ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΘΟΔΟΛΟΓΙΑ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΚΑΙ ΤΗ ΣΥΝΘΕΣΗ ΥΠΗΡΕΣΙΩΝ ΛΟΓΙΣΜΙΚΟΥ ΣΕ ΕΦΑΡΜΟΓΕΣ ΥΠΟΛΟΓΙΣΤΙΚΟΥ ΝΕΦΟΥΣ

Διπλωματική Εργασία

του

Θεόδωρου Μαϊκαντή

Θεσσαλονίκη, Φεβρουάριος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Θεόδωρος Μαϊκαντής

Μηχανικός Πληροφορικής, ΔΙΠΑΕ, 2020

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Απόστολος Αμπατζόγλου

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την __/__/____

Απόστολος Αμπατζόγλου	Αλέξανδρος Χατζηγεωργίου	Χρήστος Γεωργιάδης
.....	.....	.....

Μαϊκαντής Θεόδωρος

.....

Περίληψη

Στης μέρες μας, η πλειονότητα των εφαρμογών νέφους αναπτύσσονται με βάση το πρότυπο αρχιτεκτονικής προσανατολισμένη στην χρήση μικροϋπηρεσιών (Service-Oriented Architecture - SOA). Οι εφαρμογές μεγάλης κλίμακας είναι δομημένες ως μια συλλογή από καλά ενσωματωμένες υπηρεσίες που διατίθενται σε δημόσιο, ιδιωτικό ή υβριδικό νέφος. Παρά τα εγγενή οφέλη που παρέχει η βασισμένη σε υπηρεσίες ανάπτυξη εφαρμογών νέφους , η διαδικασία δεν είναι καθόλου απλή, με την έννοια ότι απαιτεί από τον μηχανικό λογισμικού να είναι εξοικειωμένος με τη χρήση διαφόρων τεχνολογιών ενός μεγάλου συνόλου εργαλείων: προγραμματισμός σε διάφορες γλώσσες, εργαλεία δοκιμών, εργαλεία CI/CD, αποθετήρια Git, μηχανισμοί διάθεσης λογισμικών, κτλ. Σε αυτή την εργασία προτείνουμε μια προσέγγιση που συνοδεύεται από αντίστοιχη πλατφόρμα (που ονομάζεται SmartCLIDE[1]—ως μέρος των αποτελεσμάτων ενός ερευνητικού έργου που χρηματοδοτείται από την ΕΕ) για τη διευκόλυνση δημιουργίας μιας εφαρμογής νέφους SOA επεκτείνοντας ένα γνωστό εργαλείο ανάπτυξης κώδικα της Eclipse. Η προσέγγιση στοχεύει στη συντόμευση της αλυσίδας εργαλείων που απαιτούνται για την ανάπτυξη εφαρμογών νέφους , στην απόκρυψη της πολυπλοκότητας της διαδικασίας και στη μείωση του απαιτούμενου επιπέδου γνώσης από τους μηχανικούς λογισμικού. Η προσέγγιση και η εργαλειοθήκη που αναπτύχθηκε υποβλήθηκαν σε αρχική δοκιμή από επαγγελματίες προγραμματιστές στον τομέα ανάπτυξης SOA εφαρμογών. Τα αποτελέσματα αναδεικνύουν τις δυνατότητες μιας τέτοιας προσέγγισης αυτοματισμού, καθώς και τη χρηστικότητα του πρωτότυπου

1

SmartCLIDE website: https://smartclide.eu/

Service Creation service: eclipse-researchlabs/smartclide-service-creation (github.com)

TD Interest service:eclipse-researchlabs/smartclide-TD-Interest (github.com)

TD Reusability Index service: eclipse-researchlabs/smartclide-TD-Reusability-Index (github.com)

Service Creation Theia extension: eclipse-researchlabs/smartclide-service-creation-theia (github.com)

TD Interest & Reusability Theia extension: eclipse-researchlabs/smartclide-TD-Theia

Service Discovery: smartclide-service-discovery-poc (github.com)

SmartCLIDE CI/CD: eclipse-researchlabs/smartclide-cicd (github.com)

Service Deployment: eclipse-researchlabs/smartclide-deployment-service (github.com)

SmartCLIDE-CI/CD Gitlab: eclipse-researchlabs/smartclide-cicd-gitlab (github.com)

λογισμικού, ανοίγοντας περαιτέρω ερευνητικές ευκαιρίες και παρέχοντας οφέλη για τους επαγγελματίες. Από τη συγκεκριμένη εργασία προέκυψαν δύο δημοσιεύσεις σε συνέδρια πληροφορικής (PCI2021 και ICSR22). Μία δημοσίευση αναφορικά με την συντόμευση και την διευκόλυνση της διαδικασίας δημιουργίας εφαρμογών νέφους[2] και μία που ερευνά την κατάταξη υπηρεσιών χρησιμοποιώντας μεθόδους μηχανική μάθησης[3].

Λέξεις Κλειδιά: Service-Oriented Architecture, Ανάπτυξη Εφαρμογών Νέφους, Αυτοματοποίηση Τεχνολογίας Λογισμικού

[2] Maikantis, Theodore & Chaikalis, Theodore & Ampatzoglou, Apostolos & Chatzigeorgiou, Alexander. (2021). SmartCLIDE: Shortening the Toolchain of SOA-based Cloud Software Development by Automating Service Creation, Composition, Testing, and Deployment.

[3] Zakieh Alizadehsani, Daniel Feitosa, Theodoros Maikantis, Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, David Berrocal, Alfonso González Briones, Juan M. Corchado, Marcio Mateus and Johannes Groenewold (2022), Service Classification through Machine Learning: An Industrial Case Study.

Abstract

Nowadays the majority of cloud applications are developed based on the Service-Oriented Architecture (SOA) paradigm. Large-scale applications are structured as a collection of well-integrated services that are deployed in public, private or hybrid cloud. Despite the inherent benefits that service-based cloud development provides, the process is far from trivial, in the sense that it requires the software engineer to be (at least) comfortable with the use of various technologies in the long cloud development toolchain: programming in various languages, testing tools, build / CI tools, repositories, deployment mechanisms, etc. In this thesis, we propose an approach and corresponding toolkit (termed SmartCLIDE[4]—as part of the results of an EU-funded research project) for facilitating SOA-based software development for the cloud, by extending a well-known cloud IDE from Eclipse. The approach aims at shortening the toolchain for cloud development, hiding the process complexity and lowering the required level of knowledge from software engineers. The approach and tool underwent an initial validation from professional cloud software developers. The results underline the potential of such an automation approach, as well as the usability of the research prototype, opening further research opportunities and providing benefits for practitioners. We produced two publications from this thesis, that were accepted in two different computer science conventions (PCI2021 and ICSR22). One publication is on the

[4]

SmartCLIDE website: https://smartclide.eu/

Service Creation service: eclipse-researchlabs/smartclide-service-creation (github.com)

TD Interest service:eclipse-researchlabs/smartclide-TD-Interest (github.com)

TD Reusability Index service: eclipse-researchlabs/smartclide-TD-Reusability-Index (github.com)

Service Creation Theia extension: eclipse-researchlabs/smartclide-service-creation-theia (github.com)

TD Interest & Reusability Theia extension: eclipse-researchlabs/smartclide-TD-Theia

Service Discovery: smartclide-service-discovery-poc (github.com)

SmartCLIDE CI/CD: eclipse-researchlabs/smartclide-cicd (github.com)

Service Deployment: eclipse-researchlabs/smartclide-deployment-service (github.com)

SmartCLIDE-CI/CD Gitlab: eclipse-researchlabs/smartclide-cicd-gitlab (github.com)

Shortening the Required Toolchain of SOA-based Cloud Software Development[5] and another one on Service Classification using Machine Learning techniques[6].

