ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Κατασκευή Web Service Μέτρησης Ποιότητας Λογισμικού»

Του φοιτήτη Κων/νου Τσιτούμη
Αρ. Μητρώου: 174859

Επιβλέπουσα
Ελβίρα Αρβανίτου

Ημερομηνία 1/3/2024

Κατασκευή Web Service Μέτρησης Ποιότητας Λογισμικού Κωδικός

Κωνσταντίνος Τσιτούμης
Ελβίρα Αρβανίτου

Ημερομηνία ανάληψης Δ.Ε. 27-03-2022
Ημερομηνία περάτωσης Δ.Ε. 27-03-2024

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητη Κωνσταντίνου Τσιτούμη που την εκπόνησε/αν. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης, ο συγγραφέας/δημιουργός εκχωρεί στο Διεθνές Πανεπιστήμιο της Ελλάδος άδεια χρήσης του δικαιώματος αναπαραγωγής, δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσης της εργασίας διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος. Η ανοικτή πρόσβαση στο πλήρες κείμενο της εργασίας, δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού, ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, πώληση, εμπορική χρήση, διανομή, έκδοση, μεταφόρτωση (downloading), ανάρτηση (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Στον Πατέρα και στην Μητέρα μου,

που με στήριξαν και μου έδωσαν την δυνατότητα να σπουδάσω αυτό που αγαπώ.

Πρόλογος

Στο πεδίο της πληροφορικής, νέες τεχνολογίες δημιουργούνται καθημερινά. Πέραν όμως από την ταχεία εξέλιξη των τεχνολογιών, ένας ικανός μηχανικός πληροφορικής πρέπει να διαθέτει σταθερές θεμελιώδεις γνώσεις. Ένα από τα θεμέλια αποτελεί ο τρόπος ανάπτυξης κώδικα. Με στόχο τη συνεχή εξέλιξη και την απορρόφηση νέων γνώσεων σε αυτόν τον τομέα ένα σημαντικό κίνητρο για την εκπόνηση της συγκεκριμένης πτυχιακής εργασίας είναι η γνώση γύρω από το τεχνικό χρέος, το οποίο αποτελεί ουσιώδες στοιχείο για τη δημιουργία επεκτάσιμων εφαρμογών. Ένα κλειδί εργαλείο που αντιμετωπίζει αυτό το πρόβλημα είναι το refactoring.

Η συνύπαρξη του refactoring και του τεχνικου χρέους ανοίγει για εμένα την ευκαιρία να σταθεροποιήσω τις γνώσεις μου στην δημιουργια οργανωμένου και επεκτάσιμου κώδικα και αποτελεί ένα πλαίσιο για την εξερεύνηση λεπτομερειών πανω σε αυτα τα δύο θέματα.

Ένας άλλος παράγοντας που επηρέασε την επιλογή μου είναι η εξερεύνηση των σύγχρονων τεχνολογιών όπως τα microservices, το Spring Boot και το Docker. Αυτή η εξερεύνηση δεν αποτελεί μόνο μέσο απόκτησης γνώσης, αλλά και στρατηγική επένδυση για το μέλλον μου ως μηχανικός πληροφορικής.

Με αυτήν την προσπάθεια, στοχεύω στο να συνδυάσω τη θεωρητική γνώση με την πρακτική εφαρμογή, αναπτύσσοντας ένα σύνολο δεξιοτήτων που όχι μόνο είναι σύγχρονες, αλλά και προορατικές για το μέλλον μου.

Περίληψη

Η παρούσα πτυχιακή εργασία αρχικά εστιάζει στην κατανόηση θεμελιωδών γνώσεων, όπως το τεχνικό χρέος, και στην εφαρμογή αποτελεσματικών λύσεων μέσω του refactoring. Ξεκινά με την εννοιολογική αποσαφίνιση του τεχνικού χρέους - μια συσσώρευση προβλημάτων και πολυπλοκοτήτων που συχνά επηρεάζουν τα έργα λογισμικού με την πάροδο του χρόνου. Στη συνέχεια, η μελέτη εμβαθύνει στο refactoring, μια ουσιαστική πρακτική που στοχεύει στον καθαρισμό και τη βελτίωση του κώδικα, καθιστώντας τον πιο κατανοητό και διαχειρίσιμο ως ενα αρκετα πρακτικό εργαλείο εναντίον του τεχνικού χρέους.

Η μελέτη επεκτείνεται στο πεδίο της αρχιτεκτονικής λογισμικού με προσανατολισμό στις υπηρεσίες (SOA) και τις μικροϋπηρεσίες, παρέχοντας ένα υπόβαθρο για σύγχρονες μεθοδολογίες σχεδιασμού λογισμικού. Στη συνέχεια, εστιάζει στο Spring Boot, ένα εργαλείο σχεδιασμένο να απλοποιεί τη δημιουργία μικροϋπηρεσιών, ενισχύοντας την προσβασιμότητα και την επεκτασιμότητα. Τέλος, η εργασία εξηγεί όλα τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση του microservice.

Πιο συγκεκριμένα, περιγράφεται η σταδιακή μετάβαση από μια παλαιότερη εφαρμογή που βασίζεται σε τοπικά εργαλεία (αρχεία JAR) για τον εντοπισμό και την αντιμετώπιση τεχνικού χρέους, σε μια νέα προσέγγιση που αξιοποιεί τις μικροϋπηρεσίες. Αυτή η εξέλιξη αντιπροσωπεύει ένα κρίσιμο χαρακτηριστικό, καθιστώντας αυτά τα τεχνικά εργαλεία χρέους διαθέσιμα μέσω του διαδικτύου, ενισχύοντας έτσι την προσβασιμότητα στα εργαλεία, την ευκολία και την επεκτασιμότητά τους.

Έπειτα ακολουθεί μια λεπτομερής επεξήγηση του κωδικα του microservice, που αποτελεί τον πυρήνα της πτυχιακης εργασίας. Αυτή η ενότητα προσφέρει πληροφορίες για αρχιτεκτονικές αποφάσεις, διαγράμματα και επεξήγηση κώδικα, τα εργαλεία του Spring Boot που χρησιμοποιήθηκαν και την ενοποίηση των εργαλείων semi και gea, που είναι αφιερωμένα στον εντοπισμό και την επίλυση τεχνικών χρεών.

Εκτός από τις τεχνικές πτυχές, η πτυχιακή περιλαμβάνει ένα εγχειρίδιο χρήσης, που παρέχει έναν πρακτικό οδηγό. Αυτό το εγχειρίδιο περιγράφει τη διαδικασία βήμα προς βήμα για τον τρόπο με τον οποίο οι χρήστες μπορούν να υποβάλουν συνδέσμους project repository και να ανακτήσουν τα αποτελέσματα.

«Implementation of Software Quality Measurement Web Service»

«Konstantinos Tsitoumis»

Abstract

This thesis embarks on a journey through the realm of software development, with initial focus on understanding foundational knowledge such as technical debt and implementing effective solutions through the process of refactoring. It begins with an explanation of technical debt—an accumulation of issues and complexities that often plagues software projects over time. The study then delves into refactoring, an essential practice aimed at cleaning up and enhancing code, making it more comprehensible and manageable.

The exploration extends to the landscape of Service-Oriented Architecture (SOA) and microservices, providing a backdrop for modern software design methodologies. Within this context, the thesis places a spotlight on Spring Boot later on, as a tool designed to simplify the creation of microservices, enhancing accessibility and scalability and explains all the tools used for the implementation of the microservice.

The narrative unfolds as it traces the transition from an older implementation reliant on local tools (JAR files) for identifying and addressing technical debt, to a novel approach leveraging microservices. This evolution represents a crucial feature, making these technical debt tools available over the internet, thereby enhancing the accessibility of the tools, convenience and scalability.

The core of the thesis resides in the detailed explanation of the microservice's codebase. This section offers insights into architectural decisions, diagrams and explanation of code, the used Spring Boot components, and the integration of the semi and gea tools dedicated to the detection and resolution of technical debt.

In addition to the technical aspects, the thesis includes a user manual, providing a practical guide. This manual outlines the step-by-step process for how users can submit project repository links and fetch the results.

Ευχαριστίες

Με την παρούσα διπλωματική εργασία, ολοκληρώνονται οι σπουδές μου ως φοιτητής στο Τμήμα Μηχανικών πληροφορικής Τ.Ε. του Διεθνούς Πανεπιστημίου της Ελλάδος.

Στο σημείο αυτό, θα ήθελα να ευχαριστήσω την επιβλέποντα καθηγήτρια μου, κυρία Ελβίρα Αρβανίτου, για την καθοδήγηση και βοήθεια που μου προσέφερε κατά τη διάρκεια εκπόνησης της παρούσας πτυχιακής εργασίας.

Ευχαριστώ, όλους τους καθηγητές που πλαισίωσαν το προπτυχιακό πρόγραμμα, για τη συνεισφορά που είχαν, ο καθένας με το δικό του τρόπο, στη διεύρυνση της γνώσης μου.

Επίσης, θα ήθελα να εκφράσω θερμές ευχαριστίες στους ανθρώπους που γνώρισα κατά την περίοδο των σπουδών μου, όπου μέσα από αυτές τις γνωριμίες πλάστηκε ο χαρακτήρας μου, κι έκαναν το «ταξίδι» αυτό ακόμα πιο όμορφο.

Τέλος, ένα πολύ μεγάλο ευχαριστώ αξίζει στους γονείς μου που στάθηκαν δίπλα μου σε όλες τις δύσκολες στιγμές στην ζωή μου, χωρίς την υποστήριξη τους δεν θα κατάφερνα να φτάσω στο στάδιο που έχω καταφέρει να βρίσκομαι σήμερα.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή – Θεωριτικό πλαίσιο

1.1 Τεχνικό χρέος

1.1.1 Η έννοια του τεχνικού χρέους

Το τεχνικό χρέος είναι μια σύνθετη και πολύπλευρη έννοια που έχει αναδειχθεί ως σημαντική ανησυχία στον τομέα της μηχανικής λογισμικού**[1]**. Επινοήθηκε για πρώτη φορά από τον Ward Cunningham για να εξηγήσει τις μελλοντικές επιπτώσεις στο κόστος της επιλογής πιο γρήγορων, αλλά λιγότερο ποιοτικών λύσεων κατά τη διαδικασία ανάπτυξης λογισμικού[**2]**.

Η έννοια του τεχνικού χρέους βασίζεται σε μια οικονομική αναλογία που περιγράφει τους συμβιβασμούς μεταξύ της λήψης ενός "δανείου" με την επιλογή μιας ταχύτερης, αλλά μη ποιοτικής λύσης, και της "εξόφλησης" αυτού του δανείου αργότερα με την αφιέρωση επιπλέον χρόνου για τη βελτίωση ή την αντικατάσταση της αρχικής λύσης [1][4]. Όπως ένα χρηματοοικονομικό χρέος, εάν το τεχνικό χρέος δεν "εξοφληθεί" εγκαίρως, συσσωρεύει "τόκους", καθιστώντας το σύστημα πιο δύσκολο και δαπανηρό στη συντήρηση και ενίσχυση του[**2]**.

Το τεχνικό χρέος είναι ένα αποτέλεσμα που προκύπτει σε κάθε στάδιο της ανάπτυξης λογισμικού, από την αρχική ανάλυση απαιτήσεων, μέχρι την τελική υλοποίηση **[4]**. Αυτό υπογραμμίζει τη σημασία της συνεχούς παρακολούθησης και διαχείρισης καθ' όλη τη διάρκεια του κύκλου ζωής του λογισμικού.[4]

Το τεχνικό χρέος μπορεί δημιουργηθεί τόσο σκόπιμα όσο και ακούσια δηλαδή χωρίς μια ομάδα ανάπτυξης λογισμικού να το γνωρίζει [**1][3]**. Στην περίπτωση του σκόπιμου τεχνικού χρέους, οι ομάδες ανάπτυξης συχνά λαμβάνουν αποφάσεις στρατηγικά που μπορεί να μην είναι σωστές από τεχνικής απόψεως, αλλά επιτρέπουν την ταχύτερη παράδοση του λογισμικού ή την εκπλήρωση των προθεσμιών [1][3]. Αυτό συμβαίνει συχνά όταν υπάρχει πίεση για την επίτευξη συγκεκριμένων στόχων ή όταν υπάρχουν ευκαιρίες στην αγορά που απαιτούν γρήγορη ανταπόκριση [1]. Ωστόσο, αυτές οι αποφάσεις δημιουργούν ένα "χρέος" που πρέπει να "αποπληρωθεί" στο μέλλον, δηλαδή το λογισμικό πρέπει να βελτιωθεί ή να αναδιαμορφωθεί [1].
Από την άλλη πλευρά, το ακούσιο τεχνικό χρέος προκύπτει όταν οι αποφάσεις που λαμβάνονται κατά τη διάρκεια της ανάπτυξης του λογισμικού δεν είναι ποιοτικές τεχνικά, αλλά αυτό δεν είναι γνωστό στην ομάδα ανάπτυξης λογισμικού εκείνη τη στιγμή [1][3]. Αυτο μπορεί να προκληθεί λόγω χαμηλής ποιότητας εργασίας, έλλειψη γνώσης ή εξειδίκευσης, απο λήψη πολλών σύντομων τρόπων υλοποίησης, απο την χρήση γενικευμένων ή κακών ονομασιών των μεταβλητών, αποφυγή τήρησης των κανόνων της εταιρείας πάνω στον τρόπο συγγραφής κώδικα κ.λπ.[3]

Στον κόσμο των επιχειρήσεων, εταιρείες συχνά χειρίζονται τόσο βραχυπρόθεσμα όσο και μακροπρόθεσμα οικονομικά χρέη. Αυτές οι περιπτώσεις ειναι όμοιες με την περίπτωσή μας για το τεχνικό χρέος. Τα βραχυπρόθεσμα οικονομικά χρέη προκύπτουν όταν μια επιχείρηση χρειάζεται να γεφυρώσει το χάσμα μεταξύ των εισερχόμενων πληρωμών από τους πελάτες και των εξερχόμενων δαπανών όπως οι μισθοί[3]. Ενώ τα κεφάλαια ενδέχεται να είναι διαθέσιμα στο μέλλον, η εταιρεία δεν τα διαθέτει επί του παρόντος[3]. Αυτό το είδος χρέους αναλαμβάνεται ως απάντηση σε άμεσες

ανάγκες και συνήθως διακανονίζεται γρήγορα[3]. Στο χώρο της τεχνολογίας, μια αναλογία θα ήταν μια επιδιόρθωση της τελευταίας στιγμής για την γρήγορη κυκλοφορία ενός προϊόντος στην αγορά[3].

Τα μακροπρόθεσμα οικονομικά χρέη είναι σκόπιμες επιλογές που κάνουν οι εταιρείες, συχνά ως μια προοδευτική στρατηγική[3]. Παραδείγματα περιλαμβάνουν την απόκτηση σημαντικών περιουσιακών στοιχείων, όπως εργοστάσια ή εκτεταμένους επαγγελματικούς χώρους[3]. Κάνοντας την τεχνολογική παραλληλία, το μακροπρόθεσμο τεχνολογικό χρέος είναι κι αυτό στρατηγικό[3]. Συμβαίνει όταν μια εταιρεία λαμβάνει τεχνολογικές αποφάσεις που είναι βιώσιμες για το προβλέψιμο μέλλον αλλά μπορεί να απαιτήσουν σημαντική προσπάθεια για αλλαγή αργότερα[3].

1.1.2 Οι τύποι του τεχνικού χρέους

Οι διάφορες εκδηλώσεις του τεχνικού χρέους είναι τόσο πολύπλευρες όσο και ο ίδιος ο τομέας της ανάπτυξης λογισμικού. Εκτείνονται σε διάφορες διαστάσεις, καθεμία από τις οποίες παρουσιάζει τις μοναδικές της προκλήσεις. Με βάση τη βιβλιογραφία [2][5], διαπιστώθηκαν διάφοροι τύποι τεχνικού χρέους.

- *Τεχνικό χρέος στην Αρχιτεκτονικη του κωδικα (Architecture debt)*: Πρόκειται για το τεχνικό χρέος που δημιουργείται κατά την διάρκεια της αρχιτεκτονικής ενός έργου λογισμικού [2][5]. Το αποτέλεσμα του μπορεί να παραβιάζει τις απαιτήσεις αρχιτεκτονικής λογισμικού (Architecture requirements). Συνήθως, τεχνικά χρέη αυτού του είδους δεν μπορούν να "πληρωθούν" με απλές παρεμβάσεις στον κώδικα, αλλά είναι αναγκαίες πιο εκτεταμένες ενέργειες για την επιδιόρθωση τους[2][5].
- *Τεχνικό χρέος στην διαδικασία χτισίματος (Build Debt):* Το χρέος διαδικασίας κατασκευής (Build) σχετίζεται με προβλήματα που το καθιστούν άσκοπα/αδικαιολόγητα πολύπλοκο και χρονοβόρο [2][5]. Κάποια από αυτά τα προβλήματα μπορεί να είναι ο περιττός κώδικας ή προβληματικές εξαρτήσεις (Dependencies) που επιβραδύνουν τη διαδικασία [2][5].
- *Τεχνικό χρέος στον κώδικα (Code Debt)*: Το Τεχνικό χρέος στον κώδικα αναφέρεται σε προβλήματα εντός του κώδικα. Το συγκεκριμένο τεχνικό χρέος μπορει να βρεθεί παρατηρώντας τον κώδικα[5]. Πρόκειται για τα προβλήματα τα οποία μπορούν να επηρεάσουν αρνητικά την αναγνωσιμότητα του κώδικα, καθιστώντας πιο δύσκολη τη διατήρησή του. Αυτά τα προβλήματα σχετίζονται άμεσα με την χρήση κακών πρακτικών συγγραφής κώδικα[5].
- *Τεχνικό χρέος ελαττωμάτων (Defect Debt)*: Το χρέος ελαττωμάτων αποτελείται από γνωστά ελαττώματα, συνήθως προσδιοριζόμενα από δραστηριότητες test ή από τους χρήστες που τα χρησιμοποιούν[2][5]. Η Επιτροπή Ελέγχου Αλλαγών - Change Control Board (CCB) συμφωνεί ότι θα πρέπει να διορθωθούν, αλλά λόγω ανταγωνιστικών προτεραιοτήτων και περιορισμένων πόρων πρέπει να αναβληθούν για αργότερα[2][5]. Αυτά μπορούν να συσσωρεύουν σε ένα σημαντικό ποσό τεχνικού χρέους, καθιστώντας τη διόρθωσή τους πιο δύσκολη στο μέλλον[2][5].
- *Τεχνικό χρέος σχεδιασμού (Design Debt)*: Πρόκειται για το τεχνικό χρέος που συσσωρεύεται από την αποφυγή τήρησης πρακτικών από τις αρχές της καλής σχεδίασης του αντικειμενοστραφή προγραμματισμού στον πηγαίο κώδικα[2][5]
- *Τεχνικό χρέος τεκμηρίωσης (Documentation Debt)*: Αυτή η μορφή τεχνικού χρέους αφορά τα προβλήματα που εντοπίζονται στην τεκμηρίωση των έργων λογισμικού και μπορούν να προσδιοριστούν αναζητώντας την απουσία, την ανεπάρκεια ή την ελλιπή τεκμηρίωση οποιουδήποτε είδους[2][5]. Ειδικότερα, ως "ανεπαρκής τεκμηρίωση" χαρακτηρίζεται εκείνη που,

παρά το γεγονός ότι λειτουργεί ορθώς εντός του συστήματος, δεν καλύπτει κάποια συγκεκριμένα κριτήρια ποιότητας που απαιτούνται για τα έργα λογισμικού[5].

- *Τεχνικό χρέος υποδομής (Infrastructure Debt)*: Αυτή η κατηγορία αναφέρεται σε προβλήματα υποδομής που εάν υπάρχουν σε μια εταιρία προγραμματισμού, μπορούν να καθυστερήσουν ή να εμποδίσουν κάποιες δραστηριότητες ανάπτυξης κώδικα[2][5]. Μερικά παραδείγματα αυτού του είδους χρέους είναι η καθυστέρηση μιας αναβάθμισης ή της επιδιόρθωσης της υποδομής[2][5].
- *Τεχνικό χρέος ανθρώπινου δυναμικού (People Debt)*: Πρόκειται για το τεχνικό χρέος που σχετίζεται με ζητήματα προσωπικού σε μια οργάνωση ανάπτυξης λογισμικού. Τέτοιου είδους χρέη μπορούν να προκαλέσουν καθυστερήσεις ή εμπόδια στην ανάπτυξη του κώδικα. Ενα χαρακτηριστικό παράδειγμα είναι η ύπαρξη λίγων έμπειρων μηχανικών λόγω καθυστερημένης πρόσληψης νέων μηχανικών ή εκπαίδευση[2][5].
- *Τεχνικό χρέος διαδικασιών (Process Debt)*: 'Αλλη μια διάσταση του τεχνικού χρέους ειναι το χρέος πάνω στις υπαρκτές διαδικασίες που μια οργάνωση έχει. Πρόκειται για διαδικασίες οι οποίες με το πέρασμα του χρόνου μπορεί να γίνουν ανεπαρκείς και να μην εκπληρώνουν πλέον τον σκοπό για τον οποίο σχεδιάστηκαν[2][5].
- *Τεχνικό χρέος απαιτήσεων (Requirements Debt)*: Το Τεχνικό χρέος απαιτήσεων αναφέρεται στις συμβιβαστικές επιλογές που γίνονται σχετικά με τις απαιτήσεις που πρέπει να υλοποιήσει η ομάδα ανάπτυξης λογισμικού ή τον τρόπο υλοποίησής τους. Ορισμένα παραδείγματα περιλαμβάνουν: απαιτήσεις που έχουν υλοποιηθεί μόνο εν μέρει, απαιτήσεις που έχουν υλοποιηθεί αλλά όχι για όλες τις περιπτώσεις, απαιτήσεις που έχουν υλοποιηθεί αλλά με τρόπο που δεν ικανοποιεί πλήρως όλες τις μη λειτουργικές απαιτήσεις π.χ. σε ασφάλεια ή σε απόδοση[2][5].
- *Τεχνικό χρέος υπηρεσιών (Service Debt)*: Όταν μια επιχείρηση ή ένας τεχνικός οργανισμός αντιμετωπίζει την ανάγκη να αντικαταστήσει μια διαδικτυακή υπηρεσία, αυτή η απόφαση μπορεί να οδηγήσει στη δημιουργία τεχνικού χρέους[2][5]. Αυτό το τεχνικό χρέος θα πρέπει να διαχειριστεί, να καθαριστεί και να μετατραπεί από κάτι που αποτελεί ευθύνη σε κέρδος[2][5]. Το τεχνικό χρέος μπορεί να αφορά διάφορες διαστάσεις, οι οποίες σχετίζονται με την επιλογή, την σύνθεση, και την λειτουργία της υπηρεσίας[2][5].
- *Τεχνικό χρέος αυτοματοποίησης τέστ (Test automation Debt)*: Το Τεχνικό χρέος αυτοματοποίησης τεστ αναφέρεται στην εργασία που απαιτείται για την αυτοματοποίηση των τεστ προηγούμενων αναπτυγμένων λειτουργιών με στόχο την υποστήριξη της Continuous Integration διαδικασίας και την επιτάχυνση των φάσεων ανάπτυξης του λογισμικού[2][5].
- *Τεχνικό χρέος τέστ (Test Debt)*: Στο πλαίσιο του τεχνικού χρέους που σχετίζεται με τα τεστ, περιλαμβάνονται ζητήματα που προκύπτουν κατά την εκτέλεση των τεστ και έχουν επιπτώσεις στην ποιότητα των εν λόγω δραστηριοτήτων. Για παράδειγμα, ένα στοιχείο του τεχνικού χρέους μπορεί να είναι τα προγραμματισμένα τεστ που τελικά δεν εκτελέστηκαν ή οι γνωστές ατέλειες της τεστ σουίτας, όπως η χαμηλή κάλυψη κώδικα[2][5].

1.1.3 Η διαχείριση τεχνικού χρέους

Η διαχείριση τεχνικού χρέους είναι μια κρίσιμη πτυχή της ανάπτυξης λογισμικού που περιλαμβάνει μια σειρά δραστηριοτήτων για την πρόληψη, τον εντοπισμό, τη μέτρηση, την ιεράρχηση, την παρακολούθηση και την αποπληρωμή του τεχνικού χρέους προκειμένου να διατηρηθεί σε λογικό επίπεδο. Είναι απαραίτητη για τη μακροπρόθεσμη συντήρηση και επιτυχία των συστημάτων λογισμικού. Μια ολοκληρωμένη προσέγγιση στην τεχνική διαχείριση του χρέους περιλαμβάνει διάφορες δραστηριότητες, η καθεμία με το δικό της σύνολο προσεγγίσεων και τεχνικών.

1. *Αναγνώριση τεχνικού χρέους*: Είναι ο εντοπισμός του τεχνικού χρέους που προκλήθηκε σκόπιμα ή ακούσια. Η ανάλυση κώδικα και η ανάλυση εξαρτήσεων είναι οι δύο πιο δημοφιλείς προσεγγίσεις[1].
2. *Μέτρηση τεχνικού χρέους*: Περιλαμβάνει τον ποσοτικό προσδιορισμό των οφελών και του κόστους των γνωστών χρεών μέσω τεχνικών εκτίμησης και περιλαμβάνει: μοντέλα υπολογισμού με την χρήση μαθηματικών συναρτήσεων, μετρικές κώδικα, ανθρώπινη εκτίμηση βάση την εμπειρία και την εξειδίκευση, κατηγοριοποίηση κόστους, μετρικές λειτουργίας προϊόντος και υπολογισμούς απόστασης μεταξύ πραγματικής και βέλτιστης λύσης [1].
3. *Προτεραιότητα τεχνικού χρέους*: Αυτό περιλαμβάνει την κατάταξη των προσδιορισμένων στοιχείων σύμφωνα με προκαθορισμένους κανόνες ή κριτήρια που τοποθετούν τεχνικά χρέη σε προτεραιότητα, όπως: ανάλυση κόστους/οφέλους (**Cost/benefit analysis**) όπου αν μια εξόφληση μπορεί να αποφέρει υψηλότερο όφελος από το κόστος τότε μπαίνει σε προτεραιότητα, αποκατάσταση τεχνικών χρεών με υψηλό κόστος πρώτα (**High remediation cost first**) και αποκατάσταση τεχνικών χρεών με υψηλό τόκο πρώτα (**High interest first**). Το κίνητρο είναι να καθοριστεί ποιά είδη πρέπει να αποπληρωθούν πρώτα και ποιά μπορούν να γίνουν ανεκτά. μέχρι μεταγενέστερες εκδόσεις[1].
4. *Πρόληψη τεχνικού χρέους*: Στοχεύει να αποτρέψει την δημιουργία τεχνικού χρέους[1]. Κάποιες προσεγγίσεις που εφαρμόζονται για αυτή την επίτευξη είναι: Η βελτίωση της διαδικασίας του προγραμματισμού, αξιολόγηση όλων των επιλογών αρχιτεκτονικής και εφαρμογή της επιλογής με το λιγότερο τεχνικό χρέος, δημιουργία συστημάτων που ελέγχουν το ποσοστό του τεχνικού χρέους κατά την όλη διάρκεια της ζωής του λογισμικού, η καλλιέργεια κουλτούρας (των μηχανικών της ομάδας ανάπτυξης λογισμικού) με σκοπό την ελαχιστοποίηση του ακούσιου τεχνικού χρέους που δημιουργείται από ανθρώπινους παράγοντες (π.χ., αδιαφορία και άγνοια) [1].
5. *Παρακολούθηση τεχνικού χρέους*: Περιλαμβάνει την παρακολούθηση των αλλαγών στο κόστος και το όφελος του μη επιλυμένου τεχνικού χρέους[1]. Κάποιες μέθοδοι για την παρακολούθηση του τεχνικού χρέους είναι:
    a. Καθορισμός ορίων για μετρήσεις της ποιότητας που σχετίζονται με το τεχνικό χρέος και προειδοποίηση σε περίπτωση που τα όρια δεν τηρούνται (**Threshold-based approach**)[1].
    b. Τακτικός ελεγχος αναγνωρισμένου τεχνικού χρέους και παρακολούθηση της αλλαγής του (**Planned check**)[1].
    c. Παρακολούθηση των αλλαγών στα χαρακτηριστικά ποιότητας που επηρεάζουν αρνητικά το τεχνικό χρέος, όπως η σταθερότητα (**Technical monitoring with quality attribute focus**)[1].
    d. Απεικόνιση διαφορών συγκεντρωτικών μέτρων του τεχνικού χρέους κατά τη διάρκεια του χρόνου και εξέταση της μορφής της καμπύλης που παράγεται.
    e. Παρακολούθηση επιρροών τεχνικού χρέους μεταξύ συστήματος με τεχνικό χρεος και άλλων συστηματων που εξαρτώνται απο αυτό (**propagation tracking**) [1].
6. *Αποπληρωμή τεχνικού χρέους*: Πρόκειται για την επίλυση ή τον μετριασμό του τεχνικού χρέους χρησιμοποιώντας τεχνικές όπως την ανακατασκευή, που είναι η πιο συχνά χρησιμοποιούμενη, την επανεγγραφή, την αυτοματοποίηση, τον ανασχεδιασμό, την επανασυσκευασία και τη διόρθωση σφαλμάτων[1].
7. *Τεκμηρίωση Τεχνικού Χρέους*: Περιλαμβάνει την καταγραφή και την παρουσίαση του τεχνικού χρέους ως οντότητες, με στόχο την διευθέτηση των ανησυχιών των συγκεκριμένων ενδιαφερόμενων μερών[1]. Είναι σημαντικό κάθε καταχώρηση να διατηρείται με έναν ομοιόμορφο τρόπο και να περιέχει ορισμένα βασικά πεδία με πληροφορίες για το κάθε τεχνικό

χρέος, όπως το ID, τη τοποθεσία, τον υπεύθυνο, τον τύπο τεχνικού χρέους, τη περιγραφή, και άλλες σχετικές πληροφορίες.[1].

8. *Επικοινωνία Τεχνικού Χρέους*: Είναι η έκθεση του εντοπισμένου τεχνικού χρέος στα ενδιαφερόμενα μέρη, ώστε να μπορεί να συζητηθεί και να διαχειριστεί περαιτέρω[1]. Κάποια παραδείγματα είναι: η χρήση ενός πίνακα ελέγχου που παρουσιάζει τις οντότητες τεχνικού χρέους, ένα backlog που περιέχει όλα τα τεχνικά χρέη τα οποία πρέπει να επιλυθούν, την οπτικοποίηση των ανεπιθύμητων εξαρτήσεων και την οπτικοποίηση των μετρικών κώδικα[1].

