεκριμένο email
        i. Επιστροφή γενικού μηνύματος με γενικό περιεχόμενο

Από το παραπάνω διάγραμμα παρατηρούμε ότι είτε βρεθεί εγγραφή στο σύστημα μας με τη σχετική email διεύθυνση είτε όχι το μήνυμα στο χρήστη είναι γενικού περιεχόμενου. Δεν είναι μήνυμα επιτυχίας ή αποτυχίας. Η συγκεκριμένη πρακτική εφαρμόζεται σαν ένα μέτρο προστασίας από επιθέσεις τύπου username enumeration.

Username enumeration είναι ένας τύπος επίθεσης ασφαλείας όπου ένας εισβολέας προσπαθεί να προσδιορίσει έγκυρα ονόματα χρήστη ενός συστήματος, μιας εφαρμογής ή ενός δικτύου. Αυτό είναι συνήθως ένας πρόδρομος για πιο στοχευμένες επιθέσεις, όπως οι επιθέσεις brute force με κωδικό πρόσβασης, όπου ο εισβολέας, αφού έχει μια λίστα με έγκυρα ονόματα χρήστη, προσπαθεί να σπάσει τους κωδικούς πρόσβασης που σχετίζονται με αυτά τα ονόματα χρήστη. Κατά τη διάρκεια μίας τέτοιας επίθεσης, οι εισβολείς αναζητούν μοναδικές απαντήσεις του εξυπηρετητή (server) που επιβεβαιώνουν την εγκυρότητα ενός υποβληθέντος διαπιστευτηρίου. Η πιο προφανής απάντηση είναι ένα μήνυμα ελέγχου ταυτότητας πεδίου μετά την υποβολή μιας φόρμας ιστού.

Εικόνα 14. Επίθεση τύπου Username Enumeration

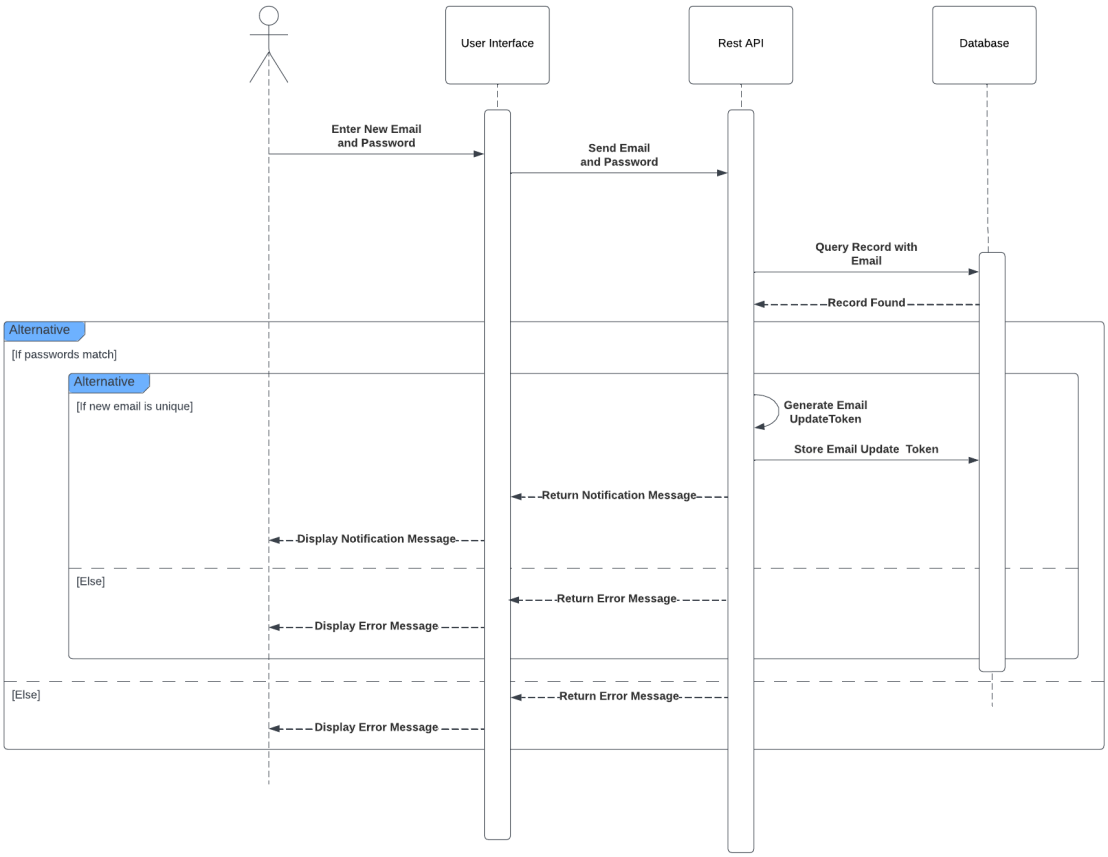
Να σημειωθεί ότι προφανώς και γίνεται επικύρωση του νέου κωδικού. Δεν αποτυπώθηκε στο διάγραμμα για να μην βγει μεγάλο σε έκταση. Επίσης στέλνεται σχετικό email στον χρήστη με σύνδεσμο επαλήθευσης, όπου περιέχει το token και στην συνέχεια ο χρήστης μπορεί να υποβάλει τον νέο κωδικό.



Εικόνα 15. Το διάγραμμα ακολουθίας για την ανανέωση κωδικού πρόσβασης

Το παραπάνω διάγραμμα περιλαμβάνει τις εξής αλληλεπιδράσεις που σχετίζονται με την ανανέωση του κωδικού πρόσβασης ενός χρήστη:

1) Υποβολή παλιού και καινούργιου κωδικού
2) Επικύρωση καινούργιου κωδικού
    a. Εάν ο καινούργιος κωδικός πληροί τις προϋποθέσεις, ανάκτηση σχετικής εγγραφής από την βάση
        i. Εάν ο παλιός κωδικός είναι ο ίδιος με τον κωδικό της εγγραφής από την βάση:
            1. Ενημέρωση της βάσης με τον καινούργιο κωδικό
            2. Επιστροφή μηνύματος επιτυχίας
            3. Θέαση μηνύματος επιτυχίας
        ii. Εάν ο παλιός κωδικός δεν είναι ο ίδιος με τον κωδικό της εγγραφής από την βάση:
            1. Επιστροφή μηνύματος λάθους
            2. Θέαση μηνύματος λάθους
    b. Εάν ο καινούργιος κωδικός δεν πληροί τις προϋποθέσεις:
            1. Επιστροφή μηνύματος λάθους
            2. Θέαση μηνύματος λάθους



Εικόνα 16. Το διάγραμμα ακολουθίας για την ενημέρωση email διεύθυνσης

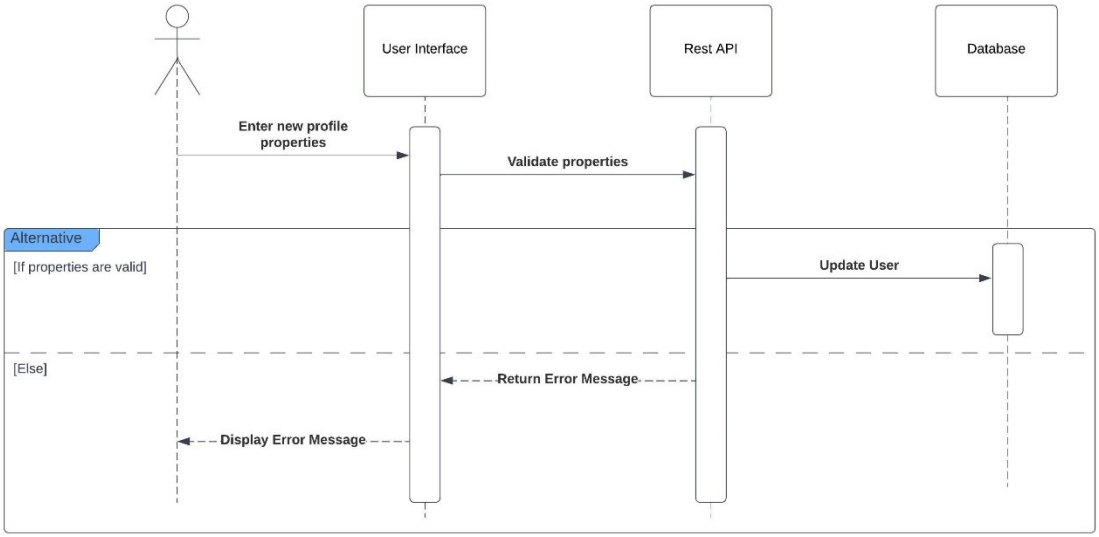
Το παραπάνω διάγραμμα ακολουθίας περιλαμβάνει τις εξής αλληλεπιδράσεις που αφορούν την ενημέρωση της email διεύθυνσης ενός χρήστη:

1) Υποβολή νέας διεύθυνσης email και τρέχοντος κωδικού
2) Η διεπαφή επικοινωνεί και ενημερώνει το Rest Api για την σχετική αίτηση
3) Ανάκτηση εγγραφής από την βάση
    a. Εάν ο κωδικός της αίτησης είναι ο ίδιος με τον κωδικό της εγγραφής από την βάση
        i. Εάν η νέα διεύθυνσης email είναι μοναδική:
            1. Δημιουργείται μοναδικό token
            2. Αποθηκεύεται στην βάση
            3. Επιστροφή σχετικού ενημερωτικού μηνύματος
            4. Θέαση σχετικού ενημερωτικού μηνύματος
        ii. Εάν η νέα διεύθυνση email δεν είναι μοναδική:

1. Επιστροφή μηνύματος λάθους
2. Θέαση μηνύματος λάθους

b. Εάν ο κωδικός της αίτησης δεν είναι ο ίδιος με τον κωδικό της εγγραφής από την βάση:
    i. Επιστροφή μηνύματος λάθους
    ii. Θέαση μηνύματος λάθους

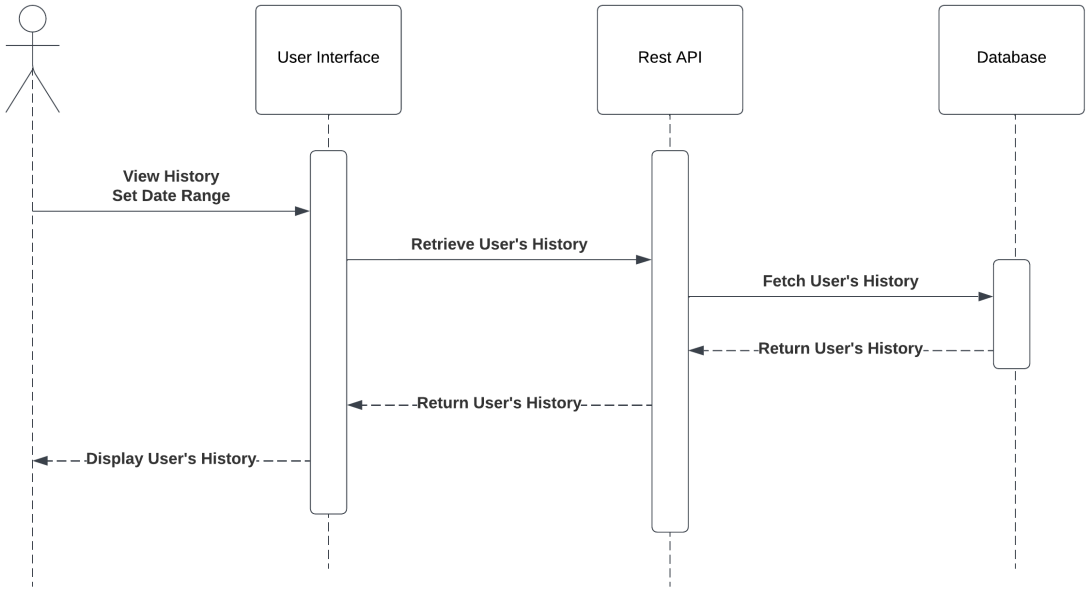
Να σημειωθεί ότι στέλνεται σχετικό email στον χρήστη με σύνδεσμο επαλήθευσης, όπου περιέχει το token. Ο χρήστης ενημερώνεται με σχετικό μήνυμα ότι για να ολοκληρωθεί η διαδικασία πρέπει να μεταβεί στο καινούργιο email και να επιλέξει τον σχετικό σύνδεσμο.