Keywords: Service-Oriented Architecture, Cloud Software Development, Automated Software Engineering

[5] Maikantis, Theodore & Chaikalis, Theodore & Ampatzoglou, Apostolos & Chatzigeorgiou, Alexander. (2021). SmartCLIDE: Shortening the Toolchain of SOA-based Cloud Software Development by Automating Service Creation, Composition, Testing, and Deployment.

[6] Zakieh Alizadehsani, Daniel Feitosa, Theodoros Maikantis, Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, David Berrocal, Alfonso González Briones, Juan M. Corchado, Marcio Mateus and Johannes Groenewold (2022), Service Classification through Machine Learning: An Industrial Case Study

Περιεχόμενα

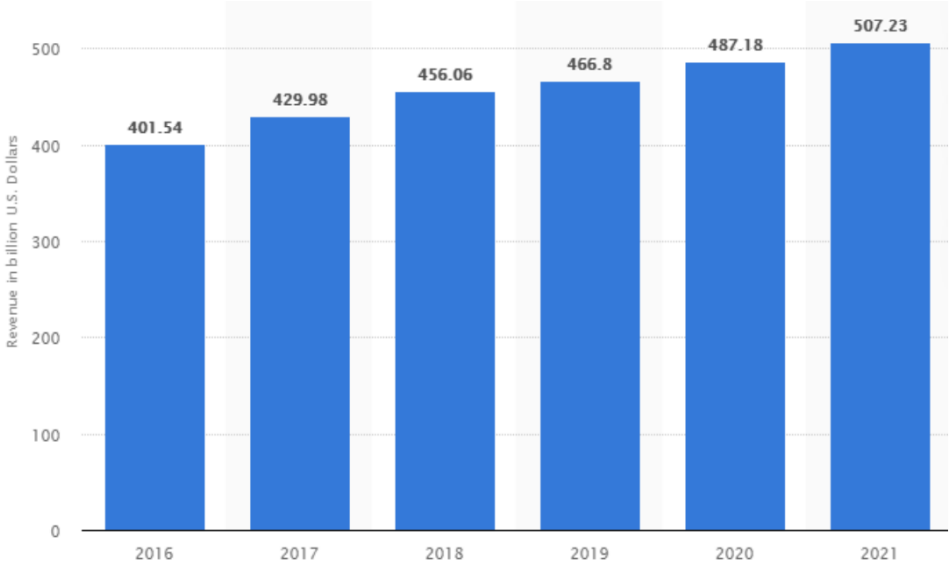
Κατάλογος Εικόνων

Κατάλογος Πινάκων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Η ανάπτυξη λογισμικού στις μέρες μας, είναι ένα από τα πιο σημαντικά και ταχύτατα οικονομικά αναπτυσσόμενα πεδία. Τις τελευταίες δεκαετίες, το γενικότερο πεδίο της πληροφορικής έχει δείξει ιδιαίτερα μεγάλη οικονομική άνθιση. Η άνθιση που παρατηρείται, δεν έχει εμφανίσει σημάδια επιβράδυνσης με το πέρασμα του χρόνου, αντιθέτως φαίνεται να αυξάνεται. Ένας από τους κύριους λόγους που οδηγούν σε αυτό το αποτέλεσμα, είναι η ενσωμάτωση διαφόρων μορφών λογισμικού σε όλες τις πτυχές τις καθημερινότητας μας, όπως για παράδειγμα στα μέσα μεταφοράς, τις πλατφόρμες επικοινωνίας, τις δημόσιες υπηρεσίες, τους τρόπους εργασίας μας, κτλ. Τα παραπάνω έχουν ως αποτέλεσμα, την ολοένα και μεγαλύτερη αύξηση της ζήτησης λογισμικού.



Εικόνα 1: Ανάπτυξη Βιομηχανίας Λογισμικού [1]

Δεδομένης της υψηλής ζήτησης λογισμικού, οι εταιρίες ανάπτυξης που το αναπτύσσουν καλούνται να παράγουν και να διαθέτουν τα προϊόντα τους, σε όλο και πιο απαιτητικά χρονικά περιθώρια.

Αξίζει επίσης να σημειωθεί πως τα τελευταία χρόνια παρατηρείται η μεταστροφή του λογισμικού, από την παραδοσιακή του μορφή σε αυτήν των εφαρμογών νέφους. Πολλές από τις υπηρεσίες που χρησιμοποιούμε στην καθημερινότητά μας, οδηγούνται

προς την ψηφιοποίηση. Αυτό συμβαίνει κατά κύριο λόγο προς όφελος του τελικού χρήστη, έτσι ώστε να βελτιωθεί η ευχρηστία και η προσβασιμότητα του σε υπηρεσίες. Όταν μία εφαρμογή λειτουργεί στο νέφος, είναι διαθέσιμη συνεχώς και προσβάσιμη από παντού. Ένα καλό παράδειγμα αυτής της μεταστροφής, είναι η ψηφιοποίηση διάφορων κρατικών υπηρεσιών, όπως έχει συμβεί τελευταία χρόνια και στην Ελλάδα.

Η διαδικασία ανάπτυξης μίας σύνθετης εφαρμογής νέφους είναι πιο περίπλοκη και χρονοβόρα από αυτήν μιας τοπικής παραδοσιακής εφαρμογής. Για την ανάπτυξη της απαιτούνται πολλά ενδιάμεσα βήματα κατά τα οποία είναι απαραίτητη η χρήση πολλών νέων και σύνθετων τεχνολογιών και εργαλείων.

Καθώς οι τεχνολογίες και οι πρακτικές ανάπτυξης είναι σχετικά καινούργιες δεν υπάρχει χαρτογραφημένη διαδικασία που μπορεί να ακολουθηθεί, επομένως η απόφαση για την επιλογή και την υιοθέτηση τεχνολογιών και εργαλείων επαφίεται στον εκάστοτε προγραμματιστή. Ως αποτέλεσμα των παραπάνω, η υψηλή ζήτηση σε συνδυασμό με την πολυπλοκότητα της ανάπτυξης εφαρμογών νέφους, απαιτούν την εύρεση νέας πιο βελτιωμένης και αποδοτικής διαδικασίας ανάπτυξης.

1.2 Σκοπός – Στόχοι

Στην παρούσα εργασία προτείνουμε μια προσέγγιση συνοδευόμενη από την αντίστοιχη εργαλειοθήκη (που ονομάζεται SmartCLIDE - ως μέρος των αποτελεσμάτων ενός ερευνητικού έργου που χρηματοδοτείται από την ΕΕ) για τη διευκόλυνση της ανάπτυξης λογισμικού που βασίζεται στην αρχιτεκτονική με επίκεντρο την χρήση υπηρεσιών (Service-oriented architecture - SOA), επεκτείνοντας ένα πολύ γνωστό εργαλείο συγγραφής κώδικα (IDE) από την Eclipse. Η προσέγγιση στοχεύει στη συντόμευση της αλυσίδας εργαλείων που απαιτούνται για την ανάπτυξη εφαρμογών νέφους, στην απόκρυψη της πολυπλοκότητας της διαδικασίας και στη μείωση του απαιτούμενου επιπέδου γνώσης από τους μηχανικούς λογισμικού. Η προσέγγιση και η εργαλειοθήκη που αναπτύχθηκε υποβλήθηκαν σε αρχική δοκιμή από επαγγελματίες προγραμματιστές στον τομέα ανάπτυξης SOA εφαρμογών. Τα αποτελέσματα αναδεικνύουν τις δυνατότητες μιας τέτοιας προσέγγισης αυτοματισμού, καθώς και τη χρηστικότητα του πρωτότυπου λογισμικού, ανοίγοντας περαιτέρω ερευνητικές ευκαιρίες και παρέχοντας οφέλη για τους επαγγελματίες.