1.2 Αναδομήσεις λογισμικού

Οι αναδομήσεις λογισμικού αντιπροσωπεύουν μια μέθοδο τροποποίησης της εσωτερικής δομής ενός λογισμικού χωρίς την αλλαγή της εξωτερικής του συμπεριφοράς. Είναι μια συστηματική προσέγγιση για τον καθαρισμό του κώδικα ελαχιστοποιώντας τον κίνδυνο εισαγωγής νέων σφαλμάτων [6]. Ουσιαστικά, η αναδόμηση βελτιώνει τον σχεδιασμό του κώδικα μετά την αρχική του ανάπτυξη.
Αντίθετα με τη δημοφιλή πεποίθηση, ο σχεδιασμός δεν περιορίζεται μόνο στην αρχή της διαδικασίας της ανάπτυξης του λογισμικού [6]. Παρόλο που συνήθως σχεδιάζουμε πρώτα και συγγράφουμε κώδικα αργότερα, η ακεραιότητα και η δομή του συστήματος συχνά υποβαθμίζονται με τον καιρό, οδηγώντας σε μια μετάβαση από οργανωμένο λογισμικό σε ένα χαώδες και ανοργάνωτο [6]. Η αναδόμηση λοιπόν αντισταθμίζει αυτή την τάση μετατρέποντας τον κακοσχεδιασμένο ή χαοτικό κώδικα σε καλά δομημένο, καλοσχεδιασμένο κώδικα [6]. Αυτή η μεταμόρφωση περιλαμβάνει απλά, ακόμη και ασήμαντα, βήματα, όπως η μετακίνηση ενός πεδίου μεταξύ κλάσεων ή η εξαγωγή ενός τμήματος μιας μεθόδου για τη δημιουργία μιας νέας [6]. Ωστόσο, αυτές οι μικρές προσαρμογές μπορούν να οδηγήσουν συλλογικά σε μια σημαντική βελτίωση του σχεδιασμού, αντιστρέφοντας έτσι την τυπική διαδικασία υποβάθμισης του λογισμικού [6].
Η αναδιάρθρωση επίσης αλλάζει την κατανομή του φόρτου εργασίας, καθώς ο σχεδιασμός γίνεται μια συνεχής διαδικασία καθ' όλη τη διάρκεια της ανάπτυξης [6]. Αυτή η συνεχής βελτίωση βοηθά στη διατήρηση ενός υψηλής ποιότητας σχεδιασμού καθ' όλη τη διάρκεια της διαδικασίας ανάπτυξης [6].

Στην ανάπτυξη λογισμικού, η αναδόμηση κατέχει μια ιδιαίτερα κρίσιμη θέση, λειτουργώντας ως ένα ευέλικτο εργαλείο που συμβάλλει στην ενίσχυση της ποιότητας του κώδικα και την ταχύτητα της ανάπτυξης. Στη συνέχεια, περιγράφονται οι λόγοι για τους οποίους η αναδόμηση είναι ένα βασικό βήμα στη διαδικασία ανάπτυξης λογισμικού:

- **Βελτιώνει τον Σχεδιασμό του Λογισμικού**: Η αναδόμηση είναι απαραίτητη για τη διατήρηση της ποιότητας και της λειτουργικότητας του σχεδιασμού ενός λογισμικού. Βοηθά στον καθαρισμό και την οργάνωση του κώδικα, αποτρέποντας τη φθορά του λόγω συχνών αλλαγών που συχνά στοχεύουν σε βραχυπρόθεσμους στόχους. Χωρίς ανακατασκευή, ο κώδικας χάνει σταδιακά τη δομή του, καθιστώντας δύσκολη τη σωστή κατανόηση και τροποποίηση στο μέλλον. Αυτή η διαδικασία δίνει έμφαση στην εξάλειψη των διπλών κωδικών, ενισχύοντας μια συνοπτική και σαφή δομή κώδικα όπου κάθε συνάρτηση ορίζεται μόνο μία φορά. Δεν επιταχύνει απαραίτητα το σύστημα, αλλά διευκολύνει σημαντικά τις μελλοντικές τροποποιήσεις του κώδικα, ενισχύοντας έναν πιο συνεκτικό και διαχειρίσιμο σχεδιασμό. Επομένως, η τακτική ανακατασκευή είναι το κλειδί για τη διατήρηση της ακεραιότητας του σχεδιασμού του λογισμικού με την πάροδο του χρόνου. [6]
- **Καθιστά το Λογισμικό Πιο Κατανοητό**: Η αναδόμηση είναι μια διαδικασία που κάνει το λογισμικό πιο κατανοητό, διευκολύνοντας έναν σαφέστερο διάλογο μεταξύ του

προγραμματιστή και του υπολογιστή. Όχι μόνο απλοποιεί τον κώδικα ώστε να αντικατοπτρίζει τους στόχους του προγραμματιστή, αλλά επίσης βελτιώνει την αναγνωσιμότητα για τους μελλοντικούς προγραμματιστές που ενδέχεται να εργαστούν σε αυτόν. Συχνά όμως, το άτομο που επωφελείται είναι ο αρχικός προγραμματιστής, ο οποίος μπορεί να επιστρέψει στον κώδικα αργότερα και να τον κατανοήσει και να τον τροποποιήσει ευκολότερα λόγω των αλλαγών που έγιναν. Επιπλέον, η αναδόμηση μπορεί να είναι ένα εργαλείο για τη βελτίωση της κατανόησης του άγνωστου κώδικα, επιτρέποντας στον προγραμματιστή να τροποποιήσει και να δοκιμάσει τον κώδικα για να κατανοήσει καλύτερα τη λειτουργικότητά του. Βοηθά στην αποσαφήνιση μικρών λεπτομερειών, αποκαλύπτοντας σταδιακά βαθύτερες γνώσεις σχετικά με τη σχεδίαση του κώδικα που μπορεί να μην ήταν εμφανείς στην αρχή. [6]

- **Βοηθά στον Εντοπισμό των Σφαλμάτων**: Η αναδόμηση χρησιμεύει ως βασικό εργαλείο για τον εντοπισμό σφαλμάτων στον κώδικα. Η διαδικασία περιλαμβάνει μια βαθιά ανάλυση του κώδικα, η οποία βοηθάει στην καλύτερη κατανόηση των λειτουργικοτήτων του. Αυτή η κατανόηση στη συνέχεια ενσωματώνεται ομαλά στον κώδικα, ενισχύοντας τη δομή του και διευκρινίζοντας τις αρχικές υποθέσεις που έγιναν κατά τη διαδικασία συγγραφής κώδικα. Αυτή η διευκρίνιση, με τη σειρά της, φέρνει σφάλματα στην επιφάνεια, καθιστώντας τα σχεδόν αναπόφευκτα να τα παρατηρήσετε. [6]
- **Επιταχύνει τον Προγραμματισμό**: Η αναδόμηση, αν και φαινομενικά αποτελεί εμπόδιο στην ταχύτητα ανάπτυξης λογισμικού, στην πραγματικότητα καταστεί ταχύτερο το ρυθμό προγραμματισμού μακροπρόθεσμα. Αρχικά, μπορεί να φαίνεται ότι επιβραδύνει τη διαδικασία ανάπτυξης καθώς περιλαμβάνει την επανεξέταση και τροποποίηση των υπαρχόντων κωδίκων για τη βελτίωση της ποιότητάς τους μέσω βελτιωμένης σχεδίασης και αναγνωσιμότητας και μειωμένων σφαλμάτων. Ωστόσο, η διατήρηση ενός καλού σχεδιασμού μέσω της ανακατασκευής διευκολύνει την ταχεία ανάπτυξη, αποτρέποντας την επιβράδυνση που αναπόφευκτα συμβαίνει με έναν υποβαθμισμένο σχεδιασμό. Αποτρέποντας την αποσύνθεση του σχεδιασμού του συστήματος και ακόμη και ενισχύοντας το, η αναδόμηση επιτρέπει στους προγραμματιστές να διατηρήσουν έναν γρήγορο ρυθμό ανάπτυξης, αποφεύγοντας τις παρατεταμένες περιόδους που δαπανώνται για τον εντοπισμό σφαλμάτων και την κατανόηση πολύπλοκων, διορθωμένων συστημάτων. Έτσι, η ενσωμάτωση της αναδόμησης στη διαδικασία ανάπτυξης κώδικα είναι μια συνετή προσέγγιση για την επίτευξη ταχύτερης και αποτελεσματικότερης ανάπτυξης λογισμικού. [6]

1.2.1 Code smells

Τα code smells είναι συμπτώματα κακών επιλογών σχεδιασμού και υλοποίησης. Τέτοια συμπτώματα μπορεί να προέρχονται από δραστηριότητες που εκτελούνται από προγραμματιστές κατά τη διάρκεια έκτακτης ανάγκης με αποτέλεσμα τις βιαστικές αλλαγές, κακές σχεδιάσεις ή λύσεις, λήψη κακών αποφάσεων ή χρήση των αντι-μοτίβων (**anti-patterns**). Τα πιο συχνά code smells με βάση τη βιβλιογραφία [6] είναι τα εξής:

1. *Διπλότυπος κώδικας (Duplicated Code)*: Ο διπλότυπος κώδικας αναφέρεται σε περιπτώσεις όπου η ίδια δομή κώδικα βρίσκεται σε περισσότερα από ένα σημεία σε ένα πρόγραμμα. Θεωρείται πρόβλημα και προτείνεται η ενοποίηση των διπλότυπων δομών για την βελτίωση της ποιότητα και της λειτουργικότητας του προγράμματος. [6]
2. *Μεγάλη μέθοδος (Long Method)*: Μια μεγάλη μέθοδος αναφέρεται σε μια διαδικασία μέσα σε ένα πρόγραμμα που είναι εκτεταμένη, καθιστώντας την πιο δύσκολη την κατανόησή της. Στο παρελθόν, απέφευγαν την δημιουργία μικρότερων μεθόδων λόγω της επιβάρυνσης κλήσεων

υπορουτίνας που σχετίζεται με παλαιότερες γλώσσες. Ωστόσο, οι σύγχρονες αντικειμενοστρεφείς γλώσσες έχουν ελαχιστοποιήσει αυτό το ζήτημα, ενθαρρύνοντας την δημιουργία συντομότερων μεθόδων για την ευκολότερη κατανόηση και συντήρηση του προγράμματος. Συνιστάται λοιπόν η κατανομή των μεθόδων σε μικρότερες μονάδες, χρησιμοποιώντας σωστές ονομασίες για να μεταφερθεί με σαφήνεια ο σκοπό τους, κάτι που τελικά βοηθά στη μείωση της πολυπλοκότητας και στη βελτίωση της αναγνωσιμότητας του κώδικα. [6]

3. *Μεγάλη κλάση (Large Class):* Μια μεγάλη κλάση χαρακτηρίζεται μια κλάση σε ένα πρόγραμμα που προσπαθεί να κάνει πάρα πολλά πράγματα, υποδεικνύοντας ότι έχει πάρα πολλές μεταβλητές ή πάρα πολύ κώδικα. Αυτό μπορεί να οδηγήσει σε επαναλαμβανόμενο κώδικα και ακαταστασία, καθιστώντας το πρόγραμμα δύσκολο στη διαχείριση και επιρρεπές σε σφάλματα. Για να το διορθωθεί αυτό, πρέπει να χωριστεί η κλάση σε μικρότερα κομμάτια ή υποκλάσεις που ομαδοποιούν σχετικές μεταβλητές ή συναρτήσεις, κάνοντας τον κώδικα πιο οργανωμένο και πιο κατανοητό. [6]
4. *Μεγάλη λίστα παραμέτρων (Long Parameter List)*: Μια "Μεγάλη λίστα παραμέτρων" παρατηρείτε όταν μια μέθοδος σε ένα πρόγραμμα απαιτεί πολλές παραμέτρους, γεγονός που την καθιστά περίπλοκη και δύσκολη στην κατανόηση ή τη χρήση της. Συχνά χρειάζεται συχνές προσαρμογές καθώς ενσωματώνονται περισσότερα δεδομένα. Αυτό το ζήτημα συμβαίνει λιγότερο στον αντικειμενοστραφή προγραμματισμό, καθώς οι μέθοδοι μπορούν να λάβουν απαραίτητα δεδομένα από την κλάση ή άλλα γνωστά αντικείμενα, μειώνοντας έτσι το μέγεθος των λιστών παραμέτρων. Ωστόσο, μερικές φορές η διατήρηση μιας μεγαλύτερης λίστας παραμέτρων είναι λογική, ειδικά όταν είναι απαραίτητη η αποφυγή δημιουργίας εξάρτησης από ένα μεγαλύτερο αντικείμενο. [6]
5. *Συσσωρεύματα δεδομένων (Data Clumps)*: Τα συσσωρεύματα δεδομένων αναφέρονται σε ομάδες στοιχείων δεδομένων που εμφανίζονται συχνά μαζί σε διάφορα σημεία, όπως πεδία σε κλάσεις ή στις παραμέτρους μεθόδων. Αυτές οι ομάδες στοιχείων δεδομένων θα πρέπει ιδανικά να ενοποιηθούν στο δικό τους αντικείμενο για να απλοποιήσουν τη δομή και να διευκολύνουν την κλήση μεθόδων. Αυτή η διαδικασία μπορεί να βοηθήσει στην μείωση των παραμέτρων σε μεθόδους και στη βελτίωση της λειτουργικότητας και της οργάνωσης του κώδικα, οδηγώντας ενδεχομένως στον εντοπισμό χαρακτηριστικών που μπορούν να ενσωματωθούν στις νέες κλάσεις. Προτείνεται η δημιουργία ενός νέου αντικειμένου εάν η κατάργηση ενός στοιχείου δεδομένων από την ομάδα καθιστά τα υπόλοιπα στοιχεία παράλογα, στην συγκεκριμένη περίπτωση που προκύπτει αυτό αναδεικνύεται η ανάγκη δημιουργίας ενός αντικειμένου ώστε να απλοποιηθεί η δομή. [6]
6. *Κακή χρήση σχολίων (Comments)*: Η κακή χρήση σχολίων αναφέρεται στην πρακτική της χρήσης σχολίων για την επεξήγηση περίπλοκων ή ασαφών ενοτήτων κώδικα, που συχνά υποδηλώνει Code smell στον ίδιο τον κώδικα. Αντί να χρησιμοποιείτε σχόλια για την αποσαφήνιση αυτών των τμημάτων, συνιστάται η αναδόμηση του κώδικα για να αφαιρεθεί η ανάγκη για τα σχόλια/το σχόλιο. Αυτό βοηθά στη δημιουργία καθαρότερου και πιο διαχειρίσιμου κώδικα, όπου τα σχόλια χρησιμοποιούνται πιο αποτελεσματικά για να υποδηλώσουν περιοχές αβεβαιότητας ή για να εξηγήσουν τον λόγο πίσω από συγκεκριμένες επιλογές, βοηθώντας τους μελλοντικούς προγραμματιστές να κατανοήσουν και να τροποποιήσουν τον κώδικα πιο εύκολα. [6]
7. *Εμμονή πρωτογενών τύπων (Primitive Obsession)*: Η εμμονή με την χρήση των πρωτογενων τύπων είναι ένα σενάριο συγγραφής κώδικα όπου οι προγραμματιστές βασίζονται υπερβολικά σε πρωτογενείς τύπους για να αναπαραστήσουν διαφορετικές έννοιες σε ένα πρόγραμμα, αντί να χρησιμοποιούν μικρά αντικείμενα ή κλάσεις για να ενθυλακώσουν

σχετικά δεδομένα. Αυτό μπορεί να οδηγήσει σε πολύπλοκες και ακατάστατες δομές κώδικα. [6]

8. *Ζήλια Ιδιοτήτων (Feature Envy)*: Το Feature Envy είναι μια κατάσταση όπου μια μέθοδος σε μια κλάση φαίνεται να ενδιαφέρεται περισσότερο για τα δεδομένα μιας άλλης κλάσης, συχνά βασίζοντας σε αυτά για να εκτελέσει υπολογισμούς ή πράξεις. Μια ενέργεια για να διορθωθεί αυτό είναι η μετακίνηση της μεθόδου στην κλάση που φαίνεται να ενδιαφέρεται περισσότερο. Ο στόχος είναι να ομαδοποιηθούν τα δεδομένα και οι συμπεριφορές που χρησιμοποιούν αυτά τα δεδομένα μαζί, καθιστώντας τον κώδικα πιο οργανωμένο και αποτρέποντας τις διάσπαρτες αλλαγές. [6]

1.2.2 Τεχνικές αναδόμησης

Οι τεχνικές αναδιαμόρφωσης είναι συστηματικές μέθοδοι που χρησιμοποιούνται για τη βελτίωση της δομής, της αναγνωσιμότητας και της αποτελεσματικότητας του υπάρχοντος κώδικα, χωρίς να αλλοιώνεται η εξωτερική συμπεριφορά ή λειτουργικότητά του. Αυτές οι τεχνικές στοχεύουν να κάνουν τον κώδικα πιο οργανωμένο, διατηρήσιμο και προσαρμόσιμο σε μελλοντικές αλλαγές. Η εφαρμογή τεχνικών αναδόμησης δεν αφορά την προσθήκη νέων λειτουργιών, αλλά τη βελτίωση των υπαρχόντων για βέλτιστη απόδοση και δυνατότητα συντήρησης. Ας εξερευνήσουμε μερικές από τις βασικές τεχνικές αναδόμησης που είναι καθοριστικές για την επίτευξη μιας καλά δομημένης, αποτελεσματικής και καθαρής βάσης κώδικα:

ΠΡΙΝ

```
void printOwing() {
  printBanner();

  // Print details.
  System.out.println("name: " + name);
  System.out.println("amount: " + getOutstanding());
}
```

ΜΕΤΑ

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails(double outstanding) {
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}
```

Σχήμα 1.1: Παράδειγμα Extract Method [11].

Extract Method: Η Εξαγωγή μεθόδου είναι μια κοινή τεχνική αναδόμησης όπου τμήματα κώδικα από μια μεγαλύτερη μέθοδο μεταφέρονται σε μια νέα, καλά ονομασμένη μέθοδο. Αυτό προάγει την αναγνωσιμότητα και την επαναχρησιμοποίηση, επιτρέποντας στις μεθόδους υψηλότερου επιπέδου να είναι πιο κατανοητές και να μοιάζουν σαν μια σειρά σχολίων. [6]

ΠΡΙΝ

```
class PizzaDelivery {
  // ...
  int getRating() {
    return moreThanFiveLateDeliveries() ? 2 : 1;
  }
  boolean moreThanFiveLateDeliveries() {
    return numberOfLateDeliveries > 5;
  }
}
```

ΜΕΤΑ

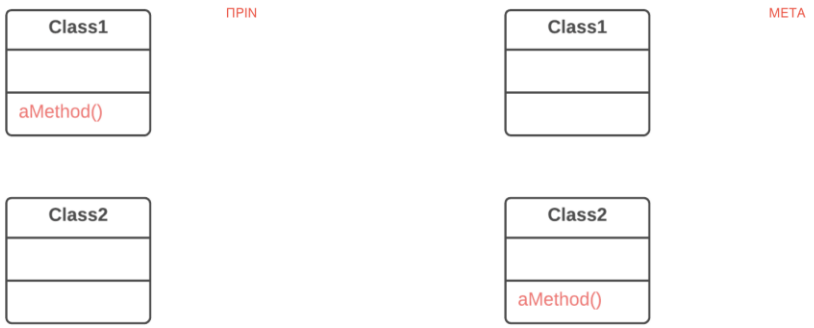
```
class PizzaDelivery {
  // ...
  int getRating() {
    return numberOfLateDeliveries > 5 ? 2 : 1;
  }
}
```

Σχήμα 1.2: Παράδειγμα Inline Method [11].

Inline Method: Είναι μια τεχνική αναδόμησης που στοχεύει στην απλοποίηση της δομής του κώδικα. Εάν διαπιστωθεί ότι το κομμάτι κωδικα μιας μεθόδου μεταφέρει ξεκάθαρα τις ίδιες πληροφορίες με το όνομά της, αυτή η τεχνική προτείνει την πλήρη κατάργηση της μεθόδου για να αποφευχθεί η περιττή έμμεση κατεύθυνση που μπορεί να κάνει τον κώδικα πιο δυσανάγνωστο. Να σημειωθεί όμως

πως η εφαρμογή της τεχνικής "Inline Method" μπορεί να είναι περίπλοκη. Παρόλο που τα βασικά βήματα απαιτούν την αντικατάσταση των κλήσεων μεθόδου με το σώμα της μεθόδου και την κατάργηση του ορισμού της, συναντώνται προκλήσεις όπως η αντιμετώπιση αναδρομών ή πολλαπλών σημείων επιστροφής, που μπορούν να περιπλέξουν τη διαδικασία. Σε περίπτωση ανακύκλωσης τέτοιων πολυπλοκότητας, ίσως είναι προτιμότερο να αποφευχθεί η χρήση αυτής της τεχνικής ανακατασκευής. [6]

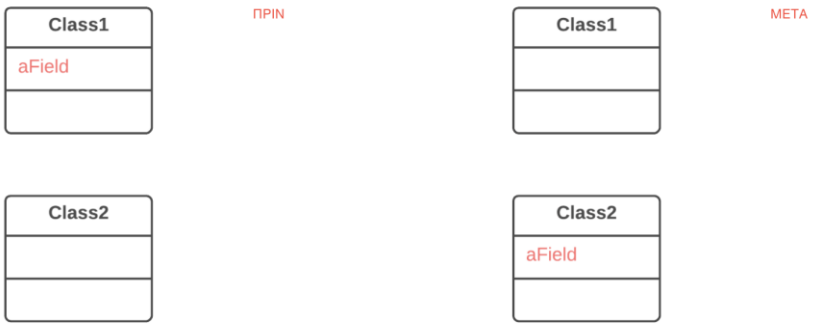
Move Method/Field: Οι τεχνικές "Μετακίνησης Μεθόδου" και "Μετακίνησης Πεδίου" είναι βασικές στρατηγικές στη διαδικασία της αναδιάρθρωσης κώδικα, βοηθώντας στην οργάνωση και απλοποίηση του κώδικα [6]



.

Σχήμα 1.3: Παράδειγμα Move Method [11].

- **Move Method:** Στη τεχνική αυτή, εντοπίζεται μια μέθοδος που φαίνεται να αλληλεπιδρά περισσότερο με μια άλλη κλάση από την κλάση στην οποία βρίσκεται τώρα. Η διαδικασία απαιτεί τη μετακίνηση της μεθόδου σε μια κλάση όπου φαίνεται να ανήκει περισσότερο, δημιουργώντας έτσι μια πιο συνεκτική και οργανωμένη δομή κώδικα. [6]



Σχήμα 1.4: Παράδειγμα Move Field [11].

- **Move Field:** Αυτή η τεχνική αφορά τη μετακίνηση ενός πεδίου που φαίνεται να είναι πιο στενά συνδεδεμένο με μια άλλη κλάση απ' ό,τι με την τρέχουσα κλάση. Η διαδικασία περιλαμβάνει την απομόνωση του πεδίου και τη μετακίνηση του στη νέα κλάση, ενημερώνοντας όλες τις αναφορές προς αυτό, για να διασφαλιστεί η σωστή λειτουργικότητα και η συνοχή του κώδικα. [6]

ΠΡΙΝ

```
double potentialEnergy(double mass, double height) {
  return mass * height * 9.81;
}
```

ΜΕΤΑ

```
static final double GRAVITATIONAL_CONSTANT = 9.81;

double potentialEnergy(double mass, double height) {
  return mass * height * GRAVITATIONAL_CONSTANT;
}
```

Σχήμα 1.5: Παράδειγμα Replace Magic Number with Symbolic Constant [11].

Replace Magic Number with Symbolic Constant: Πρόκειται για μια τεχνική αναδόμησης που ενισχύει την αναγνωσιμότητα και τη δυνατότητα συντήρησης του κώδικα. Αυτή η προσέγγιση προτείνει την αντικατάσταση των "μαγικών αριθμών" (κυριολεκτικών αριθμών με συγκεκριμένες, μη προφανείς εννοίες) με εύστοχα ονομασμένες σταθερές, καθιστώντας τον κώδικα πιο διαχειρίσιμο.
Στο πλαίσιο του προγραμματισμού, οι μαγικοί αριθμοί είναι προβληματικοί γιατί μπορούν να δημιουργήσουν σύγχυση και να καταστήσουν τον κώδικα πιο δύσκολο για προσαρμογές στο μέλλον. Η χρήση σταθερών όχι μόνο ανακουφίζει από αυτά τα ζητήματα αλλά έρχεται και χωρίς κόστος απόδοσης. Για να εφαρμοστεί αυτή η τεχνική, αρχικά, θα πρέπει να δηλωθεί μια σταθερά και να ανατεθεί σε αυτή η τιμή του μαγικού αριθμού. Επόμενο βήμα είναι η αντικατάσταση όλων των εμφανίσεων του μαγικού αριθμού στον κώδικα με την ονομασία της σταθεράς. Έπειτα, διενεργείται συντακτικός έλεγχος για να διασφαλιστεί ότι όλες οι λειτουργίες παραμένουν αμετάβλητες και λειτουργούν όπως αναμενόταν. [6]

ΠΡΙΝ

```
if (date.before(SUMMER_START) || date.after(SUMMER_END)) {
  charge = quantity * winterRate + winterServiceCharge;
}
else {
  charge = quantity * summerRate;
}
```

ΜΕΤΑ

```
if (isSummer(date)) {
  charge = summerCharge(quantity);
}
else {
  charge = winterCharge(quantity);
}
```

Σχήμα 1.6: Παράδειγμα Decompose Conditional [11].

Decompose Conditional: Η τεχνική "Decompose Conditional" αναφέρεται στη διαδικασία απλοποίησης περίπλοκων υποχρεωτικών (if-then-else) δηλώσεων μέσα σε ένα πρόγραμμα. Αυτό επιτυγχάνεται μέσω της εξαγωγής των συνθηκών, καθώς και των τμημάτων "then" και "else" σε ξεχωριστές μεθόδους. Αυτό βοηθά να επισημανθεί η συνθήκη και ο λόγος για τον οποίο γίνεται η διακλάδωση, καθιστώντας τον κώδικα πιο κατανοητό, ευανάγνωστο και διαχειρίσιμο. [6]

ΠΡΙΝ ✓

```
class Bird {
  // ...
  double getSpeed() {
    switch (type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (isNailed) ? 0 : getBaseSpeed(voltage);
    }
    throw new RuntimeException("Should be unreachable");
  }
}
```

ΜΕΤΑ ✓

```
abstract class Bird {
  // ...
  abstract double getSpeed();
}

class European extends Bird {
  double getSpeed() {
    return getBaseSpeed();
  }
}
class African extends Bird {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
  }
}
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(voltage);
  }
}

// Somewhere in client code
speed = bird.getSpeed();
```

Σχήμα 1.7: Παράδειγμα Replace Conditional with Polymorphism [11].

Replace Conditional with Polymorphism: είναι μια τεχνική ανακατασκευής κώδικα που στοχεύει στην απλοποίηση διακλαδώσεων υπό όρους χρησιμοποιώντας τον πολυμορφισμό που παρέχεται από τον αντικειμενοστραφή προγραμματισμό.
Αρχικά, πρέπει να δημιουργηθεί μια κατάλληλη ιεραρχία κληρονομικότητας με υποκλάσεις που αντιπροσωπεύουν τους διάφορους τύπους αντικειμένων που εμπλέκονται στις συνθήκες. Στη συνέχεια, πρέπει να μετακινηθείκάθε ενότητα της συνθήκης (π.χ. κάθε περίπτωση δήλωσης switch) σε μια μέθοδο που παρακάμπτεται σε κάθε υποκλάση, παρέχοντας τη συγκεκριμένη υλοποίηση για τον συγκεκριμένο τύπο αντικειμένου.
Τελικά, η αρχική μέθοδος γίνεται αφηρημένη, αναγκάζοντας τις υποκλάσεις να παρέχουν τις δικές τους συγκεκριμένες υλοποιήσεις. Αυτό επιτρέπει μια πιο καθαρή, πιο ευέλικτη και διαχειρίσιμη δομή κώδικα, καθώς αποφεύγει την ανάγκη ενημέρωσης πολλαπλών σημείων κώδικα κατά την προσθήκη ή την τροποποίηση τύπων αντικειμένων. [6]

ΠΡΙΝ

```
class Person {
  public String name;
}
```

ΜΕΤΑ

```
class Person {
  private String name;

  public String getName() {
    return name;
  }
  public void setName(String arg) {
    name = arg;
  }
}
```

Σχήμα 1.8: Παράδειγμα Encapsulate Field [11].

Encapsulate Field: Το "Encapsulate Field" είναι μια τεχνική που χρησιμοποιείται για να ελέγχει την πρόσβαση σε ένα πεδίο (field) μιας κλάσης (class), κρύβοντας τις εσωτερικές του λεπτομέρειες και παρέχοντας δημόσιες μεθόδους για την πρόσβαση και την τροποποίησή του. Με αυτόν τον τρόπο, το πεδίο γίνεται κρυφό (encapsulated) μέσα στην τάξη, προστατεύοντας το από ανεπιθύμητες προσβάσεις ή τροποποιήσεις από τον εξωτερικό κώδικα.
Αυτό επιτυγχάνεται συνήθως ορίζοντας το πεδίο ως ιδιωτικό (private) και δημιουργώντας δημόσιες μεθόδους (public methods), γνωστές ως getters και setters, για τη λήψη και τον καθορισμό της τιμής του πεδίου αντίστοιχα. [6]

ΠΡΙΝ

```
int withdraw(int amount) {
  if (amount > _balance) {
    return -1;
  }
  else {
    balance -= amount;
    return 0;
  }
}
```

ΜΕΤΑ