Εικόνα 17. Το διάγραμμα ακολουθίας για την ενημέρωση προφίλ χρήστη

Το παραπάνω διάγραμμα περιλαμβάνει τις εξής αλληλεπιδράσεις που αφορούν την ενημέρωση του προφίλ ενός χρήστη:

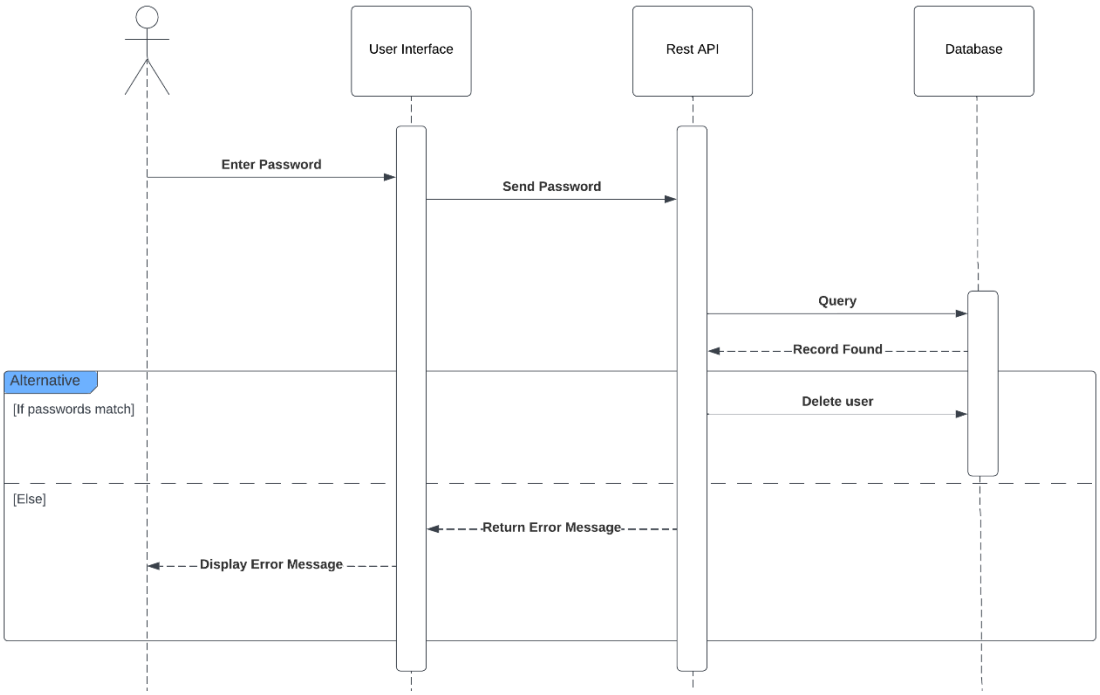
1) Υποβολή νέων στοιχείων
2) Επικύρωση στοιχείων
    a. Εάν τα στοιχεία είναι έγκυρα:
        i. Ενημέρωση χρήστη στην βάση
    b. Εάν τα στοιχεία δεν είναι έγκυρα:
        i. Επιστροφή μηνύματος λάθους
        ii. Θέαση μηνύματος λάθους



Εικόνα 18. Το διάγραμμα ακολουθίας για την θέαση ιστορικού

Το παραπάνω διάγραμμα περιλαμβάνει τις εξής αλληλεπιδράσεις που αφορούν την θέαση του ιστορικού χρήστη:

1) Αίτηση για θέαση ιστορικού από τον χρήστη
2) Η διεπαφή επικοινωνεί και ενημερώνει το Rest Api για την σχετική αίτηση
3) Ανάκτηση ιστορικού από την βάση
4) Επιστροφή λίστας αναλύσεων
5) Θέαση λίστας αναλύσεων

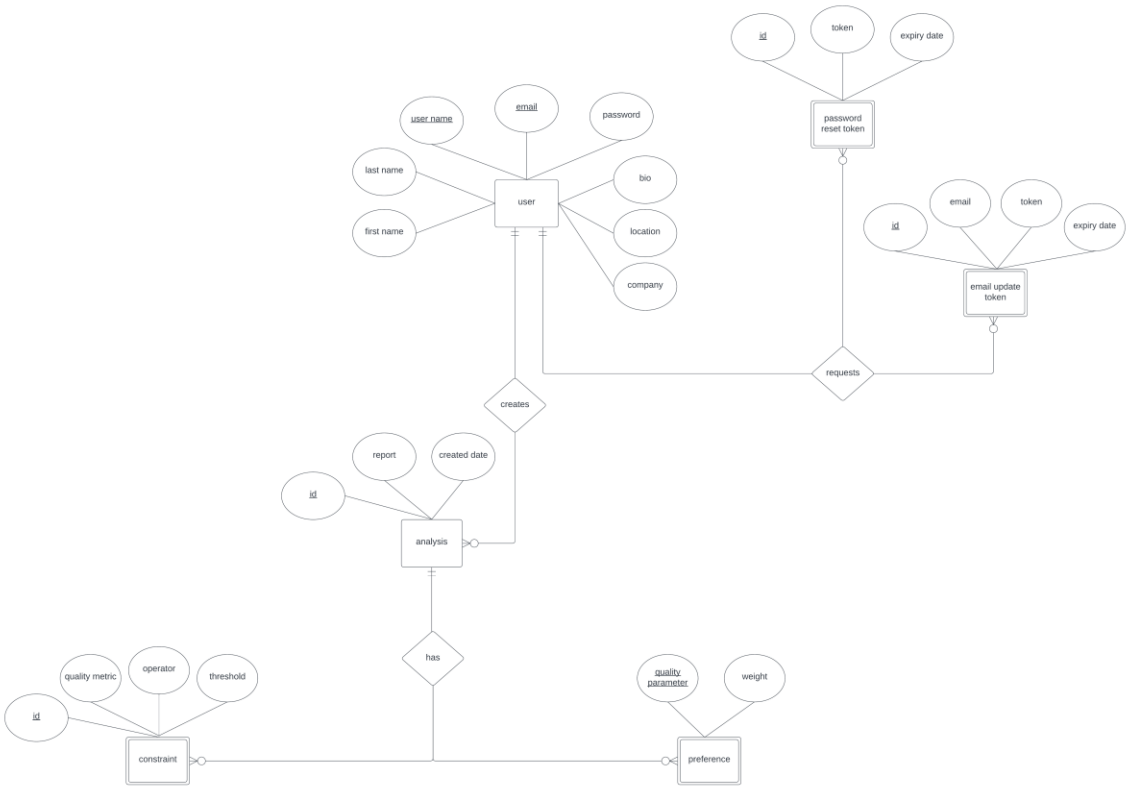


Εικόνα 19. Το διάγραμμα ακολουθίας για την διαγραφή λογαριασμού

Το παραπάνω διάγραμμα περιλαμβάνει τις εξής δραστηριότητες που αφορούν την διαγραφή του λογαριασμού ενός χρήστη:

1) Υποβολή τρέχοντος κωδικού από τον χρήστη
2) Η διεπαφή επικοινωνεί και ενημερώνει το Rest Api για την σχετική αίτηση
3) Ανάκτηση εγγραφής από την βάση
    a. Εάν ο κωδικός της αίτησης είναι ο ίδιος με τον κωδικό της εγγραφής από την βάση:
        i. Διαγραφή χρήστη
    b. Εάν ο κωδικός της αίτησης δεν είναι ο ίδιος με τον κωδικό της εγγραφής από την βάση:
        i. Επιστροφή μηνύματος λάθους
        ii. Θέαση μηνύματος λάθους

4.2.5. Διάγραμμα Οντοτήτων-Συσχετίσεων



Εικόνα 20. Το διάγραμμα Οντοτήτων-Συσχετίσεων

Το διάγραμμα Οντοτήτων-Συσχετίσεων (Entity-Relationship – ER) απεικονίζει τις ακόλουθες οντότητες και τις σχέσεις τους:

1. User:
   - Ιδιότητες: user_name (όνομα χρήστη) (μοναδική), email (διεύθυνση email) (μοναδική), password (κωδικός), first name (όνομα), last name (επώνυμο), bio (βιογραφικό), location (τοποθεσία), και company (επιχείρηση).
   - Σχέσεις:
     - Δημιουργία μίας ή πολλών αναλύσεων (Analysis).
     - Αίτηση ενός ή πολλών Password Reset Token και Email Update Token.
2. Analysis:
   - Ιδιότητες: id (αναγνωριστικό) (μοναδική), report (αναφορά), created date (ημερομηνία δημιουργίας).
   - Έχει μηδέν ή πολλούς περιορισμούς και προτιμήσεις (Constraints, Preferences)
3. Constraint:
   - Ιδιότητες: id (αναγνωριστικό) (μοναδική), quality metric (μετρική ποιότητας), operator (τελεστής), και threshold (όριο).
   - Είναι αδύναμη οντότητα ως προς την οντότητα της Ανάλυσης (Analysis).
4. Preference:
   - Ιδιότητες: quality parameter (παράμετρος ποιότητας) (μοναδική) και weight (βάρος).
   - Είναι αδύναμη οντότητα ως προς την οντότητα της Ανάλυσης (Analysis).

5. Password Reset Token:
    - Ιδιότητες: id (αναγνωριστικό) (μοναδική), token (μάρκα), και expiry date (ημερομηνία λήξης).
    - Είναι αδύναμη οντότητα ως προς την οντότητα του Χρήστη (User).
6. Email Update Token:
    - Ιδιότητες: id (αναγνωριστικό) (μοναδική), email (διεύθυνση email), token (μάρκα), και expiry date (ημερομηνία λήξης).
    - Είναι αδύναμη οντότητα ως προς την οντότητα του Χρήστη (User).

4.3. Υλοποίηση

4.3.1. Αρχιτεκτονική

Η αρχιτεκτονική με την οποία υλοποιήθηκε η συγκεκριμένη εφαρμογή είναι η N – tier αρχιτεκτονική ή διαστρωματική (layered) αρχιτεκτονική. Είναι ένα μοτίβο αρχιτεκτονικής λογισμικού στο οποίο μια εφαρμογή χωρίζεται σε ξεχωριστά στρώματα/επίπεδα και κάθε ένα εκτελεί μια συγκεκριμένη λειτουργία. Κάθε στρώμα μπορεί να επικοινωνήσει μόνο με τα στρώματα ακριβώς πάνω και κάτω από αυτό.