1.3 Διάρθρωση της μελέτης

Στο δεύτερο κεφάλαιο, πραγματοποιείται μια βιβλιογραφική επισκόπηση και παρουσιάζεται το θεωρητικό υπόβαθρο της έρευνας. Σε αυτό το σημρίο, αναλύονται τα βασικά βήματα, τεχνολογίες και εργαλεία που απαιτούνται για την ανάπτυξη εφαρμογών νέφους, καθώς και οι μέθοδοι ανάπτυξης που πρέπει να ακολουθηθούν.

Στο τρίτο κεφάλαιο ορίζεται η μεθοδολογία που επιλέξαμε να ακολουθήσουμε, καθώς και τα εργαλεία και οι τεχνολογίες που υιοθετήθηκαν για την χρήση στην πλατφόρμα του SmartCLIDE.

Στο τέταρτο κεφάλαιο, παρουσιάζεται η προσέγγιση μας για την αυτοματοποίηση της διαδικασία υαλοποίησης και αναλύονται τα δομικά κομμάτια της εργαλειοθήκης που αναπτύχθηκε. Επίσης παρουσιάζεται το γραφικό περιβάλλον της πλατφόρμας του SmartCLIDE, καθώς και τα βασικά κομμάτια από τα οποία αποτελείται.

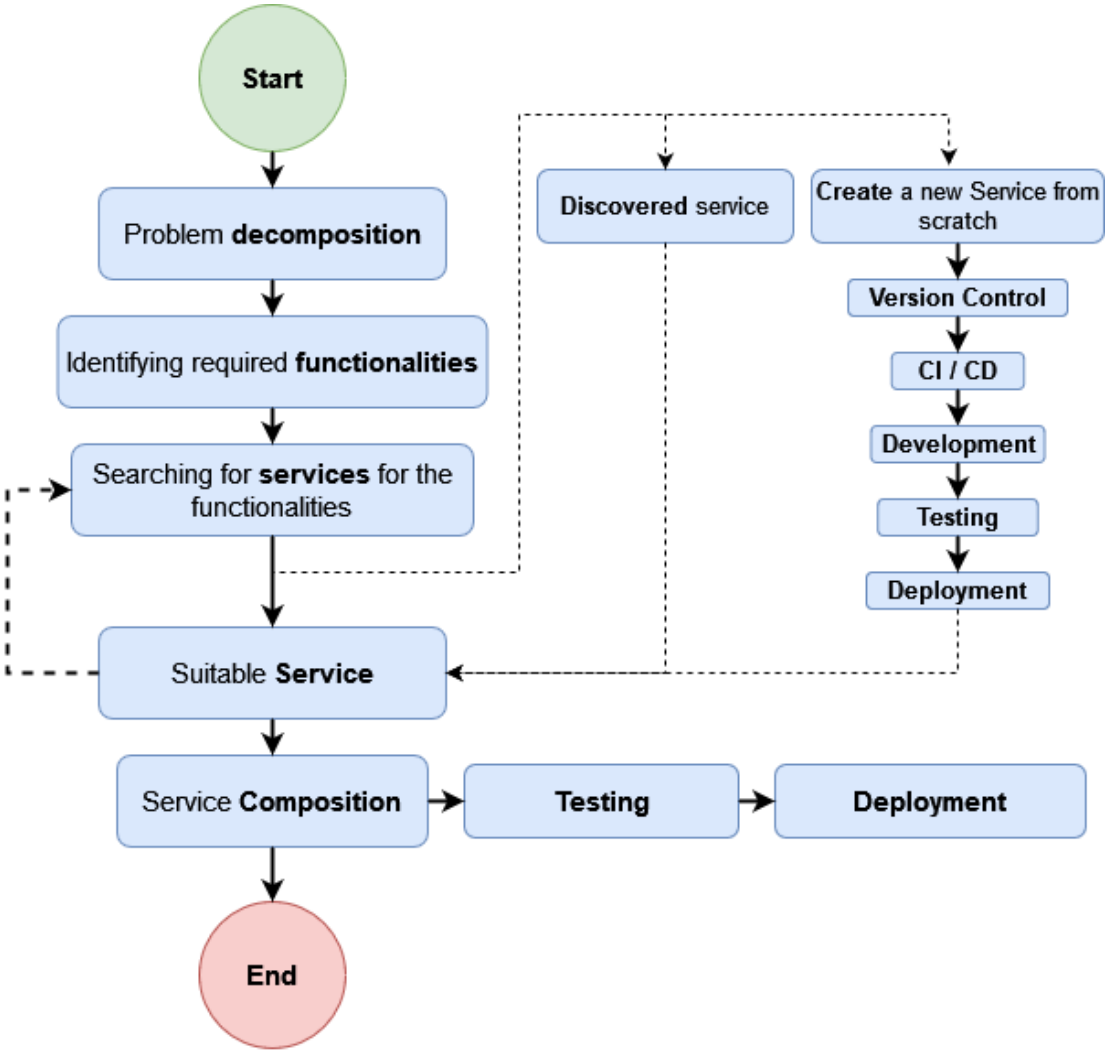
Στο πέμπτο κεφάλαιο, πραγματοποιείται εμπειρική μελέτη για την αξιολόγηση της αποτελεσματικότητας της προσέγγισης που έχουμε προτείνει και παρουσιάζονται τα αποτελέσματα της αρχικής δοκιμής του πρωτότυπου εργαλείου.

2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

Η διαδικασία ανάπτυξης σύνθετων εφαρμογών νέφους είναι μία αρκετά πολύπλοκη και χρονοβόρα διαδικασία στην οποία απαιτείται η γνώση και η χρήση πολλών διαφορετικών εργαλείων και τεχνολογιών.

Η αρχιτεκτονική που είναι βασισμένη στην χρήση μικροϋπηρεσιών (SOA) στοχεύει στο να επιτρέψει τη γρήγορη και εύκολη δημιουργία εφαρμογών λογισμικού που είναι προσβάσιμες μέσω δικτύου [2]. Για το σκοπό αυτό, μια εφαρμογή που χρησιμοποιεί SOA αποτελείται από πολλές μικροϋπηρεσίες, καθεμία από τις οποίες ικανοποιεί μια συγκεκριμένη λειτουργικότητα [3]. Δεδομένου ότι κάθε υπηρεσία αντιπροσωπεύει έναν ενιαίο σκοπό, η σύνθεση μπορεί να θεωρηθεί ως μια ακολουθία βημάτων προς την ολοκλήρωση ενός γενικού στόχου [4]. Στο πλαίσιο του SOA, οι μικροϋπηρεσίες θεωρούνται ως ανεξάρτητα στοιχεία που είναι πλήρως επαναχρησιμοποιήσιμα μέσω μιας διεπαφής.