```
void withdraw(int amount) throws BalanceException {
  if (amount > _balance) {
    throw new BalanceException();
  }
  balance -= amount;
}
```

Σχήμα 1.9: Παράδειγμα Replace Error Code with Exception [11].

Replace Error Code with Exception: Ο όρος "Replace Error Code with Exception" αναφέρεται στην αντικατάσταση των κωδικών σφαλμάτων με exceptions, για να διευκολύνει τη διαχείριση σφαλμάτων και να καταστήσει τον κώδικα πιο καθαρό και κατανοητό. [6]

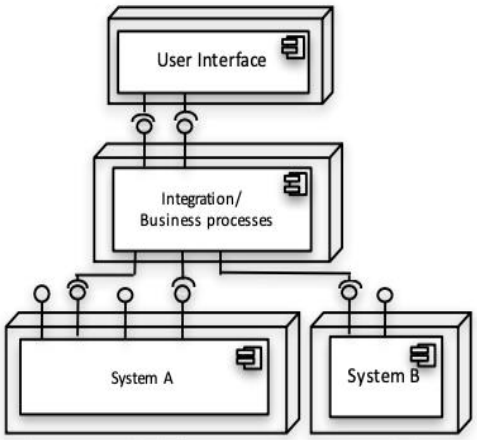
1.3 Ανάπτυξη λογισμικού βασισμένο σε υπηρεσίες και μικροϋπηρεσίες

Στο πλαίσιο των εξελισσόμενων απαιτήσεων του κλάδου και της αυξανόμενης πολυπλοκότητας των εταιρικών εφαρμογών, η Αριτεκτονική με προσανατολισμό στις υπηρεσίες – Service Oriented Architecture (SOA) αναδείχθηκε ως μια σημαντική λύση [7]. Έχει διαδραματίσει σοβαρό ρόλο στην αντιμετώπιση των αναγκών των μεγάλων επιχειρήσεων και έχει αντικαταστήσει προηγούμενες αρχιτεκτονικές προσεγγίσεις όπως το Common Object Request Broker Architecture (CORBA) και το Enterprise Service Bus [7].

Το SOA έχει σχεδιαστεί για να παρέχει μια δομημένη προσέγγιση στο σχεδιασμό και την αρχιτεκτονική λογισμικού, εστιάζοντας στην αποσύνθεση συστημάτων σε αρθρωτές υπηρεσίες που μπορούν να προσπελαστούν μέσω δικτύου και να ενσωματωθούν σε διάφορες πλατφόρμες [7]. Αυτές οι υπηρεσίες συνεργάστηκαν για την παροχή συνολικής λειτουργικότητας του συστήματος, δίνοντας έμφαση στην απλότητα στο σχεδιασμό και την ενσωμάτωση των υπηρεσιών μέσω έξυπνων μηχανισμών δρομολόγησης. [7]

Οι υπηρεσίες SOA λειτουργούν ανεξάρτητα, αντιδρώντας σε συμβάντα χωρίς προηγούμενη γνώση των ενεργειών, επιτρέποντας την απρόσκοπτη ενσωμάτωση νέων υπηρεσιών χωρίς να διακόπτονται οι υπάρχουσες. Ωστόσο, είναι σημαντικό να κατανοήσουμε ότι ενώ το SOA μοιράζεται κοινούς στόχους με την Αρχιτεκτονική Microservice, ακολουθεί μια ξεχωριστή διαδρομή για την επίτευξη των στόχων του. Παμε λοιπόν να εμβαθύνουμε σε αυτές τις αρχιτεκτονικές. [7]

1.3.1 Αρχιτεκτονική με προσανατολισμό στις υπηρεσίες



Σχήμα 1.10: Διάγραμμα παραδείγματος αρχιτεκτονικής SOA [7].

Η Service-Oriented Architecture (SOA) είναι μια ολοκληρωμένη αρχιτεκτονική προσέγγιση που στοχεύει να μετατρέψει πολύπλοκα συστήματα σε ένα δίκτυο αυτόνομων υπηρεσιών [7]. Αυτές οι υπηρεσίες ενσωματώνουν συγκεκριμένες λειτουργίες και είναι προσβάσιμες μέσω ενός δικτύου, επιτρέποντάς τους να εκτελούν καλά καθορισμένες εργασίες ή λειτουργίες. Αυτό το αρχιτεκτονικό παράδειγμα προωθεί πολλές βασικές αρχές και χαρακτηριστικά, τα οποια περιγράφονται παρακάτω:

1. Υπηρεσιακή αρθρότητα: Στην καρδιά της SOA βρίσκεται η έννοια της αρθρωτής υπηρεσίας. Αυτό σημαίνει διάσπαση ενός πολύπλοκου συστήματος σε διακριτές, αυτόνομες υπηρεσίες.

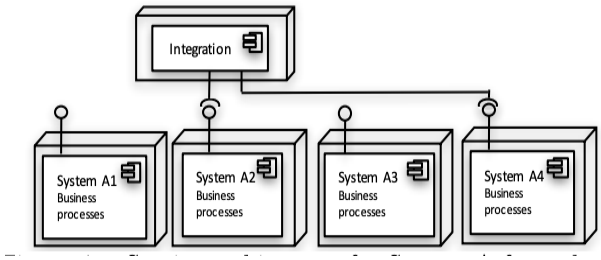
Κάθε υπηρεσία εστιάζει σε μια συγκεκριμένη πτυχή της συνολικής λειτουργικότητας, επιτρέποντας μια αρθρωτή και οργανωμένη προσέγγιση στο σχεδιασμό του συστήματος. [7]

2. Προκαταρκτικές Δεσμεύσεις: Η εφαρμογή SOA απαιτεί συχνά μια σημαντική αρχική δέσμευση. Οι εταιρίες που υιοθετούν το SOA πρέπει να αναλάβουν μια διαδικασία μετασχηματισμού που περιλαμβάνει την αναδιάρθρωση ολόκληρης της υποδομής πληροφορικής τους σε ξεχωριστές υπηρεσίες. Ενώ η εισαγωγή νέων υπηρεσιών είναι σχετικά απλή, η πιο προηγμένη υιοθέτηση περιλαμβάνει τη διαίρεση των υπαρχουσών εφαρμογών σε πολλαπλές υπηρεσίες για την προώθηση μεγαλύτερης επαναχρησιμοποίησης και ενορχήστρωσης υπηρεσιών. [7]
3. Κεντρική Διακυβέρνηση: Αναπόσπαστο στοιχείο του SOA είναι η καθιέρωση κεντρικής διακυβέρνησης. Αυτό το πλαίσιο διακυβέρνησης ορίζει και επιβάλλει κανόνες, πολιτικές και πρότυπα που διέπουν το σχεδιασμό, την υλοποίηση και τη διαχείριση των υπηρεσιών εντός του οικοσυστήματος SOA. Η κεντρική οργανωτική διακυβέρνηση διασφαλίζει ότι οι υπηρεσίες ευθυγραμμίζονται με τους στόχους διατηρώντας παράλληλα την ποιότητα, την ασφάλεια και το compliance. [7]
4. Ένταξη και επικοινωνία: Η αποτελεσματική ενοποίηση και επικοινωνία μεταξύ των υπηρεσιών είναι θεμελιώδεις για την επιτυχία της SOA. Το Enterprise Service Bus (ESB) χρησιμεύει συχνά ως το βασικό στοιχείο που είναι υπεύθυνο για τη διαχείριση της επικοινωνίας μεταξύ των υπηρεσιών. Το ESB διευκολύνει όχι μόνο τη δρομολόγηση μηνυμάτων αλλά και την ενορχήστρωση υπηρεσιών, την επεξεργασία συμβάντων, τη συσχέτιση και την παρακολούθηση της επιχειρηματικής δραστηριότητας. Οι υπηρεσίες μπορούν να αλληλεπιδρούν μέσω μηνυμάτων ή συμβάντων, επιτρέποντας σε πολλές υπηρεσίες να ανταποκρίνονται σε συγκεκριμένα συμβάντα, ακόμη και όταν δεν έχουν επίγνωση των ενεργειών. [7]
5. Ευελιξία και επιχειρηματικές διαδικασίες: Το SOA προσφέρει σημαντική ευελιξία στον καθορισμό και την προσαρμογή των επιχειρηματικών διαδικασιών. Επιτρέπει τη δυναμική σύνθεση των υπηρεσιών, επιτρέποντας τη δημιουργία προσαρμόσιμων ροών εργασίας. Αυτή η ευελιξία είναι ιδιαίτερα πολύτιμη για την εισαγωγή εναλλακτικών ροών διεργασιών ή την προσθήκη νέων λειτουργιών πάνω από τις υπάρχουσες υπηρεσίες. [7]
6. Ανοιχτότητα σε τρίτους: Η SOA παρέχει το πλαίσιο για να κάνει τις υπηρεσίες προσβάσιμες σε εξωτερικούς φορείς, όπως συνεργάτες ή εξωτερικούς προγραμματιστές. Αυτό το άνοιγμα μπορεί να προωθήσει τη συνεργασία και την καινοτομία επεκτείνοντας τη χρήση υπηρεσιών πέρα από τα όρια του οργανισμού. [7]
7. Πολυπλοκότητα και μονολιθικές αναπτύξεις: Ενώ το SOA προσφέρει πολλά πλεονεκτήματα, μπορεί να εισάγει πολυπλοκότητα, ειδικά κατά την αρχική φάση ανάπτυξης. Η πλατφόρμα γεφλυρωσης, συνήθως η ESB, μπορεί να γίνει πολύπλοκη και ενδεχομένως να λειτουργήσει ως εμπόδιο λόγω των γενικών εξόδων επικοινωνίας και της διαχείρισης των κατανεμημένων συναλλαγών. [7]
8. Εκδόσεις και συντήρηση: Η έκδοση εκδόσεων συστήματος αποτελεί σημαντική πρόκληση στο SOA. Καθώς οι υπηρεσίες ενδέχεται να μην γνωρίζουν τους χρήστες τους, οι οργανισμοί διατηρούν συχνά πολλαπλές εκδόσεις της ίδιας υπηρεσίας με ελαφρώς τροποποιημένες διεπαφές για να φιλοξενούν διαφορετικές μορφές δεδομένων. Η διαχείριση αυτών των εκδόσεων απαιτεί προσεκτική παρακολούθηση και συντήρηση για την εξασφάλιση απρόσκοπτης λειτουργίας. [7]
9. Συνεργασία ομάδας: Οι αναπτύξεις SOA συχνά περιλαμβάνουν διαφορετικές ομάδες υπεύθυνες για διάφορες πτυχές, όπως διεπαφές χρήστη και μεμονωμένες υπηρεσίες. Ο συντονισμός αλλαγών και ενημερώσεων σε αυτές τις ομάδες μπορεί να είναι δύσκολος και

ζητήματα σε μία υπηρεσία μπορεί να επηρεάσουν ολόκληρη την ανάπτυξη της εφαρμογής, απαιτώντας περίπλοκες διαδικασίες επαναφοράς. [7]

10. Κεντρική διαχείριση: Για τη διαχείριση αλλαγών στην υποδομή υπηρεσιών SOA, οι οργανισμοί συχνά δημιουργούν μια κεντρική μονάδα διαχείρισης. Αν και αυτό μπορεί να προσφέρει διακυβέρνηση και επίβλεψη, μπορεί επίσης να οδηγήσει σε συντηρητισμό και να περιορίσει την εξέλιξη των μεμονωμένων υπηρεσιών. [7]

1.3.2 Μικροϋπηρεσίες



Σχήμα 1.11: Διάγραμμα παραδείγματος Αρχιτεκτονικής Μικροϋπηρεσιών για την υπηρεσία Α[7]

Οι Μικροϋπηρεσίες, αντιπροσωπεύουν ένα αρχιτεκτονικό παράδειγμα που αντλεί έμπνευση από τρεις θεμελιώδεις αρχές του Unix:

- **Modularization**: Κάθε μικροϋπηρεσία έχει σχεδιαστεί για να εκπληρώνει μια συγκεκριμένη εργασία, τηρώντας τη φιλοσοφία Unix ότι ένα πρόγραμμα πρέπει να εκτελεί μια εργασία καλά. Αυτή η ιδέα προωθεί μια αρθρωτή προσέγγιση στο σχεδιασμό του συστήματος, όπου πολύπλοκες εφαρμογές αποδομούνται σε μικρότερες, αυτόνομες μονάδες. [7]
- **Interoperability**: Οι μικροϋπηρεσίες έχουν σχεδιαστεί για να συνεργάζονται απρόσκοπτα, ευθυγραμμίζοντας με την ιδέα ότι τα προγράμματα πρέπει να μπορούν να συνεργάζονται αποτελεσματικά. Αυτή η διυπηρεσιακή επικοινωνία επιτρέπει τη σύνθεση εξελιγμένων λειτουργιών από μεμονωμένες μικροϋπηρεσίες. [7]
- **Universal Interface**: Οι μικροϋπηρεσίες χρησιμοποιούν μια καθολική διεπαφή, η οποία τους επιτρέπει να αλληλεπιδρούν μεταξύ τους ανεξάρτητα από την υποκείμενη στοίβα τεχνολογίας. Αυτή η αρχή ενθαρρύνει ένα συνεκτικό οικοσύστημα όπου οι υπηρεσίες επικοινωνούν αποτελεσματικά. [7]

Η βασική φιλοσοφία των μικροϋπηρεσιών περιστρέφεται γύρω από τη δημιουργία επαναχρησιμοποιήσιμων και καλά καθορισμένων στοιχείων που υποστηρίζουν τη αυτονομία.
Σε αντίθεση με τις παραδοσιακές μονολιθικές αρχιτεκτονικές, οι μικροϋπηρεσίες ασπάζονται την ιδέα ότι οι υπηρεσίες μπορούν να εισαχθούν ανεξάρτητα στην παραγωγή. Αυτή η ανεξαρτησία δεν επηρεάζει μόνο την ανάπτυξη αλλά επίσης επηρεάζει τις προσπάθειες εξέλιξης και τροποποίησης.
Οι υποστηρικτές των Microservices συχνά επικαλούνται τον νόμο του Conway, ο οποίος δηλώνει ότι "Οι οργανισμοί που σχεδιάζουν συστήματα περιορίζονται να παράγουν σχέδια που είναι αντίγραφα των δομών επικοινωνίας αυτών των οργανισμών." Στο πλαίσιο των μικροϋπηρεσιών, αυτός ο νόμος υπογραμμίζει τη σχέση μεταξύ της οργανωτικής δομής και του σχεδιασμού του συστήματος που προκύπτει. [7]

Οι μικροϋπηρεσίες εξελίσσονται προς ελαφριές εικονικές μηχανές, που υλοποιούνται ως κοντέινερ (π.χ. Docker) ή μεμονωμένες διαδικασίες. Αυτή η αρχιτεκτονική επιλογή εξαλείφει τις εξαρτήσεις από συγκεκριμένες τεχνολογίες, επιτρέποντας υποδομές για συγκεκριμένες υπηρεσίες προσαρμοσμένες στις ανάγκες κάθε μικροϋπηρεσίας. [7]
Σε αντίθεση με την Service-Oriented Architecture (SOA), οι μικροϋπηρεσίες προτιμούν τη χορογραφία έναντι της ενορχήστρωσης. Οι επιχειρηματικές διαδικασίες είναι ενσωματωμένες σε μεμονωμένες μικροϋπηρεσίες, εξαλείφοντας την ανάγκη για κεντρική λογική στην ενοποίηση. Κάθε μικροϋπηρεσία είναι αυτόνομη, υπεύθυνη για τις αλληλεπιδράσεις της με άλλες υπηρεσίες. [7]
Οι μικροϋπηρεσίες δίνουν έμφαση στην απομόνωση, με κάθε υπηρεσία να διαχειρίζεται συνήθως μία ομάδα προγραμματιστών. Για την αποτελεσματική διαχείριση των αλλαγών, τόσο η διεπαφή χρήστη όσο και η ίδια η μικροϋπηρεσία βρίσκονται υπό τον έλεγχο της ίδιας ομάδας. Αυτή η προσέγγιση προσφέρει ευελιξία στην προσαρμογή στις εξελισσόμενες απαιτήσεις. [7]
Η ανεξαρτησία στην ανάπτυξη και την εγκατάσταση αποτελεί χαρακτηριστικό γνώρισμα των μικροϋπηρεσιών. Υποστηρίζουν την ατομική επεκτασιμότητα, τη συνεχή παράδοση και την ευρωστία, με αιτήματα που κατανέμονται έξυπνα σε πολλαπλές περιπτώσεις υπηρεσιών. [7]

1.3.3 Microservices vs SOA

Οι μικροϋπηρεσίες και η Service Oriented Architecture (SOA) στοχεύουν και οι δύο να χωρίσουν τα συστήματα σε υπηρεσίες, αλλά το κάνουν με θεμελιωδώς διαφορετικούς τρόπους. Αυτή η ενότητα θα παρέχει μια άμεση σύγκριση μεταξύ αυτών των δύο αρχιτεκτονικών παραδειγμάτων χρησιμοποιώντας τις πληροφορίες που παρείχατε, επισημαίνοντας τις βασικές τους διακρίσεις:

Προοπτική ανάπτυξης:Το SOA μπορεί συχνά να εκληφθεί ως μονόλιθος από την άποψη της ανάπτυξης.Οι μServices υποστηρίζουν ανεξάρτητες αναπτύξεις, στενά ευθυγραμμισμένες με τεχνολογίες κοντέινερ, απλοποιώντας την αυτοματοποιημένη ανάπτυξη. [7]
Αυτονομία σχεδίασης:Η SOA διατηρεί ένα ευρύ επιχειρηματικό εύρος και ευελιξία μέσω της κεντρικής διαχείρισης. Οι μικροϋπηρεσίες δίνουν έμφαση στο να κάνουμε «ένα πράγμα καλά» και προάγουν την αυτονομία του σχεδιασμού, με αποτέλεσμα συχνά την ετερογένεια των στοιχείων. [7]
Cloud-Native Alignment: Οι μικροϋπηρεσίες σχετίζονται στενά με τις εγγενείς αρχιτεκτονικές του cloud, επωφελούμενοι από την ανάπτυξη μεμονωμένων και αυτοματοποιημένων υπηρεσιών. [7]
Το SOA σπάνια αναφέρεται ως «εγγενές στο σύννεφο» στη βιβλιογραφία, χωρίς το ίδιο επίπεδο ευθυγράμμισης με τις αρχές του υπολογιστικού νέφους. [7]
Μελλοντική Κατεύθυνση: Οι τάσεις του κλάδου και η πρόσφατη έρευνα ευνοούν τις μικροϋπηρεσίες ως τη μελλοντική κατεύθυνση, με το SOA να θεωρείται όλο και περισσότερο ως κληρονομιά. Ωστόσο, υπάρχουν αντεπιχειρήματα που υποδηλώνουν ότι το SOA και οι μικροϋπηρεσίες είναι θεμελιωδώς διακριτά, με βασικά κομμάτια να λείπουν. [7]
Θεμελιώδεις έννοιες: Η αρχιτεκτονική μικροϋπηρεσίες ακολουθεί ένα μοτίβο "κοινή χρήση όσο το δυνατόν λιγότερο", δίνοντας μεγάλη έμφαση στο περιορισμένο πλαίσιο. [7]
Το SOA ακολουθεί ένα μοτίβο "κοινή χρήση όσο το δυνατόν λιγότερο", δίνοντας έμφαση στην αφαίρεση και την επαναχρησιμοποίηση της επιχειρηματικής λειτουργικότητας. [7]
Καταλληλότητα επιχείρησης: Το SOA είναι πιο κατάλληλο για μεγάλες, πολύπλοκες υποδομές σε επίπεδο επιχείρησης, ιδιαίτερα εκείνες με πολλά κοινά στοιχεία. [7]
Οι μικροϋπηρεσίες είναι πιο κατάλληλες για μικρότερες, καλά διαμερισμένες εφαρμογές που βασίζονται στον ιστό και ενδέχεται να μην περιλαμβάνουν ενδιάμεσο λογισμικό ανταλλαγής μηνυμάτων. [7]

Αποσύνδεση συμβολαίου: Η SOA υπερέχει στην υποστήριξη των καταναλωτών υπηρεσιών και υπηρεσιών που μπορούν να εξελιχθούν χωριστά, διατηρώντας παράλληλα συμβατικές συμφωνίες.
Οι μικροϋπηρεσίες μπορεί να δυσκολεύονται να επιτύχουν την αποσύνδεση συμβάσεων, μια κύρια δυνατότητα του SOA. [7]
Δυνατότητες ενσωμάτωσης: Το SOA είναι ανώτερο όταν πρόκειται για την ενοποίηση ετερογενών συστημάτων και υπηρεσιών, προσφέροντας ευρύτερες επιλογές για την ενοποίηση υπηρεσιών.
Οι Υπηρεσίες τείνουν να μειώνουν τις επιλογές για ενσωμάτωση υπηρεσιών. [7]
Συνοπτικά, η σύγκριση μεταξύ των μικροϋπηρεσίων και του SOA υπογραμμίζει τις αντίθετες προσεγγίσεις τους στην αρχιτεκτονική προσανατολισμένη στις υπηρεσίες. Ενώ οι μΥπηρεσίες ευνοούν την αυτονομία, την ατομική ανάπτυξη και την εγγενή ευθυγράμμιση στο cloud, το SOA λάμπει σε πολύπλοκα επιχειρηματικά σενάρια, αποσύνδεση συμβολαίων και ετερογενή ενοποίηση συστημάτων. Αυτές οι διακρίσεις απεικονίζουν τα μοναδικά πλεονεκτήματα και τις ανταλλαγές που σχετίζονται με κάθε αρχιτεκτονικό παράδειγμα, βοηθώντας τους οργανισμούς να κάνουν συνειδητές επιλογές με βάση τις συγκεκριμένες ανάγκες και τα πλαίσιά τους.

1.4 Στόχος εργασίας

Ο βασικός προσανατολισμός της παρούσας εργασίας κατέστησε αναγκαία τη δημιουργία μιας εξειδικευμένης υπηρεσίας API, εστιάζοντας αποκλειστικά στις λειτουργίες των εργαλείων GEA και Semi. Η πρωταρχική επιδίωξη ήταν η ανάπτυξη ενός συστήματος που θα προσέφερε όχι μόνο λειτουργικότητα, αλλά και ευκολία στη συντήρηση, αναγνώση και χρήση.

Ένας επιπλέον στόχος αποτελούσε η διασφάλιση της προσβασιμότητας αυτών των δύο εργαλείων μέσω του Διαδικτύου, καθιστώντας τα πιο φιλικά προς τον χρήστη και εύχρηστα για εξωτερικές πλατφόρμες. Η ιδέα ήταν να απλοποιηθεί η χρήση τους και να ενισχυθεί η συμβατότητά τους.

Εκτός από αυτά, η εργασία επιδίωκει τη βελτίωση του κώδικα των εργαλείων, εφαρμόζοντας μια στρατηγική αναδιάρθρωσης ορισμένων τμημάτων κώδικα. Αυτή η προσέγγιση συνέβαλε στην απλούστευση και βελτίωση της διαχείρισης του κώδικα, προσφέροντας μια πιο βιώσιμη προσέγγιση στην συντήρηση.

Κεφάλαιο 2ο: Μεθοδολογία

2.1 Εισαγωγή στο Spring Framework και τα εργαλεία που χρησιμοποιήθηκαν

2.1.1 Τι είναι Java;

Η Java είναι μια ευρέως χρησιμοποιούμενη αντικειμενοστραφής γλώσσα προγραμματισμού που λειτουργεί σε δισεκατομμύρια συσκευές, από φορητούς υπολογιστές και smartphones έως παιχνιδομηχανές και ιατρικό εξοπλισμό, τρέχουν εφαρμογές που έχουν αναπτυχθεί με αυτή τη γλώσσα[8]. Οι κανόνες και η σύνταξη της Java βασίζονται στις γλώσσες C και C++[8].

Ένα από τα πρωταρχικά προτερήματα της Java ήταν η ικανότητα φορητότητας των εφαρμογών της. Οποιοσδήποτε κώδικας αναπτυχθεί με Java σε έναν υπολογιστή μπορεί εύκολα να προσαρμοστεί και σε μια κινητή συσκευή[8]. Ο στόχος της γλώσσας, όταν δημιουργήθηκε το 1991 από τον James Gosling στη Sun Microsystems (η οποία αργότερα εντάχθηκε στην Oracle), ήταν να προσφέρει μια λύση που θα επέτρεπε τον κώδικα να "γράφεται μία φορά και να εκτελείται οπουδήποτε."[8].

2.1.2 Τι είναι το Java Spring Boot;

Το Java Spring Boot, που είναι γνωστότερο ως Spring Boot, είναι ένα προηγμένο εργαλείο ανάπτυξης που έχει κατασκευαστεί πάνω στο Java Spring Framework γνωστό και ως Spring Framework. Αποτελεί μία ανοικτού κώδικα πλατφόρμα, ιδιαίτερα εκτιμητέα στον χώρο της επιχειρησιακής τεχνολογίας, που επιτρέπει τη δημιουργία αυτόνομων εφαρμογών οι οποίες εκτελούνται στο Java Virtual Machine (JVM)[9].

Το Spring Boot δημιουργήθηκε με τον σκοπό να καταστήσει την ανάπτυξη διαδικτυακών εφαρμογών και μικροϋπηρεσιών πιο απλή και γρήγορη χρησιμοποιώντας το Spring Framework. Αυτό επιτυγχάνεται χάρη σε τρεις κύριες δυνατότητες που προσφέρει:

1. **Autoconfiguration**: Αυτή η δυνατότητα προετοιμάζει εφαρμογές με προκαθορισμένες εξαρτήσεις που απαλλάσσουν τους προγραμματιστές από τη χειροκίνητη διαμόρφωση[9].
2. **Opinionated Approach to Configuration**: Αντί οι προγραμματιστές να λαμβάνουν αποφάσεις σχετικά με τα πακέτα και τις διαμορφώσεις τους, το Spring Boot λαμβάνει αυτές τις αποφάσεις με βάση τις ανάγκες του project. Αυτή η προσέγγιση εκδηλώνεται κατά τη διαδικασία προετοιμασίας, όπου οι προγραμματιστές επιλέγουν από μια σειρά εξαρτήσεων εκκίνησης, γνωστές ως Spring Starters[9].
3. **Ability to Create Standalone Applications**: Οι εφαρμογές Spring Boot μπορούν να λειτουργούν ανεξάρτητα χωρίς να απαιτείται εξωτερικός διακομιστής web, κυρίως επειδή μπορεί να ενσωματώσει διακομιστές όπως το Tomcat ή το Netty κατά τη διαδικασία προετοιμασίας[9].

Αυτές οι δυνατότητες διασφαλίζουν συλλογικά ότι οι προγραμματιστές μπορούν να δημιουργήσουν μια εφαρμογή που βασίζεται στο Spring με ελάχιστο κόπο.

Η δημοτικότητα του Spring Framework μπορεί να αποδοθεί στο πρωτοποριακό χαρακτηριστικό του το dependency injection[9]. Αυτό επιτρέπει στα αντικείμενα να δηλώνουν τα dependencies τους, τα οποία στη συνέχεια εκπληρώνονται από το Spring container[9]. Αυτός ο μηχανισμός διευκολύνει την ανάπτυξη modular εφαρμογών με χαλαρά συνδεδεμένα στοιχεία, καθιστώντας τον ιδανικό για μικροϋπηρεσίες και εφαρμογές κατανεμημένου δικτύου[9].
Επιπλέον το Spring Framework παρέχει εξαιρετική υποστήριξη για διάφορες κοινές λειτουργίες των εφαρμογών, όπως το "data binding" (δέσμευση δεδομένων), "data conversion" (μετατροπή δεδομένων), "validation" (επικύρωση) και "exception handling" (διαχείριση εξαιρέσεων)[9]. Επιπλέον, το framework ενσωματώνει τέλεια με διάφορες τεχνολογίες Java EE, όπως το RMI, το AMQP και τα Java Web Services[9]. Αυτό καθιστά το Spring Framework ένα ιδανικό εργαλείο για την ανάπτυξη εφαρμογών Java EE σε διάφορες πλατφόρμες και περιβάλλοντα[9].

Τα πιο σημαντικά πλεονεκτήματα του Spring Boot σε σχέση με το Spring Framework είναι η φιλικότητα προς τον χρήστη και οι δυνατότητες ταχείας ανάπτυξης. Αν και η απευθείας εργασία με το Spring Framework μπορεί να προσφέρει μεγαλύτερη ευελιξία, στα περισσότερα πρακτικά σενάρια, η ευκολία και η ταχύτητα του Spring Boot υπερτερούν των πλεονεκτημάτων αυτής της ευελιξίας[9].
Το Spring Boot δεν συμβιβάζεται με τα χαρακτηριστικά γνωρίσματα του Spring Framework[9]. Εξακολουθεί να αξιοποιεί το διάσημο σύστημα σχολιασμού του Spring Framework, διευκολύνοντας την εύκολη έγχυση εξάρτησης πέρα από τα Spring Starters[9]. Επιπλέον, οι προγραμματιστές διατηρούν πλήρη πρόσβαση στις διάφορες δυνατότητες του Spring Framework, από τη διαχείριση συμβάντων έως τα ενσωματωμένα μέτρα ασφαλείας[9]. Ουσιαστικά, εάν το εύρος ενός έργου ευθυγραμμίζεται έστω και με ένα μόνο Spring Starter, το Spring Boot μπορεί να επιταχύνει σημαντικά τη διαδικασία ανάπτυξης[9].

2.1.3 Τι είναι το Hibernate

Το Hibernate είναι ένα εργαλείο αντικειμενο-σχεσιακής αντιστοίχισης (ORM) ανοιχτού κώδικα που έχει σχεδιαστεί για να απλοποιεί την αλληλεπίδραση μεταξύ αντικειμενοστρεφών εφαρμογών Java και σχεσιακών βάσεων δεδομένων εντός εφαρμογών Ιστού. Αυτή η τεχνολογία αντιμετωπίζει τις προκλήσεις της γεφύρωσης του χάσματος μεταξύ του αντικειμενοστρεφούς προγραμματισμού και του μοντέλου της σχεσιακής βάσης δεδομένων[10].
Στον πυρήνα του, το Hibernate χρησιμεύει ως πλαίσιο που διευκολύνει την αντιστοίχιση κλάσεων Java σε πίνακες βάσεων δεδομένων και τύπους δεδομένων Java σε τύπους δεδομένων SQL[10]. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να εργαστούν με αντικείμενα Java στον κώδικά τους και το Hibernate φροντίζει για τις υποκείμενες λειτουργίες της βάσης δεδομένων, όπως η αποθήκευση, η ανάκτηση και η αναζήτηση δεδομένων[10]. Το Hibernate προσφέρει πολλά πλεονεκτήματα, συμπεριλαμβανομένης της μείωσης του όγκου του κώδικα που πρέπει να γράψουν οι προγραμματιστές με τη διαχείριση της αντιστοίχισης του πίνακα αντικειμένων και τον χειρισμό της μετάφρασης μεταξύ αντικειμένων Java και τύπων δεδομένων SQL. Ως αποτέλεσμα, απλοποιεί τη διαδικασία ανάπτυξης, εξοικονομεί χρόνο και μειώνει το κόστος συντήρησης[10].
Σε σύγκριση με το παραδοσιακό JDBC (Java Database Connectivity), το οποίο είναι ένας μηχανισμός αλληλεπίδρασης βάσεων δεδομένων, χαμηλότερου επιπέδου. Το Hibernate παρέχει υψηλότερο επίπεδο αφαιρετικότητας, απλοποιεί τις εργασίες διατήρησης δεδομένων και είναι ιδιαίτερα χρήσιμο όταν εμπλέκονται πολύπλοκα αντικειμενοστραφή μοντέλα σε εφαρμογές Java[10].
Συνολικά, το Hibernate διαδραματίζει κρίσιμο ρόλο στη γεφύρωση του χάσματος μεταξύ αντικειμενοστρεφούς προγραμματισμού και σχεσιακών βάσεων δεδομένων, καθιστώντας το ένα πολύτιμο εργαλείο για τους προγραμματιστές Java που κατασκευάζουν εφαρμογές web. Η ιστορία

του χρονολογείται από το 2001 και έχει εξελιχθεί με τα χρόνια για να γίνει ένα ευρέως χρησιμοποιούμενο και αξιόπιστο πλαίσιο ORM στο οικοσύστημα της Java[10].

2.1.4 Τι είναι οι HTTP αιτήσεις

Οι HTTP αιτήσεις (HTTP requests) αποτελούν βασικά στοιχεία της επικοινωνίας μεταξύ client και server σε συστήματα που βασίζονται στο web[13]. Όταν ένας client επιχειρεί να αποκτήσει πρόσβαση σε έναν πόρο σε έναν server, εκκινεί μια αίτηση HTTP συνθέτοντας ένα δομημένο μήνυμα. Αυτό το μήνυμα περιλαμβάνει μια γραμμή αίτησης, ένα σύνολο HTTP επικεφαλίδων (HTTP Headers) και αν είναι εφαρμοστέο, ένα σώμα μηνύματος[13]. Η γραμμή αίτησης λειτουργεί ως πρόλογος, περιέχοντας μια μέθοδο αίτησης (HTTP Method), το στοιχείο διαδρομής του URL που υποδηλώνει τον επιθυμητό πόρο και τον αριθμό έκδοσης του HTTP. Είναι σημαντικό να σημειωθεί ότι η μέθοδος καθορίζει την ενέργεια που ο server πρέπει να πραγματοποιήσει, όπως η ανάκτηση ή η δημιουργία ενός πόρου[13].

Οι HTTP επικεφαλίδες, που ακολουθούν τη γραμμή αίτησης, παρέχουν μεταδεδομένα σχετικά με το μήνυμα, τον αποστολέα και τις επιθυμητές παραμέτρους επικοινωνίας[13]. Κάθε επικεφαλίδα αποτελείται από ένα ζευγάρι όνοματος και τιμής, συμμορφούμενο με το προτυποποιημένο σύνολο επικεφαλίδων του πρωτοκόλλου HTTP [13]. Αυτές οι επικεφαλίδες μεταφέρουν κρίσιμες πληροφορίες για την επεξεργασία του server, επηρεάζοντας τις απαντήσεις βάσει των προτιμήσεων ή των περιορισμών του client [13]. Για παράδειγμα, επικεφαλίδες όπως "Accept-Language" καθορίζουν τις γλωσσικές προτιμήσεις, ενώ η "If-Modified-Since" διευκολύνει συνθήκες αιτήσεις, υποδεικνύοντας την τελευταία ημερομηνία τροποποίησης[13].

Μια κενή γραμμή, που σηματοδοτείται από ένα CRLF, διαχωρίζει τις επικεφαλίδες από το προαιρετικό σώμα μηνύματος[13]. Το σώμα, επίσης αναφερόμενο ως σώμα οντότητας, περιέχει το πραγματικό περιεχόμενο του αιτήματος. Ενώ κάποιες μέθοδοι, όπως η POST, απαιτούν ένα σώμα μηνύματος για τη μετάδοση δεδομένων στον εξυπηρετητή, άλλες, όπως η GET, συνήθως δεν περιλαμβάνουν σώμα[13].

Ουσιαστικά, μια HTTP αίτηση περιλαμβάνει τη μέθοδο για μια λειτουργία (π.χ. GET, POST), τη διαδρομή προς τον πόρο, την έκδοση του HTTP, προαιρετικές επικεφαλίδες που μεταφέρουν πρόσθετες πληροφορίες και, αν είναι εφαρμοστέο, ένα σώμα μηνύματος[13]. Αυτή η δομημένη μορφή διευκολύνει την αποτελεσματική και τυποποιημένη επικοινωνία μεταξύ client και server, επιτρέποντας την ανάκτηση και τον χειρισμό πόρων σε ένα περιβάλλον βασισμένο στο web[13].

2.1.5 Τι είναι το Postman

Το Postman είναι μια πλατφόρμα API (API platform) που απλοποιεί κάθε βήμα του κύκλου ζωής των API, επιταχύνοντας τη συνεργασία για τη δημιουργία καλύτερων API με μεγαλύτερη ταχύτητα[14]. Είναι ένα σύστημα με ενσωματωμένα εργαλεία και διαδικασίες για τη δημιουργία, διαχείριση, δημοσίευση και χρήση του API[14]. Το Postman επιτρέπει σε ομάδες να διαχειρίζονται ολόκληρο τον κύκλο ζωής του API, από τον σχεδιασμό έως την παραγωγή, και διευκολύνει την άμεση επικοινωνία με τους χρήστες του API[14].

2.2 Αρχική υλοποίηση

Η υπηρεσία αποτελεί μια εξέλιξη δύο αλγορίθμων που σχεδιάστηκαν για την αντιμετώπιση τεχνικού χρέους. Παρακάτω θα δώσουμε περιληπτικές εξηγήσεις για τον κάθε έναν από αυτούς τους αλγορίθμους:

2.2.1 GEA

Ο αλγόριθμος DeRecGEA σχεδιάστηκε για την εφαρμογή της μετακίνησης κλάσης (Move Class)[11]. Ομαδοποιεί τις κλάσεις σε modular components με αναδρομικό τρόπο, δημιουργώντας ιεραρχική δομή[11]. Ο αλγόριθμος αξιολογεί με βάση μετρικές cohesion και coupling, με στόχο την αύξηση της ενότητας[11].
Κάθε άτομο αντιπροσωπεύει μια διάταξη class-component, με την ποιότητα να υπολογίζεται χρησιμοποιώντας Modularity equation[11]. Ο αλγόριθμος χρησιμοποιεί αρχές γενετικής εξέλιξης (Ge), περιλαμβάνοντας διεργασίες population initialization, selection, crossover και mutation[11]. Επιβάλλει περιορισμούς στον αριθμό των component και των κλάσεων, λαμβάνοντας υπόψη της καλές πρακτικές στη μηχανική λογισμικού[11]. Η διαδικασία crossover ενώνει γονείς-άτομα για τη δημιουργία νέων, με τα νέα άτομα να κληρονομούν χαρακτηριστικά[11]. Το Mutation περιλαμβάνει την τυχαία μεταβαλλόμενες αναθέσεις component για κλάσεις, με συμπληρωματικές συναρτήσεις που παρουσιάζουν τοπικά βέλτιστα[11]. Ο αλγόριθμος εκτελείται αναδρομικά για ολόκληρο το έργο, με τις συνθήκες τερματισμού να βασίζονται στη βελτίωση της ποιότητας και της αναδρομής[11]. Συνολικά, ο DeRecGEA αλγόριθμος βελτιστοποιεί τη διάταξη των κλάσεων σε συνιστώσες για την ενίσχυση του modularity του λογισμικού.

2.2.2 SEMI

Ο αλγόριθμος SEMI (Single Responsibility Principle-Based Extract Method Identification) λειτουργεί με το να αναγνωρίζει δυνητικές ευκαιρίες για την εφαρμογή της μετατροπής Extract Method με βάση την Αρχή της Ενιαίας Ευθύνης (Single Responsibility Principle ή SRP)[12]. Ξεκινά αναλύοντας μια δεδομένη μέθοδο και στοχεύει στο να εντοπίσει συνεκτικά σύνολα διαδοχικών εντολών[12]. Η συνεκτίκοτητα καθορίζεται μέσω δύο κύριων κριτηρίων: εντολές που έχουν πρόσβαση στην ίδια μεταβλητή και εντολές που καλούν μια μέθοδο για τον ίδιο ή διαφορετικό αντικείμενο[12].

Ο αλγόριθμος δημιουργεί έναν πίνακα για την οπτικοποίηση μεταβλητών και κλήσεων μεθόδων ανά εντολή, βοηθώντας στον εντοπισμό συνεκτικών συνόλων[12]. Αφού εντοπιστούν οι υποψήφιες ευκαιρίες Extract Method, ο αλγόριθμος τις ομαδοποιεί και τις κατατάσσει[12]. Η ομαδοποίηση περιλαμβάνει την κατηγοριοποίηση ευκαιριών με heavy overlap και παρόμοιο μέγεθος[12]. Η κατάταξη λαμβάνει υπόψιν παράγοντες όπως οι cohesion μετρικές (LCOM2) και το μέγεθος του εξαγόμενου κώδικα, υπολογίζοντας το όφελος από την εφαρμογή της μετατροπής[12].

Παράμετροι όπως το max_size_difference και το min_overlap καθοδηγούν τη διαδικασία ομαδοποίησης. Το τελικό αποτέλεσμα παρουσιάζει ομαδοποιημένες ευκαιρίες Extract Method στον χρήστη, με κύριες προτάσεις και εναλλακτικές εντός κάθε ομάδας[12].

Η ευελιξία του αλγορίθμου έγκειται σε παραμέτρους που καθορίζονται από τον χρήστη, επιτρέποντας την προσαρμογή βάσει κριτηρίων όπως η διαφορά μεγέθους και το overlap[12]. Η κύρια cohesion μετρική εξασφαλίζει ότι η μετατροπή συνεισφέρει θετικά στην ποιότητα του κώδικα[12].

Κεφάλαιο 3ο: Προτεινόμενη λύση

3.1 Σχεδιαστικές αποφάσεις

Η νέα υπηρεσία Spring Boot επιτρέπει στους χρήστες να έχουν δυνατότητα να χρησιμοποιήσουν τους αλγορίθμους Semi και GEA μέσω του ιστού. Αυτό απλοποιεί τη διαδικασία χρήσης τους, καθώς οι χρήστες χρειάζεται μόνο να υποβάλλουν http requests για να σαρώσουν τα repository τους και να λάβουν τα αποτελέσματα τους.
Αυτό επιτυγχάνεται με την ενσωμάτωση του Semi codebase. Έπειτα από βελτιώσεις στη συντηρησιμότητά του, εισάγοντας μια υλοποίηση επικοινωνίας βάσης δεδομένων με την χρήση του Hibernate αντικαθιστώντας την προηγούμενη μέθοδο χρήσης της.
Όταν γίνεται ένα αίτημα στην υπηρεσία, ενεργοποιείται η εκτέλεση της σάρωσης του αλγορίθμου Semi, εκτελώντας τον κώδικα του και χρησιμοποιώντας τα διαπιστευτήρια σύνδεσης που έχουμε αποθηκευμένα στην υπηρεσία μας.
Για τον αλγόριθμο GEA, χρησιμοποιούμε ένα προεπιλεγμένο αρχείο JAR για να πραγματοποιήσουμε τη σάρωση. Το JAR είναι η compiled version του αλγοριθμου το οποίο μπορείι να χρησιμοποιηθεί στο command line μας, δινοντας του τα απαραίτητα διαπιστευτήρια για τη σύνδεση με τη βάση δεδομένων, πληροφορίες του repository, τη γλώσσα προγραμματισμού και άλλα το τρέχουμε και αποθηκεύει τα αποτελέσματα στην βάση δεδομένων. Για την εφαρμογή των αλγορίθμων, κλωνοποιούμε το αποθετήριο τοπικά.
Ένα σημαντικό σημείο που πρέπει να τονιστεί είναι ότι αυτή η υλοποίηση εκθέτει και τους δύο αλγορίθμους στο διαδίκτυο, επιτρέποντας στους χρήστες να μπορούν να τους προσεγγίζουν εύκολα μέσω αιτημάτων HTTP.
Αυτή η μετάβαση αντιπροσωπεύει μια σημαντική βελτίωση στην προσβασιμότητα και τη χρηστικότητα, καθώς οι χρήστες μπορούν τώρα να αλληλεπιδρούν απροβλημάτιστα και αποτελεσματικά με αυτούς τους αλγορίθμους.

3.2 Παρουσίαση κώδικα

3.2.1 Εργαλεία / Utilities Scope

Μέθοδος runCmd

(src/main/java/gr/tsitoumis/geasemi/utils/Commands.java)

Η συνάρτηση runCmd πρόκειται για μια function η οποία εκτελεί μια εξωτερική εντολή στο σύστημα και επιστρέφει τον κωδικό εξόδου της. Είναι μια στατική μέθοδος που δέχεται δύο παραμέτρους:

1. cmd: Αυτή είναι η εντολή που θέλουμε να εκτελέσουμε. Είναι η εντολή που θα δώσουμε στο σύστημα για εκτέλεση.
2. directory: Αυτή είναι η διαδρομή (φάκελος) στον οποίο θέλουμε να εκτελεστεί η εντολή.

Η συνάρτηση χρησιμοποιεί τον ProcessBuilder για να δημιουργήσει μια νέα διεργασία που θα εκτελέσει την εντολή. Ανάλογα με το λειτουργικό σύστημα που χρησιμοποιείται, η συνάρτηση διαμορφώνει την εντολή χρησιμοποιώντας τον ProcessBuilder. Αν το λειτουργικό σύστημα είναι Windows, χρησιμοποιεί το "cmd.exe /c" ως το πρόγραμμα που θα εκτελέσει την εντολή. Διαφορετικά, χρησιμοποιεί το "sh -c" για άλλα λειτουργικά συστήματα.

Έπειτα, η συνάρτηση δημιουργεί μια νέα διεργασία (Process) μέσω του ProcessBuilder και την στέλνει να εκτελέσει την εντολή που έχει καθοριστεί στο προηγούμενο βήμα.
Για να διαχειριστεί την έξοδο της εντολής, η συνάρτηση χρησιμοποιεί έναν StreamGobbler. Η κλάση StreamGobbler υλοποιεί την Runnable interface, δημιουργεί ένα BufferedReader για να διαβάσει την έξοδο της εντολής από την είσοδο της διεργασίας και να την προωθήσει (εκτυπώσει) σε έναν Consumer. Στην προκειμένη περίπτωση, η έξοδος εκτυπώνεται στο System.out.
Τέλος, η διεργασία περιμένει να ολοκληρωθεί με τη χρήση της process.waitFor(), και ο κωδικός εξόδου της εντολής αποθηκεύεται στη μεταβλητή exitCode, η οποία επιστρέφεται από τη συνάρτηση.

Συνοψίζοντας, η συνάρτηση runCmd χρησιμοποιείται για να εκτελέσει εντολές στο σύστημα μέσω της Java και να λάβει τον κωδικό εξόδου της εντολής. Η συγκεκριμένη function θα είναι ένα εργαλείο για τις παρακάτω εργασίες διότι θα χρειαστε'ι να τρέξουμε εντολές όπως είναι το Git και τα jar files

Μέθοδος gitClone

(src/main/java/gr/tsitoumis/geasemi/utils/Commands.java)

Η μέθοδος, gitClone χρησιμοποιείται για να κάνει clone ένα project repository με την βοήθεια του Git.
Πρόκειται για μια στατική μέθοδο που δέχεται την παράμετρο project η οποία είναι τύπου string.
Πρώτα καλεί την runCmd (που παρουσιάστηκε προηγουμένως) για να εκτελέσει την εντολή git clone [project] στον directory git-repositories όπου για το project αυτό είναι ο κατάλογος που αποθηκεύει όλα τα projects προσωρινά μέχρι να γινει η ανάλυσή τους.
Περνάει στην μέθοδο την παράμετρο project το οποίο αντιπροσωπεύει το project που θέλουμε να κανουμε clone.
Έπειτα ελέγχει τον κωδικό εξόδου από την runCmd. Αν ο κωδικός εξόδου είναι 128, σημαίνει ότι το project έχει ήδη γίνει cloned. Σε αυτήν την περίπτωση, ένα GeaSemiException εκτίνεται με το μήνυμα "Project already cloned" και κωδικό κατάστασης HTTP 500 (Internal Server Error).
Αν ο κωδικός εξόδου δεν είναι 0, τότε η εντολή git clone απέτυχε. Σε αυτήν την περίπτωση, ένα GeaSemiException εκτίνεται με το μήνυμα "Git clone FAILED" και κωδικό κατάστασης HTTP 500 (Internal Server Error).

Μέθοδος semiRun

(src/main/java/gr/tsitoumis/geasemi/utils/Commands.java)

Η μέθοδος semiRun εκτελεί ένα σημαντικό κομμάτι λογικής στο πλαίσιο της εφαρμογής. Χρησιμοποιείται για την εκτέλεση του εργαλείου "semi" jar το οποίο αναλύει το project repository που εχουμε κανει clone. Η μέθοδος δέχεται τρία σημαντικά ορίσματα:

- name: Το όνομα του project που θα αναλυθεί.
- lang: Η γλώσσα προγραμματισμού που χρησιμοποιείται για την ανάλυση.
- version: Η έκδοση του project.

Η μέθοδος ξεκινά δημιουργώντας τρεις διαδρομές (paths):

- projectPath: Αναπαριστά τον φάκελο του project που θα αναλυθεί.
- semiPath: Αναπαριστά το JAR αρχείο του εργαλείου "semi" με τις εξαρτήσεις του.
- semiExtractPosition: Αναπαριστά τον φάκελο όπου θα εκτελεστεί το JAR αρχείο "semi".

Στη συνέχεια, δημιουργεί μια εντολή εκτέλεσης που περιλαμβάνει την εκτέλεση του "semi" JAR αρχείου. Αυτή η εντολή περιλαμβάνει τα ορίσματα που περνάνε το όνομα της γλώσσας, το όνομα του

project, την έκδοση, διαδρή του project και τα δεδομενα για συνδεθει το jar file στην βαση δεδομενων.
Στη συνέχεια, χρησιμοποιεί την runCmd για να εκτελέσει την εντολή που δημιούργησε προηγουμένως. Η εκτέλεση γίνεται στον φάκελο semiExtractPosition.
Τέλος, ελέγχει τον κωδικό εξόδου της εκτέλεσης. Αν ο κωδικός εξόδου δεν είναι μηδέν, τότε καταλαβαίνουμε ότι κάτι δεν πήγε καλά κατά τη διάρκεια της εκτέλεσης του "semi." Σε αυτήν την περίπτωση, δημιουργείται μια εξαίρεση τύπου "GeaSemiException" με συγκεκριμένο μήνυμα για να δείξουμε ότι κάτι πήγε λάθος και κωδικό κατάστασης HTTP 500 (σφάλμα εσωτερικού διακομιστή).

Μέθοδος geaRun

(src/main/java/gr/tsitoumis/geasemi/utils/Commands.java)

Η συνάρτηση geaRun εχει κοινό τρόπο αντιμετώπισης με την semiRun. Προκειται για μια στατικη μεθοδο η οποια δεχεται δυο παραμέτρους:

- name: Το όνομα του project που θα αναλυθεί.
- lang: Η γλώσσα προγραμματισμού που χρησιμοποιείται για την ανάλυση.

και σκοπος της ειναι να τρεξει το gea jar εργαλειο σε ενα project repository που εχουμε κανει clone.
Η μέθοδος ξεκινά δημιουργώντας τρεις διαδρομές (paths):

- projectPath: Αναπαριστά τον φάκελο του project που θα αναλυθεί.
- semiPath: Αναπαριστά το JAR αρχείο του εργαλείου "semi" με τις εξαρτήσεις του.
- semiExtractPosition: Αναπαριστά τον φάκελο όπου θα εκτελεστεί το JAR αρχείο "semi".

Επειτα δημιουργουμε την string εντολη που θελουμε να εκτελεσουμε και την τρεχουμε με την βοηθεια της runCmd η οποία παρουσιάστηκε προηγουμένως.
Αν το exit code δεν είναι ίσο με το μηδέν τότε όπως και στο semiRun δημιουργούμε ένα exception με κατάλληλο μήνυμα και το κάνουμε throw

Μέθοδος deleteProject

(src/main/java/gr/tsitoumis/geasemi/utils/Commands.java)

Η μέθοδος deleteProject είναι μία μέθοδος που θα χρησιμοποιηθεί στην εφαρμογή μας για να διαγράφουμε τα cloned projects τα οποία έχουν περάσει από την ανάλυση που θέλουμε και πλέον μας είναι μη αναγκαία.
Αρχικά είναι μια στατική μέθοδος που χρειάζεται ένα μόνο όρισμα, το projectName, το οποίο είναι το όνομα του έργου που πρέπει να διαγραφεί.
Ξεκινά ελέγχοντας το λειτουργικό σύστημα του συστήματος όπου εκτελείται ο κώδικας. Χρησιμοποιεί το System.getProperty ("os.name") για να ανακτήσει το όνομα του λειτουργικού συστήματος και στη συνέχεια το μετατρέπει σε πεζά. Εάν το όνομα του λειτουργικού συστήματος ξεκινά με "windows", προϋποθέτει ότι το σύστημα που το εκτελεί είναι Windows.
Ανάλογα με το λειτουργικό σύστημα, αν είναι Windows, χρησιμοποιεί τη μέθοδο runCmd για να εκτελέσει την εντολή rd /s /q <projectName> στον κατάλογο "./git-repositories". Η εντολή rd χρησιμοποιείται στα Windows για την αφαίρεση καταλόγων. Η σημαία /s το κάνει να διαγράφει τον κατάλογο και όλους τους υποκαταλόγους του, ενώ η σημαία /q το κάνει αθόρυβα χωρίς να ζητά επιβεβαίωση.
Εάν δεν είναι Windows (προφανώς κάποιο σύστημα τύπου Unix), χρησιμοποιεί τη μέθοδο runCmd για να εκτελέσει την εντολή rm -rf <projectName> στον κατάλογο "./git-repositories". Η εντολή rm χρησιμοποιείται σε συστήματα τύπου Unix για την αφαίρεση αρχείων και καταλόγων. Οι σημαίες -rf επιβάλλουν την αφαίρεση του καταλόγου και των περιεχομένων του χωρίς να ζητηθεί επιβεβαίωση.

Η μέθοδος δεν επιστρέφει καμία τιμή (void) αλλά δημιουργεί μια Εξαίρεση εάν παρουσιαστούν σφάλματα κατά την εκτέλεση της λειτουργίας διαγραφής.

GeaSemiException

(src/main/java/gr/tsitoumis/geasemi/utils/GeaSemiException.java)

Αυτή η κλάση έχει σχεδιαστεί για να αντιπροσωπεύει εξαιρέσεις ειδικά για την εφαρμογή GeaSemi.
Η κλάση επεκτείνει την ενσωματωμένη κλάση Exception, καθιστώντας την μια επιλεγμένη εξαίρεση. Έχει ένα ιδιωτικό πεδίο που ονομάζεται httpStatus τύπου HttpStatus. Το HttpStatus είναι μια κλάση Enum που χρησιμοποιείται για την αναπαράσταση τυπικών κωδικών κατάστασης HTTP (π.χ. 200 για OK, 404 για Not Found και 500 για Internal Server Error).
Η κλάση παρέχει έναν constructor που παίρνει δύο παραμέτρους:

- μήνυμα (String): Αυτή η παράμετρος αντιπροσωπεύει ένα περιγραφικό μήνυμα για την εξαίρεση, που συνήθως περιέχει ένα μήνυμα σφάλματος ή σχετικές πληροφορίες.
- httpStatus (HttpStatus): Αυτή η παράμετρος συσχετίζει την εξαίρεση με έναν συγκεκριμένο κωδικό κατάστασης HTTP.

Μέσα στον constructor, χρησιμοποιεί την κλήση super(message) για να ορίσει το μήνυμα σφάλματος για την εξαίρεση. Αρχικοποιεί επίσης το πεδίο httpStatus με την παρεχόμενη παράμετρο httpStatus.
Η κλάση περιλαμβάνει μεθόδους getter και setter για το πεδίο httpStatus, επιτρέποντας στον εξωτερικό κώδικα να έχει πρόσβαση και να τροποποιεί την κατάσταση HTTP που σχετίζεται με την εξαίρεση.
Συμπερασματικά, η κλάση GeaSemiException είναι μια εξαίρεση προσαρμοσμένη για την εφαρμογή GeaSemi. Περιλαμβάνει πληροφορίες μηνύματος σφάλματος και συνδέει κάθε εξαίρεση με έναν σχετικό κωδικό κατάστασης HTTP. Αυτός ο σχεδιασμός είναι ιδιαίτερα πολύτιμος κατά τον χειρισμό εξαιρέσεων και τη δημιουργία κατάλληλων αποκρίσεων HTTP, ειδικά σε ένα πλαίσιο εφαρμογών που βασίζεται στον ιστό.

GitTools - getProjectName

(src/main/java/gr/tsitoumis/geasemi/utils/GitTools.java)

Ο σκοπός αυτής της μεθόδου είναι να παρέχει μια βοηθητική μέθοδο για την εξαγωγή του ονόματος ενος git repository project από μια διεύθυνση URL.
Η μέθοδος getProjectName ειναι μια στατική μέθοδος, χρησιμοποιείται για την εξαγωγή του ονόματος του έργου από μια διεύθυνση URL και την επιστροφή του ως string. Η μέθοδος παίρνει μία παράμετρο url (String) οπου αυτή είναι η διεύθυνση URL από την οποία θα εξαχθεί το όνομα του έργου. Επειτα αφου λάβει το url, το χωρίζει σε μια σειρά από υποσυμβολοσειρές χρησιμοποιώντας την κάθετο προς τα εμπρός / ως οριοθέτη. Αυτό διαχωρίζει αποτελεσματικά τη διεύθυνση URL στα στοιχεία της.
Στη συνέχεια εξάγει το τελευταίο στοιχείο από τον πίνακα (το τμήμα της διεύθυνσης URL που αντιπροσωπεύει το όνομα του έργου) με πρόσβαση στο splittedUrl[splittedUrl.length - 1].
Η μέθοδος επιστρέφει το εξαγόμενο όνομα του έργου ως συμβολοσειρά.
Η συγκεκριμένη μέθοδος θα μας βοηθήσει να βρούμε το όνομα ενος git repository project στα endpoints.

MessageResponseBody

(src/main/java/gr/tsitoumis/geasemi/utils/MessageResponseBody.java)

Η κλάση MessageResponseBody χρησιμεύει ως μια απλή δομή δεδομένων για τον περιορισμό και τη διαχείριση ενός μεμονωμένου μηνύματος. Χρησιμοποιείται κυρίως για επιστροφή πληροφοριών στους χρήστες.

Pagination

(src/main/java/gr/tsitoumis/geasemi/utils/Pagination.java)

Η κλάση Pagination χρησιμεύει ως αναπαράσταση των ρυθμίσεων σελιδοποίησης, διασφαλίζοντας ότι ο αριθμός σελίδας και το μέγεθος σελίδας βρίσκονται εντός έγκυρων ορίων. Αυτό γενικά είναι χρήσιμο για τη διαχείριση και τη διαμόρφωση της σελιδοποίησης σε εφαρμογές Ιστού, ενώ παράλληλα επιβάλλονται ορισμένοι περιορισμοί στις παραμέτρους που δίνονται.

PaginationResponseBody

(src/main/java/gr/tsitoumis/geasemi/utils/PaginationResponseBody.java)

Η κλάση PaginationResponseBody χρησιμεύει ως δομή για τη μεταφορά πληροφοριών σελιδοποίησης στο response ενός paginated query. Περιλαμβάνει δεδομένα σχετικά με την τρέχουσα σελίδα, την τελευταία σελίδα και το μέγεθος των δεδομένων ανά σελίδα, καθιστώντας το χρήσιμο για την ενημέρωση των χρηστών σχετικά με τις λεπτομέρειες σελιδοποίησης ενός συνόλου αποτελεσμάτων.

Slf4jMDCFilter

(src/main/java/gr/tsitoumis/geasemi/utils/filters/Slf4jMDCFilter.java)

Η κλάση Slf4jMDCFilter ειναι ενα Filter το οποίο προορίζεται να χρησιμοποιηθεί για να προσθέτει requestIDs στα responses της εφαρμογής.
Η κλάση Slf4jMDCFilter ορίζεται ως ένα στοιχείο @Component και επεκτείνει το OncePerRequestFilter, το οποίο διασφαλίζει ότι το φίλτρο εκτελείται μόνο μία φορά ανά αίτημα.
Η μέθοδος doFilterInternal είναι η βασική λογική του φίλτρου και εκτελείται για κάθε αίτημα HTTP. Εξάγει ένα id από το αίτημα (είτε από μια καθορισμένη κεφαλίδα αιτήματος είτε δημιουργεί ένα τυχαίο).
Εξάγει τη διεύθυνση IP του πελάτη από το αίτημα (λαμβάνοντας υπόψη την κεφαλίδα X-Forwarded-For εάν υπάρχει).
Τοποθετεί το εξαγόμενο id και την IP πελάτη στο MDC για καταγραφή.
Προσθέτει το διακριτικό στην κεφαλίδα της απόκρισης (εάν έχει καθοριστεί το answerHeader). Καλεί το επόμενο φίλτρο ή το servlet στην αλυσίδα φίλτρου. Τέλος, αφαιρεί το διακριτικό και την IP πελάτη από το MDC. Οι μέθοδοι isAsyncDispatch και shouldNotFilterErrorDispatch παρακάμπτονται για να καθοριστεί ότι το φίλτρο δεν χειρίζεται ασύγχρονα αιτήματα και θα πρέπει να φιλτράρει τις αποστολές σφαλμάτων. Συνοπτικά, το Slf4jMDCFilter είναι ένα φίλτρο Servlet που εμπλουτίζει τις εγγραφές καταγραφής με πληροφορίες συμφραζομένων, όπως ένα μοναδικό id και τη διεύθυνση IP του πελάτη, καθιστώντας το χρήσιμο για τον εντοπισμό σφαλμάτων και την παρακολούθηση αιτημάτων σε μια εφαρμογή Ιστού. Αυτό το φίλτρο ενσωματώνεται με την καταγραφή SLF4J και μπορεί να διαμορφωθεί με διαφορετικές επιλογές με βάση τις παρεχόμενες μεταβλητές παρουσίας.

Slf4jMDCFilterConfiguration

(src/main/java/gr/tsitoumis/geasemi/utils/filters/Slf4jMDCFilterConfiguration.java)

Η Slf4jMDCFilterConfiguration κλάση είναι υπεύθυνη για τη διαμόρφωση και τη δημιουργία μιας παρουσίας του φίλτρου Slf4jMDCFilter και την καταχώρισή του με το Spring Boot.
Η κλάση Slf4jMDCFilterConfiguration ορίζεται ως κλάση διαμόρφωσης Spring με τον σχολιασμό @Configuration.
Ο σχολιασμός @Bean χρησιμοποιείται για να ορίσει μια μέθοδο που ονομάζεται servletRegistrationBean που επιστρέφει ένα FilterRegistrationBean. Αυτή η μέθοδος διαμορφώνει και δημιουργεί ένα στιγμιότυπο του φίλτρου Slf4jMDCFilter και το καταχωρεί με το Spring Boot.
Δημιουργείται ένα FilterRegistrationBean για να κρατήσει τη διαμόρφωση του φίλτρου.
Δημιουργείται μια παρουσία του φίλτρου Slf4jMDCFilter, περνώντας τα διαμορφωμένα answerHeader, mdcTokenKey, mdcClientIpKey και requestHeader.
Το FilterRegistrationBean επιστρέφεται για το Spring για να διαμορφώσει και να καταχωρήσει το φίλτρο.
Συνοπτικά, η κλάση Slf4jMDCFilterConfiguration είναι υπεύθυνη για τη διαμόρφωση του Slf4jMDCFilter και την καταχώρισή του με το Spring Boot. Επιτρέπει την προσαρμογή διαφόρων παραμέτρων, όπως οι κεφαλίδες απόκρισης και τα κλειδιά MDC, για τον εμπλουτισμό του logging. Το φίλτρο καταχωρείται με μια καθορισμένη σειρά για τον προσδιορισμό της ακολουθίας εκτέλεσης στην αλυσίδα φίλτρου.

3.2.2 Πεδίο SEMI

Entities

Opportunities

(src/main/java/gr/tsitoumis/geasemi/semi/entities/Opportunities.java)

Η οντότητα Opportunities είναι ένα component εφαρμογής Java Persistence, που χρησιμεύει ως γέφυρα μεταξύ της εφαρμογής και της βάσης δεδομένων. Τα χαρακτηριστικά, οι σχολιασμοί και οι μέθοδοι του έχουν σχεδιαστεί σχολαστικά για να διευκολύνουν τη διαχείριση και την ανάκτηση δεδομένων.
Επισημαίνεται με τον σχολιασμό "@Entity", υποδηλώνοντας το ρόλο του ως κλάση οντοτήτων JPA (Java Persistence API). Αντιστοιχίζεται σε έναν πίνακα βάσης δεδομένων που ονομάζεται "OPPORTUNITIES" προσδιορίζεται χρησιμοποιώντας τον σχολιασμό "@Table" με το χαρακτηριστικό name ορισμένο σε "OPPORTUNITIES".
Το χαρακτηριστικό "id" χρησιμεύει ως πρωτεύον κλειδί και προσδιορίζει μοναδικά κάθε Opportunity object. Σημειώνεται με "@Id" και "@GeneratedValue" για αυτόματη δημιουργία τιμών
Η οντότητα Opportunities περιέχει πολλά attributes, όπως "projectName", "projectVersion", "classPath", "className", "methodName", "lineStart", "lineEnd", "cohesionBenefit", "methodOriginalCohesion" και "linesOfCode". Αυτά τα χαρακτηριστικά περιέχουν σημαντικές πληροφορίες που σχετίζονται με την αναλυση του έργου από το εργαλείο semi.
Τα attributes "projectName", "classPath", "className" και "methodName" είναι τύπου String και χρησιμοποιούν τον σχολιασμό "@Lob" για την αποθήκευση μεγάλων τιμών κειμένου. Τα χαρακτηριστικά "projectVersion", "lineStart", "lineEnd", "cohesionBenefit", "methodOriginalCohesion" και "linesOfCode" περιέχουν αριθμητικά δεδομένα.

Η οντότητα κανει override την μεθοδο hashCode και την μεθοδο equals, οι οποίες είναι απαραίτητες για τον εντοπισμό και τη σύγκριση περιπτώσεων εντός collections. Η υλοποίηση βασίζεται στο χαρακτηριστικό "id" για την ισότητα αντικειμένων. Με λίγα λόγια αυτή είναι η αντιστοίχιση των εισαγωγών πίνακα μιας βάσης δεδομένων με αντικείμενα στην εφαρμογή μας. Το συγκεκριμένο entity θα μας χρησιμεύσει όταν θα ανακτήσουμε πληροφορίες από την βάση.

SemiOpportunitiesResponseBody

(src/main/java/gr/tsitoumis/geasemi/semi/entities/SemiOpportunitiesResponseBody.java)

Αυτή η κλάση έχει σχεδιαστεί για να ενσωματώνει μια λίστα Opportunities και PaginationResponseBody, σχηματίζοντας μια δομημένη απόκριση για το χειρισμό και τη μεταφορά δεδομένων από το backend στο frontend.

SemiRepository

(src/main/java/gr/tsitoumis/geasemi/semi/SemiRepository.java)

Πρόκειται για ενα interface το οποίο μας δίνει την δυνατότητα να επικοινωνούμε με την βάση δεδομένων. Κανει extend το PagingAndSortingRepository οπου είναι μια διεπαφή Spring Data JPA που παρέχει μεθόδους για λειτουργίες CRUD (Create, Read, Update, Delete) και δυνατότητες ταξινόμησης/σελιδοποίησης. Παραμετροποιείται με το Opportunities entity ως τύπο οντότητας και Integer ως τύπο του πρωτεύοντος κλειδιού.Το repository περιεχει τρεις μεθόδους για custom ερωτήματα προς την βάση.

1. findByProjectName(String projectName, Pageable pageable): Η μέθοδος επισημαίνεται με τον σχολιασμό @Query, ο οποίος υποδεικνύει ότι πρόκειται για μια custom μέθοδο ερωτήματος βάσης δεδομένων. Ανακτά μια σελιδοποιημένη λίστα αντικειμένων Opportunities από τη βάση δεδομένων όπου το projectName ταιριάζει με την παρεχόμενη παράμετρο projectName. Η παράμετρος Pageable ελέγχει τη σελιδοποίηση. Η συγκεκριμένη μέθοδος έχει σκοπό να ανακτά όλα τα Opportunity database entries ενος project μετά από ανάλυση.
2. findByProjectNameCount(String projectName): Αυτή η μέθοδος είναι μια άλλη custom μέθοδος ερωτήματος με τον σχολιασμό @Query. Ανακτά τον αριθμό των αντικειμένων Opportunities στη βάση δεδομένων για ένα project. Επιστρέφει τον αριθμό ως ακέραιο.
3. deleteByProjectNameAndProjectVersion(String projectName, int projectVersion) Αυτή η μέθοδος ορίζει μια λειτουργία διαγραφής για αντικείμενα Opportunities. Διαγράφει εγγραφές με βάση τις τιμές του projectName και του projectVersion που παρέχονται ως παραμέτρους. Η μέθοδος επιστρέφει τον αριθμό των γραμμών που διαγράφηκαν ως long.

SemiService

(src/main/java/gr/tsitoumis/geasemi/semi/SemiService.java)

Το service είναι ένα από τα βασικά αρχιτεκτονικά στοιχεία στην ανάπτυξη λογισμικού. Ο σκοπός του SemiService ειναι να χρησιμοποιεί το repository για να ανακτήσει δεδομένα και να τα επεξεργαστεί βάση του business logic της εφαρμογής.
Ο σχολιασμός @Service υποδεικνύει ότι η κλάση είναι ένα bean υπηρεσίας Spring. Χρησιμοποιείται για το χειρισμό της business logic και μπορεί να ανακαλυφθεί αυτόματα από το component scanning. Η class εισάγει ένα SemiRepository χρησιμοποιώντας τον σχολιασμό @Autowired. Αυτό το

repository παρέχει μεθόδους αλληλεπίδρασης με την βάση δεδομένων για τον πίνακα OPPORTUNITIES. Στην κλάση αυτή, έχουν υλοποιηθεί τρεις μέθοδοι:

1. **public SemiOpportunitiesResponseBody getProjectOpportunities(String projectName, int page, int pageSize)**: Σκοπός αυτής της μεθόδου είναι η ανάκτηση γραμμών του πίνακα Opportunities με βάση το όνομα του project που έγινε η ανάλυση με το semi εργαλείο. Αρχικά η μέθοδος λαμβάνει τρεις παραμέτρους: "projectName", "page" και "pageSize". Η παράμετρος projectName αντιπροσωπεύει το όνομα του έργου για το οποίο πρόκειται να ανακτηθούν ευκαιρίες. Τα "page" και "pageSize" χρησιμοποιούνται για την υλοποίηση σελιδοποίησης, επιτρέποντας την ανάκτηση δεδομένων σε διαχειρίσιμα κομμάτια. Ένα από τα πρώτα πράγματα που κάνει αυτή η μέθοδος είναι η ρύθμιση της σελιδοποίησης. Δημιουργεί ένα αντικείμενο Pagination, το οποίο οπως εχει αναφερθει ο σκοπος του ειναι οι ελεγχοι για τις παραμετρους σελιδοποιησης που δόθηκαν page και pageSize. Το επόμενο βήμα περιλαμβάνει τη διαμόρφωση ενός αντικειμένου Pageable με την βοηθεια της μεθοδου PageRequest.of(). Το Pageable είναι ένα σημαντικό μέρος του Spring Data JPA που χρησιμοποιείται για τον καθορισμό του τρόπου ανάκτησης των δεδομένων, συμπεριλαμβανομένων των λεπτομερειών σελιδοποίησης όπως ο τρέχων αριθμός σελίδας και ο αριθμός των στοιχείων ανά σελίδα. Με τη ρύθμιση της σελιδοποίησης, η μέθοδος προχωρά στην ανάκτηση των σχετικών δεδομένων. Αυτό το κάνει χρησιμοποιώντας τη μέθοδο repository.findByProjectName(projectName, pageable). Αυτή η μέθοδος αναμένεται να επιστρέψει μια Σελίδα Opportunities που σχετίζεται με το καθορισμένο έργο. Η χρήση της σελίδας διασφαλίζει ότι ανακτάται μόνο ένα υποσύνολο αποτελεσμάτων, που αντιστοιχεί στην τρέχουσα σελίδα. Για να παρέχει πληροφορίες σχετικά με την σελιδοποίηση και να βοηθήσει στην πλοήγηση ανάμεσα στις σελίδες, ο κώδικας υπολογίζει τον συνολικό αριθμό των διαθέσιμων σελίδων. Αυτό γίνεται με την επίκληση της opportunities.getTotalPages(). Έτσι η μεταβλητή allPages ορίζεται και αποθηκεύει το συνολικό αριθμό σελίδων ώστε να προσφερθεί ως πληροφορία για την σελιδοποίηση. Στη συνέχεια, δημιουργείται ένα αντικείμενο PaginationResponseBody, που ονομάζεται paginationResponseBody, για την αποθήκευση πληροφοριών που σχετίζονται με τη σελιδοποίηση. Αυτό περιλαμβάνει την τρέχουσα σελίδα, το μέγεθος σελίδας και τον συνολικό αριθμό σελίδων. Τέλος, η μέθοδος δημιουργεί ένα αντικείμενο reponse, που ονομάζεται SemiOpportunitiesResponseBody. Αυτό το αντικείμενο reponse δημιουργείται συμπεριλαμβάνοντας το περιεχόμενο των Opportunities που λαμβάνονται από το repository και το paginationResponseBody που δημιουργήθηκε νωρίτερα.
2. **public boolean saveOpportunitiesArray(ArrayList<Opportunities> array)**: Ο σκοπός αυτής της μεθόδου είναι να αποθηκεύει μια λιστα από opportunities στην βάση δεδομένων. Η μέθοδος σημειώνεται με @Transactional και καθορίζει το επίπεδο απομόνωσης READ_UNCOMMITTED. Η μέθοδος λαμβάνει ως παράμετρο εισόδου μια ArrayList απο Opportunities, η οποία συμβολίζεται ως “array”. Η βασική λογική της μεθόδου περιστρέφεται γύρω από την κλήση του repository.saveAll(array). Αυτή η μέθοδος αναμένεται να αποθηκεύσει όλα τα αντικείμενα Opportunities που περιέχονται στην μεταβλητή “array” στην βάση δεδομένων. Το αποτέλεσμα αυτής της λειτουργίας αποθηκεύεται σε μια μεταβλητή με το όνομα result. Μετά την αποθήκευση των δεδομένων, εκτελείται ένας έλεγχος για να εξασφαλιστεί η επιτυχία της λειτουργίας αποθήκευσης. Εξετάζει εάν το αποτέλεσμα είναι instance της κλάσης ArrayList. Εάν είναι, αυτό σημαίνει ότι η λειτουργία αποθήκευσης επέστρεψε μια μη κενή συλλογή αποθηκευμένων αντικειμένων. Επομένως, η μέθοδος επιστρέφει true για να υποδείξει ότι η λειτουργία αποθήκευσης ήταν επιτυχής. Αντίθετα, εάν

το αποτέλεσμα δεν είναι μια instance της ArrayList ή εάν η ArrayList είναι κενή, η μέθοδος επιστρέφει false. Αυτό σημαίνει ότι η λειτουργία αποθήκευσης απέτυχε.

3. ο κύριος σκοπός αυτής της μεθόδου είναι να διαγράψει δεδομένα από την βάση δεδομένων με βάση συγκεκριμένα κριτήρια. Η μέθοδος σημειώνεται με @Transactional το οποίο καθορίζει το επίπεδο απομόνωσης ως READ_UNCOMMITTED.
   Η μέθοδος παίρνει δύο παραμέτρους εισόδου:
   - projectName - Ενα string που αντιπροσωπεύει το όνομα του έργου για το οποίο τα δεδομένα πρέπει να διαγραφούν.
   - projectVersion - Ένας integer αριθμός που αντιπροσωπεύει την έκδοση του έργου για την οποία τα δεδομένα πρέπει να διαγραφούν.

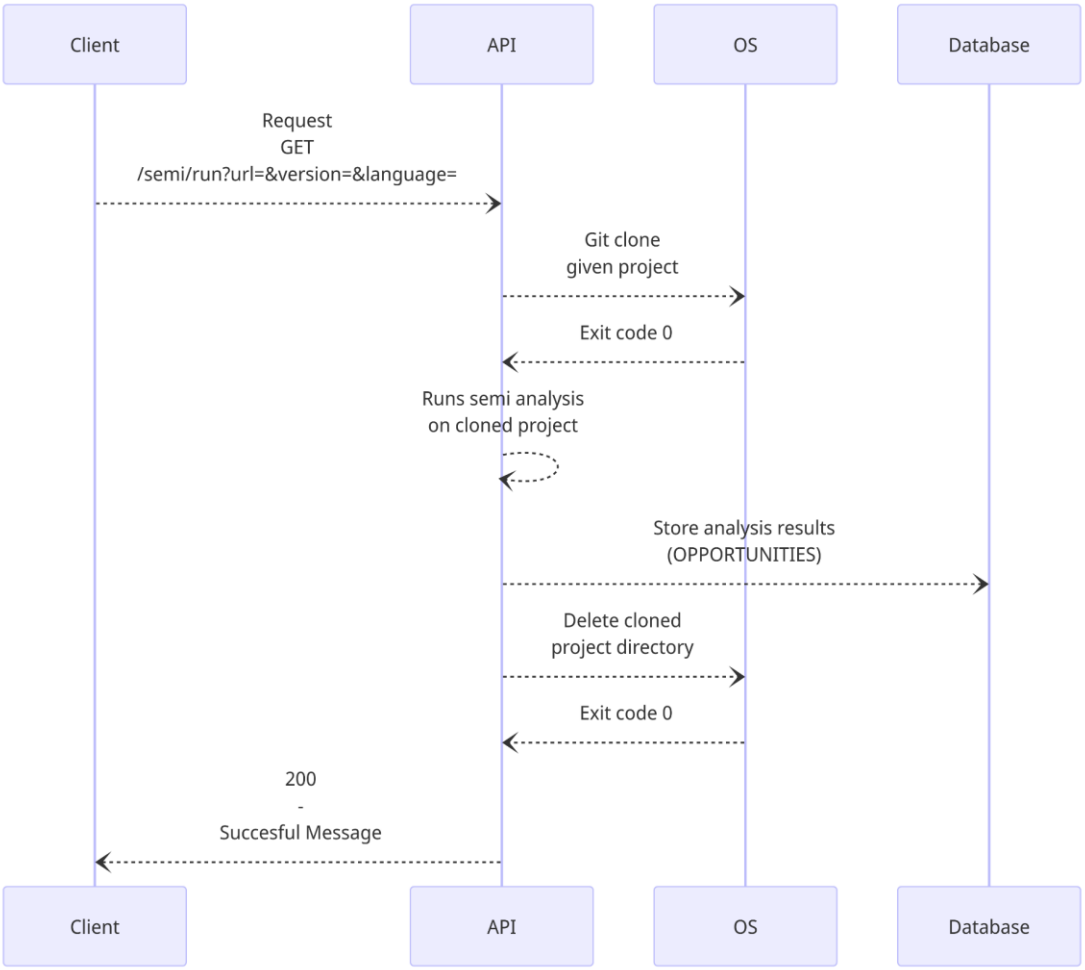
   Η βασική λογική της μεθόδου περιστρέφεται γύρω από την κλήση της μεθόδου repository.deleteByProjectNameAndProjectVersion(projectName, projectVersion). Αυτή η μέθοδος αναμένεται να διαγράψει εγγραφές από την βάση δεδομένων που ταιριάζουν με το παρεχόμενο projectName και το projectVersion. Το αποτέλεσμα αυτής της λειτουργίας είναι μια καταμέτρηση των εγγραφών που διαγράφηκαν.

SemiController

(src/main/java/gr/tsitoumis/geasemi/utils/semi/SemiController.java)\

Η κλάση SemiController είναι υπεύθυνη για το χειρισμό αιτημάτων HTTP. Σχολιάζεται με @Controller, υποδεικνύοντας ότι είναι ένας controller στο πλαίσιο του Spring MVC, που χρησιμεύει ως σημείο εισόδου για την επεξεργασία των αιτημάτων των user request.
Ο σχολιασμός @RequestMapping καθορίζει τη βασική διαδρομή για όλα τα endpoints εντός του controller. Σε αυτήν την περίπτωση, η διαδρομή βάσης ορίζεται με “/semi”, καθιστώντας όλα τα επόμενα endpoint σε σχέση με αυτήν τη διαδρομή.
Η βασική συνεργασία σε αυτή την κατηγορία είναι με το SemiService. Το SemiService είναι μια κατηγορία υπηρεσιών που είναι υπεύθυνη για τη διαχείριση της επιχειρηματικής λογικής της εφαρμογής, την επεξεργασία δεδομένων και τις αλληλεπιδράσεις με τα αποθετήρια δεδομένων. Ο SemiController βασίζεται στην SemiService για την εκτέλεση σύνθετων λειτουργιών, όπως η ανάλυση των αποθετηρίων Git και η ανάκτηση ευκαιριών έργου.
Στην κλαση υπαρχει μια μεταβλητη με το ονομα service. Ο σχολιασμός @Autowired είναι ένα fundamental aspect του μηχανισμού "dependency injection" του Spring Framework. Επιτρέπει στο Spring να εισάγει αυτόματα dependencies σε μια κλάση το Spring θα δημιουργήσει, θα διαμορφώσει και θα διαχειριστεί τον κύκλο ζωής του service, διασφαλίζοντας ότι είναι διαθέσιμο για χρήση όταν χρειάζεται.

GET - /RUN endpoint



Σχήμα 3.1: Διάγραμμα Sequence για το Endpoint get - semi/run.

Το endpoint "run" ορίζεται για να χειρίζεται HTTP GET (Χρησιμοποιώντας το σχόλιο RequestMapping) αιτήματα. Όταν ένας client στέλνει ένα αίτημα GET σε αυτό το Endpoint, αναμένει να λάβει το αποτελέσματα semi analysis για ένα συγκεκριμένο github project. Αρχικά το Endpoint δέχεται τρια query parameters:

1. url: παράμετρος συμβολοσειράς που αντιπροσωπεύει τη διεύθυνση URL του έργου Git προς ανάλυση.
2. version: παράμετρος συμβολοσειράς που αντιπροσωπεύει την έκδοση του έργου.
3. language: παράμετρος συμβολοσειράς που υποδεικνύει τη γλώσσα προγραμματισμού που χρησιμοποιείται στο έργο.

Ο Controller ξεκινάει με πρώτη ενέργεια την εξαγωγή του ονόματος έργου από την παρεχόμενη διεύθυνση URL χρησιμοποιώντας απο τα utils το GitTools.getProjectName.
Στη συνέχεια, ο κώδικας εκτελεί λειτουργία κλωνοποίησης Git. Κατεβάζει ένα αντίγραφο του Git project repository από τη διεύθυνση URL σε έναν τοπικό κατάλογο. Αυτό το βήμα επιτυγχάνεται χρησιμοποιώντας τη μέθοδο Commands.gitClone(url).
Μετά την κλωνοποίηση του repository, ο κώδικας προχωρά στην εκτέλεση του εργαλείου ανάλυσης semi, που αντιπροσωπεύεται από την κλάση Semitool. Στο εργαλείο δίνονται οι παράμετροι: - -

1. service: Το service του semiController το οποίο εχει γινει injected με το σχόλιο Autowired.
2. language: Η γλώσσα που δεχτήκαμε στα query params.
3. name: Το όνομα του project που καναμε extract από το url.

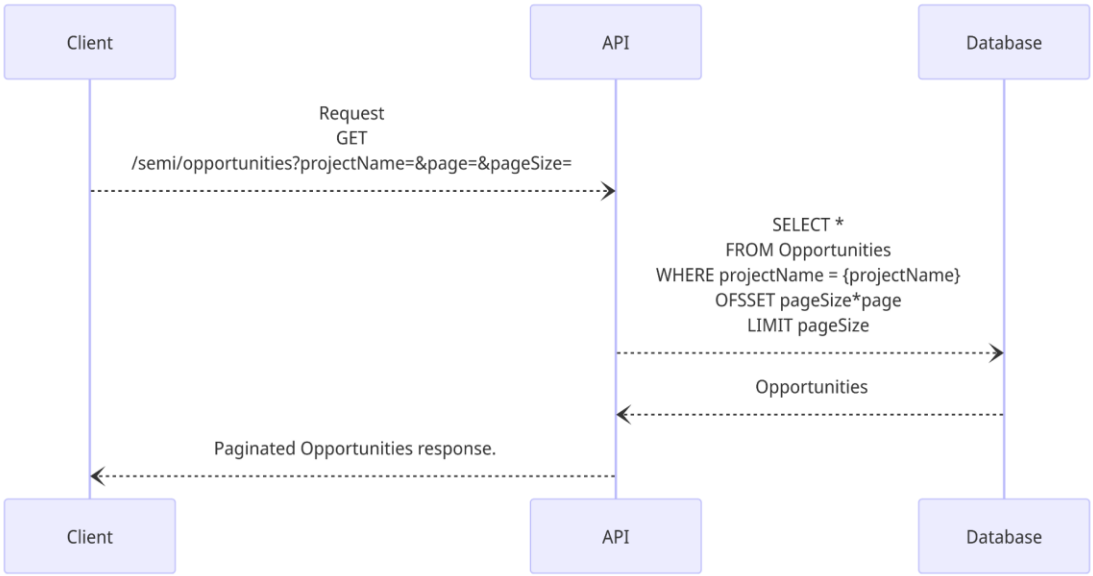
4. version: Το version που δόθηκε στα query params.
5. path: Το path που αντιστοιχεί στο φάκελο του cloned project όπου βρίσκεται στο φάκελο που αποθηκεύονται τα project

Μόλις ολοκληρωθεί η ανάλυση, ο κώδικας δημιουργεί μια απάντηση HTTP για αποστολή πίσω στον client. Η απάντηση περιλαμβάνει:

1. Ένας κωδικός κατάστασης HTTP 200, που υποδεικνύει μια επιτυχημένη απόκριση (OK).
2. Ένα μήνυμα στο response body, που υποδεικνύει ότι η "Ανάλυση ολοκληρώθηκε."

Τέλος, ο κώδικας εκτελεί εκκαθάριση διαγράφοντας τον κλωνοποιημένο φακελο του project με την βοήθεια του Commands.deleteProject και έπειτα κάνει return το response που δημιούργησε.

GET - /OPPORTUNITIES Endpoint



Σχήμα 3.2: Διάγραμμα Sequence για το Endpoint get - semi/opportunities.

Το endpoint "/opportunities" ορίζεται για να χειρίζεται HTTP GET αιτήματα (Χρησιμοποιώντας το σχόλιο RequestMapping). Όταν ένας client στέλνει ένα αίτημα GET σε αυτό το Endpoint, αναμένει να λάβει δεδομένα που σχετίζονται με με το αποτελεσμα της semi analysis για ένα συγκεκριμένο έργο.Δέχεται τρια query parameters:

1. projectName: Μια παράμετρος συμβολοσειράς που αντιπροσωπεύει το όνομα του έργου για το οποίο ζητούνται ευκαιρίες.
2. page: Μια ακέραια παράμετρος που υποδεικνύει τον αριθμό σελίδας των opportunities για ανάκτηση.
3. pageSize: Μια ακέραια παράμετρος που καθορίζει τον αριθμό των opportunities για εμφάνιση ανά σελίδα.

Μέσα στο controller καλειται το SemiService με το projectName, page και pageSize και ανακτα τα δεδομενα που δημιουργηθηκαν για το project που δοθηκε στα query params. Τα αποτελέσματα της ανάκτησης επιστρέφονται ως απάντηση από το enpoint. Αυτή η απάντηση είναι του τύπου SemiOpportunitiesResponseBody.

DBController - SemiTool

Η κλάση DbController είναι μια κλάση υπεύθυνη για τη διαχείριση και τη διασύνδεση της βάσης δεδομένων με το Semi analysis εργαλείο, ειδικά φτιαγμένη για να διαχειρίζεται opportunities. Οι

κύριες λειτουργίες του περιλαμβάνουν την εισαγωγή και διαγραφή δεδομένων και λειτουργεί ως γέφυρα μεταξύ της λογικής της semi analysis εργαλείου και της υπηρεσίας βάσης δεδομένων, που αντιπροσωπεύεται από την κλάση SemiService.

Η μέθοδος **dbActions** στην κλάση DbController είναι ένα κρίσιμο μέρος αυτής της κλάσσης. Χρησιμεύει ως μηχανισμός ελέγχου υψηλού επιπέδου για την αλληλεπίδραση με την βάση δεδομένων. Η μέθοδος λαμβάνει δύο παραμέτρους, το projectName και το C_ProjectVersion. Ξεκινά με την εκτέλεση ελέγχων. Διασφαλίζει ότι το projectName δεν είναι μηδενικό ή κενό και ότι το C_ProjectVersion δεν είναι αρνητικό. Εάν κάποια από αυτές τις προϋποθέσεις δεν πληρούται, η μέθοδος επιστρέφει false, υποδεικνύοντας ότι η λειτουργία δεν ήταν επιτυχής.
Η μέθοδος ορίζει τη μεταβλητή επιπέδου κλάσης this.version στην παρεχόμενη C_ProjectVersion. Αυτή η μεταβλητή αντιπροσωπεύει την έκδοση του έργου.
Στη συνέχεια καλεί τη μέθοδο deletePreviousInsertsOfProject για να αφαιρέσει τυχόν υπάρχουσες εγγραφές που σχετίζονται με το συγκεκριμένο έργο και την έκδοση. Αυτή η ενέργεια είναι σημαντική για τη διασφάλιση ότι η βάση δεδομένων δεν περιέχει διπλότυπες ή ξεπερασμένες πληροφορίες. Τα αποτελέσματα αυτής της λειτουργίας αποθηκεύονται στη μεταβλητή ret.
Μετά την επιτυχή διαγραφή προηγούμενων εγγραφών, η μέθοδος καλεί το doInserts για την εισαγωγή νέων εγγραφών στη βάση δεδομένων. Αυτή η μέθοδος είναι υπεύθυνη για την αποθήκευση των δεδομένων που σχετίζονται με τα opportunities objects που εχει αποθηκευμένα ήδη στην μεταβλητή opportunitiesList.
Η μέθοδος dbActions επιστρέφει true μόνο εάν και οι δύο λειτουργίες διαγραφής και εισαγωγής είναι επιτυχείς. Εάν κάποιο από αυτά αποτύχει, επιστρέφει false, υποδεικνύοντας ότι η συνολική ενέργεια της βάσης δεδομένων δεν ήταν επιτυχής.

Η μέθοδος **doInserts** εντός της κλάσης DbController είναι υπεύθυνη για την εισαγωγή νέων εγγραφών στη βάση δεδομένων. Είναι ένα κρίσιμο μέρος της διαδικασίας για τη διαχείριση δεδομένων.
Πριν από την εισαγωγή των εγγραφών, η μέθοδος κάνει loop τα στοιχεία στη λίστα opportunities.
Για κάθε opportunities, ορίζει το χαρακτηριστικό ProjectVersion στην τιμή που C_ProjectVersion. Αυτό το βήμα διασφαλίζει ότι όλα τα opportunities στη λίστα συσχετίζονται με τη έκδοση του έργου.
Στη συνέχεια, η μέθοδος επιχειρεί να εισαγάγει την ενημερωμένη λίστα opportunities στη βάση δεδομένων χρησιμοποιώντας τη μέθοδο service.saveOpportunitiesArray που παρέχεται από την κλάση SemiService. Αυτή η μέθοδος αποθηκεύει τα δεδομένα στη βάση δεδομένων. Εάν αυτή η λειτουργία είναι επιτυχής, σημαίνει ότι οι νέες εγγραφές έχουν προστεθεί με επιτυχία στη βάση δεδομένων.
Σε περίπτωση που δημιουργηθεί Exception κατά τη διαδικασία εισαγωγής, η μέθοδος συλλαμβάνει την εξαίρεση και τη χειρίζεται. Εκτυπώνει ένα μήνυμα σφάλματος που υποδεικνύει ότι η λειτουργία INSERT δεν μπορεί να εκτελεστεί, μαζί με πρόσθετες πληροφορίες που παρέχονται από το Exception.

Η μέθοδος **deletePreviousInsertsOfProject** εντός της κλάσης DbController είναι υπεύθυνη για τη διαγραφή προηγούμενων εγγραφών που σχετίζονται με ένα συγκεκριμένο project και έκδοση project στη βάση δεδομένων. Αυτή η μέθοδος παίρνει δύο παραμέτρους:

- projectName: Το όνομα του project για το οποίο πρέπει να διαγραφούν οι προηγούμενες εγγραφές.
- projectVersion: Η έκδοση του project για την οποία πρέπει να διαγραφούν οι εγγραφές.

Ο σκοπός αυτής της μεθόδου είναι η αλληλεπίδραση με τη βάση δεδομένων, καλώντας συγκεκριμένα τη μέθοδο service.deleteByProjectNameAndProjectVersion. Αυτή η μέθοδος αναμένεται να

διαγράψει εγγραφές από τη βάση δεδομένων με βάση το παρεχόμενο όνομα και την έκδοση του project.
Εάν η λειτουργία διαγραφής είναι επιτυχής, επιστρέφει true. Επιπλέον, εκτυπώνει ένα μήνυμα στην κονσόλα που υποδεικνύει πόσες σειρές διαγράφηκαν.
Εάν παρουσιαστεί ένα exception κατά τη διαδικασία διαγραφής (για παράδειγμα, λόγω προβλήματος σύνδεσης βάσης δεδομένων ή σφάλματος ερωτήματος), η μέθοδος εντοπίζει το Exception και το χειρίζεται. Εκτυπώνει ένα μήνυμα σφάλματος που υποδεικνύει ότι δεν μπόρεσε να διαγράψει παλιές πληροφορίες βάσης δεδομένων για το έργο και περιλαμβάνει πρόσθετες πληροφορίες από την εξαίρεση, όπως ένα μήνυμα σφάλματος ή ένα ίχνος στοίβας.
Σε περίπτωση Exception, η μέθοδος επιστρέφει false για να σηματοδοτήσει ότι η λειτουργία διαγραφής δεν ήταν επιτυχής.

Η μέθοδος **insertMethodToDatabase** εισάγει opportunities σχετικά με μια συγκεκριμένη μέθοδο λογισμικού, συμπεριλαμβανομένου του έργου, της κλάσης, του ονόματος της μεθόδου, των αριθμών γραμμών, των μέτρων συνοχής, των Γραμμών κώδικα και της διαδρομής κλάσης, σε ένα πίνακα αντικειμένων Opportunities. Αυτός ο πίνακας αργότερα θα χρησιμοποιηθεί για περαιτέρω επεξεργασία και για εισαγωγή δεδομένων στη βάση δεδομένων.
Η μέθοδος εκτελεί ελέγχους για να διασφαλίσει ότι όλες οι παράμετροι δεν είναι null, δεν είναι κενές και έχουν έγκυρες τιμές. Εάν κάποια παράμετρος αποτύχει στην επικύρωση, η μέθοδος επιστρέφει false για να υποδείξει μια ανεπιτυχή εισαγωγή.
Εάν περάσουν όλοι οι έλεγχοι, δημιουργεί ένα νέο αντικείμενο Opportunities. Οι ιδιότητες του αντικειμένου Opportunities ορίζονται με τις τιμές που παρέχονται ως παραμέτρους. Στη συνέχεια, το αντικείμενο Opportunities προστίθεται σε έναν πίνακα αντικειμένων. Αυτός ο πίνακας χρησιμοποιείται για τη συλλογή Opportunities. Εάν η εισαγωγή είναι επιτυχής η μέθοδος επιστρέφει true. Εάν υπάρχουν προβλήματα κατά την εισαγωγή ή στους ελέγχους, επιστρέφει false.

3.2.3 Πεδίο GEA

Entities

GeaClasses

(src/main/java/gr/tsitoumis/geasemi/gea/entities/GeaClasses.java)

Η κλάση GeaClasses είναι μια κλάση οντοτήτων Java που έχει σχεδιαστεί για χρήση Java Persistence API (JPA) για λειτουργίες βάσης δεδομένων. Αυτή η κλάση επισημαίνεται με τον σχολιασμό @Entity, που σημαίνει ότι αντιπροσωπεύει μια οντότητα βάσης δεδομένων και ότι τα αντικείμενά της μπορούν να διατηρηθούν σε μια βάση δεδομένων. Ο σχολιασμός @Table καθορίζει το όνομα του πίνακα βάσης δεδομένων που σχετίζεται με αυτήν την οντότητα. Σε αυτήν την περίπτωση, ονομάζεται "gea_classes." Η κλάση περιλαμβάνει πολλά πεδία για την αποθήκευση δεδομένων που σχετίζονται με κλάσεις λογισμικού:

- id: Ένα μοναδικό αναγνωριστικό για κάθε κλάση.
- name: Το όνομα της τάξης, αποθηκευμένο ως κείμενο.
- packageID: Ένας ακέραιος αριθμός που αντιπροσωπεύει το πακέτο στο οποίο ανήκει η κλάση.
- projectName: Το όνομα του έργου στο οποίο ανήκει η κλάση, αποθηκευμένο ως κείμενο.
- cohesion: Αριθμητική τιμή που αντιπροσωπεύει τη μέτρηση cohesion της κλάσης.

- coupling: Μια αριθμητική τιμή που αντιπροσωπεύει τη μετρική coupling της κλάσης.
- isNew: Ένα boolean πεδίο που υποδεικνύει εάν η κλάση είναι νέα.

Το πεδίο id επισημαίνεται με τον σχολιασμό @Id, προσδιορίζοντάς το ως το πρωτεύον κλειδί για τον πίνακα της βάσης δεδομένων. Ο σχολιασμός @GeneratedValue με τη στρατηγική GenerationType.IDENTITY διασφαλίζει ότι το πεδίο αναγνωριστικού δημιουργείται αυτόματα από τη βάση δεδομένων. Τα πεδία name και projectName σημειώνονται με @Lob, υποδεικνύοντας ότι μπορούν να αποθηκεύσουν μεγάλα κομμάτια δεδομένων κειμένου. Η κλάση παρέχει πολλούς constructors, συμπεριλαμβανομένου ενός προεπιλεγμένου constructor και ενός constructor που αρχικοποιεί όλα τα πεδία. Αυτά χρησιμοποιούνται για τη δημιουργία αντικειμένων της κλάσης με τα απαραίτητα δεδομένα.

Η κλάση κάνει override τις τυπικές μεθόδους Java:

1. hashCode(): Υπολογίζει έναν μοναδικό hash κωδικό για κάθε παρουσία με βάση το αναγνωριστικό του.
2. equals(): Συγκρίνει περιπτώσεις με βάση τα πεδία τους, επιτρέποντας ελέγχους ισότητας.
3. toString(): Παρέχει μια αναγνώσιμη από τον άνθρωπο αναπαράσταση συμβολοσειράς της κλάσης.

GeaPackages

(src/main/java/gr/tsitoumis/geasemi/gea/entities/GeaPackages.java)

Η κλάση GeaPackages είναι μια κλάση οντοτήτων που έχει σχεδιαστεί για χρήση Java Persistence API (JPA) για λειτουργίες βάσης δεδομένων.

Ακριβώς όπως η κλάση GeaClasses που συζητήθηκε προηγουμένως, αυτή η κλάση φέρει τον σχολιασμό @Entity, υποδεικνύοντας τον ρόλο της ως οντότητα βάσης δεδομένων που μπορεί να αποθηκευτεί σε μια βάση δεδομένων. Ο σχολιασμός @Table καθορίζει το όνομα του πίνακα βάσης δεδομένων, επισημαίνεται ως "gea_packages". Η κλάση περιλαμβάνει πολλαπλά πεδία για την αποθήκευση δεδομένων σχετικά με πακέτα λογισμικού:

- id: Ένα αποκλειστικό αναγνωριστικό για κάθε πακέτο.
- name: Το όνομα του πακέτου, αποθηκευμένο ως κείμενο.
- coupling: Μια αριθμητική τιμή που υποδηλώνει τη μέτρηση συνοχής του πακέτου.
- cohesion: Μια αριθμητική τιμή που υποδεικνύει τη μέτρηση σύζευξης του πακέτου.
- projectName: Το όνομα του έργου στο οποίο ανήκει το πακέτο, αποθηκευμένο ως κείμενο.
- isNew: Ένα boolean πεδίο που υποδηλώνει εάν το πακέτο είναι νέο.

Το πεδίο id σημειώνεται με @Id, προσδιορίζοντάς το ως το πρωτεύον κλειδί για τον συσχετισμένο πίνακα βάσης δεδομένων. Ο σχολιασμός @GeneratedValue με τη στρατηγική GenerationType.IDENTITY διασφαλίζει την αυτόματη δημιουργία τιμών για το πεδίο id από τη βάση δεδομένων όπως και στην προηγούμενή μας περίπτωση.

Όπως και στην προηγούμενη κλάση κανει override τις μεθόδους hashCode, equals, toString.

GeaPackagesWithClasses

(src/main/java/gr/tsitoumis/geasemi/gea/entities/GeaPackagesWithClasses.java)

Η κλάση GeaPackagesWithClasses είναι μια επέκταση της GeaPackages, με πρόσθετη λειτουργικότητα για τη συσχέτιση μιας λίστας GeaClasses. Αυτή η επέκταση επιτρέπει τη μοντελοποίηση σχέσεων μεταξύ των πακέτων λογισμικού και των κλάσεων που περιέχουν.

Η κλάση παρέχει δύο constructors: Ο πρώτος constructor παίρνει ένα αντικείμενο GeaPackages (geaPackage) και μια λίστα αντικειμένων GeaClasses (κλάσεις). Εξάγει δεδομένα από το αντικείμενο

geaPackage και ορίζει τη λίστα των κλάσεων. Αυτός ο constructor επιτρέπει τη δημιουργία μιας παρουσίας GeaPackagesWithClasses με βάση μια υπάρχουσα παρουσία GeaPackages.
Ο δεύτερος κατασκευαστής δέχεται απευθείας τιμές για το id, το name, το cohesion, το coupling, το projectName, το isNew και μια λίστα κλάσεων του πακέτου. Αυτός ο κατασκευαστής επιτρέπει τη δημιουργία μιας νέας παρουσίας GeaPackagesWithClasses με συγκεκριμένα δεδομένα.

GeaProjects

(src/main/java/gr/tsitoumis/geasemi/gea/entities/GeaProjects.java)

Η κλάση GeaProjects είναι άλλη μια κλάση οντοτήτων Η κλάση σημειώνεται με @Entity, υποδεικνύοντας τον ρόλο της ως οντότητα βάσης δεδομένων. Οι παρουσίες αυτής της κλάσης μπορούν να διατηρηθούν σε μια βάση δεδομένων. Ο σχολιασμός @Table καθορίζει το όνομα του συσχετισμένου πίνακα βάσης δεδομένων, ο οποίος ονομάζεται "gea_projects".
Η κλάση περιλαμβάνει πολλά πεδία για την αποθήκευση δεδομένων που σχετίζονται με έργα λογισμικού:

- name: Ένα πεδίο συμβολοσειράς που αντιπροσωπεύει το όνομα του έργου.
- couplingOld: Ένα αριθμητικό πεδίο που αποθηκεύει μια μετρική τιμή που σχετίζεται με τη coupling στην παλιά έκδοση του project.
- cohesionOld: Ένα αριθμητικό πεδίο που αποθηκεύει μια μετρική τιμή που σχετίζεται με τη cohesion στην παλιά έκδοση του project.
- couplingNew: Ένα αριθμητικό πεδίο που αποθηκεύει μια μετρική τιμή που σχετίζεται με τη coupling στη νέα έκδοση του project.
- cohesionNew: Ένα αριθμητικό πεδίο που αποθηκεύει μια μετρική τιμή που σχετίζεται με τη cohesion στη νέα έκδοση του project.

Η κλάση παρέχει τρεις constructors: Ο προεπιλεγμένος constructor δεν έχει παραμέτρους και χρησιμοποιείται για τη δημιουργία αντικειμένων της κλάσης. Ένας constructor με την παράμετρο name χρησιμοποιείται για την προετοιμασία του ονόματος έργου.
Ένας άλλος κατασκευαστής αποδέχεται το όνομα του έργου και τις μετρικές τιμές για το coupling και τη cohesion. Η κλάση περιλαμβάνει μεθόδους getter και setter για κάθε πεδίο, επιτρέποντάς την ανάκτηση και να την τροποποίηση των τιμών των πεδίων.
Όπως και στις προηγούμενες κλάσεις κανει override τις μεθόδους hashCode, equals, toString.

Response Classes

Στο πλαίσιο της εφαρμογής, οι response classes διαδραματίζουν κρίσιμο ρόλο στη διευκόλυνση της ανταλλαγής δεδομένων μεταξύ των server και client. Αυτές οι κλάσεις απόκρισης έχουν σχεδιαστεί για να ενσωματώνουν και να δομούν δεδομένα, καθιστώντας ευκολότερη τη μεταφορά πληροφοριών σχετικά με πακέτα λογισμικού, κλάσεις, έργα και σχετικές μετρήσεις. Αυτή η ενότητα εξετάζει τρεις βασικές κατηγορίες απόκρισης που βοηθούν στην επικοινωνία δεδομένων εντός της εφαρμογής:

GeaClassesResponseBody

(src/main/java/gr/tsitoumis/geasemi/gea/entities/GeaClassesResponseBody.java)

Η κλάση GeaClassesResponseBody έχει σχεδιαστεί για να χειρίζεται απαντήσεις που σχετίζονται με GeaClasses. Περιέχει τα ακόλουθα πεδία:

- geaClasses: Αυτό το πεδίο είναι μια λίστα με αντικείμενα GeaClasses. Επιτρέπει στην εφαρμογή να στείλει μια συλλογή δεδομένων κλάσης ως απάντηση σε αιτήματα client.

- pagination: Το πεδίο pagination είναι τύπου PaginationResponseBody, το οποίο χρησιμοποιείται για τη διαχείριση πληροφοριών σελιδοποίησης για τα response data. Αυτό είναι χρήσιμο όταν ασχολείστε με μεγάλα σύνολα δεδομένων, καθώς επιτρέπει την αποτελεσματική πλοήγηση στα δεδομένα.

GeaPackagesResponse

(src/main/java/gr/tsitoumis/geasemi/gea/entities/GeaPackagesResponse.java)

Η κλάση GeaPackagesResponse ασχολείται με responses που σχετίζονται με GeaPackages.
Τα πεδία της είναι τα εξής:

- geaPackages: Αυτό το πεδίο περιέχει μια λίστα με αντικείμενα GeaPackagesWithClasses, που αντιπροσωπεύουν πακέτα λογισμικού μαζί με τις κλάσεις που περιέχουν.
- pagination: Παρόμοια με το GeaClassesResponseBody, το πεδίο σελιδοποίησης χρησιμοποιείται για τη διαχείριση πληροφοριών σελιδοποίησης.

GeaProjectsResponse

(src/main/java/gr/tsitoumis/geasemi/gea/entities/GeaProjectsResponse.java)

Η τάξη GeaProjectsResponse χειρίζεται responses που αφορούν GeaProjects και τις σχετικές μετρήσεις τους. Περιλαμβάνει τα ακόλουθα χαρακτηριστικά:

- name: Ένα πεδίο συμβολοσειράς που αντιπροσωπεύει το όνομα του project, επιτρέποντας στην εφαρμογή να καθορίσει το έργο στο οποίο αναφέρεται η απάντηση.
- coupling_old: Αντιπροσωπεύει μια μέτρηση που σχετίζεται με το coupling στην παλιά έκδοση του έργου.
- cohesion_old: Αντιπροσωπεύει μια μέτρηση που σχετίζεται με το cohesion στην παλιά έκδοση του έργου.
- coupling_new: Αντιπροσωπεύει μια μέτρηση που σχετίζεται με το coupling στη νέα έκδοση του έργου.
- cohesion_new: Αντιπροσωπεύει μια μέτρηση που σχετίζεται με το cohesion στη νέα έκδοση του έργου.
- coupling_difference: Αντιπροσωπεύει τη διαφορά στις μετρήσεις coupling μεταξύ παλαιών και νέων εκδόσεων του project.
- cohesion_difference: Αντιπροσωπεύει τη διαφορά στις μετρήσεις cohesion μεταξύ παλαιών και νέων εκδόσεων του project.

GeaRepository

(src/main/java/gr/tsitoumis/geasemi/gea/GeaRepository.java)

Αυτή η διεπαφή αποτελεί ουσιαστικό μέρος Spring Data JPA για το Gea analysis.
Η διεπαφη Επεκτείνει το PagingAndSortingRepository. Αυτό σημαίνει ότι το GeaRepository είναι μια διεπαφή repository που παρέχει υποστήριξη για σελιδοποίηση και ταξινόμηση των αποτελεσμάτων.
Η διεπαφή repository ορίζει διάφορες μεθόδους. Αυτές οι μέθοδοι δημιουργούνται χρησιμοποιώντας τον σχολιασμό @Query, ο οποίος επιτρέπει να γράφουν προσαρμοσμένα ερωτήματα βάσης δεδομένων χρησιμοποιώντας JPQL (Java Persistence Query Language).
findGeaClassesByProjectName: Αυτή η μέθοδος ανακτά μια σελιδοποιημένη λίστα οντοτήτων GeaClasses με βάση την παράμετρο projectName. Είναι χρήσιμο για την εύρεση geaClasses που σχετίζονται με ένα συγκεκριμένο project.

findGeaPackagesByProjectName: Παρόμοια με την προηγούμενη μέθοδο, αυτή ανακτά μια σελιδοποιημένη λίστα οντοτήτων GeaPackages με βάση το projectName. Βοηθά να βρεθούν πακέτα που σχετίζονται με ένα συγκεκριμένο project.
findGeaClassesByPackageId: Αυτή η μέθοδος ανακτά μια λίστα GeaClasses που βασίζεται στις παραμέτρους packageId και isNew. Αυτό θα χρησιμοποιείται για την ανάκτηση GeaClasses ενός συγκεκριμένου πακέτου που πληρούν ορισμένα κριτήρια.
findGeaProject: Αυτή η μέθοδος ανακτά μια μεμονωμένη οντότητα GeaProjects με βάση το projectName. Είναι χρήσιμο όταν χρειάζονται πληροφορίες για ένα συγκεκριμένο έργο.

GeaService

(src/main/java/gr/tsitoumis/geasemi/gea/GeasService.java)

Αυτή η κλάση είναι υπεύθυνη για την αλληλεπίδραση με την βάση δεδομένων χρησιμοποιώντας το repository GeaRepository και την παροχή διαφόρων μεθόδων για την ανάκτηση και το χειρισμό δεδομένων. Ο σχολιασμός @Service υποδεικνύει ότι αυτή η κλάση είναι ένα στοιχείο υπηρεσίας Spring και εντοπίζεται αυτόματα από το Spring για dependency injection. Το GeaRepository συνδέεται αυτόματα σε αυτήν την κλάση, επιτρέποντάς της να έχει πρόσβαση και να χρησιμοποιεί τις μεθόδους πρόσβασης δεδομένων που ορίζονται στη διεπαφή του repository.

- ***getGeaClasses(String projectName, int page, int pageSize)***: Αυτή η μέθοδος ανακτά μια σελιδοποιημένη λίστα οντοτήτων GeaClasses που σχετίζονται με ένα δεδομένο projectName. Χρησιμοποιεί ένα αντικείμενο σελιδοποίησης για να χειριστεί τις παραμέτρους της σελίδας και του μεγέθους σελίδας και επιστρέφει ένα GeaClassesResponseBody που περιέχει τα δεδομένα που ανακτήθηκαν μαζί με πληροφορίες σελιδοποίησης.
  Ξεκινά με τη δημιουργία ενός αντικειμένου σελιδοποίησης με την παρεχόμενη σελίδα και το μέγεθος σελίδας. Αυτό το αντικείμενο βοηθά στη διαχείριση των παραμέτρων σελιδοποίησης. Δημιουργεί ένα αντικείμενο με Pegable χρησιμοποιώντας το PageRequest.of() μεταβιβάζοντας τον αριθμό και το μέγεθος σελίδας από το αντικείμενο pagination. Το αντικείμενο Pageable χρησιμοποιείται για τον καθορισμό της σελίδας και του μεγέθους του συνόλου αποτελεσμάτων.
  Καλεί τη μέθοδο repository.findGeaClassesByProjectName(projectName, pageable). Αυτή η μέθοδος ορίζεται στη διεπαφή repository και είναι υπεύθυνη για την υποβολή ερωτημάτων στη βάση δεδομένων για την ανάκτηση οντοτήτων GeaClasses που σχετίζονται με το συγκεκριμένο όνομα project. Τα αποτελέσματα επιστρέφονται ως Page<GeaClasses>.
  Υπολογίζει τον συνολικό αριθμό σελίδων (όλες τις σελίδες) που είναι διαθέσιμες για το σύνολο δεδομένων και έπειτα κατασκευάζει ένα αντικείμενο PaginationResponseBody, παρέχοντας τον τρέχοντα αριθμό σελίδας, το μέγεθος σελίδας και τον συνολικό αριθμό σελίδων. Στη συνέχεια, δημιουργεί ένα αντικείμενο GeaClassesResponseBody, ενσωματώνοντας το περιεχόμενο των ανακτημένων οντοτήτων GeaClasses και τις πληροφορίες σελιδοποίησης. Τέλος, η μέθοδος επιστρέφει το GeaClassesResponseBody, το οποίο περιέχει τη λίστα των οντοτήτων GeaClasses και τις πληροφορίες σελιδοποίησης.

- ***getGeaPackagesWithMovableClasses(String projectName, int page, int pageSize)***: Αυτή η μέθοδος ανακτά μια σελιδοποιημένη λίστα οντοτήτων GeaPackages που σχετίζονται με ένα δεδομένο projectName. Επιστρέφει GeaPackagesResponse που περιέχει πακέτα με τις αντίστοιχες κλάσεις τους.

Ο κώδικας ξεκινά με τη δημιουργία ενός αντικειμένου pagination με τον παρεχόμενο αριθμό σελίδας και το μέγεθος σελίδας. Δημιουργεί ένα Pageable αντικείμενο χρησιμοποιώντας το PageRequest.of(). Αυτό το αντικείμενο χρησιμοποιείται για τον καθορισμό της σελίδας και του μεγέθους του συνόλου αποτελεσμάτων. Καλεί τη μέθοδο repository.findGeaPackagesByProjectName(projectName, pageable). Αυτή η μέθοδος ορίζεται στη διεπαφή του repository και είναι υπεύθυνη για την υποβολή ερωτημάτων στη βάση δεδομένων για την ανάκτηση οντοτήτων GeaPackages που σχετίζονται με το συγκεκριμένο όνομα του project. Τα αποτελέσματα επιστρέφονται ως Page<GeaPackages>.
Υπολογίζει τον συνολικό αριθμό σελίδων (όλες τις σελίδες) που είναι διαθέσιμες για το σύνολο δεδομένων. Δημιουργεί μια λίστα GeaPackagesWithClasses και επειτα για κάθε ένα GeaPackages object ανακτα τα GeaClasses objects που εχουν isNew=True (το οποίο σημαίνει οτι θα ανακτήσει moveable classes) και αντιστοιχούν στο package id του και τα προσθέτει στην λίστα με τα GeaPackagesWithClasses ώστε να τα αποτελέσματα που γυρνάμε στον client να περιέχουν και τις πληροφορίες του package και τις moveable classes τους. Στη συνέχεια, δημιουργεί ένα αντικείμενο PaginationResponseBody. Και τέλος, ενσωματώνοντας το περιεχόμενο των ανακτημένων οντοτήτων και τις πληροφορίες σελιδοποίησης δημιουργεί ένα GeaPackagesResponse και το επιστρέφει σαν αποτέλεσμα.

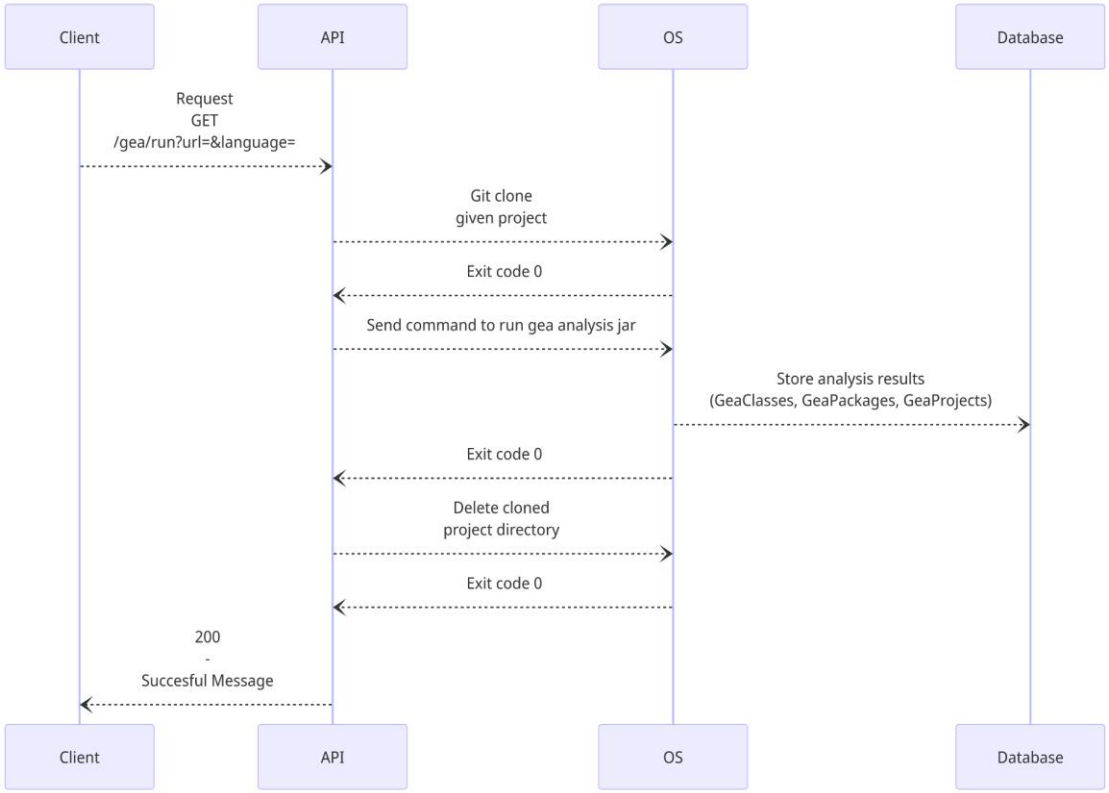
- ***getGeaProject(String projectName)***: Αυτή η μέθοδος ανακτά πληροφορίες για ένα συγκεκριμένο project με το όνομά του. Υπολογίζει τις διαφορές στο cohesion και τις τιμές coupling μεταξύ της παλαιάς και της νέας έκδοσης του έργου και επιστρέφει ένα GeaProjectsReponse που περιέχει τις λεπτομέρειες του έργου και αυτές τις διαφορές.
  Ο κώδικας ξεκινά με την κλήση της μεθόδου repository.findGeaProject(projectName). Αυτή η μέθοδος ορίζεται στη διεπαφή repository και είναι υπεύθυνη για την αναζήτηση στη βάση δεδομένων για την ανάκτηση του GeaProjects object του project με το καθορισμένο όνομα. Το αποτέλεσμα επιστρέφεται οντότητα GeaProjects. Αφού λάβει τις λεπτομέρειες του έργου, υπολογίζει τις διαφορές μεταξύ των παλαιών και των νέων τιμών για δύο βασικές μετρήσεις, το cohesion και το coupling. Οι υπολογιζόμενες διαφορές αποθηκεύονται στις μεταβλητές cohesion_difference και coupling_difference. Τέλος, δημιουργεί ένα αντικείμενο GeaProjectsReponse, ενσωματώνοντας τις λεπτομέρειες του έργου, καθώς και τις απόλυτες τιμές των διαφορών συνοχής και σύζευξης. Αυτό το αντικείμενο απόκρισης είναι δομημένο ώστε να παρουσιάζει τα δεδομένα του έργου μαζί με τις υπολογιζόμενες διαφορές.

GeaController

(src/main/java/gr/tsitoumis/geasemi/gea/GeaController.java)

Προκειται για μια κλάση controller Spring MVC. Αυτός ο controller είναι υπεύθυνος για το χειρισμό αιτημάτων HTTP που σχετίζονται με το εργαλείο ανάλυσης "Gea".
Ο σχολιασμός @Controller επισημαίνει αυτήν την κλάση ως controller Spring MVC. Ο σχολιασμός @RequestMapping σε επίπεδο κλάσης καθορίζει τη βασική διαδρομή διεύθυνσης URL για όλα τα endpoints που ορίζονται σε αυτόν τον ελεγκτή.
Το GeaService συνδέεται αυτόματα στον controller, πράγμα που σημαίνει ότι μπορεί να χρησιμοποιήσει τις μεθόδους που ορίζονται για να χειριστεί τα αιτήματα.

Get - gea/run endpoint



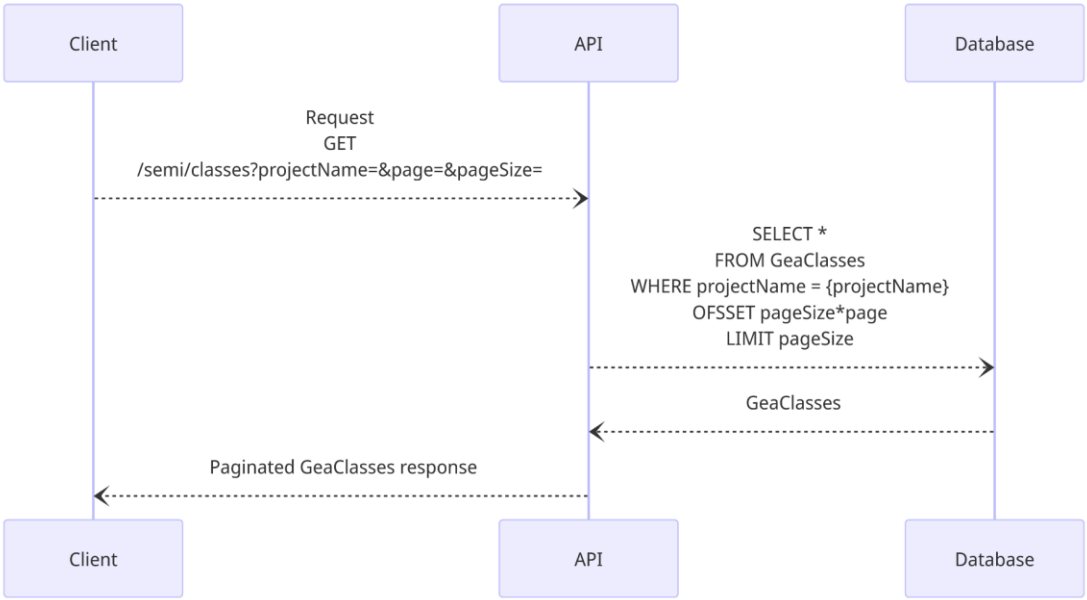
Σχήμα 3.3: Διάγραμμα Sequence για το Endpoint get - gea/run.

Αντιστοιχίζεται στο endpoint /gea/run χρησιμοποιώντας τον σχολιασμό @RequestMapping. Διαχειρίζεται αιτήματα HTTP GET.

Χρειάζονται δύο query params, url και language, οι οποίες παρέχονται στο συμβολοσειρά ερωτήματος της διεύθυνσης URL. Συνοπτικα Αυτο το endpoint εκτελεί ανάλυση σε ένα project που καθορίζεται από το git repository url. Η μέθοδος ξεκινά ανακτώντας το όνομα του έργου από το παρεχόμενο URL χρησιμοποιώντας την GitTools.getProjectName(url) μέθοδο. Στη συνέχεια, κλωνοποιεί το Git repository στο τοπικό σύστημα αρχείων χρησιμοποιώντας το παρεχόμενο URL. Αυτό επιτυγχάνεται μέσω της Commands.gitClone(url) μεθόδου. Στη συνέχεια, εκτελεί την "Gea analysis" για το project με βάση το όνομα του project και τη γλώσσα που παρασχέθηκε. Αυτό επιτυγχάνεται μέσω της Commands.geaRun(name, language) μεθόδου. Τέλος, διαγράφει το πρότζεκτ από το τοπικό σύστημα αρχείων. Αυτό επιτυγχάνεται μέσω της Commands.deleteProject(name) μεθόδου.

Η μέθοδος επιστρέφει ένα αντικείμενο τύπου ResponseEntity<MessageResponseBody> με HTTP staus 200 OK και ένα αντικείμενο MessageResponseBody που περιέχει το μήνυμα "Gea analysis COMPLETED".

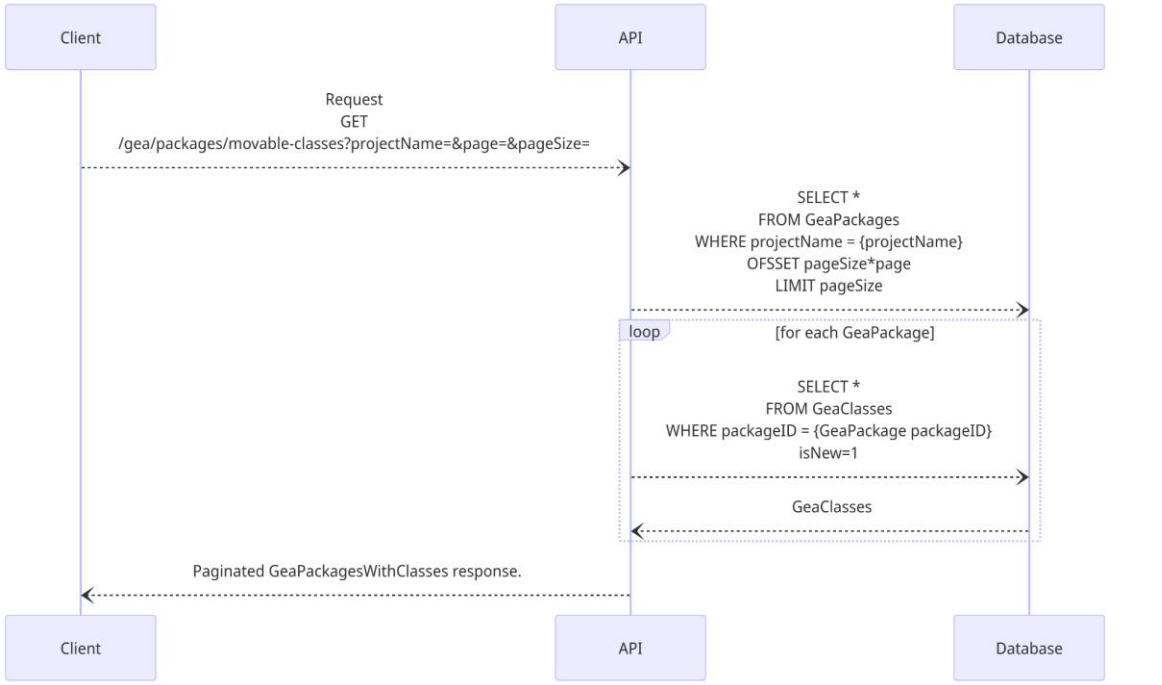
Get - /gea/classes



Σχήμα 3.4: Διάγραμμα Sequence για το Endpoint get - gea/classes

Αντιστοιχίζεται στο endpoint διεύθυνσης URL /gea/classes χρησιμοποιώντας τον σχολιασμό @RequestMapping και διαχειρίζεται αιτήματα HTTP GET.
Απαιτούνται τρεις παράμετροι αιτήματος, το projectName, το page και το pageSize, οι οποίες παρέχονται στη συμβολοσειρά ερωτήματος της διεύθυνσης URL.
Αυτή η μέθοδος καλεί τη μέθοδο service.getGeaClasses από το GeaService για να ανακτήσει μια σελιδοποιημένη λίστα οντοτήτων GeaClasses που σχετίζονται με ένα συγκεκριμένο project. Οι παράμετροι σελιδοποίησης παρέχονται στο αίτημα. Απαντά με ένα GeaClassesResponseBody, που περιέχει τις ανακτημένες GeaClasses και πληροφορίες σελιδοποίησης.

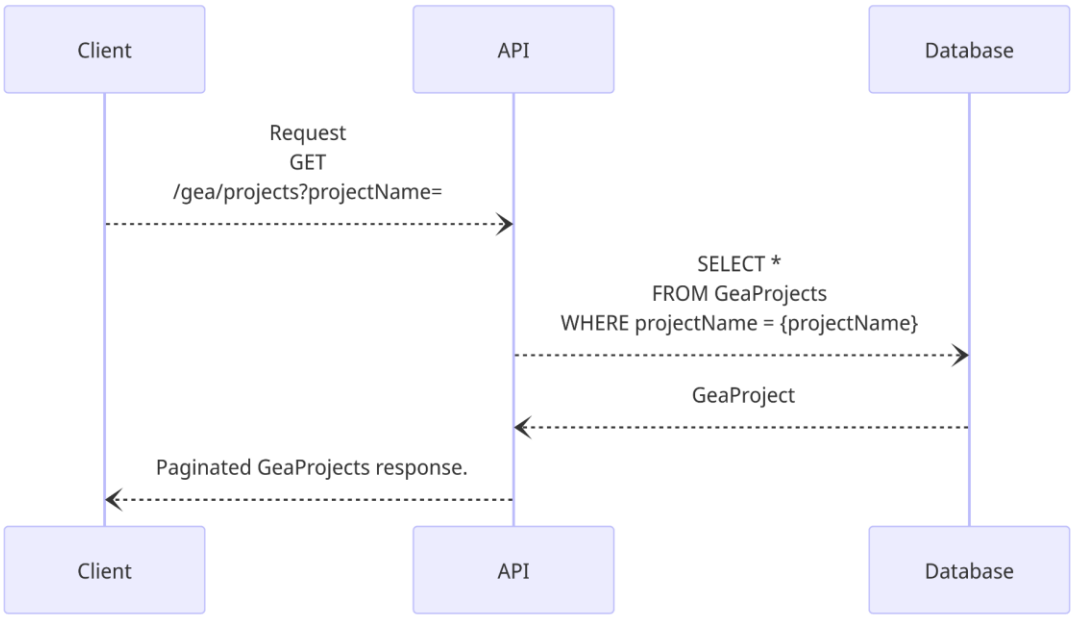
GET - /gea/packages/movable-classes



Σχήμα 3.5: Διάγραμμα Sequence για το Endpoint get - gea/packages/movable-classes

Αυτή η μέθοδος αντιστοιχίζεται στο endpoint διεύθυνσης URL /gea/packages/movable-classes χρησιμοποιώντας τον σχολιασμό @RequestMapping. Διαχειρίζεται αιτήματα HTTP GET.
Απαιτούνται τρεις παράμετροι αιτήματος, το projectName, το page και το pageSize, οι οποίες παρέχονται στη συμβολοσειρά ερωτήματος της διεύθυνσης URL.
Αυτή η μέθοδος καλεί τη μέθοδο service.getGeaPackagesWithMovableClasses από το GeaService για να ανακτήσει μια σελιδοποιημένη λίστα οντοτήτων GeaPackagesWithClasses που σχετίζονται με ένα συγκεκριμένο project, μαζί με τις κλάσεις τους. Απαντά με ένα GeaPackagesResponse, που περιέχει τα ανακτημένα πακέτα με τις κλάσεις και τις πληροφορίες σελιδοποίησης.

Get - /gea/projects



Σχήμα 3.6: Διάγραμμα Sequence για το Endpoint get - gea/projects

RequestMapping: Αυτή η μέθοδος αντιστοιχίζεται στο endpoint διεύθυνσης URL /gea/projects χρησιμοποιώντας τον σχολιασμό @RequestMapping. Διαχειρίζεται αιτήματα HTTP GET. Χρειάζεται ένα query param, το projectName, το οποίο παρέχεται στη συμβολοσειρά ερωτήματος της διεύθυνσης URL.Αυτή η μέθοδος καλεί τη μέθοδο service.getGeaProject από το GeaService για να ανακτήσει τις λεπτομέρειες του project και να υπολογίσει τις διαφορές στις μετρήσεις cohesion και coupling.
Απαντά με ένα GeaProjectsReponse, που περιέχει τις λεπτομέρειες του έργου και τις υπολογιζόμενες διαφορές στις μετρήσεις cohesion και coupling.

3.2.4 Service Health

(src/main/java/gr/tsitoumis/geasemi/health/HealthController.java)

Πρόκειται για ένα controller οπου χειρίζεται αιτήματα ελέγχου υγείας για την εφαρμογή.
Ο σχολιασμός @Controller επισημαίνει αυτήν την κλάση ως ελεγκτή Spring MVC. Ο
Η μέθοδος health αντιστοιχίζεται σε πολλαπλά τελικά σημεία διεύθυνσης URL, συμπεριλαμβανομένων των /health, /healthy, /healthz χρησιμοποιώντας τον σχολιασμό @RequestMapping. Διαχειρίζεται αιτήματα HTTP GET για αυτά τα τελικά σημεία.
Αυτή η μέθοδος χρησιμεύει ως τελικό σημείο ελέγχου υγείας. Κατά την πρόσβαση, αποκρίνεται με κατάσταση HTTP 200 OK, υποδεικνύοντας ότι η εφαρμογή είναι υγιής. Επιστρέφει ένα MessageResponseBody με το μήνυμα "Ok" ως body.

3.2.5 Service ExceptionHandling

(src/main/java/gr/tsitoumis/geasemi/ExceptionHandling.java)

Πρόκειται για τον μηχανισμό χειρισμού exception για την εφαρμογή.

Ο σχολιασμός @ControllerAdvice επισημαίνει αυτήν την κλάση ως χειριστή εξαιρέσεων που μπορεί να εφαρμοστεί σε όλους τους ελεγκτές της εφαρμογής.
Η κλάση αρχικοποιεί έναν logger χρησιμοποιώντας το LogManager.getLogger(ExceptionHandling.class)
@ExceptionHandler(Exception.class): Αυτός ο σχολιασμός υποδεικνύει ότι η ακόλουθη μέθοδος είναι υπεύθυνη για το χειρισμό εξαιρέσεων του τύπου Exception ή των υποκλάσεων του.
Η κλάση ExceptionHandling χρησιμεύει ως global χειριστής εξαιρέσεων για την εφαρμογή Spring. Χειρίζεται εξαιρέσεις, καθορίζει την κατάλληλη κατάσταση HTTP και το μήνυμα σφάλματος, καταγράφει την εξαίρεση και δημιουργεί μια απάντηση με τις λεπτομέρειες του σφάλματος.
Λαμβάνει ένα αντικείμενο Exception (εξαίρεση) και ένα αντικείμενο WebRequest (webRequest).
Χειρίζεται εξαιρέσεις που μπορεί να προκύψουν κατά την εκτέλεση ενεργειών στους controller της εφαρμογής. Καθορίζει πρώτα τον κωδικό κατάστασης HTTP που θα χρησιμοποιηθεί στην απάντηση (httpStatus) και στο μήνυμα σφάλματος (μήνυμα). Εάν η εξαίρεση είναι τυπου GeaSemiException, εξάγει την κατάσταση HTTP και το μήνυμα από την εξαίρεση. Διαφορετικά, ορίζεται από προεπιλογή ένα εσωτερικό σφάλμα διακομιστή.
Η μέθοδος καταγράφει την εξαίρεση χρησιμοποιώντας το logger.
Τέλος δημιουργεί ένα αντικείμενο ExceptionResponse που περιέχει την κατάσταση και το μήνυμα HTTP και στη συνέχεια δημιουργεί ένα ResponseEntity με αυτήν την απόκριση και την καθορισμένη κατάσταση HTTP.
Η ExceptionResponse είναι μια απλή κλάση που αντιπροσωπεύει το response για εξαιρέσεις. Έχει χαρακτηριστικά για τον κωδικό κατάστασης HTTP και το μήνυμα σφάλματος .
Ο constructor αρχικοποιεί αυτά τα χαρακτηριστικά και υπάρχουν μέθοδοι getter και setter.

3.2.6 Cronjob για την αυτόματη διαγραφή Project

(src/main/java/gr/tsitoumis/geasemi/Scheduler.java)

Αυτή η κλάση περιέχει μια προγραμματισμένη μέθοδο που ονομάζεται cleanGitProjects. Ο σκοπός αυτού του scheduler είναι να καθαρίζει περιοδικά τον φάκελο που περιέχει git repositories.
Ο σχολιασμός @Scheduled χρησιμοποιείται για την επισήμανση της μεθόδου cleanGitProjects ως προγραμματισμένης εργασίας. Αυτή η μέθοδος θα εκτελείται σε τακτά χρονικά διαστήματα με βάση την έκφραση cron που παρέχεται.
Η μέθοδος cleanGitProjects εκτελείται σε προγραμματισμένη βάση για τον καθαρισμό ενός φακέλου που περιέχει Git repositories.
Σημειώνεται με @Scheduled(cron = "0 0 */3 * * *"), που σημαίνει ότι θα εκτελείται κάθε 3 ώρες. Η έκφραση cron "0 0 */3 * * *" καθορίζει το χρονοδιάγραμμα για την εργασία.
Μέσα στη μέθοδο, εκτελεί τις ακόλουθες εργασίες:

1. Λαμβάνει την απόλυτη διαδρομή του καταλόγου που πρόκειται να καθαριστεί. Η διαδρομή καταλόγου ορίζεται ως "git-repositories" στον τρέχοντα κατάλογο εργασίας.
2. Χρησιμοποιεί το FileUtils.cleanDirectory του Apache Commons IO για να διαγράψει τα περιεχόμενα του καθορισμένου καταλόγου. Αυτό καθαρίζει αποτελεσματικά όλα τα αρχεία και τους υποκαταλόγους στον κατάλογο "git-repositories".
3. Καταγράφει ένα μήνυμα που υποδεικνύει ότι η εργασία καθαρισμού εκτελέστηκε με επιτυχία.
4. Εντοπίζει τυχόν εξαιρέσεις που ενδέχεται να προκύψουν κατά τη διαδικασία καθαρισμού και καταγράφει ένα μήνυμα σφάλματος.

3.3 Παρουσίαση εργαλείου

3.3.1 Ρυθμίσεις σύνδεσης της βάσης δεδομένων

Το configuration της σύνδεσης στην βάση δεδομένων για την εφαρμογή Spring Boot αποθηκεύεται στο αρχείο configuration που ονομάζεται application.properties. Αυτό το αρχείο βρίσκεται στον κατάλογο src/main/resources του έργου.
Άνοιξε το αρχείο application.properties χρησιμοποιώντας ένα πρόγραμμα επεξεργασίας κειμένου ή ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE).
Μέσα στο αρχείο application.properties, μπορείς να διαμορφώσεις τη σύνδεση της βάσης δεδομένων καθορίζοντας τις ακόλουθες ιδιότητες:

- ***spring.datasource.url***: Χρησιμοποιείται για τον καθορισμό της διεύθυνσης URL της βάσης δεδομένων.
    - Παράδειγμα:
      spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
- ***spring.datasource.username***: Όρισε αυτήν την ιδιότητα για να καθορίσεις το όνομα χρήστη που απαιτείται για τη σύνδεση στη βάση δεδομένων.
    - Παράδειγμα:
      spring.datasource.username=myusername
- ***spring.datasource.password***: Καθόρισε τον κωδικό πρόσβασης που απαιτείται για τον έλεγχο ταυτότητας και τη δημιουργία σύνδεσης με τη βάση δεδομένων. Εισηγάγε τον σωστό κωδικό πρόσβασης για τη βάση δεδομένων σας.
    - Παράδειγμα:
      spring.datasource.password=mypassword

Αφού ρυθμίσεις αυτές τις ιδιότητες στο αρχείο application.properties, αποθήκευσε τις αλλαγές σου.

3.3.2 Έναρξη Project

Βήμα 1: Δημιουργια Docker Image
Το πρώτο βήμα για το deploy του project Spring Boot με το Docker είναι να δημιουργηθεί μια εικόνα Docker.
Για να δημιουργήσεις μια εικόνα Docker, ανοίξε το τερματικό σου ή τη γραμμή εντολών και μεταφέρσου στον κατάλογο του project GeaSemi όπου βρίσκεται το Dockerfile. Το Dockerfile περιέχει οδηγίες για τον τρόπο δημιουργίας της εικόνας.

docker build -t springio/geasemi .

Docker build: Αυτή η εντολή λέει στον Docker να δημιουργήσει μια εικόνα.
Η σημαία -t σάς επιτρέπει να προσθέσεις ετικέτα στην εικόνα με ένα όνομα και προαιρετικά μια ετικέτα (σε αυτήν την περίπτωση, το springio/geasemi χρησιμοποιείται ως όνομα εικόνας).
Η τελεία (.) στο τέλος καθορίζει το περιβάλλον κατασκευής, που είναι ο τρέχων κατάλογος που περιέχει το Docker file.
Το Docker θα διαβάσει το Dockerfile και θα εκτελέσει τις οδηγίες μέσα σε αυτό για να δημιουργήσει ενα image για την εφαρμογή Spring Boot.

Βήμα 2: Εκτελέστε το Docker Container
Τώρα που έχεις δημιουργήσει την εικόνα Docker, το επόμενο βήμα είναι να εκτελέσεις το container Docker από το image.
Για να ξεκινήσεις την εφαρμογή Spring Boot σε ένα κοντέινερ Docker, χρησιμοποιήσε την ακόλουθη εντολή:

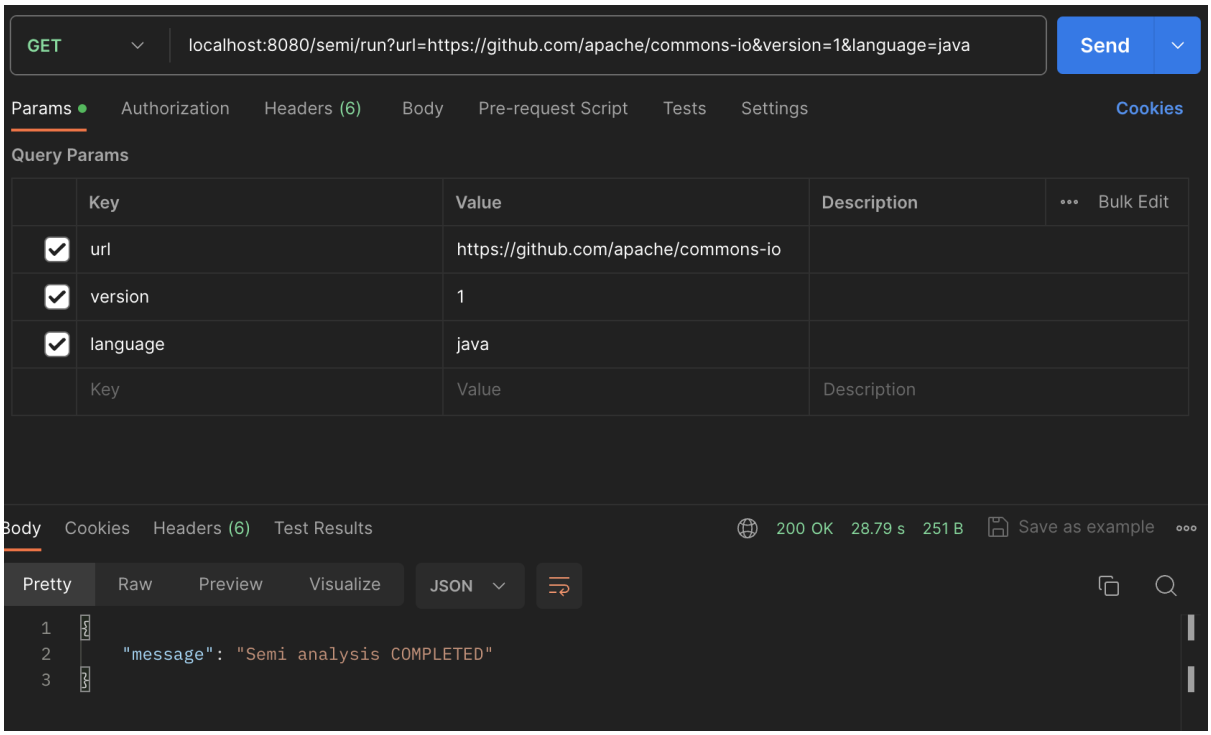
docker run -p 8080:8080 springio/geasemi

Αυτή η εντολή λέει στο Docker να εκτελέσει ένα κοντέινερ με βάση την καθορισμένη εικόνα.
Η σημαία -p καθορίζει τις θύρες. Λέει στο Docker να εκθέσει τη θύρα 8080 από το κοντέινερ στη θύρα 8080 του συστήματος υπολογιστή σου. Αυτό είναι απαραίτητο για την πρόσβαση στην εφαρμογή Spring Boot μέσω του δικτύου.
ΤΟ springio/geasemi είναι το όνομα της εικόνας Docker που θέλεις να εκτελέσεις ως κοντέινερ.
Μόλις εκτελέσεις αυτήν την εντολή, το Docker θα ξεκινήσει ένα κοντέινερ που εκτελεί την εφαρμογή Spring Boot.

3.3.3 Κλήσεις endpoint

Στο web development, εχεις στη διάθεσή σου μια σειρά από εργαλεία για να αλληλεπιδράσεις με τα API. Μπορείς να επιλέξεις έναν client ή ένα εργαλείο HTTP που ταιριάζει καλύτερα στις ανάγκες και τις προτιμήσεις σου. Ορισμένες κοινές επιλογές περιλαμβάνουν το cURL για τη γραμμή εντολών, τον Postman για μια φιλική προς το χρήστη γραφική διεπαφή ή τη χρήση γλωσσών προγραμματισμού με βιβλιοθήκες όπως το Python requests η το Axios της JavaScript.
Στις παρακάτω παρουσιάσεις της χρήσης του εργαλείου θα χρησιμοποιηθεί σαν παράδειγμα το πρόγραμμα Postman.

GET - /semi/run



Σχήμα 3.7: Παράδειγμα κλήσης Endpoint get - semi/run.

Βήμα 1: Καθορισμός λεπτομερειών αιτήματος
Για αρχή πρέπει να διαλέξεις τη μέθοδο HTTP για το αίτημα. Για το endpoint semi/run, χρησιμοποίησε τη μέθοδο GET. Μπορείς να επιλέξεις τη μέθοδο από το μενού που παρέχεται δίπλα στο πεδίο εισαγωγής URL.
Καθόρισε τη διεύθυνση URL του endpoint σου. Η διεύθυνση URL πρέπει να δείχνει στο semi/run, το οποίο είναι http://localhost:8080/semi/run εάν εκτελείς την εφαρμογή Spring Boot τοπικά. Φρόντισε να το αντικαταστήσεις με το πραγματικό URL εάν η εφαρμογή σου φιλοξενείται αλλού.

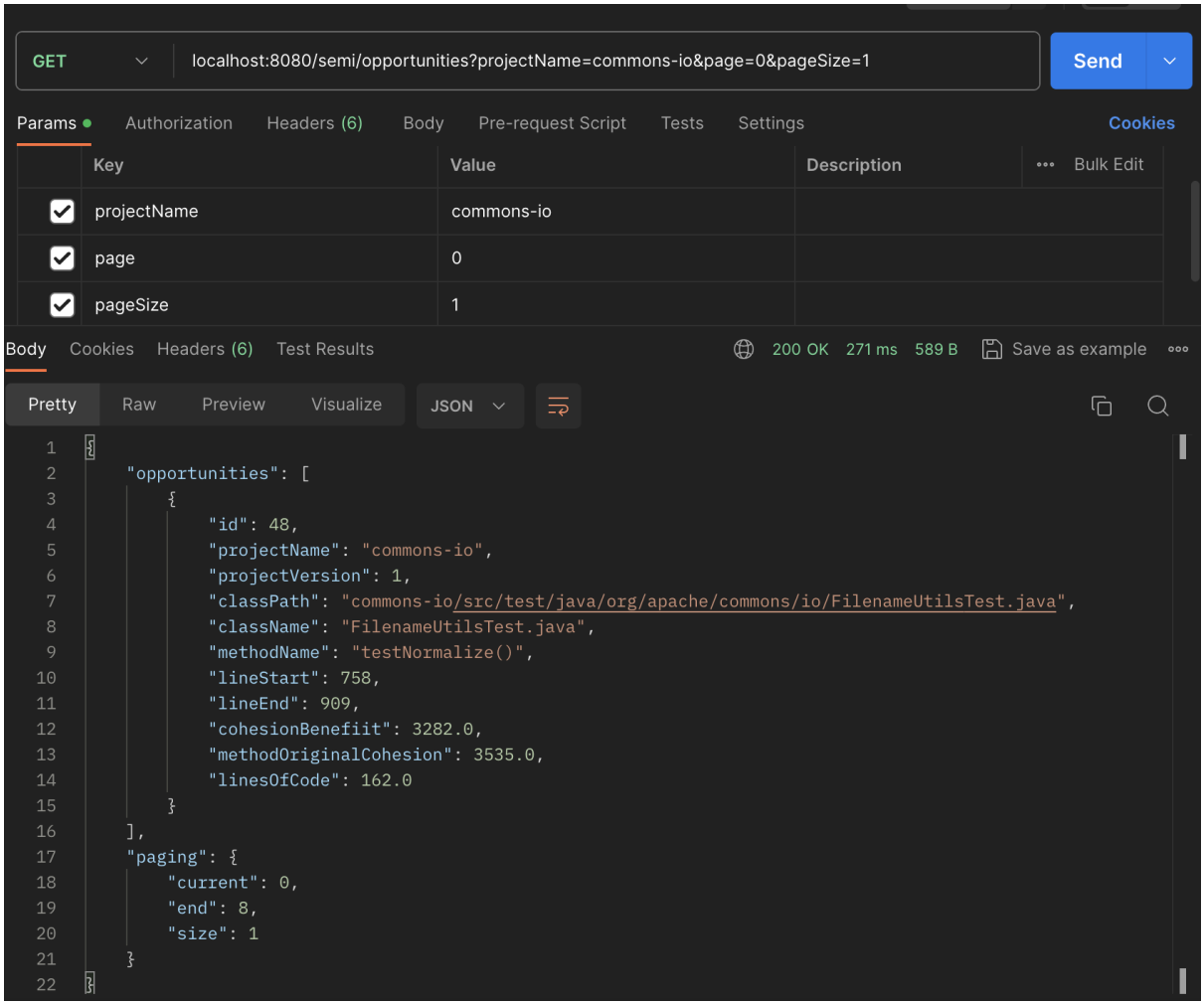
Βήμα 2: Προσθήκη query params
Το τελικό σημείο semi/run στην εφαρμογή Spring Boot αναμένει query params. Για να συμπεριλάβεις αυτές τις παραμέτρους στο αίτημά σου:
Μεταφέρσου στην καρτέλα "Params" που βρίσκεται κάτω από τη διεύθυνση URL αιτήματος στο εργαλείο δημιουργίας αιτημάτων και άρχισε να προσθέτεις παραμέτρους ερωτήματος.
Στο πεδίο "Key", πληκτρολόγησε "url". Αυτό το κλειδί αντιστοιχεί στην παράμετρο URL που αναμένει το τελικό σημείο του API.
Στο πεδίο "Value", καταχώρισε την πραγματική διεύθυνση URL του Git repository που θέλεις να τρέξεις την semi analysis. Για παράδειγμα, μπορείς να εισαγάγεις το "https://github.com/apache/commons-io" εδώ.
Στη νέα σειρά, όρισε το "Κλειδί" σε "language". Αυτό το κλειδί αντιστοιχεί στην παράμετρο που καθορίζει τη γλώσσα προγραμματισμού που θέλεις να αναλύσεις. Στο πεδίο "Value", μπορείς να εισαγάγεις μια γλώσσα όπως η "Java". Αντίστοιχα στη επόμενη νέα σειρά, το κλειδί "version" όπου αντιστοιχεί στην version του project. Στο Value, βάλε τον αριθμό της version του project σου.

Βήμα 3: Αποστολή request και response
Κάνε κλικ στο κουμπί "Send" στο πρόγραμμα δημιουργίας αιτημάτων. Αυτή η ενέργεια ενεργοποιεί τον Postman να στείλει το αίτημα GET στο τελικό σημείο semi/run της εφαρμογής Spring Boot.
Καθώς το αίτημά σου φτάνει στο API, η εφαρμογή Spring Boot το επεξεργάζεται, τρέχει το semi analysis στο project που του έδωσες και δημιουργεί μια απάντηση. Αυτή η απάντηση περιλαμβάνει βασικές πληροφορίες που θα θελεις να ελέγξεις:
Αρχικά, θα δεις έναν κωδικό κατάστασης HTTP, όπως "200 OK", που υποδηλώνει το αποτέλεσμα του αιτήματός σπυ. Η κατάσταση "200 OK" υποδηλώνει ότι το αίτημά σου ήταν επιτυχές, ενώ άλλοι κωδικοί κατάστασης ενδέχεται να σηματοδοτούν σφάλματα ή Exceptions.
Το response body περιέχει ένα μήνυμα. Στην περίπτωση του endpoint semi/run, επιστρέφει ένα μήνυμα "semi analysis COMPLETED".

Παράδειγμα response body:

```
{
   "message": "semi analysis COMPLETED"
}
```

GET - semi/opportunities



Σχήμα 3.8: Παράδειγμα κλήσης Endpoint get - semi/opportunities.

Βήμα 1: Καθορισμός λεπτομερειών αιτήματος

Καθόρισε τη διεύθυνση URL και χρησιμοποίηστε τη μέθοδο GET. Η διεύθυνση θα πρέπει να οδηγεί στο endpoint semi/opportunities, παράδειγμα http://localhost:8080/semi/opportunities εάν η εφαρμογή Spring Boot εκτελείται τοπικά. Αντικατεστήσε το με το πραγματικό URL εάν η εφαρμογή σου φιλοξενείται αλλού.

Βήμα 2: Προσθήκη query params

Το endpoint semi/opportunities στην εφαρμογή Spring Boot χρειάζεται κάποια συγκεκριμένα query params.

Μεταφέρσου στην καρτέλα "Params" που βρίσκεται κάτω από τη διεύθυνση URL και πρόσθεσε τις παρακάτω παραμέτρους:

- projectName: Το όνομα του project που έτρεξες το semi analysis endpoint.
- page: Ο αριθμός σελίδας. Ξεκινάει από το 0.
- pageSize: ο αριθμος αντικειμενων opportunities που θα επιστρεψει το api.

Για παράδειγμα, μπορείς να εισαγάγεις ένα όνομα έργου όπως "commons-io" για την προηγούμενη περίπτωση semi analysis, να ορίσεις τη σελίδα σε "0" για να πάρεις την αρχική σελίδα και το μέγεθος σελίδας σε "10".

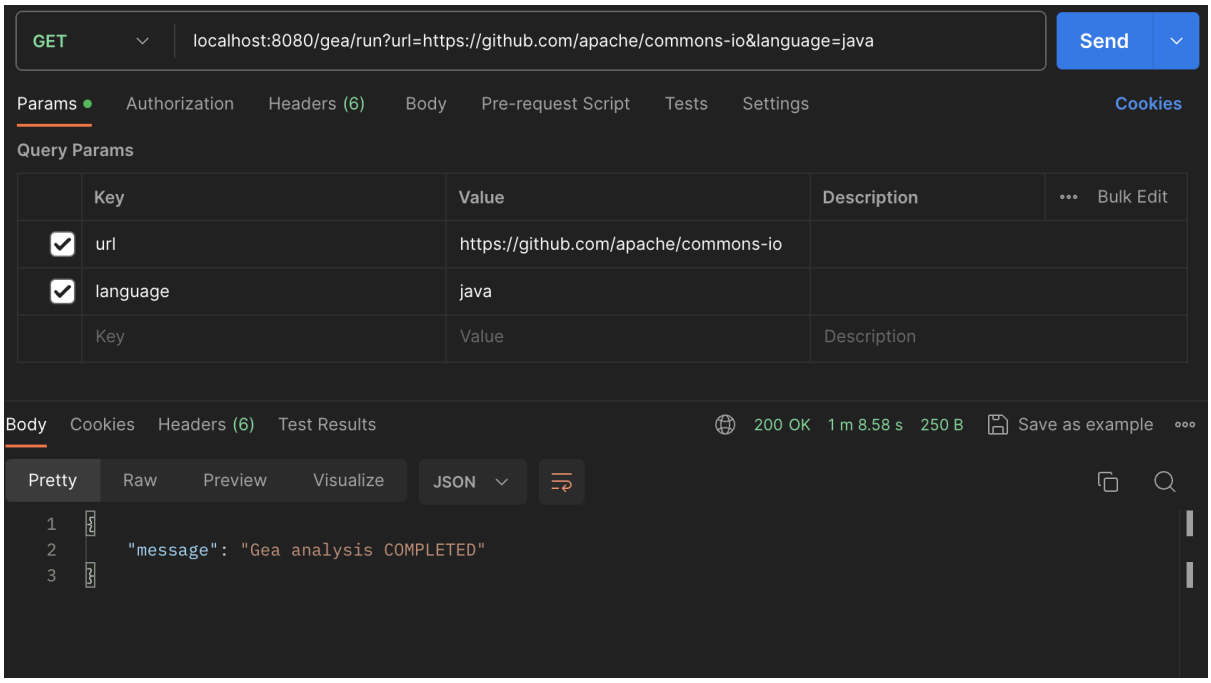
Βήμα 3: Αποστολή request και response

Μετά τη διαμόρφωση του αιτήματός σου κάνε κλικ στο κουμπί "Αποστολή". Αυτή η ενέργεια ζητά από το Postman να στείλει το αίτημα GET στο enpoint semi/opportunities της εφαρμογής Spring Boot.
Καθώς το αίτημά φτάνει στο API, η εφαρμογή Spring Boot το επεξεργάζεται και βρίσκει τα opportunities entries στην βάση δεδομένων για το projectName που του έχει δοθεί και δημιουργεί μια απάντηση. Απο το endpoint semi/opportunities θα λάβεις μια λίστα με τα opportunities objects. Το πλήθος τους θα είναι καθορισμένο από την παράμετρο pageSize που έδωσες στο request και μαζί με την λίστα των opportunities περιλαμβάνεται ένα βοηθητικό paging αντικείμενο το οποίο περιλαμβάνει την σελίδα που ανακτήθηκε, τον αριθμό της τελευταίας σελίδας και το pageSize που δόθηκε.

Παράδειγμα response body:


```
{
  "opportunities": [
    {
      "id": 48,
      "projectName": "commons-io",
      "projectVersion": 1,
      "classPath": "commons-io/src/test/java/org/apache/commons/io/FilenameUtilsTest.java",
      "className": "FilenameUtilsTest.java",
      "methodName": "testNormalize()",
      "lineStart": 758,
      "lineEnd": 909,
      "cohesionBenefiit": 3282.0,
      "methodOriginalCohesion": 3535.0,
      "linesOfCode": 162.0
    },
    {
      "id": 49,
      "projectName": "commons-io",
      "projectVersion": 1,
      "classPath": "commons-io/src/test/java/org/apache/commons/io/FilenameUtilsTest.java",
      "className": "FilenameUtilsTest.java",
      "methodName": "testNormalizeNoEndSeparator()",
      "lineStart": 999,
      "lineEnd": 1109,
      "cohesionBenefiit": 2388.0,
      "methodOriginalCohesion": 2898.0,
      "linesOfCode": 141.0
    }
  ],
  "paging": {
    "current": 0,
    "end": 4,
    "size": 2
  }
}
```

GET - /gea/run



Σχήμα 3.9: Παράδειγμα κλήσης Endpoint get - gea/run.

Βήμα 1: Καθορισμός λεπτομερειών αιτήματος

Καθόρισε τη διεύθυνση URL και χρησιμοποίησε τη μέθοδο GET για το endpoint gea/run της εφαρμογής. Αυτή η διεύθυνση URL έχει κατασκευαστεί ως εξής:

Το βασικό URL της εφαρμογής σας (π.χ. http://localhost:8080/ εάν εκτελείται τοπικά) πρόσθεσε "gea/run" στο base URL (π.χ. http://localhost:8080/gea/run).

Βήμα 2: Προσθήκη query params

Το τελικό σημείο /run, αναμένει δύο query params σε αντιθεση με το semi run:

I. url: Αυτή η παράμετρος πρέπει να περιέχει τη διεύθυνση URL του Git repository που θέλεις να αναλύσεις. (π.χ. https://github.com/apache/commons-io)

II. language: Καθόρισε τη γλώσσα προγραμματισμού για την ανάλυση. Ρύθμισε το στην γλώσσα του project σου (π.χ. "Java").

Βήμα 3: Αποστολή request και response

Εκτέλεσε το αίτημα χρησιμοποιώντας το εργαλείο HTTP που έχεις επιλέξει. Αυτή η ενέργεια στέλνει το αίτημα GET στο τελικό σημείο gea /run της εφαρμογής Spring Boot.

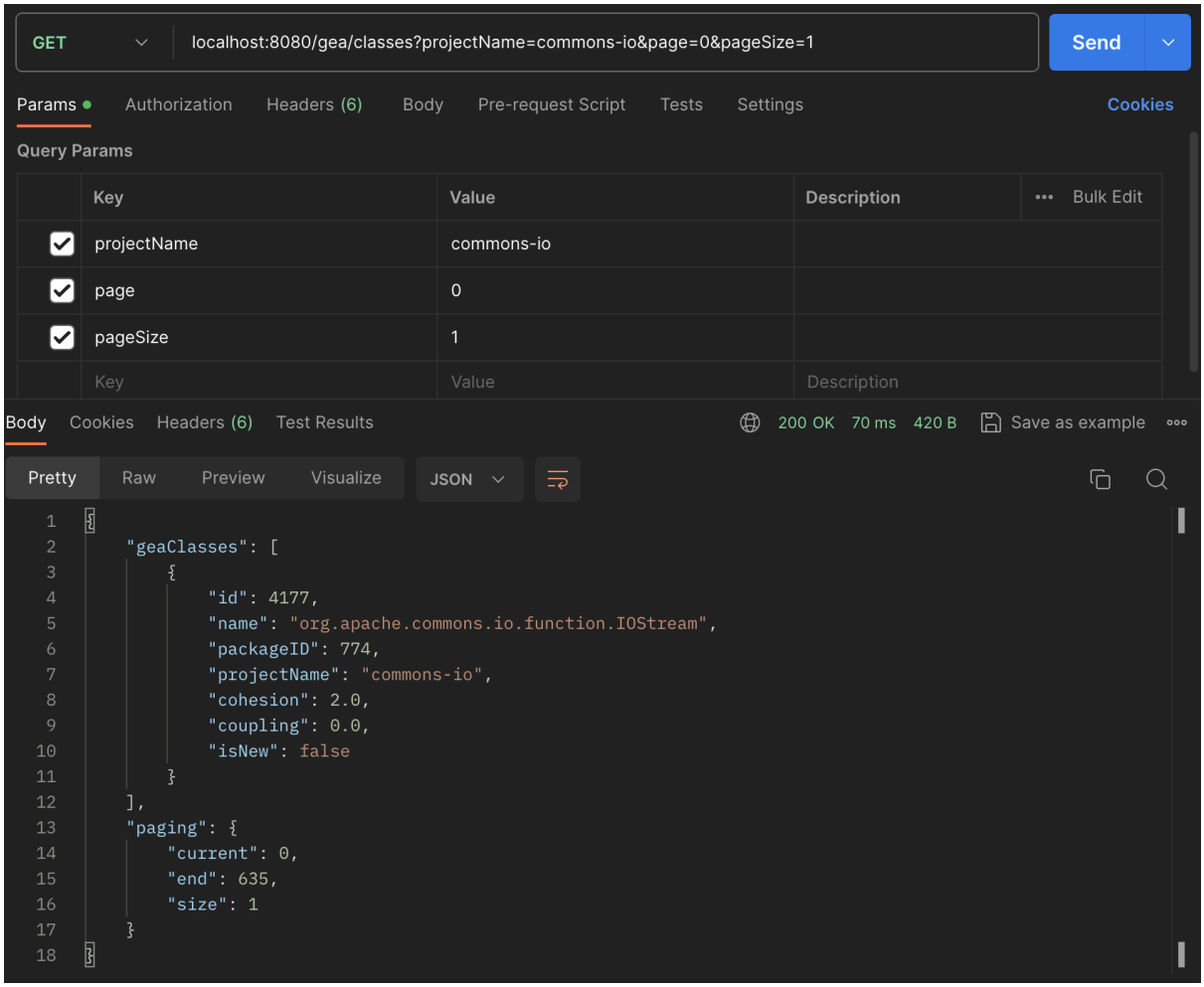
Μόλις το αίτημα υποβληθεί σε επεξεργασία από την εφαρμογή Spring Boot, θα ξεκινήσει την gea analysis για το project που του εδώσες και μόλις ολοκληρωθεί θα επιστρέψει μια απάντηση. Η απάντηση περιλαμβάνει κωδικό κατάστασης HTTP (π.χ. 200 OK για επιτυχία) και ένα μήνυμα στο body της απάντησης. Σε περίπτωση επιτυχής ανάλυσης γυρνάει HTTP CODE: 200 και μήνυμα στο body: "Gea analysis COMPLETED". Έλεγξε την απάντηση λοιπόν για να βεβαιωθείς ότι η ανάλυση ήταν επιτυχής.

Παράδειγμα response body:

```
{
  "message": "Gea analysis COMPLETED"
```

}

GET - gea/classes



Σχήμα 3.10: Παράδειγμα κλήσης Endpoint get - gea/classes.

Βήμα 1: Καθορισμός λεπτομερειών αιτήματος
Για το τελικό σημείο /classes, θα χρησιμοποίησε τη μέθοδο GET και για να κατασκευάσεις τη διεύθυνση URL χρησιμοποίησε την βασική διεύθυνση URL της εφαρμογής σου, όπως http://localhost:8080/ εάν εκτελείται τοπικά και πρόσθεσε "gea/classes" στο base URL για να σχηματίσεις την πλήρη διεύθυνση URL τελικού σημείου, για παράδειγμα: http://localhost:8080/gea/classes.

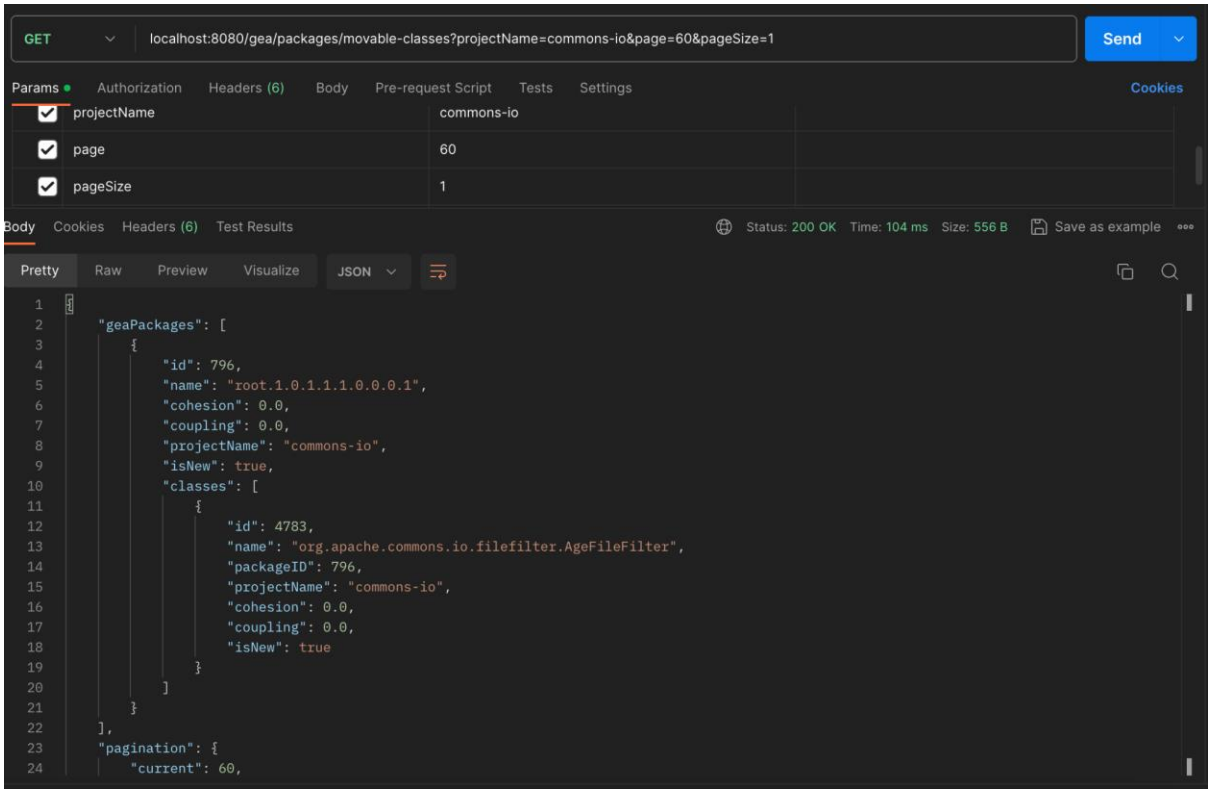
Βήμα 2: Προσθήκη query params
Το τελικό σημείο /classes αναμένει τρεις παραμέτρους ερωτήματος:

- projectName: Καθόρισε το όνομα του project για το οποίο θέλεις να ανακτήσεις τις GeaClasses. Όρισε αυτήν την παράμετρο στο όνομα του project σου.
- page: Δώσε τον αριθμό σελίδας που θέλεις να αποκτήσεις πρόσβαση στα αποτελέσματα ξεκινώντας από το 0.
- pageSize: Καθόρισε τον αριθμό των αποτελεσμάτων που θέλεις να ανακτας ανά σελίδα.

Βήμα 3: Αποστολή request και response
Εκτέλεσε το αίτημα χρησιμοποιώντας το εργαλείο HTTP που έχεις επιλέξει. Αυτή η ενέργεια στέλνει το αίτημα GET στο endpoint gea/classes της εφαρμογής Spring Boot. Αφού η εφαρμογή Spring Boot επεξεργαστεί το αίτημά σου, θα ανακτήσει από την βάση δεδομένων τα GeaClasses για το project σου και θα επιστρέψει μια απάντηση. Η απάντηση περιλαμβάνει έναν κωδικό κατάστασης HTTP (π.χ. 200 OK για επιτυχία) και ένα μήνυμα στο σώμα απάντησης. Σε αυτήν την περίπτωση, θα λάβεις μια λίστα με τις GeaClasses που σχετίζονται με το καθορισμένο Project και πληροφορίες για την σελιδοποίηση των αποτελεσμάτων.

Παράδειγμα response body:


```
{
   "geaClasses": [
      {
         "id": 4177,
         "name": "org.apache.commons.io.function.IOStream",
         "packageID": 774,
         "projectName": "commons-io",
         "cohesion": 2.0,
         "coupling": 0.0,
         "isNew": false
      },
      {
         "id": 4178,
         "name": "org.apache.commons.io.input.TaggedInputStream",
         "packageID": 770,
         "projectName": "commons-io",
         "cohesion": 0.0,
         "coupling": 0.0,
         "isNew": false
      }
   ],
   "paging": {
      "current": 0,
      "end": 317,
      "size": 2
   }
}
```



Σχήμα 3.11: Παράδειγμα κλήσης Endpoint get - gea/packages/movable-classes.

Βήμα 1: Καθορισμός λεπτομερειών αιτήματος
Για το τελικό σημείο /packages/movable-classes, θα επιλέξεις τον τύπο αιτήματος GET και θα καθορίσεις τη διεύθυνση URL για το endpoint /packages/movable-classes χρησιμοποιώντας την βασική διεύθυνση URL της εφαρμογής και προσθέτοντας την κατάληξη του endpoint gea/packages/movable-classes (π.χ. http://localhost:8080/gea/packages/movable-classes εάν εκτελείται τοπικά).

Βήμα 2: Προσθήκη query params
Το τελικό σημείο /packages/movable-classes αναμένει τρια query params:

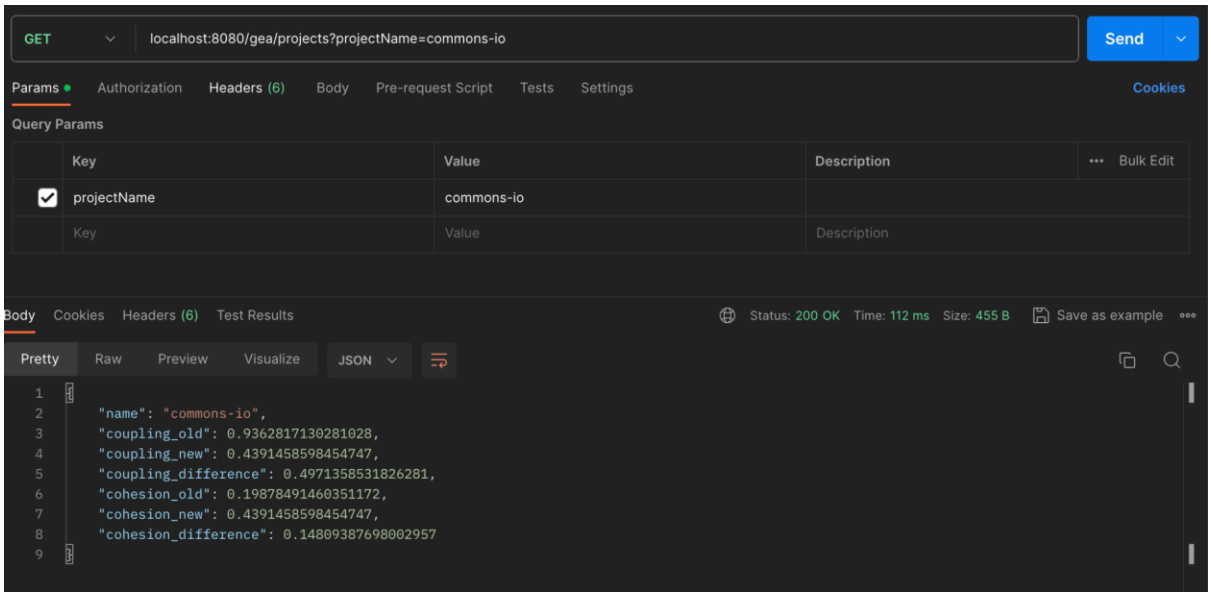
- projectName: Καθόρισε το όνομα του έργου για το οποίο θέλεις να ανακτήσεις πακέτα Gea με κινητές κλάσεις. Όρισε αυτήν την παράμετρο στο όνομα του project σου.
- page: Υπόδειξε τον αριθμό σελίδας που θέλεις να αποκτήσεις πρόσβαση στα αποτελέσματα, ξεκινώντας από το 0.
- pageSize: Καθόρισε τον αριθμό των αποτελεσμάτων που θέλεις να ανακτήσεις ανά σελίδα.

Βήμα 3: Αποστολή request και response
Εκτέλεσε το αίτημα χρησιμοποιώντας το εργαλείο HTTP που έχεις επιλέξει. Αφού η εφαρμογή Spring Boot επεξεργαστεί το αίτημα θα βρει τα αποθηκευμενα GeaPackages στην βάση δεδομένων και έπειτα θα ανακτήσει τις GeaClasses για καθε ένα και θα επιστρέψει μια απάντηση. Η απάντηση περιλαμβάνει έναν κωδικό κατάστασης HTTP (π.χ. 200 OK για επιτυχία), στο response body θα λάβετε μια λίστα με GeaPackages και της GeaClasses τους για το project που δόθηκε και πληροφορίες για την σελιδοποίηση των αποτελεσμάτων.

Παράδειγμα response body:


```
{
   "geaPackages": [
      {
         "id": 796,
         "name": "root.1.0.1.1.1.0.0.0.1",
         "cohesion": 0.0,
         "coupling": 0.0,
         "projectName": "commons-io",
         "isNew": true,
         "classes": [
            {
               "id": 4783,
               "name": "org.apache.commons.io.filefilter.AgeFileFilter",
               "packageID": 796,
               "projectName": "commons-io",
               "cohesion": 0.0,
               "coupling": 0.0,
               "isNew": true
            }
         ]
      }
   ],
   "pagination": {
      "current": 60,
      "end": 99,
      "size": 1
   }
}
```

GET - gea/projects



Σχήμα 3.12: Παράδειγμα κλήσης Endpoint get - gea/projects.

Βήμα 1: Καθορισμός λεπτομερειών αιτήματος
Για το τελικό σημείο gea/projects θα επιλέξεις τον τύπο αιτήματος GET και θα θέσεις τη διεύθυνση URL για το endpoint gea/projects χρησιμοποιώντας την βασική διεύθυνση URL της εφαρμογής και προσθέτοντας την κατάληξη του endpoint gea/packages/movable-classes (π.χ. http://localhost:8080/gea/projects εάν εκτελείται τοπικά).

Βήμα 2: Προσθήκη query params
Το τελικό σημείο gea/projects αναμένει ενα query param:

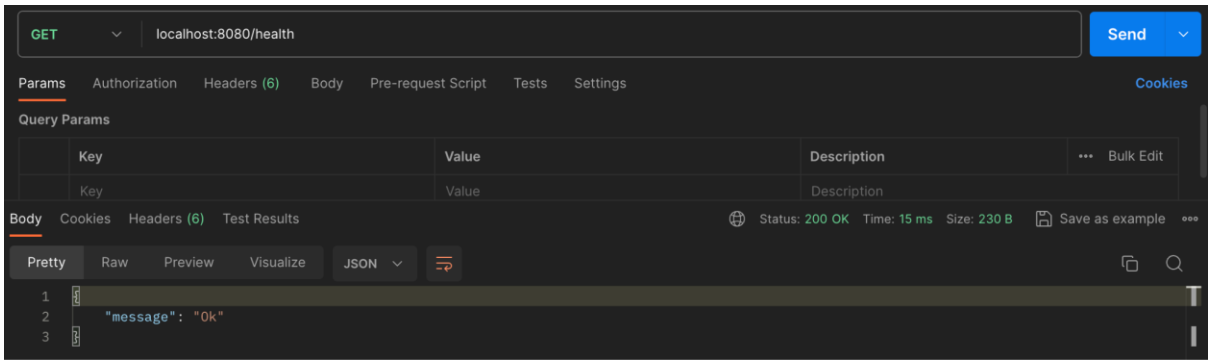
- projectName: Καθορίζει το όνομα του project για το οποίο θέλεις να ανακτήσεις λεπτομέρειες. Όρισε αυτήν την παράμετρο στο όνομα του project σου.

.
Βήμα 3: Αποστολή request και response
Εκτέλεσε το αίτημα χρησιμοποιώντας το εργαλείο HTTP που έχεις επιλέξει. Αφού η εφαρμογή Spring Boot επεξεργαστεί το αίτημα θα βρει το GeaProject στην βάση δεδομένων και αφού ανακτήσει τις πληροφορίες θα σου γυρίσει μια απάντηση. Η απάντηση περιλαμβάνει έναν κωδικό κατάστασης HTTP και ενα response body στο οποίο θα περιλαμβάνονται όλες οι πληροφορίες για το project σου με τιμές πριν και μετά την ανάλυση.

Παράδειγμα response body:


```
{
  "name": "commons-io",
  "coupling_old": 0.9362817130281028,
  "coupling_new": 0.4391458598454747,
  "coupling_difference": 0.4971358531826281,
  "cohesion_old": 0.19878491460351172,
  "cohesion_new": 0.4391458598454747,
  "cohesion_difference": 0.14809387698002957
}
```

Έλεγχος υγείας API



Σχήμα 3.13: Παράδειγμα κλήσης Endpoint get - /health.

Ο σκοπός του endpoint health είναι να παρέχει ένα απλό και αποτελεσματικό μέσο για τον προσδιορισμό του εάν η εφαρμογή είναι online.
Για να ελέγξεις την υγεία της εφαρμογής θα χρειαστεί να υποβάλλεις ένα αίτημα HTTP GET σε ένα από τα παρακάτω endpoints:

- /health
- /healthy
- /healthz
- / (Χρησημοποιώντας μόνο το base url π.χ. localhost:8080 αν το τρέχεις τοπικά)

Η απάντηση περιλαμβάνει έναν κωδικό κατάστασης HTTP, όπως 200 OK για μια υγιή κατάσταση, και ένα μήνυμα στο response body, το οποίο απλώς αναφέρει "Ok" για να επιβεβαιώσει την καλή κατάσταση της εφαρμογής.
Παράδειγμα response body:

```
{
   "message": "Ok"
}
```

Κεφάλαιο 4ο: Συμπεράσματα

Συμπερασματικά, η διαχείριση και αντιμετώπιση του τεχνικού χρέους αποτελούν κρίσιμες προϋποθέσεις για τη μακροχρόνια πρόοδο της ανάπτυξης λογισμικού. Όμοια με το οικονομικό χρέος, το τεχνικό χρέος προκαλεί σύσσωμο επιτόκιο, οδηγώντας στη συσσώρευση προκλήσεων που μπορούν να επηρεάσουν την αποτελεσματικότητα, την προσαρμοστικότητα και τη συνολική υγεία ενός συστήματος λογισμικού με τον χρόνο.

Οι στρατηγικές για τη διαχείριση του τεχνικού χρέους περιλαμβάνουν πολυδιάστατη προσέγγιση που περιλαμβάνει τακτική συντήρηση, στρατηγικό refactoring και συστηματικό Documentation. Με την προτεραιότητα αυτών των δραστηριοτήτων, οι ομάδες ανάπτυξης μπορούν να αποτρέψουν την διόγκωση του επιτοκίου και να διατηρήσουν έναν πιο ευέλικτο και ανθεκτικό κώδικα. Αυτή η προσέγγιση ενισχύει όχι μόνο τη μακροχρόνια βιωσιμότητα των έργων λογισμικού, αλλά θεμελιώνει και μια βάση για διαρκή καινοτομία και προσαρμοστικότητα στις εξελισσόμενες απαιτήσεις.
Είναι κρίσιμο για τις οργανώσεις να καλλιεργούν μια κουλτούρα συνεχούς βελτίωσης, όπου η αναγνώριση και διαχείριση του τεχνικού χρέους ενσωματώνεται στον κύκλο ζωής της ανάπτυξης. Με την υιοθέτηση αυτής τής νοοτροπίας, οι ομάδες μπορούν να βρουν ένα σημείο τομής μεταξύ της γρήγορης παράδοσης feature και της συνεχούς συντήρησης ενός υγιούς, βιώσιμου κώδικα. Η αναγνώριση και η υπεύθυνη διαχείριση του τεχνικού χρέους γίνονται, επομένως, ουσιώδεις συνιστώσες στην προσπάθεια για αριστεία στην ανάπτυξη λογισμικού.

Η διαδικασία του refactoring αποτελεί θεμέλιο λίθο στην προσπάθεια αντιμετώπισης του τεχνικού χρέους και της διατήρησης ενός υγιούς κώδικα. Στον κόσμο της ανάπτυξης λογισμικού, το refactoring αναφέρεται στη διαδικασία βελτίωσης της δομής του κώδικα χωρίς να αλλάζει τη συμπεριφορά του προγράμματος. Κατά τη διάρκεια του refactoring, ο κώδικας υποβάλλεται σε αλλαγές που βελτιώνουν την αναγνωσιμότητα, τη συντηρησιμότητα, και την αποτελεσματικότητα, χωρίς να επηρεάζουν την εξωτερική συμπεριφορά του συστήματος.

Το refactoring δεν είναι απλώς μια τεχνική, αλλά μια φιλοσοφία ανάπτυξης που ενισχύει την ευελιξία του κώδικα και προάγει την αειφορία του συστήματος. Με την εφαρμογή του οι προγραμματιστές μπορούν να εντοπίζουν και να απομακρύνουν περιττή πολυπλοκότητα, να βελτιώσουν τη δομή του κώδικα, και να δημιουργήσουν μια βάση για μελλοντικές εξελίξεις. Κατά το refactoring, η επανεξέταση των σχεδιαστικών επιλογών και η επαναφορά σε βέλτιστες πρακτικές αποτελούν ουσιαστικά βήματα προς την καλύτερη ποιότητα του κώδικα.
Το refactoring συμβάλλει στη μείωση του τεχνικού χρέους, καθώς επιτρέπει στις ομάδες ανάπτυξης λογισμικού να αντιμετωπίζουν συστηματικά θέματα και να διορθώνουν προβλήματα πριν επιδεινωθούν. Με τη συχνή εφαρμογή του refactoring, η εξέλιξη του κώδικα γίνεται πιο ευέλικτη, η συντηρησιμότητα βελτιώνεται, και οι ομάδες ανάπτυξης αποκτούν τη δυνατότητα να προσαρμόζονται γρήγορα σε νέες απαιτήσεις και προκλήσεις.

Επιπλέον, η υιοθέτηση των αρχών της Ανάπτυξης με Προσανατολισμό στις Υπηρεσίες ενισχύει την αποτελεσματικότητα της αντιμετώπισης του τεχνικού χρέους και συμβάλλει στη συνολική ανθεκτικότητα των λογισμικών συστημάτων. Η Ανάπτυξη με Προσανατολισμό στις Υπηρεσίες τονίζει την ανάλυση των συστημάτων σε ανεξάρτητες υπηρεσίες, που επικοινωνούν μεταξύ τους μέσω καλά καθορισμένων διεπαφών. Κάθε υπηρεσία αντιπροσωπεύει ξεχωριστό business logic και

λειτουργεί ανεξάρτητα, προωθώντας την ενότητα, την κλιμάκωση και τη συντηρησιμότητα εντός του συνολικού συστήματος.

Βιβλιογραφία

Άρθρα

[1] Li, Zengyang & Avgeriou, Paris & Liang, Peng. (2014). A Systematic Mapping Study on Technical Debt and Its Management. Journal of Systems and Software.

[2] Nicolli S.R. Alves, Thiago S. Mendes, Manoel G. de Mendonça, Rodrigo O. Spínola, Forrest Shull, Carolyn Seaman,Identification and management of technical debt: A systematic mapping study

[3] McConnell, S. (2008). Managing technical debt. Construx Software Builders, Inc, 1-14.

[4] Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Paris Avgeriou,
The financial aspect of managing technical debt: A systematic literature review

[5] Rios, Nicolli & Ribeiro, Leilane & Caires, Vivyane & Mendes, Thiago & Spínola, Rodrigo. (2014). Towards an Ontology of Terms on Technical Debt. Proceedings - 2014 6th IEEE International Workshop on Managing Technical Debt, MTD 2014. 1-7. 10.1109/MTD.2014.9.

[11] Theodoros Maikantis, Angeliki-Agathi Tsintzira, Apostolos Ampatzoglou, Elvira-Maria Arvanitou, Alexander Chatzigeorgiou, Ioannis Stamelos, Stamatia Bibi, and Ignatios Deligiannis. 2021. Software Architecture Reconstruction via a Genetic Algorithm: Applying the Move Class Refactoring. In Proceedings of the 24th Pan-Hellenic Conference on Informatics (PCI '20). Association for Computing Machinery, New York, NY, USA, 135–139.

[12] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, A. Gkortzis and P. Avgeriou, "Identifying Extract Method Refactoring Opportunities Based on Functional Relevance," in IEEE Transactions on Software Engineering, vol. 43, no. 10, pp. 954-974, 1 Oct. 2017, doi: 10.1109/TSE.2016.2645572.

Internet Sites

[8] IBM. (n.d.). What is java? IBM. https://www.ibm.com/topics/java

[9] What is Java spring boot? IBM. https://www.ibm.com/topics/java-spring-boot

[10] What is Hibernate? TheServerSide.com. https://www.theserverside.com/definition/Hibernate

[11] Refactoring.Guru – Refactoring and design patterns. https://refactoring.guru

[13] HTTP requests. https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests

[14] What is postman? Postman API Platform. https://www.postman.com/product/what-is-postman/

Βιβλία

[6] Fowler, M. (1999). Refactoring: Improving the Design of Existing Code. Boston, MA, USA: Addison-Wesley. ISBN: 0-201-48567-2

[7] Bieber, G. (2001). Introduction to Service-Oriented Programming (Rev 2 . 1) by Guy Bieber , Lead Architect , Motorola ISD Jeff Carpenter , Software Engineer , Motorola ISD.