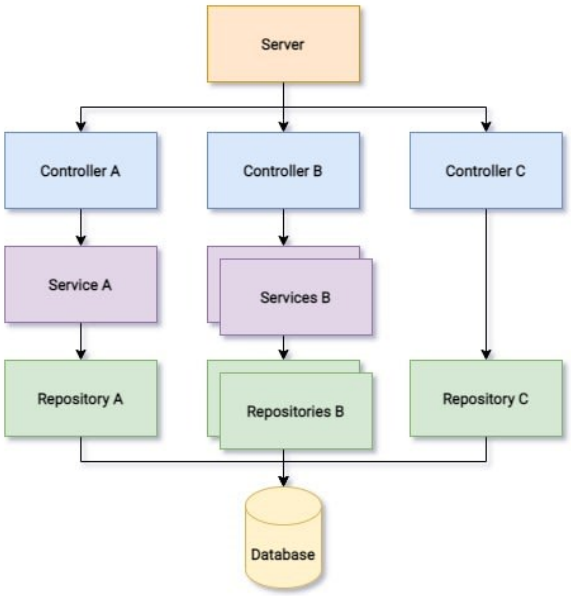
Τα επίπεδα/στρώματα σε μια τυπική N – tier αρχιτεκτονική είναι:

Presentation Layer (Στρώμα Παρουσίασης): Το ανώτερο επίπεδο που αλληλοεπιδρά με τον τελικό χρήστη. Χειρίζεται τη λογική παρουσίασης της εφαρμογής και επικοινωνεί με το service layer (στρώμα υπηρεσιών) για ανάκτηση ή ενημέρωση δεδομένων.

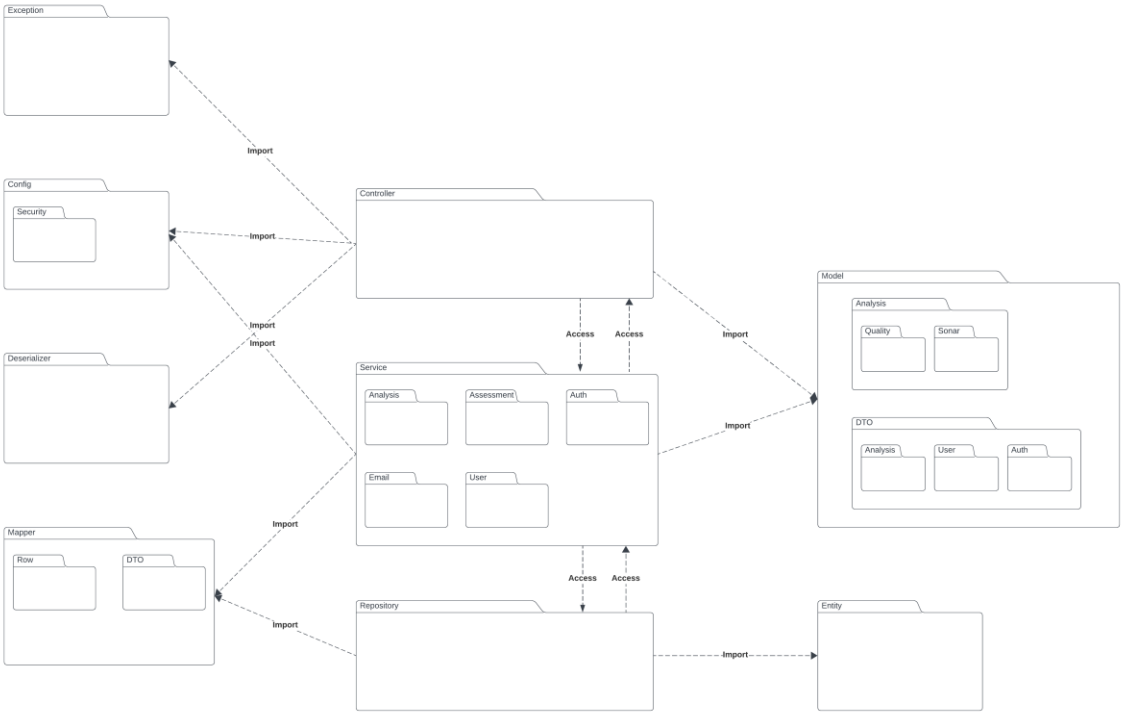
Service Layer (Στρώμα Υπηρεσιών): Το μεσαίο στρώμα που χειρίζεται τη λογική της εφαρμογής. Λαμβάνει δεδομένα από το επίπεδο παρουσίασης, εκτελεί επιχειρηματική λογική (business logic) και αλληλοεπιδρά με το data layer για ανάκτηση ή ενημέρωση δεδομένων.

Data Layer (Στρώμα Δεδομένων): Το κατώτατο επίπεδο που χειρίζεται τη διατήρηση δεδομένων. Αλληλοεπιδρά με τη βάση δεδομένων για την αποθήκευση και την ανάκτηση δεδομένων.

Διαχωρίζοντας την εφαρμογή σε ξεχωριστά επίπεδα/στρώματα, η αρχιτεκτονική παρέχει πλεονεκτήματα, όπως επεκτασιμότητα και δυνατότητα συντήρησης. Οι αλλαγές σε ένα επίπεδο έχουν ελάχιστο αντίκτυπο στα άλλα επίπεδα, καθιστώντας ευκολότερη την τροποποίηση και επέκταση της εφαρμογής.



Εικόνα 21. Η N-tier αρχιτεκτονική



Εικόνα 22. Το διάγραμμα πακέτων του συστήματος

Ένα διάγραμμα πακέτου απεικονίζει πώς το σύστημα χωρίζεται σε λογικές ομαδοποιήσεις χρησιμοποιώντας πακέτα. Τα πακέτα χρησιμοποιούνται για την ομαδοποίηση σχετικών κλάσεων, διεπαφών ή άλλων στοιχείων σε μια ενιαία μονάδα, η οποία βοηθά στην οργάνωση και διαχείριση μεγάλων συστημάτων λογισμικού. Η δομή των πακέτων του συστήματός μας απεικονίζεται στην παραπάνω εικόνα.

Backend: Η γλώσσα προγραμματισμού που επιλέχθηκε είναι η Java, συγκεκριμένα η έκδοση 17, ενώ το πλαίσιο ανάπτυξης ήταν το Spring Boot έκδοση 3.1.0. Το εργαλείο αυτοματοποίησης κατασκευής (build automation tool) που επιλέχθηκε είναι το Maven. Είναι ένα εργαλείο αυτοματοποίησης κατασκευής και διαχείρισης εξαρτήσεων για έργα που βασίζονται σε Java καθώς απλοποιεί τις προαναφερόμενες διαδικασίες, παρέχοντας τη δυνατότητα χειρισμού των κατασκευών του έργου (λογισμικού) και των εξαρτήσεων του μέσω μιας προσέγγισης βασιζόμενης σε μοντέλα διαμόρφωσης κατασκευής. Για τη βάση δεδομένων χρησιμοποιήθηκε η PostgreSQL, έκδοση 15.2, που είναι ένα δωρεάν και ανοιχτού κώδικα σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων ενώ για τη μετανάστευση βάσεων δεδομένων (database migrations) χρησιμοποιήθηκε το Flyway. Για την αναγνώριση των γλωσσών του κάθε αποθετηρίου χρησιμοποιήθηκε το GitHub Linguist ενώ για την στατική ανάλυση του κώδικα το SonarQube.

Frontend: Η γλώσσα προγραμματισμού που επιλέχθηκε είναι η Typescript, συγκεκριμένα η έκδοση 5.2, ενώ το πλαίσιο ανάπτυξης ήταν η Angular 16. Για CSS επιλέχθηκε Bootstrap 5 που αποτελείται από στοιχεία HTML, CSS και JavaScript ενώ προσφέρει προσχεδιασμένα πρότυπα και δυνατότητες για την ταχεία ανάπτυξη ιστού.

4.3.2. Ανάλυση βασικών κλάσεων

RankingService (Υπηρεσία Ταξινόμησης)

```java
@Service
@RequiredArgsConstructor
public class RankingService {
    private final TreeService treeService;

    /*
        If any empty list of preferences is provided, the weights are equally distributed among the child nodes.
     */
    4 usages  ThanosL
    public double rankTree(Map<QualityMetric, Double> qualityMetricsReport, List<Preference> preferences) {
        TreeNode root = treeService.buildTree();

        assignWeight(root, preferences);
        assignLeafNodeValue(root, qualityMetricsReport);
        assignParentNodeValue(root);

        return root.getValue();
    }

    2 usages  ThanosL
    private void assignWeight(TreeNode node, List<Preference> preferences) {
        if (treeService.isLeafNode(node)) {
            return;
        }

        /*
            If there is only 1 child node then it has the max weight.
         */
        if (node.getChildren().size() == 1) {
            node.getChildren().get(0).setWeight(1.0);

            return;
        }
```

Εικόνα 23. Η κλάση της υπηρεσίας ταξινόμησης

```java
        List<TreeNode> childNodesWithoutWeight = new ArrayList<>();
        double sum = 1.0;
        Optional<Preference> matchingPreference;

        for (TreeNode child : node.getChildren()) {
            matchingPreference = preferences.stream()
                    .filter(preference -> preference.getQualityAttribute().name().equals(child.getName()))
                    .findFirst();

            /*
                Case 1: The user submitted weight for the specific node.
                Case 2: The user did not submit weight for the specific node.
             */
            if (matchingPreference.isPresent()) {
                child.setWeight(matchingPreference.get().getWeight());
                sum -= child.getWeight();
            } else {
                childNodesWithoutWeight.add(child);
            }
        }

        /*
            Weight is distributed dynamically to the child nodes without weight
         */
        double weightToDistribute = sum / childNodesWithoutWeight.size();
        childNodesWithoutWeight.forEach(nodeWithoutWeight -> nodeWithoutWeight.setWeight(weightToDistribute));

        for (TreeNode child : node.getChildren()) {
            assignWeight(child, preferences);
        }
    }
```

Εικόνα 24. Η κλάση της υπηρεσίας ταξινόμησης (συνέχεια)

```
/*
    Assigning the values to the leaf nodes(metrics), to calculate the value of the parent node.
 */
2 usages  ThanosL
private void assignLeafNodeValue(TreeNode node, Map<QualityMetric, Double> qualityMetricsReport) {
    if (treeService.isLeafNode(node)) {
        node.setValue(qualityMetricsReport.get(QualityMetric.valueOf(node.getName())));

        return;
    }

    for (TreeNode child : node.getChildren()) {
        assignLeafNodeValue(child, qualityMetricsReport);
    }
}

/*
    The value of a parent node is the sum of the products of each child node's weight and value
 */
2 usages  ThanosL
private void assignParentNodeValue(TreeNode node) {
    if (treeService.isLeafNode(node)) {
        return;
    }

    double parentValue = 0.0;
    for (TreeNode child : node.getChildren()) {
        assignParentNodeValue(child);
        parentValue += child.getWeight() * child.getValue();
    }

    node.setValue(parentValue);
}
```

Εικόνα 25. Η κλάση της υπηρεσίας ταξινόμησης (δεύτερη συνέχεια)

Η κλάση RankingService είναι μία από τις πιο σημαντικές κλάσεις της εφαρμογής καθώς είναι υπεύθυνη για τον υπολογισμό του βαθμού (rank) του αποθετηρίου με βάση τις προτιμήσεις του χρήστη.
Αφού έχουμε βεβαιωθεί ότι ισχύουν 2 βασικοί κανόνες για τα βάρη:

- Όταν έχουνε δοθεί από τον χρήστη βάρη για όλα τα παιδιά ενός κόμβου πατέρα, το άθροισμα πρέπει να ισούται με 1.
- Ότι το άθροισμα των βαρών των παιδιών ενός κόμβου πατέρα δεν ξεπερνά το 1.