Προκειμένου οι προγραμματιστές να αναπτύξουν εφαρμογές νέφους που βασίζονται στην χρήση μικροϋπηρεσιών με αποτελεσματικό και παραγωγικό τρόπο, αναμένεται να έχουν ένα ευρύ φάσμα δεξιοτήτων που είναι απαραίτητο για την χρήση διαφόρων εργαλείων και τεχνολογιών. Ειδικότερα, για να υλοποιήσει ένα σύστημα υψηλής ποιότητας, ο μηχανικός λογισμικού πρέπει να διαθέτει γνώσεις και δεξιότητες σχετικές με τον τρόπο διαμόρφωσης και χρήσης πολλών τεχνολογιών, πολλές από τις οποίες είναι άγνωστες και απαιτητικές για έναν αρχάριο προγραμματιστή. Κάθε διαφορετική τεχνολογία, πολλές φορές εξυπηρετεί έναν μεμονωμένο σκοπό της διαδικασίας ανάπτυξης, ωστόσο όμως απαιτεί αρκετό χρόνο, προσπάθεια και γνώση για τη χρήση της. Ακόμα κι αν ένας προγραμματιστής δεν διαθέτει τις απαραίτητες δεξιότητες, συνήθως αναγκάζεται να τις αποκτήσει κάτω από πιεστικές συνθήκες, πράγμα που μπορεί να οδηγήσει σε σφάλματα και σε εκτώσεις στην ποιότητα τόσο της διαδικασίας όσο και του τελικού προϊόντος.



Εικόνα 2: Διαδικασία ανάπτυξης λογισμικού SOA

Παρακάτω παρουσιάζουμε την τυποποιημένη διαδικασία για την ανάπτυξη μιας εφαρμογής λογισμικού νέφους που βασίζεται στην χρήση μικροϋπηρεσιών:

- Η Σύνθεση Υπηρεσιών θεωρείται ως η ενοποίηση πολλαπλών μικροϋπηρεσιών με ακολουθιακό τρόπο. Ο προγραμματιστής πρέπει να αναλύσει ένα πρόβλημα σε μικρότερες εργασίες, οι οποίες με τη σειρά τους υλοποιούνται ως υπηρεσίες. Στη συνέχεια, πρέπει να δομήσει το πρόβλημα χρησιμοποιώντας την κατάλληλη αρχιτεκτονική, προκειμένου το τελικό αποτέλεσμα να λειτουργεί σωστά και να είναι καλής ποιότητας.
- Η διαδικασία της Εύρεσης είναι η αναζήτηση μιας υπηρεσίας κατάλληλης για τις ανάγκες μιας συγκεκριμένης λειτουργικότητας. Αυτή η διαδικασία μπορεί να είναι αρκετά χρονοβόρα, καθώς απαιτεί εκτενή και μη αυτόματη αναζήτηση από μεριάς του προγραμματιστή. Ο

προγραμματιστής πρέπει να ερευνήσει πολλά αποθετήρια υπηρεσιών για να βρει μια κατάλληλη, λειτουργική και προσβάσιμη υπηρεσία.

- Η Δημιουργία Υπηρεσίας είναι η εκ του μηδενός ανάπτυξη μιας νέας υπηρεσίας. Αυτό γίνεται για τις ανάγκες της Σύνθεσης Υπηρεσιών και σε περίπτωση που η διαδικασία Εύρεσης αποτύχει, κατά τη διαδικασία δημιουργίας, ο προγραμματιστής υλοποιεί την απαιτούμενη λειτουργικότητα, αναπτύσσοντας κώδικα.
- Η Δοκιμή αντιστοιχεί στον έλεγχο μιας μεμονωμένης υπηρεσίας αλλά και της Σύνθεσης Υπηρεσιών. Η δοκιμή και η αξιολόγηση ποιότητας αποτελούν αναπόσπαστο κομμάτι του κύκλου ζωής και της χρησιμότητας ενός λογισμικού, καθώς ένα λογισμικό καλύτερης ποιότητας τείνει να χρησιμοποιεί καλύτερα τον χρόνο και του πόρους που επενδύθηκαν για την ανάπτυξή του.
- Η Διάθεση λογισμικού μπορεί να θεωρηθεί το τελικό βήμα της ανάπτυξης, όπου μια υπηρεσία διατίθεται μέσω δικτύου. Είναι μια σημαντική πτυχή της διαδικασίας καθώς απαιτεί εξειδικευμένες γνώσεις προκειμένου να αξιοποιηθούν οι δυνατότητες του νέφους και να επιτευχθεί μια δυναμική εκτέλεση που προσφέρει απόδοση και επεκτασιμότητα.

Κάθε στάδιο της ανάπτυξης περιέχει πολλά μη τετριμμένα και χρονοβόρα βήματα ανάπτυξης και διαμόρφωσης. Ως εκ τούτου, μια μη αυτόματη προσέγγιση είναι επιρρεπής σε σφάλματα που μπορούν να οδηγήσουν σε μη βέλτιστες αρχιτεκτονικές και μη αποδοτικές υπηρεσίες, οι οποίες δεν εκμεταλλεύονται πλήρως την υποδομή του νέφους.

2.1 Σύνθεση Υπηρεσιών

Η αυτοματοποίηση επιχειρησιακών διεργασιών (Business process automation – BPA) είναι ένας τεχνολογικός τομέας που παρουσιάζει ραγδαία αύξηση και υποστηρίζεται από πλήθος διαφορετικών διαθέσιμων υλοποιήσεων και τεχνολογιών. Η αυτοματοποίηση διεργασιών γίνεται μέσω της μοντελοποίησης επιχειρησιακών διεργασιών (Business process management - BPM), που είναι δυνατόν να

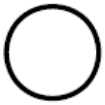
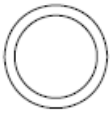
αναπαραστήσει απλές διεργασίες ρουτίνας αλλά και περίπλοκες διεργασίες όπως για παράδειγμα διάφορους μηχανισμούς κρατικών συστημάτων. Υπάρχουν πολλοί τρόποι για την επίτευξη της αυτοματοποίησης, από τους πιο απλούς, μέσω της συγγραφής κώδικα για την δημιουργία αλγορίθμων ενορχήστρωσης, μέχρι και τους πλέον μοντέρνους που χρησιμοποιούν ειδικά εργαλεία και προγράμματα.

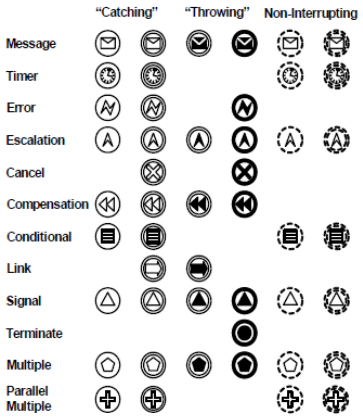
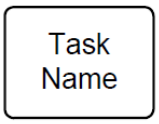
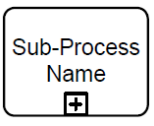
2.1.1 Σύνθεση υπηρεσιών χρησιμοποιώντας το Business Process Model and Notation (BPMN)

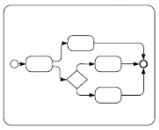
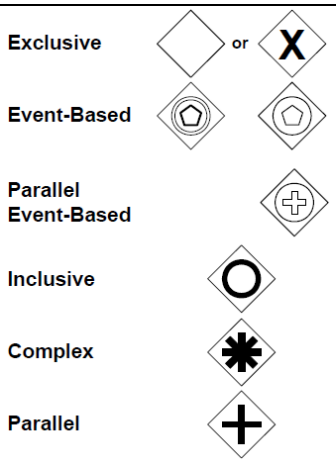
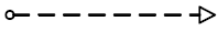
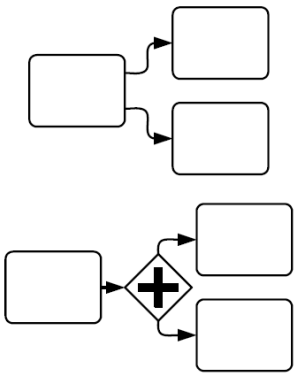
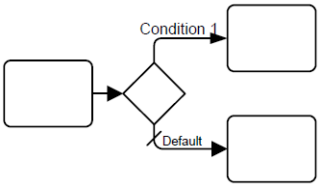
2.1.1.1 Βασικά στοιχεία μοντελοποίησης του BPMN