Αρχικά η RankingService αναλαμβάνει την κατανομή βαρών σε όλους τους κόμβους του δέντρου. Για κάθε πατρικό κόμβο, διατηρούμε μια συνολική τιμή βάρους, η οποία αρχικά είναι 1.0 και έπειτα μειώνεται για κάθε κόμβο-παιδί για τον οποίο παραδόθηκε τιμή βάρους από τον χρήστη. Η συνολική τιμή βάρους είναι απαραίτητη για τον υπολογισμό του βάρους σε κόμβους-παιδιά που δεν έχουν «στολιστεί» με τιμή βάρους από τον χρήστη. Επομένως, για την ανάθεση βάρους σε 1 κόμβο-παιδί έχουμε 2 περιπτώσεις:

- Ο χρήστης έδωσε βάρος για τον συγκεκριμένο κόμβο. Ανατίθεται η τιμή του χρήστη και αφαιρείται η τιμή αυτή από την προαναφερόμενη, συνολική τιμή βάρους ώστε στην

συνέχεια να μπορούμε να αναθέσουμε βάρη σε παιδιά-κόμβους, για τα οποία δεν παρείχε σχετική τιμή (βάρους) ο χρήστης. Τυπικά:

$$sum = sum - node.weight$$

όπου sum είναι η συνολική τιμή βάρους

- Ο χρήστης δεν έδωσε βάρος για τον συγκεκριμένο κόμβο-παιδί. Εάν ο κόμβος είναι το μοναδικό παιδί, τότε θα ανατεθεί η μέγιστη τιμή 1.0 σε αυτό. Διαφορετικά, ο κόμβος αυτός θα προστεθεί σε μία λίστα *childNodesWithoutWeight*.

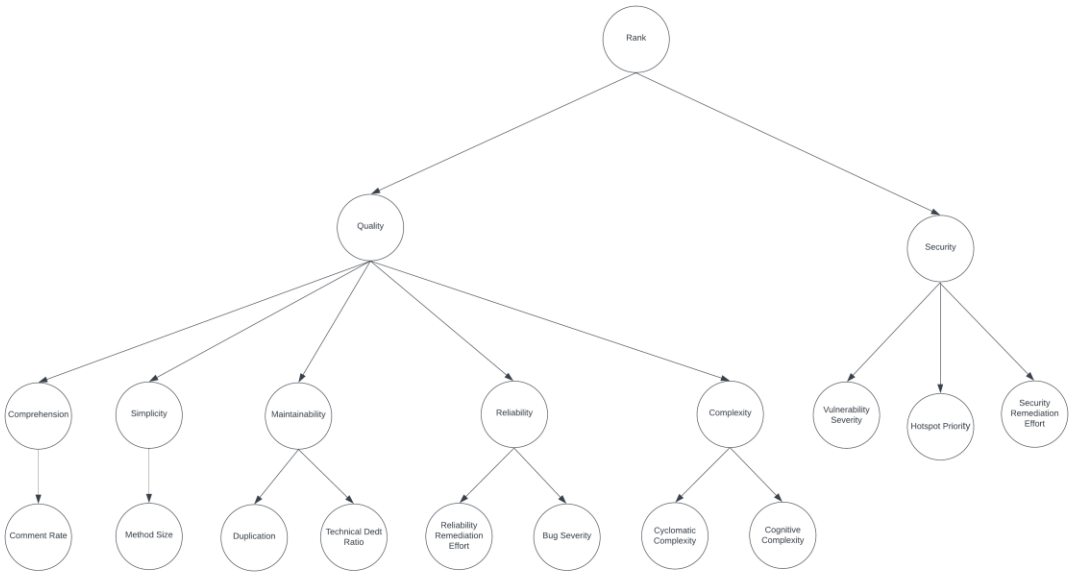
Όταν ολοκληρωθεί ο έλεγχος για όλα τα παιδιά ενός κόμβου πατέρα, θα πρέπει να ανατεθεί βάρος στους κόμβους-παιδιά που δεν έδωσε ο χρήστης σχετική τιμή (βάρους). Το μέγεθος της λίστας είναι και το πλήθος των παιδιών χωρίς βάρος. Το βάρος που πρέπει να κατανεμηθεί όμοια υπολογίζεται από τον τύπο:

$$weightToDistribute = \frac{sum}{childNodesWithoutWeight.size()}$$

Αφού ανατεθεί βάρος για όλους τους κόμβους του δέντρου και δεδομένου ότι έχουμε τιμή για όλες τις μετρικές (κόμβους-φύλλα του δέντρου), υπολογίζουμε την τιμή ωφέλειας του κόμβου πατέρα και αναδρομικά τον τελικό βαθμό (τιμή ωφέλειας) συνολικά του αποθετηρίου. Η τιμή ωφέλειας του κόμβου πατέρα υπολογίζεται από τον εξής τύπο:

$$u_j = \sum_{i=1}^{n} (w_i * u_i)$$

n: Συνολικός αριθμός από κόμβους παιδιά
w_i: Βάρους του i κόμβου-παιδιού
u_i: Τιμή ωφέλειας του i κόμβου-παιδιού



Εικόνα 26. Το δένδρο ταξινόμησης αποθετηρίων

Εφόσον ένας κόμβος-παιδί είναι φύλλο, τότε αντιστοιχεί σε μετρικές, των οποίων οι τιμές παρέχονται από το Sonarqube κατά την στατική ανάλυση του κώδικα (του τρέχοντος αποθετηρίου). Οι τιμές αυτές εφαρμόζονται στην αντίστοιχη συνάρτηση ωφέλειας (utility function) της μετρικής ώστε να υπολογιστεί η τιμή ωφέλειας της μετρικής. Η τιμή αυτή είναι προσεγμένα σχεδιασμένη να βρίσκεται εντός του εύρους [0.0, 1.0]. Επίσης, όσο πιο μεγάλη είναι η τιμή της, τόσο το καλύτερο, δηλ. τόσο πιο υψηλή είναι η απόδοση του αποθετηρίου ως προς την προκείμενη μετρική. Παρακάτω παραθέτουμε τις συναρτήσεις ωφέλειας για όλες τις μετρικές / κόμβους φύλλα που εμφανίζονται στο προηγούμενο δένδρο ταξινόμησης.

- Comment Rate (Αναλογία σχολίων): $\boldsymbol{f(x) = \begin{cases} 1.0, & x = 30.0 \\ \frac{100 - x}{70}, & x > 30.0 \\ e^{(0.023 \times x) - 1}, & x < 30.0 \end{cases}}$ όπου x είναι το ποσοστό των γραμμών σχολίων κατά μήκος όλων των γραμμών του κώδικα. Παρατηρούμε πως όταν το ποσοστό είναι ίσο με 30%, τότε έχουμε την μέγιστη ωφέλεια. Όταν το ποσοστό είναι μεγαλύτερο από 30%, η ωφέλεια μειώνεται γραμμικά. Ενώ όταν είναι μικρότερη από 30%, τότε μειώνεται εκθετικά.
- Method Size (Μέγεθος Μεθόδων): $\boldsymbol{f(x) = \begin{cases} 1.0, & x \leq 37 \\ 0.0, & x \geq 162 \\ \frac{2^{\frac{(70 - x)}{21}}}{3.0}, & 37 < x < 162 \end{cases}}$ όπου x είναι το μέγεθος των μεθόδων ως προς τις γραμμές κώδικα. Παρατηρούμε πως όταν ο αριθμός των γραμμών είναι μικρότερος από 37 έχουμε την μέγιστη ωφέλεια και όταν είναι μεγαλύτερος από 162, έχουμε την ελάχιστη ωφέλεια. Ενδιάμεσα, η ωφέλεια μειώνεται εκθετικά μέχρι την τιμή 0.0.
- Duplication (Ποσοστό διπλοτύπων): $\boldsymbol{f(x) = 1 - x}$ όπου x είναι το ποσοστό διπλοτύπων στον κώδικα.
- Technical Dept Ratio (Αναλογία Τεχνικού Χρέους): $\boldsymbol{f(x) = 1 - x}$ όπου x είναι το technical dept ratio.
- Reliability Remediation Effort (Προσπάθεια Αποκατάστασης Αξιοπιστίας): $\boldsymbol{f(x) = 1 - \frac{x}{60 \times 8 \times NCLOC \times 0.06}}$ όπου x είναι η εκτιμώμενη προσπάθεια διόρθωσης των σφαλμάτων σε ώρες και NCLOC είναι ο αριθμός των γραμμών καθαρού κώδικα. Ουσιαστικά με τον δεύτερο όρο της αφαίρεσης υπολογίζουμε τη διαίρεση μεταξύ της προσπάθειας για την διόρθωση του κώδικα σε λεπτά (x/(60 * 8) και του συνολικού κόστους συγγραφής του κώδικα (είναι το κόστος ανά λεπτό της ώρας, 0.06, επί το συνολικό αριθμό των γραμμών κώδικα). Η διαίρεση αυτή μας δίνει επομένως την αναλογία του κόστους επιδιόρθωσης του κώδικα (σε σχέση με το συνολικό κόστος συγγραφής του κώδικα). Οπότε, η αφαίρεση μας δίνει την αναλογία (κόστους) καθαρού κώδικα χωρίς την ανάγκη επιδιορθώσεων. Επομένως, όσο πιο μεγάλη είναι αυτή η αναλογία, τόσο πιο μεγάλη θα είναι και η ωφέλεια.

- Bug Severity (Κρισιμότητα σφαλμάτων): $f(x) = \begin{cases} \mathbf{1.0}, \boldsymbol{no\ bugs} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{BL} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{CR}) + \mathbf{10}^{-\mathbf{2}} \times \boldsymbol{utf}(\boldsymbol{MA}) + \mathbf{10}^{-\mathbf{3}} \times \boldsymbol{utf}(\mathbf{MI}) + \mathbf{10}^{-\mathbf{4}} \times \boldsymbol{utf}(\boldsymbol{IN})\right)}, \boldsymbol{Blocker} > \mathbf{0} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{CR} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{MA}) + \mathbf{10}^{-\mathbf{2}} \times \boldsymbol{utf}(\boldsymbol{MI}) + \mathbf{10}^{-\mathbf{3}} \times \boldsymbol{utf}(\mathbf{IN})\right) + \mathbf{0.2}}, \boldsymbol{Critical} > \mathbf{0} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{MA} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{MI}) + \mathbf{10}^{-\mathbf{2}} \times \boldsymbol{utf}(\boldsymbol{IN})\right) + \mathbf{0.4}}, \boldsymbol{Major} > \mathbf{0} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{MI} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{IN})\right) + \mathbf{0.6}}, \boldsymbol{Minor} > \mathbf{0} \\ \mathbf{0.19} \times \frac{\mathbf{1}}{\boldsymbol{IN} + \mathbf{0.8}}, \boldsymbol{Info} > \mathbf{0} \end{cases}, \boldsymbol{utf}(\boldsymbol{x}) = \mathbf{1}\frac{\mathbf{1}}{\mathbf{1}/\boldsymbol{x}}$ όπου BL ή Blocker είναι ο αριθμός των σφαλμάτων αποκλεισμού (blocker bugs), CR ή Critical είναι ο αριθμός των κρίσιμων σφαλμάτων (critical bugs), MA ή Major είναι ο αριθμός των σημαντικών σφαλμάτων (major bugs), MI ή Minor είναι ο αριθμός των μη σημαντικών σφαλμάτων (minor bugs) και IN ή Info είναι ο αριθμός των ενημερωτικών σφαλμάτων (info bugs). Παρατηρούμε πως η ωφέλεια εξαρτάται πρώτα από την κρισιμότητα ενός σφάλματος και έπειτα από τον αριθμό των σφαλμάτων ανά βαθμό κρισιμότητας. Για παράδειγμα, όταν έχουμε τουλάχιστον ένα σφάλμα αποκλεισμού, τότε η ωφέλεια είναι από 0.2 έως σχεδόν 0.0 όπου η ωφέλεια μειώνεται εκθετικά ανάλογα με τον αριθμό των σφαλμάτων αποκλεισμού. Σε περίπτωση ιδίου αριθμού σφαλμάτων αποκλεισμού, η ωφέλεια έπειτα εξαρτάται από τον αριθμό των σφαλμάτων του επόμενου επιπέδου, κοκ. Πχ., όταν έχουμε δύο περιπτώσεις με ίδιο αριθμό σφαλμάτων αποκλεισμού όπου η πρώτη έχει 2 κρίσιμα σφάλματα και η δεύτερη 3, τότε η πρώτη θα έχει λίγο μεγαλύτερη ωφέλεια από την δεύτερη. Όταν δεν έχουμε καθόλου σφάλματα αποκλεισμού αλλά κρίσιμα σφάλματα, μεταβαίνουμε στο επόμενο επίπεδο ωφέλειας, μεταξύ 0.4 και 0.2. Πάλι εδώ η ωφέλεια θα εξαρτάται πρωτίστως από τον αριθμό των κρίσιμων σφαλμάτων και έπειτα από τον αριθμό των άλλων ειδών σφαλμάτων.
- Cyclomatic Complexity (Κυκλωματική Πολυπλοκότητα): $f(x) = \mathbf{1} - \frac{x}{NCLOC}$ όπου x είναι η κυκλωματική πολυπλοκότητα και NCLOC είναι ο αριθμός των γραμμών κώδικα.
- Cognitive Complexity (Γνωστική Πολυπλοκότητα): $f(x) = \mathbf{1} - \frac{x}{NCLOC}$ όπου x είναι η γνωστική πολυπλοκότητα και NCLOC είναι ο αριθμός των γραμμών κώδικα.
- Vulnerability Severity (Κρισιμότητα Τρωτοτήτων): $f(x) = \begin{cases} \mathbf{1.0}, \boldsymbol{no\ vulnerabilities} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{BL} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{CR}) + \mathbf{10}^{-\mathbf{2}} \times \boldsymbol{utf}(\boldsymbol{MA}) + \mathbf{10}^{-\mathbf{3}} \times \boldsymbol{utf}(\mathbf{MI}) + \mathbf{10}^{-\mathbf{4}} \times \boldsymbol{utf}(\boldsymbol{IN})\right)}, \boldsymbol{Blocker} > \mathbf{0} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{CR} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{MA}) + \mathbf{10}^{-\mathbf{2}} \times \boldsymbol{utf}(\boldsymbol{MI}) + \mathbf{10}^{-\mathbf{3}} \times \boldsymbol{utf}(\mathbf{IN})\right) + \mathbf{0.2}}, \boldsymbol{Critical} > \mathbf{0} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{MA} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{MI}) + \mathbf{10}^{-\mathbf{2}} \times \boldsymbol{utf}(\boldsymbol{IN})\right) + \mathbf{0.4}}, \boldsymbol{Major} > \mathbf{0} \\ \mathbf{0.2} \times \frac{\mathbf{1}}{\mathbf{MI} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf}(\boldsymbol{IN})\right) + \mathbf{0.6}}, \boldsymbol{Minor} > \mathbf{0} \\ \mathbf{0.19} \times \frac{\mathbf{1}}{\boldsymbol{IN} + \mathbf{0.8}}, \boldsymbol{Info} > \mathbf{0} \end{cases}, \boldsymbol{utf}(\boldsymbol{x}) =$

$\mathbf{1}\frac{\mathbf{1}}{\mathbf{1}/\boldsymbol{x}}$ – υπολογίζεται παρόμοια με την Κρισιμότητα Σφαλμάτων με βάση τον βαθμό της κρισιμότητας των τρωτοτήτων που έχουν εντοπιστεί καθώς και τον αριθμό τους.

- Hotspot priority: $\boldsymbol{f(x)} = \begin{cases} \mathbf{1.0},\ \boldsymbol{no\ hotspots} \\ \mathbf{0.33} \times \frac{\mathbf{1}}{\mathbf{H} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf(M)} + \mathbf{10}^{-\mathbf{2}} \times \boldsymbol{utf(L)}\right)},\ \boldsymbol{High} > \mathbf{0} \\ \mathbf{0.33} \times \frac{\mathbf{1}}{\mathbf{M} \times \left(\mathbf{1}+\mathbf{10}^{-\mathbf{1}} \times \boldsymbol{utf(L)}\right) + \mathbf{0.33}},\ \boldsymbol{Medium} > \mathbf{0} \\ \mathbf{0.33} \times \frac{\mathbf{1}}{\boldsymbol{L} + \mathbf{0.66}},\ \boldsymbol{Low} > \mathbf{0} \end{cases}$ – υπολογίζεται πάλι παρόμοια με την Κρισιμότητα Σφαλμάτων & Τρωτοτήτων ανάλογα με το επίπεδο προτεραιότητας των hotspots που έχουν εντοπιστεί και τον αριθμό τους. Η ή High είναι ο αριθμός των hotspot υψηλής προτεραιότητας, M ή Medium είναι ο αριθμός των hotspot μέτριας προτεραιότητας και L ή Low είναι ο αριθμός των hotspot χαμηλής προτεραιότητας.
- Security Remediation Effort: $\boldsymbol{f(x)} = \mathbf{1} - \frac{\boldsymbol{x}}{\boldsymbol{NCLOC} \times \mathbf{0.6}}$ όπου x είναι η προσπάθεια αποκατάστασης της ασφάλειας. Υπολογίζεται παρόμοια με τον Βαθμό Αποκατάστασης Αξιοπιστίας με την διαφορά πως η τιμή της x είναι αμέσως σε λεπτά και όχι σε ώρες.

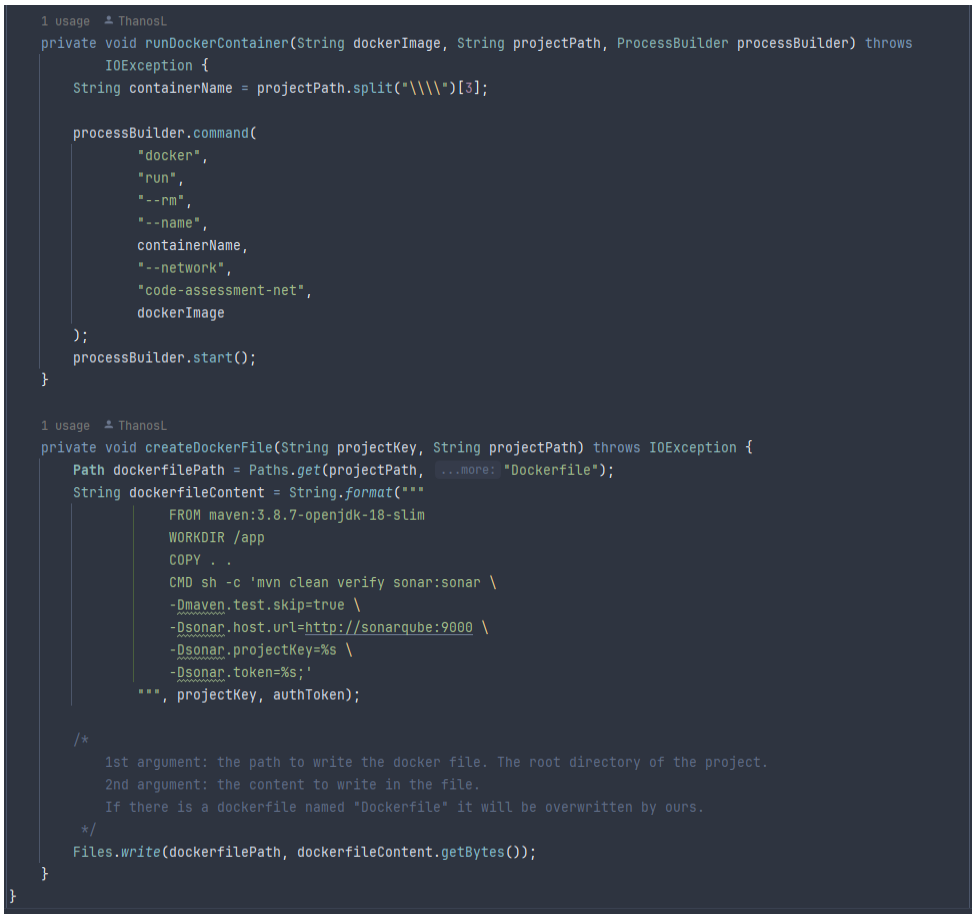
DockerService (Υπηρεσία Docker)

```java
@Service
public class DockerService {
    /*
        This is the access token for SonarQube
     */
    @Value("squ_9419ed961f6410e3bd558cafb7c9fb0eb72a5a61")
    private String authToken;

    2 usages  ThanosL
    public String createLinguistContainer(String path) {
        ProcessBuilder processBuilder = new ProcessBuilder();
        processBuilder.command(
                "docker",
                "run",
                "--rm",
                "-v",
                path + ":/code",
                "linguist",
                "github-linguist",
                "/code"
        );

        StringBuilder reportBuilder = new StringBuilder();

        try {
            Process process = processBuilder.start();
            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            String line;

            while ((line = reader.readLine()) != null) {
                reportBuilder.append(line).append("\n");
            }
        } catch (IOException ioe) {
            throw new ServerErrorException("The server encountered an internal error and was unable to complete your " +
                    "request. Please try again later");
        }

        return reportBuilder.toString();
    }
```

Εικόνα 27. Η κλάση DockerService

```java
public void analyzeMavenProject(String projectKey, String projectPath) throws
        IOException,
        InterruptedException {
    ProcessBuilder processBuilder = new ProcessBuilder();

    createDockerFile(projectKey, projectPath);
    String dockerImage = buildDockerImage(projectPath, processBuilder);
    runDockerContainer(dockerImage, projectPath, processBuilder);
}

1 usage  ThanosL
private String buildDockerImage(String projectPath, ProcessBuilder processBuilder) throws
        IOException,
        InterruptedException {
    /*
        The project key is the UUID that was assigned to the project folder. Splitting with the escape character
        which is also the file separator in Windows.
     */
    String dockerImage = projectPath.split("\\\\")[3];

    processBuilder.command(
            "docker",
            "build",
            "-t",
            dockerImage,
            "."
    );

     /*
        Setting the directory of the command execution to be the projects directory, so we can use .
     */
    processBuilder.directory(new File(projectPath));
    Process process = processBuilder.start();
    process.waitFor();

    return dockerImage;
}
```

Εικόνα 28. Η κλάση DockerService (συνέχεια)



```
    1 usage  ThanosL
    private void runDockerContainer(String dockerImage, String projectPath, ProcessBuilder processBuilder) throws
            IOException {
        String containerName = projectPath.split("\\\\")[3];

        processBuilder.command(
                "docker",
                "run",
                "--rm",
                "--name",
                containerName,
                "--network",
                "code-assessment-net",
                dockerImage
        );
        processBuilder.start();
    }

    1 usage  ThanosL
    private void createDockerFile(String projectKey, String projectPath) throws IOException {
        Path dockerfilePath = Paths.get(projectPath,  ...more: "Dockerfile");
        String dockerfileContent = String.format("""
                    FROM maven:3.8.7-openjdk-18-slim
                    WORKDIR /app
                    COPY . .
                    CMD sh -c 'mvn clean verify sonar:sonar \
                    -Dmaven.test.skip=true \
                    -Dsonar.host.url=http://sonarqube:9000 \
                    -Dsonar.projectKey=%s \
                    -Dsonar.token=%s;'
                """, projectKey, authToken);

        /*
            1st argument: the path to write the docker file. The root directory of the project.
            2nd argument: the content to write in the file.
            If there is a dockerfile named "Dockerfile" it will be overwritten by ours.
         */
        Files.write(dockerfilePath, dockerfileContent.getBytes());
    }
}
```

Εικόνα 29. Η κλάση DockerService (δεύτερη συνέχεια)

Η κλάση DockerService είναι υπεύθυνη για τις αλληλεπιδράσεις με τη μηχανή ενορχήστρωσης δοχείων (containers) Docker. Πιο συγκεκριμένα ξεκινάει ένα δοχείο με βάση την εικόνα δοχείου (container image) του GitHub Linguist, που ενσωματώνει το εργαλείο που χρησιμοποιούμε για την αναγνώριση των γλωσσών προγραμματισμού.