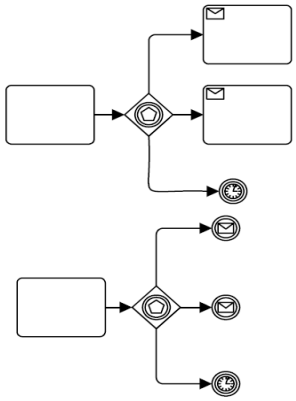
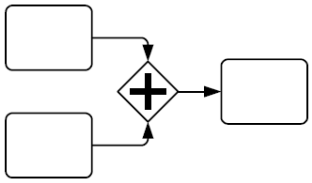
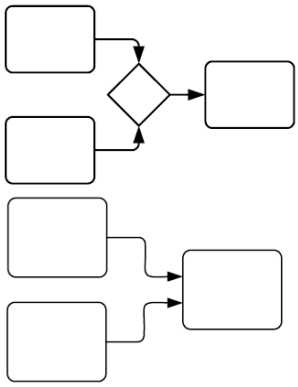
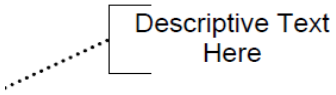
Το Business Process Model and Notation (BPMN) αποτελεί το πιο διαδεδομένο πρότυπο παγκοσμίως για την μοντελοποίηση διεργασιών. Δημιουργήθηκε από την Business Process Management Initiative (BPMI) και πλέον συντηρείται από το ινστιτούτο Object Management Group (OMG). Η τελευταία διαθέσιμη έκδοση του BPMN, είναι το BPMN 2.0, το οποίο περιλαμβάνει βελτιώσεις και πληθώρα διαφορετικών σχεδιαστικών στοιχείων. Χρησιμοποιώντας το BPMN 2.0, μπορούμε να μοντελοποιήσουμε ακόμα και τις πιο περίπλοκες διεργασίες, με τρόπο που παρομοιάζει την Ενοποιημένη Γλώσσα Σχεδίασης Προτύπων (Unified Modeling Language - UML). Στον παρακάτω πίνακα παρουσιάζουμε τα πιο συχνά χρησιμοποιούμενα στοιχεία του BPMN 2.0.

Πίνακας 1 Βασικά στοιχεία - κόμβοι μοντελοποίησης του BPMN

Element	Description	Notation
Event	Το Event είναι ένα γεγονός που συμβαίνει κατά την διάρκεια εκτέλεσης μίας διεργασίας. Τα συμβάντα επηρεάζουν την δομή του μοντέλου και συνήθως δρουν ως η εναρκτήρια αιτία κάποιας λειτουργίας. Τα συμβάντα χωρίζονται σε τρία είδη, βάσει της επιρροής τους στην ροή της διεργασίας: Start, Intermediate, and End.	
Start	Όπως καταλαβαίνουμε και από το όνομα, αυτό το συμβάν είναι το εναρκτήριο μίας διεργασίας ροής και επιδεικνύει το σημείο έναρξής της.	
Intermediate	Τα ενδιάμεσα συμβάντα συμβαίνουν ανάμεσα στο συμβάν εκκίνησης και στο συμβάν τερματισμού. Αυτά επηρεάζουν την εκτέλεση της ροής.	

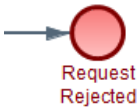
Element	Description	Notation
End	Όπως καταλαβαίνουμε από το όνομα, αυτό το συμβάν είναι ο καταληκτικός κόμβος μία διεργασίας ροής και επιδεικνύει το σημείο τερματισμού της.	
Διάφοροι πιο ειδικοί τύποι Event	Υπάρχουν διάφοροι τύποι ενδιάμεσων συμβάντων, όπως π.χ. η ακύρωση μία άλλης σύνθεσης, ή η πάροδος μίας συγκεκριμένης χρονικής περιόδου.	
Activity	Ένα Activity είναι η γενική απεικόνιση μίας λειτουργίας. Τα Activities χωρίζονται σε δύο υποκατηγορίες, τα Sub-Process και τα Task και απεικονίζονται σαν στρογγυλεμένα παραλληλόγραμμα.	
Task (Atomic)	Το Task είναι ένα ατομικό Activity ενός διαγράμματος. Επίσης, είναι η μικρότερη οντότητα αναπαράστασης μίας λειτουργικότητας, μη λαμβάνοντας υπόψη την αναπαράσταση σε πολύ χαμηλά επίπεδα.	
Process/ Sub-Process (non-atomic)	Ένα Sub-Process είναι ένα σύνθετο Activity. Η λειτουργία του Sub-Process μπορεί να αναλυθεί περαιτέρω, αντικαθιστώντας το με περισσότερους κόμβους.	
Collapsed Sub-Process	Τα δομικά στοιχεία ενός Sub-Process δεν είναι ορατά κατά την διάρκεια σχεδίασης του διαγράμματος. Το σύμβολο "+" υποδεικνύει ότι ο κόμβος έχει και χαμηλότερο επίπεδο αναπαράστασης.	

Element	Description	Notation
Expanded Sub-Process	Ένα Sub-Process που έχει διασταλεί και τα δομικά του στοιχεία είναι πλέον ορατά	
Gateway	Ένα Gateway ή πύλη, χρησιμοποιείται για τον έλεγχο της ροής. Με αυτό, αποφασίζουμε την διακλάδωση, ένωση, κτλ της ροής εκτέλεσης.	
Gateway Control Types	Τα εσωτερικά εικονίδια ενός Gateway καθορίζουν την συμπεριφορά του.	
Sequence Flow	Η ροή ακολουθίας απεικονίζει την σειρά με την οποία θα εκτελεστούν τα βήματα – κόμβοι του διαγράμματος.	
Message Flow	Η ροή μηνυμάτων απεικονίζει την κατεύθυνση μηνυμάτων ανάμεσα σε δύο κόμβους	
Fork	Το BPMN χρησιμοποιεί τον όρο «fork» αναφερόμενο σε μία διακλάδωση. Η διακλάδωση μπορεί να οδηγεί σε δύο ή παραπάνω κόμβους. Ως αποτέλεσμα, μετατρέπουμε την ροή του διαγράμματος και η εκτέλεση των κόμβων πλέον γίνεται παράλληλη αντί ακολουθιακή.	
Exclusive	Αυτή η διακλάδωση συμπεριφέρεται σαν ένας κόμβος απόφασης, επομένως μόνο ένα μονοπάτι της διακλάδωσης θα εκτελεστεί.	

Element	Description	Notation
Event-Based	Η απόφαση αυτού του κόμβου επηρεάζεται από ένα συμβάν. Ανάλογα με το συμβάν, ενεργοποιείται και το κατάλληλο μονοπάτι.	
Join	Το BPMN χρησιμοποιεί το Join για να κάνει την ένωση δύο μονοπατιών. Μία χρήση είναι το AND-Join που στην ουσία συγχρονίζει την εκτέλεση του διαγράμματος και των διαφορετικών ροών.	
Merging	Το BPMN χρησιμοποιεί το Merge για να κάνει την σύμπτυξη δύο μονοπατιών, προσομοιάζοντας την λειτουργία ενός OR-Join.	
Text Annotation (attached with an Association)	Ο σχολιασμός μέσω κειμένου είναι ένας τρόπος με τον οποίο ο σχεδιαστής του διαγράμματος παρέχει σχόλια και πληροφορίες για τους μελλοντικούς αναγνώστες του διαγράμματος.	

2.1.1.2 Events - Συμβάντα

Πίνακας 2 Συμβάντα

Notation	Description
	START Κάθε διεργασία συνήθως ξεκινά με έναν κόμβο εκκίνησης και συμβολίζει το συμβάν που εκκινεί την διαδικασία, όπως για παράδειγμα την λήψη ενός e-mail ή την κατάθεση μίας φόρμας. Ένα διάγραμμα μπορεί να έχει περισσότερους από έναν κόμβο εκκίνησης, για να εκτελείται υπό πολλές διαφορετικά περιπτώσεις.
	END Κάθε διεργασία μπορεί να περιλαμβάνει έναν ή περισσότερους καταληκτικούς κόμβους. Κάθε κόμβος δηλώνει το τέλος εκτέλεσης του διαγράμματος.
	INTERMEDIATE Τα ενδιάμεσα συμβάντα χωρίζονται σε δύο κατηγορίες: Catch και Throw. Ένα ενδιάμεσο Catch σταματάει την ροή της εκτέλεσης, μέχρις ότου ικανοποιηθεί. Αντιθέτως, ένα ενδιάμεσο Throw δεν σταματάει την εκτέλεση, απλώς εκτελείται από την ροή, η οποία μετά συνεχίζει.

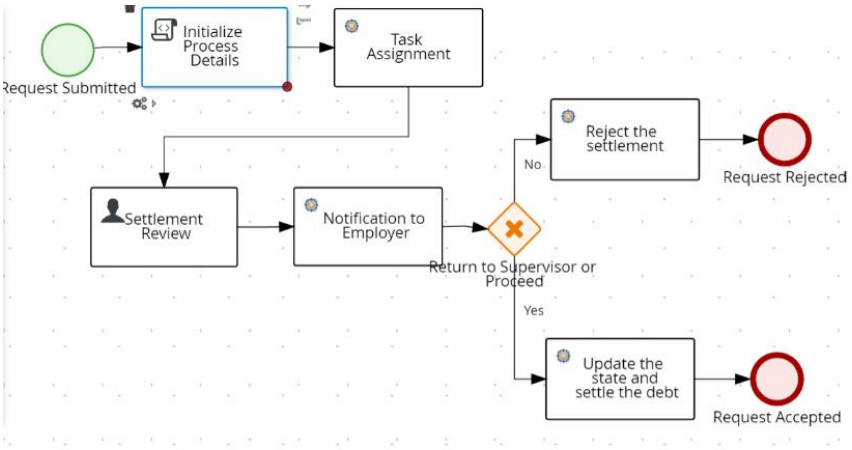
2.1.1.3 Tasks

Τα Task μπορούν να διαχωριστούν σε δύο βασικές κατηγορίες, ανάλογα με το είδος λειτουργίας τους:

1. Τα Service Tasks ή Task υπηρεσιών εκτελούνται από τον μηχανισμό εκτέλεσης BPMN και η ροή συνεχίζει κανονικά την εκτέλεσή της (Εκτός αν συμβεί κάποιο σφάλμα κατά την διάρκεια εκτέλεσης του Service Task). Τα Service Task υλοποιούνται με την χρήση των Work Item Handlers, που αποτελούν κλάσεις πηγαίου κώδικα. Ένας σχεδιαστής διαγράμματος μπορεί να επεκτείνει ένα AbstractWorkItemHandler και να δημιουργήσει μία νέα, δικιά του προσαρμοσμένη υλοποίηση.
2. Τα Manual Tasks για την εκκίνηση και την εκτέλεσή τους απαιτούν την εμπλοκή κάποιου ατόμου. Όταν ο μηχανισμός εκτέλεσης BPMN φτάσει σε έναν κόμβο Manual Task, σταματάει προσωρινά την ροή εκτέλεσης μέχρις ότου αυτός να ολοκληρωθεί από ένα άτομο. Όταν το άτομο ολοκληρώσει το Manual Task, ο μηχανισμός εκτέλεσης BPMN ειδοποιείται και η ροή συνεχίζει την εκτέλεση της.

2.1.1.4 Script Task

Τα Script Task είναι μία υποκατηγορία των Service Task και συνήθως εκτελούν ένα σύνολο εντολών κατά την εκκίνηση και αρχικοποίηση της εκτέλεσης του διαγράμματος. Μέσω αυτών, μπορούν να αρχικοποιηθούν μεταβλητές του διαγράμματος και άλλα λειτουργικά χαρακτηριστικά. Αυτό βοηθάει, καθώς όλα τα απαραίτητα αρχικά συστατικά της ροής είναι συγκεντρωμένα σε ένα σημείο. Όπως φαίνεται και στην παρακάτω εικόνα, ο κόμβος «Initialize Process Details» κάνει ακριβώς αυτό που αναφέραμε. Οι παράμετροι που αρχικοποιεί, χρησιμοποιούνται αργότερα από τους υπόλοιπους κόμβους του διαγράμματος.



Εικόνα 3: Παράδειγμα χρήσης Script Task κατά την εκκίνηση ενός διαγράμματος

2.2 Κατάταξη και Εύρεση Υπηρεσιών

Χρησιμοποιώντας την λογική της σύνθεσης υπηρεσιών για την παραγωγή ενός αποτελέσματος μέσω της χρήσης αρχιτεκτονικής υψηλότερων επιπέδων, η κατάταξη και έπειτα η αναζήτηση και εύρεση υπηρεσιών προσφέρεται σαν ένα νέο πεδίο έρευνας. Η χειρωνακτική αναζήτηση και κατάταξη των υπηρεσιών, δεν μπορεί πλέον να θεωρείται ως βιώσιμη λύση για το πρόβλημα της σύνθεσης. Με την ραγδαία αύξηση των διαθέσιμων υπηρεσιών, η εύρεση κατάλληλης υπηρεσίας που ικανοποιεί τις ανάγκες μιας λειτουργικότητας γίνεται ολοένα και δυσκολότερη, ακόμα και όταν η αναζήτηση γίνεται πάνω σε καλά τεκμηριωμένες και καταταγμένες υπηρεσίες. Για την ευκολότερη και γρηγορότερη επαναχρησιμοποίηση υπηρεσιών, είναι αναγκαίο η αναζήτηση να γίνεται πάνω σε σύνολα υπηρεσιών με καλή κατάταξη.

Φαίνεται πως το πρόβλημα της κατάταξης και έπειτα της αναζήτησης και εύρεσης υπηρεσιών, ενδιαφέρει αρκετά την επιστημονική κοινότητα καθώς υπάρχει αρκετό ερευνητικό υλικό διαθέσιμο. Προσφέρεται πληθώρα πιθανών μεθόδων που προσπαθούν να επιλύσουν το πρόβλημα, βασισμένες στην σημασιολογία [5, 6] και την τεχνητή νοημοσύνη [7, 8, 9, 10, 11].

Η λογική της σημασιολογικής προσέγγισης είναι ότι η αποθήκευση των υπηρεσιών γίνεται χρησιμοποιώντας κάποια γλώσσα οντολογίας, έτσι ώστε να είναι εφικτή μέσω αυτής η μετέπειτα αναζήτηση στα δεδομένα. Η βιβλιογραφία δείχνει πως η χρήση σημασιολογίας είναι πιο αποδοτική συγκριτικά με την χρήση αναζήτησης λέξεων κλειδιών, αλλά ενέχει μερικά προβλήματα, καθώς χρειάζεται χαρτογράφηση [12].