Για Maven έργα ο κώδικας πρέπει να μεταγλωττιστεί πρώτου γίνει η ανάλυση. Επομένως, η κλάση DockerService αναλαμβάνει επίσης να δημιουργήσει ένα περιβάλλον απομόνωσης για την αποφυγή παραβιάσεων ασφάλειας. Αυτή δημιουργεί ένα DockerFile στον ριζικό φάκελο του έργου λογισμικού (δηλ. του αποθετηρίου) προς ανάλυση με βάση το οποίο θα κατασκευαστεί μια εικόνα από την οποία θα δημιουργηθεί ένα σχετικό, κατάλληλο δοχείο. Το δοχείο αυτό θα εκτελέσει την ενότητα (module) του Sonarqube (έναν δηλαδή πελάτη Sonarqube) για το Maven ώστε να πραγματοποιηθεί η στατική ανάλυση. Τοποθετώντας το δοχείο αυτό στο ίδιο δίκτυο που εκτελείται ο εξυπηρετητής του SonarQube, καθίσταται δυνατή η αποστολή των αποτελεσμάτων της ανάλυσης από τον πελάτη στον εξυπηρετητή του Sonarqube.

SonarService

```java
@Service
@RequiredArgsConstructor
public class SonarService {
    @Value("squ_9419ed961f6410e3bd558cafb7c9fb0eb72a5a01")
    private String authToken;
    @Value("http://localhost:9000/api")
    private String baseUrl;
    1 usage
    private static final Logger LOG = LoggerFactory.getLogger(SonarService.class);
    4 usages
    private static final String SERVER_ERROR_MSG = "The server encountered an internal error and was unable to " +
            "complete your request. Please try again later";

    1 usage  ThanosL
    public void analyzeProject(String projectKey, String projectDirectory) {
        ProcessBuilder processBuilder = new ProcessBuilder();
        processBuilder.command(
                "sonar-scanner.bat",
                "-Dsonar.projectKey=" + projectKey,
                "-Dsonar.sources=.",
                "-Dsonar.host.url=http://localhost:9000",
                "-Dsonar.token=" + authToken
        );

        /*
            Setting the directory of the command execution to be the projects directory, so we can use
            sources=.
         */
        try {
            processBuilder.directory(new File(projectDirectory));
            Process process = processBuilder.start();

            /*
                Each process builder has an associated output buffer. We have to keep reading from those buffers, as
                the process writes enough data to them, the buffers can fill up, and the process will block, waiting for
                space to become available.
             */
            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                LOG.info(line);
            while ((line = reader.readLine()) != null) {
                LOG.info(line);
            }
            /*
                Waiting for the analysis to end, not to upload the analysis to the server.
             */
            process.waitFor();
        } catch (IOException ioe) {
            throw new ServerErrorException(SERVER_ERROR_MSG);
        } catch (InterruptedException ie) {
            Thread.currentThread().interrupt();
            throw new ServerErrorException(SERVER_ERROR_MSG);
        }
    }

    1 usage  ThanosL
    public AnalysisReport fetchAnalysisReport(String projectKey) {
        RestTemplate restTemplate = new RestTemplate();
        HttpHeaders headers = new HttpHeaders();
        headers.set("Authorization", "Bearer " + authToken);
        HttpEntity<String> entity = new HttpEntity<>( body: "parameters", headers);

        /*
            Wait for the key to be created on the server.
         */
        while (!projectExists(restTemplate, entity, projectKey)) {
            await().pollDelay(Duration.ofSeconds(8)).until(() -> true);
        }

        /*
            Wait for the project to be uploaded.
         */
        await().pollDelay(Duration.ofSeconds(8)).until(() -> true);

        IssuesReport issuesReport = fetchIssues(restTemplate, entity, projectKey);
        HotspotsReport hotspotsReport = fetchHotspots(restTemplate, entity, projectKey);
        Map<QualityMetric, Double> qualityMetricReport = getQualityMetrics(
                restTemplate,
                entity,
                projectKey);
```

Εικόνα 30. Η κλάση SonarService

```
        return new AnalysisReport(issuesReport, hotspotsReport, qualityMetricReport);
    }

    1 usage  ThanosL
    private boolean projectExists(RestTemplate restTemplate, HttpEntity<String> entity, String projectKey) {
        String projectUrl = String.format("%s/projects/search?projects=%s", baseUrl, projectKey);

        /*
            The request to check if a project is on the server does not return 404 but an empty components array if the
             project is not on the sonarqube server.
         */
        try {
            ResponseEntity<String> response = restTemplate.exchange(
                    projectUrl,
                    HttpMethod.GET,
                    entity,
                    String.class);

            if (response.getStatusCode() != HttpStatus.OK) {
                throw new ServerErrorException(SERVER_ERROR_MSG);
            }

            JSONObject responseObject = new JSONObject(response.getBody());
            JSONArray components = responseObject.getJSONArray( name: "components");

            return components.length() != 0;
        } catch (JSONException e) {
            throw new ServerErrorException(SERVER_ERROR_MSG);
        }
    }
private IssuesReport fetchIssues(RestTemplate restTemplate, HttpEntity<String> entity, String projectKey) {
    int page = 1;
    int pageSize = 100;
    int totalNumberOfPages = Integer.MAX_VALUE;
    IssuesReport issuesReport = null;
    IssuesReport tmp;

    /*
        SonarQube uses pagination. It returns 100 issues per request. If we have more than a 100 we need to perform
        more than 1 request to get all the issues from a single report.
     */
    while (page <= totalNumberOfPages) {
        String issuesUrl = String.format(
                "%s/issues/search?componentKeys=%s&p=%d&ps=%d",
                baseUrl,
                projectKey,
                page,
                pageSize);
        ResponseEntity<IssuesReport> response = restTemplate.exchange(
                issuesUrl,
                HttpMethod.GET,
                entity,
                IssuesReport.class);

        tmp = response.getBody();

        if (issuesReport != null && tmp != null) {
            issuesReport.getIssues().addAll(tmp.getIssues());
        } else {
            issuesReport = tmp;
        }

        totalNumberOfPages = (issuesReport.getIssues().size() / pageSize) + 1;
        page++;
    }

    return issuesReport;
}
```

Εικόνα 31. Η κλάση SonarService (συνέχεια)

Η κλάση SonarService είναι υπεύθυνη για την ανάλυση των έργων λογισμικού και την αλληλεπίδραση με τον εξυπηρετητή του SonarQube και τον SonarScanner (πρόγραμμα που

εκτελεί την στατική ανάλυση). Αρχικά καλείται ο Scanner για την ανάλυση τοπικά και τα αποτελέσματα στέλνονται στον εξυπηρετητή. Στην συνέχεια η κλάση αναλαμβάνει την ανάκτηση των αποτελεσμάτων της ανάλυσης από τον εξυπηρετητή σε 2 φάσεις. Στην 1η φάση ανακτάμε τα διάφορα ζητήματα (σφάλματα, οσμές κώδικα και τρωτότητες) και στην 2η τα Security Hotspots. Στην συνέχεια γίνεται αίτηση για την ανάκτηση μετρικών, όπως duplication, και reliability remediation effort, για την μετέπειτα εφαρμογή των συναρτήσεων ωφέλειας πάνω στις τιμές των μετρικών αυτών για το έργο λογισμικού υπό ανάπτυξη.

4.4. Δοκιμή

Δύο είδη δοκιμών πραγματοποιήθηκαν:

- Unit Testing (Δοκιμή Μονάδων): Εστίασε στα 2 ακόλουθα σημεία του κώδικα του συστήματος:
    - Στην επιχειρησιακή λογική, όπου γράφθηκαν δοκιμές για την επικύρωση των αιτήσεων, την κατανομή των βαρών, το φιλτράρισμα των αποθετηρίων με βάση τους περιορισμούς, την αναγνώριση γλωσσών προγραμματισμού, την αποθήκευση και διαγραφή ανάλυσης, την ανάκτηση αποθετηρίων, περιορισμών και προτιμήσεων συγκεκριμένης ανάλυσης, την αντιστοίχιση των οντοτήτων σε Java αντικείμενα, την αποσειριακοποίηση (deserialization) του σώματος της αίτησης (request body) σε Java αντικείμενο, την ταυτοποίηση των χρηστών και την εφαρμογή των συναρτήσεων ωφέλειας στις μετρικές.
    - Στο σώμα της απάντησης και στον κωδικό/κατάσταση της απάντησης (response status/code), όπου γράφθηκαν δοκιμές για να βεβαιωθούμε ότι πάντοτε επιστρέφεται το σωστό σώμα και κωδικός/κατάσταση στην απάντηση οποιασδήποτε αίτησης. Σε αυτό το είδος των δοκιμών καλύπτει τις περιπτώσεις οπού μη-ταυτοποιημένοι χρήστες προσπαθούν να έχουνε πρόσβαση σε πόρους του συστήματος.

  Για τη συγγραφή και διενέργεια των προαναφερόμενων δοκιμών χρησιμοποιήθηκαν Junit5 και AssertJ. Έγιναν mock οι εξαρτήσεις όπου χρειαζότανε χρησιμοποιώντας το Mockito.

- Integration Testing (Δοκιμή Ενοποίησης): Εφαρμόστηκε η εξής διαμόρφωση περιβάλλοντος (setup). Μια βάση δεδομένων που να είναι αντίγραφο της βάσης στην οποία έγινε η ανάπτυξη της εφαρμογής ώστε οι δοκιμές να είναι αξιόπιστες, επαναλαμβανόμενες και απομονωμένες από το περιβάλλον ανάπτυξης καθώς και έναν test email server για τα emails επικύρωσης. Για την βάση επιλέχθηκαν test containers (δοχεία δοκιμών), τα οποία μας επιτρέπουν, σε συνδυασμό με το Junit5, την δημιουργία βάσης σε 1 δοχείο Docker ενώ την διαχείριση του κύκλου ζωής του δοχείου την αναλαμβάνει το Junit5. Αφού ξεκινήσει το δοχείο, το Flyway αναλαμβάνει να μεταναστεύσει το σχήμα στην βάση μέσα στο δοχείο εξασφαλίζοντας ότι το περιβάλλον δοκιμής είναι παρόμοιο με το περιβάλλον ανάπτυξης. Χρησιμοποιήθηκε ο Greenmail ως test email server διότι μας επιτρέπει εύκολα να ελέγξουμε εάν τα email αποστέλλονται και λαμβάνονται σωστά και να επιθεωρήσουμε το περιεχόμενο τους.

- Οι δοκιμές ενοποίησης εστίασαν στα happy paths (θετικά μονοπάτια) για τις ενέργειες που μπορεί να κάνει ένας χρήστης εκτός της αίτησης ανάλυσης, δηλαδή ενημέρωση κωδικού, ενημέρωση email, ενημέρωση προφίλ και διαγραφή λογαριασμού.