Οι μέθοδοι που χρησιμοποιούν τεχνητή νοημοσύνη, μπορούν να ακολουθήσουν πολλές διαφορετικές προσεγγίσεις όπως:

1. Κλασικούς αλγορίθμους μηχανικής μάθησης, από τους οποίους ο SVM και ο Naive Bayes ήταν αποδοτικοί και οδήγησαν σε καλά αποτελεσματικά.
2. Τεχνητά νευρωνικά δίκτυα και προσεγγίσεις που χρησιμοποιούν ασαφή λογική (fuzzy logic).
3. Υβριδικές προσεγγίσεις που συνδυάζουν μεθόδους και λογικές από τα παραπάνω.

Οι πλειοψηφία των υλοποιήσεων αυτόματης κατάταξης χρησιμοποιούν κλασικούς αλγορίθμους μηχανικής μάθησης, το αποτέλεσμα των οποίων εξαρτάται σημαντικά από την χειρωνακτική παραμετροποίησή τους. Για τον λόγο αυτό, οι νεότερες μελέτες έχουν προσανατολιστεί προς στην χρήση νευρωνικών δικτύων και μεθόδων Βαθιάς Μάθησης (Deep Learning). Τα δίκτυα Βαθιάς Μάθησης, δεν απαιτούν χειρωνακτική παραμετροποίηση καθώς εκπαιδεύονται εισάγοντας τους δεδομένα, αλλά η απαίτηση τους σε δεδομένα εκπαίδευσης για την παραγωγή αξιόπιστων αποτελεσμάτων είναι πολύ μεγάλη.

Η πλειοψηφία των προσφερόμενων υπηρεσιών διαδικτύου, είναι υλοποιημένες χρησιμοποιώντας δύο διαδεδομένα πρωτόκολλα, το SOAP (Simple Object Access Protocol) και το REST (Representational State Transfer). Οι δύο αυτές μέθοδοι υλοποίησης διαφέρουν και στην υλοποίηση αλλά και στην τεκμηρίωσή τους. Η τεκμηρίωση είναι ιδιαιτέρως σημαντική για την διαδικασία της κατάταξης, καθώς παρέχει το μεγαλύτερο πλήθος τυποποιημένης πληροφορίας σχετικά με την υπηρεσία.

Στις SOAP υπηρεσίες έχουμε τεκμηρίωση μέσω των SOAP notes και της WSDL (Web Services Description Language), μέσω των οποίων αποκτούμε την περιγραφή και τον τρόπο επικοινωνίας μιας υπηρεσίας. Στις REST υπηρεσίες έχουμε την τεκμηρίωση μέσω του OpenAPI Specification (OAS) ή αλλιώς Swagger. Το OpenAPI Specification είναι ένας τυποποιημένος τρόπος παρουσίασης των APIs μίας RESTful υπηρεσίας. Η πιο συνηθισμένη πληροφορία που χρησιμοποιείται για την κατάταξη υπηρεσιών, βάσει της βιβλιογραφίας, είναι η WSDL, καθώς μπορεί να περιλαμβάνει αρκετά χρήσιμη και επεξηγηματική πληροφορία για την υπηρεσία, όπως το όνομα, την τεκμηρίωση, τα περιεχόμενα, το σχήμα, τα μηνύματα, τις πόρτες επικοινωνίας κ.α. [8, 13, 14, 15].

Ο Crasso κ.α. υποστηρίζουν πως η δυσκολία της κατάταξης υπηρεσιών επιδρά αρνητικά στον βαθμό διάδοσης και χρήσης τους. Προς επίλυση αυτού του προβλήματος, παρέχουν μια αυτοματοποιημένη μέθοδο (AWSC) η οποία εκμεταλλεύεται τις παρεχόμενες WSDL περιγραφές των υπηρεσιών. Η μέθοδος εξορύσσει κείμενο από τις περιγραφές, και σε συνδυασμό με τεχνικές τεχνητής νοημοσύνης κατατάσσει τις υπηρεσίες. Η μέση ευστοχία που επιτεύχθηκε από την παραπάνω μέθοδο είναι 85%. [13]

Η τακτική που πρότειναν οι Kamath και Ananthanarayana στο πρόβλημα της κατάταξης υπηρεσιών προέρχεται από την επιστήμη των δεδομένων και είναι βασισμένη σε μορφολογική ανάλυση και τεχνητή νοημοσύνη. Η μεθοδολογία τους χωρίζεται σε τρία στάδια, (1) την εξαγωγή της σημασιολογίας μίας υπηρεσίας χρησιμοποιώντας τεχνικές επεξεργασίας φυσικής γλώσσας, (2) την μοντελοποίηση των χαρακτηριστικών της υπηρεσίας και τέλος (3) την κατάταξη της. Κατά την δοκιμή και αξιολόγησή τους, οι Kamath και Ananthanarayana συμπέραναν πως η χρήση μεθόδων Ensemble για την συγκεκριμένη τακτική, οδήγησε σε καλύτερα αποτελέσματα συγκριτικά με την χρήση πολυωνυμικού Näive Bayes και SVM (Support Vector Machine). Η αξιολόγηση της προτεινόμενης τακτικής έγινε με την χρήση των περιγραφών από 1076 υπηρεσίες, οι οποίες ανήκουν σε 9 διαφορετικά τομείς. Τα αποτελέσματα ήταν ιδιαίτερα καλά, καθώς ξεπέρασαν το 90% ευστοχία. [11]

Οι Yuan-jie κ.α. [8] χρησιμοποίησαν SVM (Support Vector Machine) και μεθόδους Ensemble, παρόμοια με τους Kamath και Ananthanarayana [11]. Βάσει των αποτελεσμάτων τους, συμπέραναν πως η κατάταξη μέσω Ensemble, παρείχε το καλύτερο αποτέλεσμα ευστοχίας.

Αντιθέτως, οι Yang κ.α. [16] χρησιμοποίησαν μεθόδους Βαθιάς Μάθησης, ώστε να αποσπάσουν μία χαμηλού επιπέδου απεικόνιση της περιγραφής των υπηρεσιών. Με

το εκπαιδευμένο μοντέλο Βαθιάς Μάθησης, προχώρησαν στην κατάταξη υπηρεσιών σε 50 διακριτές κατηγορίες.

Ο Alshafaey κ.λ. πρότειναν μια νέα μέθοδο κατάταξης με το όνομα Cloud-Based Classification Methodology (CBCM). Η μέθοδος, όπως και αυτή του Kamath και Ananthanarayana, χωρίζεται σε τρία βασικά στάδια:

1. το Concepts Preparation Module (CPM) που σαρώνει και φιλτράρει τις εισαγόμενες μεθόδους, έτσι ώστε να εντοπίσει σημαντικές λέξεις κλειδιά για να δημιουργήσει βασικές έννοιες.
2. το Tree Creation Module (TCM) που δημιουργεί ένα δέντρο που περιέχει τις βασικές έννοιες του πρώτου σταδίου, σταθμισμένες.
3. την μονάδα κατάταξης με το όνομα CEAM (Change, Edit, Add Module), η οποία υπολογίζει και επιστρέφει το καταλληλότερο αποτέλεσμα βασισμένο στο αίτημα που εισήγαγε ο χρήστης.