Με το συγκεκριμένο setup δεν ήταν αρκετό να γίνουνε δοκιμές πάνω στην ανάλυση αποθετηρίων ακόμα και εάν χρησιμοποιούσαμε wiremock που θα μας επέτρεπε να έχουμε mocks για τις απαντήσεις από το SonarQube Api. Το σύστημα έχει άλλες εξωτερικές εξαρτήσεις, όπως SonarScanner, GitHub και Docker. Σε αυτή την περίπτωση, ένα κατάλληλο περιβάλλον δοκιμής (test environment) θα μας επέτρεπε να πραγματοποιήσουμε τέτοιες δοκιμές.

4.5. Ικανοποίηση Απαιτήσεων

Απαίτηση	Βαθμός Ικανοποίησης	Δικαιολόγηση
1. Το σύστημα θα πρέπει να επιτρέπει στους χρήστες να υποβάλλουν μια λίστα αποθετηρίων, με προαιρετικούς περιορισμούς ποιότητας κώδικα και προτιμήσεις χαρακτηριστικών.	Πλήρως Ικανοποιημένη	
2. Ο κώδικας κάθε αποθετηρίου αποθηκεύεται τοπικά και αξιολογείται χρησιμοποιώντας επιλεγμένα εργαλεία ανάλυσης. Τα εργαλεία παρέχουν τιμές για διάφορα χαρακτηριστικά ποιότητας και ασφάλειας, που χρησιμοποιούνται για το φιλτράρισμα και την κατάταξη των αποθετηρίων.	Πλήρως Ικανοποιημένη	
3. Μετά το φιλτράρισμα και την αποτίμηση των αποθετηρίων, αυτά θα πρέπει να εμφανίζονται σε ταξινομημένη μορφή.	Πλήρως Ικανοποιημένη	
4. Οι χρήστες μπορούν να ενημερώνουν δυναμικά τα κριτήριά τους και τα αποτελέσματα θα ανανεώνονται αμέσως για να αντικατοπτρίζουν τις	Πλήρως Ικανοποιημένη	

αλλαγές.		
5. Το σύστημα θα πρέπει να επιτρέπει την διαχείριση των χρηστών. Η διαχείριση των χρηστών περιλαμβάνει: εγγραφή χρήστη, διαγραφή χρήστη, τροποποίηση στοιχειών χρήστη, τροποποίηση κωδικού χρήστη, επαναφορά κωδικού χρήστη, ταυτοποίηση χρήστη.	Πλήρως Ικανοποιημένη	
6. Η πρώτη παραγωγή αποτελεσμάτων ταξινόμησης δεν πρέπει να παίρνει πάνω από 30 δευτερόλεπτα. Οι επόμενες παραγωγές αποτελεσμάτων λόγω της ανανέωσης των περιορισμών/προτιμήσεων του χρήστη για την ίδια λίστα αποθετηρίων δεν θα πρέπει να ξεπερνά τα 10 δευτερόλεπτα,	Μερικώς Ικανοποιημένη	Η διαδικασία ανάλυσης ξεπερνά τα 30 δευτερόλεπτα
7. Θα πρέπει να υποστηρίζονται τουλάχιστον 20 ταυτόχρονοι χρήστες,	Πλήρως Ικανοποιημένη	
8. Η διεπαφή χρήσης πρέπει να είναι χρηστική και αποκρίσιμη κατά μήκος διαφορετικών συσκευών,	Μερικώς Ικανοποιημένη	Η τρέχουσα έκδοση της εφαρμογής δεν είναι multi-platform (ως προς το μέρος της διεπαφής χρήστη που διαθέτει)
9. Η διεπαφή χρήσης πρέπει να έχει ενιαίο look-n-feel,	Μερικώς Ικανοποιημένη	Απειρία στο κομμάτι του web design
10. Διαχωρισμένο API (Back-End) και Front-End,	Πλήρως Ικανοποιημένη	
11. Δοχειοποίηση της εφαρμογής,	Μη ικανοποιημένη	Το επόμενο κομμάτι που θα υλοποιηθεί
12. Υλοποίηση backend σε REST με επίπεδο ωριμότητας 3,	Πλήρως Ικανοποιημένη	
13. Κατάλληλος χειρισμός λαθών/εξαιρέσεων με παροχή αυτο-επεξηγούμενων μηνυμάτων,	Πλήρως Ικανοποιημένη	
14. Κατάλληλη επικύρωση φορμών/δεδομένων τόσο στην πλευρά του πελάτη όσο	Πλήρως Ικανοποιημένη	

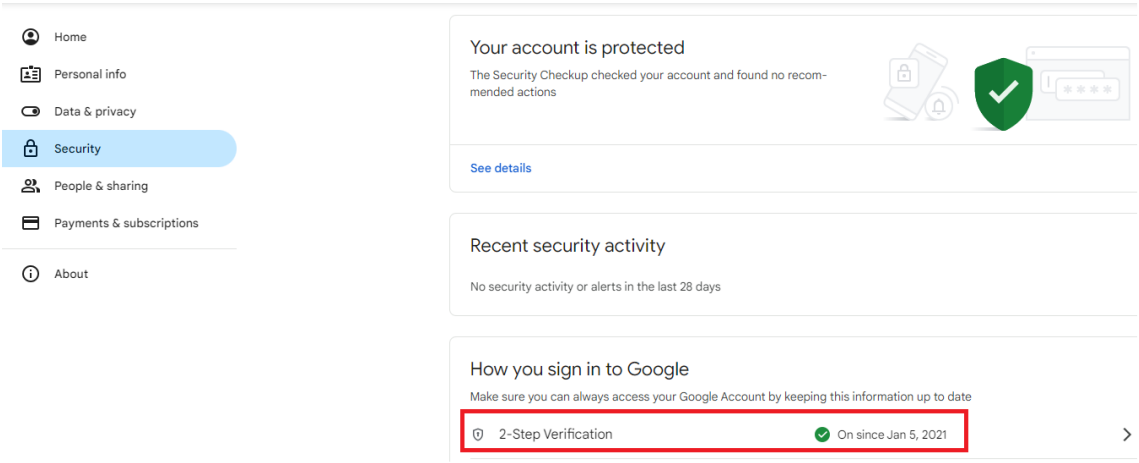
και του εξυπηρετητή,		
15. Υποστήριξη TLS & προστασία CSRF όσον αφορά την ασφάλεια της εφαρμογής ιστού,	Μερικώς Ικανοποιημένη	Υπάρχει υποστήριξη για CSRF αλλά όχι για TLS. Το Let's Encrypt δίνει SSL certificates σε public domains

5. Εγκατάσταση Εφαρμογής

Για την εγκατάσταση της εφαρμογής θα πρέπει να γίνουν τα εξής βήματα:

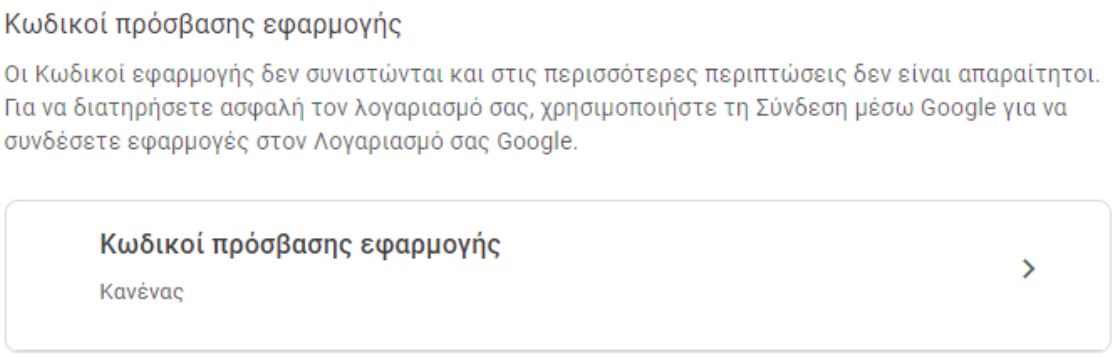
- Ο κώδικας για τον server βρίσκεται εδώ: "https://github.com/ThLentzas/code-assessment-server"
- Ο κώδικας για τον client βρίσκεται εδώ: "https://github.com/ThLentzas/code-assessment-client"
- Εγκατάσταση Node.js, έκδοση 18 ή μεταγενέστερη, και npm
- Αφού βεβαιωθείτε ότι Node.js και npm έχουν εγκατασταθεί σωστά, για την εγκατάσταση της Angular εκτελείτε την εντολή **"npm install -g @angular/cli "** https://angular.io/guide/setup-local
- Με τις επόμενες 3 εντολές θα εγκαταστήσετε πακέτα για notifications icons και materials: **npm install angular-notifier, npm i @ng-icons/core @ng-icons/bootstrap-icons, ng add @angular/material**
- Εγκατάσταση της Java έκδοση 17 ή μεταγενέστερη
- Εγκατάσταση Docker
- Εγκατάσταση της PostgresQL έκδοση 15.2 ή μεταγενέστερη. Δημιουργία βάσης με όνομα code_assessment.
- Εγκατάσταση του SonarScanner. https://docs.sonarsource.com/sonarqube/9.9/analyzing-source-code/scanners/sonarscanner/
- Η εγκατάσταση του SonarQube server θα γίνει σε docker. Αφού βεβαιωθούμε ότι έχει γίνει σωστά η εγκατάσταση του docker εκτελούμε την εντολή.
    - **"docker run -d --name sonarqube -p 9000:9000 sonarqube"**. Θα χρησιμοποιηθεί το latest version της εικόνας που βρίσκεται στο DockerHub. https://hub.docker.com/_/sonarqube
- Η αναγνώριση γλωσσών προγραμματισμού γίνεται με το GitHub Linguist. Πρέπει πρώτα να δημιουργηθεί ένα DockerFile. Βρίσκουμε το περιεχόμενο του αρχείου. https://github.com/github-linguist/linguist/blob/master/Dockerfile. Αφού δημιουργηθεί το αρχείο εκτελείται η εντολή "**docker build -t linguist .**" Πριν εκτελέσετε την εντολή βεβαιωθείτε ότι βρίσκετε στον ίδιο φάκελο με το DockerFile.

- Η εφαρμογή στέλνει σχετικά email επικύρωσης για συγκεκριμένες λειτουργίες. Για αυτό χρειάζεται να δημιουργήσετε μια email διεύθυνση από την οποία θα στέλνονται τα σχετικά emails και στην συνέχεια θα γίνει παραμετροποίηση της εφαρμογής ώστε να στέλνει τα emails μέσω αυτής της διεύθυνσης. Το παρακάτω παράδειγμα αφορά ένα λογαριασμό gmail.



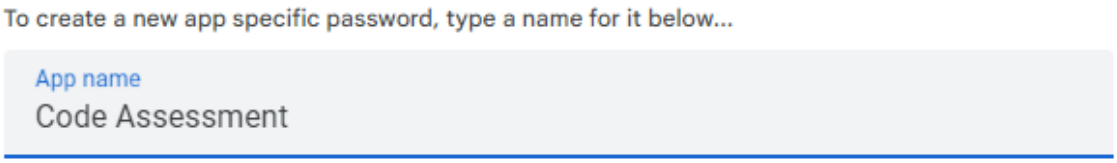
Εικόνα 32. Παραμετροποίηση εφαρμογής με λογαριασμό gmail

  - Στο Security tab από το google λογαριασμό σας βεβαιωθείτε ότι έχετε enable 2FA (ταυτοποίηση 2 παραγόντων).
  - Αφού το επιλέξετε, θα σας ζητηθεί ο κωδικός πρόσβασης και στο τέλος της σελίδας που θα μεταβείτε επιλέξτε την παρακάτω επιλογή



Εικόνα 33. Ρύθμιση κωδικών πρόσβασης εφαρμογής

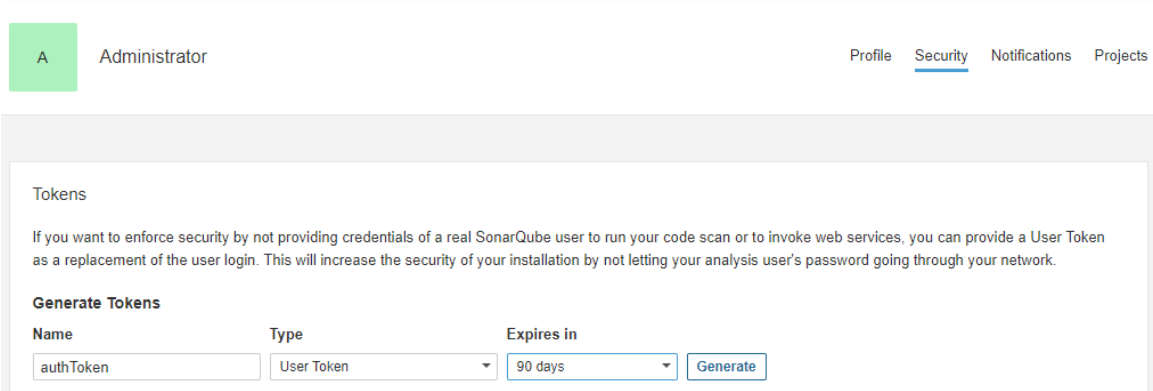
- Δώστε όποιο όνομα επιθυμείτε και στην συνέχεια θα δημιουργηθεί ένας 16ψήφιος κωδικός.



Εικόνα 34. Παροχή ονόματος εφαρμογής

- Στο application.yaml αρχείο θα αντικατασταθούνε τα εξής properties με το email που δημιουργήσατε και το password που σας δόθηκε **spring.mail.username, spring.mail.password**

- Για την αποθήκευση των αποθετηρίων τοπικά χρειάζεται να δημιουργηθεί ένας φάκελο και να αντικατασταθεί στο application.yaml το projects.base-directory με το absolute path του φακέλου
- Για την ανάλυση των αποθετηρίων θα χρειαστείτε ένα authentication token. Αφού βεβαιωθείτε ότι ο SonarQube server τρέχει και είναι προσβάσιμος στο "http://localhost:9000" συνδεθείτε με τα default credentials. https://docs.sonarsource.com/sonarqube/latest/instance-administration/security/ **Login: admin, Password: admin.** Πάνω αριστερά επιλέγοντας "My Account" και στην συνέχεια "Security" δημιουργούμε το σχετικό token. Η τιμή του token θα είναι στην μορφή squ_xxxx και θα αντικαταστήσει την τιμή sonar.token στο application.yaml



Εικόνα 35. Δημιουργία authentication token

- Στο root directory του εξυπηρετητή εκτελέστε την εντολή "mvn spring-boot:run" και θα βρίσκετε στο "http://localhost:8080"
- Στο root directory του πελάτη (client) εκτελέστε την εντολή "ng serve" και θα βρίσκετε στο "http://localhost:4200"

6. Συμπεράσματα

6.1. Συνεισφορά της διπλωματικής

Η κύρια συνεισφορά της προτεινόμενης εφαρμογής είναι στο κομμάτι του αυτοματισμού και της ευελιξίας στην ανάλυση κώδικα. Επιτρέπει εξατομικευμένες αξιολογήσεις που ευθυγραμμίζονται με τις ατομικές ανάγκες, διασφαλίζοντας ότι οι χρήστες λαμβάνουν αποτελέσματα που συνάδουν με τα κριτήρια τους. Οι χρήστες μπορούν να τροποποιούν και να βλέπουν τις ενημερωμένες αξιολογήσεις σε πραγματικό χρόνο, ενισχύοντας ένα δυναμικό περιβάλλον αξιολόγησης. Με την απλή εισαγωγή αποθετηρίων, το σύστημα αναγνωρίζει αυτόματα τις σχετικές γλώσσες προγραμματισμού και τα εργαλεία κατασκευής, απλοποιώντας τη διαδικασία και μειώνοντας τα πιθανά σφάλματα. Αυτά τα χαρακτηριστικά συνδυάζονται σε μια εμπειρία με επίκεντρο τον χρήστη, που χαρακτηρίζεται από ακρίβεια, αποτελεσματικότητα και ευκολία στη χρήση.

6.2. Προσωπική Εμπειρία

Μέσω της διπλωματικής έγινε κατανοητό πόσο σημαντικό ρόλο παίζει η στατική ανάλυση κώδικα στην ανάπτυξη λογισμικού. Εξετάζοντας συστηματικά τον κώδικα χωρίς την εκτέλεσή του, η στατική ανάλυση μπορεί να εντοπίσει από απλά συντακτικά σφάλματα έως πολύπλοκα τρωτά σημεία ασφαλείας βελτιώνοντας την ποιότητα και την ασφάλεια του κώδικα και διασφαλίζοντας τη συμμόρφωση με τα τις βέλτιστες πρακτικές προγραμματισμού. Επιπλέον, εντοπίζοντας προβλήματα νωρίς στον κύκλο ανάπτυξης, αυτό μπορεί να μειώσει σημαντικά το κόστος και την προσπάθεια που απαιτούνται για διορθώσεις μεταγενέστερων σταδίων, καθιστώντας την στατική ανάλυση ένα ανεκτίμητο εργαλείο για αποτελεσματική και αξιόπιστη δημιουργία λογισμικού.

Κατά την ανάπτυξη της εφαρμογής, δοκιμάσθηκαν διάφορα εργαλεία για την ανάλυση κώδικα. Η ενσωμάτωση τους ήταν μια ιδιαίτερα επίπονη διαδικασία καθώς η τεκμηρίωση (documentation) στην πλειοψηφία των εργαλείων ήτανε ανύπαρκτη, σχεδόν ανύπαρκτη ή κακογραμμένη. Επομένως, έγινε κατανοητό πόσο σημαντικό ρόλο παίζει η σωστή τεκμηρίωση (documentation). Χωρίς σαφή και ολοκληρωμένη τεκμηρίωση, υπάρχει δυσκολία κατανόησης των δυνατότητων του εργαλείου, οδηγώντας σε αναποτελεσματικότητα, σφάλματα και πιθανές καθυστερήσεις στο έργο. Επιπλέον, η απουσία επαρκούς τεκμηρίωσης μπορεί να αυξήσει την καμπύλη εκμάθησης (learning curve), να εμποδίσει την αντιμετώπιση προβλημάτων και να μειώσει τη συνολική εμπιστοσύνη στο εργαλείο.

6.3. Μελλοντικές βελτιώσεις

Σε 3 τομείς θα γίνει εστίαση για μελλοντικές βελτιώσεις της εφαρμογής: στην δοχειοποίηση (containerization), τη σχεδίαση του UI και το υλικό (hardware).

> **Δοχειοποίηση:** Είναι φανερό ότι η εγκατάσταση της εφαρμογής αποτελεί πολύπλοκη διαδικασία. Βασικός στόχος είναι η απλοποίηση της με την δοχειοποίηση της εφαρμογής, μειώνοντας δραστικά τον χρόνο εγκατάστασης και ελαχιστοποιώντας τα προβλήματα συμβατότητας. Μελλοντικά, η δοχειοποίηση της εφαρμογής θα βοηθήσει και στο κομμάτι της επεκτασιμότητας.

Σχεδίαση UI: Η βελτίωση της διεπαφής αυξάνει την ικανοποίηση των χρηστών, τους καθοδηγεί προς τις επιθυμητές ενέργειες ευκολότερα, διευκολύνει την πλοήγηση και το χειρισμό της εφαρμογής, μειώνοντας έτσι τα λάθη και την πιθανή απογοήτευση.

Υλικό (hardware): Η εφαρμογή μας αντιμετώπισε περιορισμούς πόρων στο κομμάτι του υλικού, ιδιαίτερα όταν ασχολούμαστε με πολλαπλά νήματα (multithreading). Λόγω περιορισμένων πόρων CPU και μνήμης, η απόδοση της εφαρμογής μας υποβαθμίστηκε κατά την ταυτόχρονη εκτέλεση πολλών νημάτων. Στο μέλλον θα πρέπει να γίνει επένδυση σε καλύτερης ποιότητας υλικό που θα βελτιώνει την απόδοση εφαρμογής.

Αναφορές

[1] «The Importance of Code Quality» https://medium.com/emblatech/the-importance-of-code-quality-ac7afa598c0d

[2] «Importance of Code Quality» https://www.tatvasoft.com/blog/importance-code-quality/

[3] «What is software?» https://www.britannica.com/technology/software

[4] «Software Characteristics» https://www.dspmuranchi.ac.in/pdf/Blog/sw_characteristics.pdf

[5] «Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements» https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson

[6] «The Impact of Poor Software Quality in Business» https://www.apexgloballearning.com/blog/impact-poor-software-quality-business-infographic/

[7] «Learning from history: The case of Software Requirements Engineering» https://re-magazine.ireb.org/articles/learning-from-history-the-case-of-software-requirements-engineering/

[8] «ISO/IEC 9126-1:2001» https://www.iso.org/standard/22749.html

[9] «ISO/IEC 25010:2011» https://www.iso.org/standard/35733.html

[10] «Consortium for IT Software Quality (CISQ)» https://en.wikipedia.org/wiki/CISQ

[11] «What is code quality?» https://www.perforce.com/blog/sca/what-code-quality-overview-how-improve-code-quality

[12] «SEI CERT Coding Standards» https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

[13] «Software Maintainability: What it Means to Build Maintainable Software» https://www.sealights.io/software-quality/software-maintainability-what-it-means-to-build-maintainable-software/

[14] «Function points – comparison» https://www.scopemaster.com/blog/function-points/

[15] «A gentle introduction to static code analysis» https://wwç.infoworld.com/article/3705568/a-gentle-introduction-to-static-code-analysis.html

[16] «What is Static Code Analysis? Static Code Analysis Overview» https://re-magazine.ireb.org/learning-from-history-the-case-of-software-requirements-engineering/

[17] « What is Static Analysis Within CI/CD Pipelines?» https://cloudacademy.com/blog/what-is-static-analysis-within-ci-cd-pipelines/

[18] «Static Code Analysis» https://owasp.org/www-community/controls/Static_Code_Analysis#

[19] «What is Dynamic Analysis» https://www.checkpoint.com/cyber-hub/cloud-security/what-is-dynamic-code-analysis/

[20] «Dynamic Code Analysis» https://www.codium.ai/glossary/dynamic-code-analysis/

[21] «Dynamic Application security Testing(DAST)» https://www.invicti.com/blog/websecurity/dynamic-code-analysis-in-application-security-testing/

[22] «The Squale Quality Model» https://www.squale.org/quality-models-site/research-deliverables/WP1.3_Practices-in-the-Squale-Quality-Model_v2.pdf

[23] «FLOSSMetrics: Free/libre/Open source software metrics and benchmarking» https://www.researchgate.net/publication/224400655_FLOSSMetrics_FreeLibreOpen_Source_Software_Metrics

[24] «SonarQube» https://www.sonarsource.com/products/sonarqube/

[25] «Codacy» https://www.codacy.com/

[26] «OWASP ZAP» https://www.zaproxy.org/