Το σύνολο δεδομένων που χρησιμοποιήθηκε για την επαλήθευση της μεθόδου, περιέχει 1007 υπηρεσίες τεσσάρων διαφορετικών τομέων. Τα αποτελέσματα της επαλήθευσης δείχνουν πως το CBCM είχε καλύτερη απόδοση και ακρίβεια από άλλες διαδεδομένες μεθόδους κατάταξης, όπως για παράδειγμα SVM, SWSC κ.α. [5]

Ο Das κ.ά. [9] πραγματοποιώντας έρευνα για την κατάταξη υπηρεσιών, χρησιμοποίησαν το σύνολο δεδομένων QWS (Quality Web Services) [17]. Το σύνολο δεδομένων QWS περιλαμβάνει της διευθύνσεις διαφόρων υπηρεσιών, καθώς και τις μη λειτουργικές τους παραμέτρους. Χρησιμοποιώντας εξόρυξη δεδομένων στις διαθέσιμες διευθύνσεις των υπηρεσιών, κατάφεραν να εξάγουν τα λειτουργικά τους δεδομένα. Οι μέθοδοι που χρησιμοποίησαν για την κατάταξη είναι:

- Random Forest,
- eXtreme Gradient Boosting
- Decision Tree
- Support Vector Machine(SVM).

Βάσει των αποτελεσμάτων τους, συμπεραίνουμε πως η χρήση eXtreme Gradient Boosting παρέχει αποτελέσματα μεγαλύτερης ακρίβειας.

Ο Kamath κ.ά. υποστηρίζουν πως η κατάταξη υπηρεσιών βάσει των περιγραφών και των παραμέτρων εισόδου και εξόδου τους δεν είναι αρκετά αποτελεσματική. Επομένως, προτείνουν μία προσέγγιση για την κατάταξη υπηρεσιών, η οποία βασίζεται στην μετατροπή των υπηρεσιών σε διανύσματα και στην μετέπειτα βαθμολόγηση της

σημασιολογικής τους συνάφειας με ένα αίτημα αναζήτησης. Επιπρόσθετα, παρουσιάζουν και συγκρίνουν τα αποτελέσματα άλλων τεχνικών, βασισμένων στην χρήση Multilayer Perceptron (MLP), Naïve Bayes κ.α. [18]

Τέλος, μετά την διαδικασία της κατάταξης υπηρεσιών, ακολουθεί η διαδικασία αναζήτησης και εύρεσης τους. Για να επιτευχθεί αυτό, πρέπει να έχουμε πρόσβαση στις διαθέσιμες υπηρεσίες έτσι ώστε να μπορούμε να αναζητήσουμε βάσει κάποιας λειτουργικότητας ή περιγραφής. Το παραπάνω επιτυγχάνεται με την χρήση διαφόρων διαδικτυακά διαθέσιμων αποθετηρίων, όπως το Programmableweb, GitHub, AWS Services, Azure, GitLab, Bitbucket etc. Για την χρήση ενός αποθετηρίου, είναι αναγκαία η επικοινωνία με αυτό. Τα περισσότερα μοντέρνα αποθετήρια, δέχονται επικοινωνία μέσω Πρωτοκόλλου Μεταφοράς Υπερκειμένου (Http - HyperText Transfer Protocol). Η επικοινωνία συνήθως γίνεται μέσω προσφερόμενων, από την μεριά του αποθετηρίου, διασυνδέσεων API, τα οποία δέχονται αιτήσεις. Η απόκριση σε μία αίτηση, είναι συνήθως σε τυποποιημένη μορφή, για παράδειγμα JSON, XML, κτλ. Έχοντας στην διάθεση μας τα προσφερόμενα API, μπορούμε να κάνουμε ανίχνευση ιστού (web crawl) και να αποσπάσουμε μεγάλο αριθμό χρήσιμης πληροφορίας. Για τον λόγο αυτό, η έρευνα των τελευταίων ετών, βασίζεται κατά κύριο λόγο σε πληροφορία αντλημένη μέσω ανίχνευσης ιστού [19]. Παρόλα αυτά, τα δεδομένα που αντλούνται μέσω της ανίχνευσης ιστού, συνήθως περιέχουν σημαντικό βαθμό αχρείαστης πληροφορίας, επομένως είναι αναγκαία η επεξεργασία των δεδομένων. Αυτό επιτυγχάνεται με την επεξεργασία φυσικής γλώσσας (NLP - Natural Language Processing) και την εξαγωγή χαρακτηριστικών από τα δεδομένα [20, 21, 22]. Τα δεδομένα που αντλούνται μπορούν να είναι δύο ειδών:

1. Δεδομένα λογισμικού
2. Δεδομένα υπηρεσίας

Τα δεδομένα λογισμικού αντλούνται από αποθετήρια όπως το GitHub και το GitLab, και περιλαμβάνουν τον πηγαίο κώδικα και τα μεταδιδόμενα μιας υπηρεσίας [23, 24, 25, 26]. Τα δεδομένα υπηρεσίας αντλούνται από αποθετήρια όπως το Programmableweb και το AWS Services, και συνήθως περιλαμβάνουν απλώς τα μεταδιδόμενα για μία υπηρεσία [27, 28]. Η βασική διαφορά στα δύο είδη δεδομένων είναι πως μόνο στην μία περίπτωση έχουμε πρόσβαση και στον πηγαίο κώδικα της υπηρεσίας, πράγμα που βοηθά στην καλύτερη κατανόηση της λειτουργίας της.

Τελευταία, έχει δοθεί ιδιαίτερη έμφαση στα API του GitHub [29], έτσι ώστε να καθορίζονται καλύτερα τα χαρακτηριστικά των υπηρεσιών. Μέσω των API μπορούμε να πάρουμε πληροφορίες για το λογισμικό όπως την βαθμολογία του, την περιγραφή και τις περαιτέρω λεκτικές πληροφορίες για αυτό, την άδεια χρήσης, ενδεχόμενες ευπάθειες που μπορεί να υπάρχουν, τα γνωστά λειτουργικά θέματα, την ημερομηνία ενημέρωσης, τις απαιτήσεις κ.α.

2.3 Δημιουργία Υπηρεσιών

Για την ανάπτυξη μίας εφαρμογής, η δημιουργία μιας υπηρεσίας θεωρείται ως ένα από τα βασικότερα κομμάτια, καθώς απαιτείται η σύνθεση πολλών διαφορετικών και ανεξάρτητων υπηρεσιών για την επίτευξη του τελικού αποτελέσματος. Ερευνώντας και ακλουθώντας τα απαραίτητα βήματα για την δημιουργία μία νέας υπηρεσίας μπορούμε να εντοπίσουμε τις τεχνολογίες και το είδος των εργαλείων που απαιτούνται για την ολοκλήρωση της διαδικασίας ανάπτυξης. Μετά από αυτό, μπορούμε να μελετήσουμε τα διαθέσιμα εργαλεία που υπάρχουν στην αγορά και να αποφασίσουμε ποιο από αυτά ικανοποιεί καλύτερα τις απαιτήσεις μας.

2.3.1 Απαιτούμενη διαδικασία για την ανάπτυξη λογισμικού

Η σωστή διαδικασία ανάπτυξης μίας νέας υπηρεσίας, απαιτεί κάποιες βασικές ενέργειες, καθώς και την εμπλοκή απαραίτητων τεχνολογιών για:

1. Την συνεργατική ανάπτυξη του κώδικα
2. Την συγγραφή κώδικα
3. Την δημιουργία, την εκτέλεση και την αξιολόγηση δοκιμών
4. Την αξιολόγηση της συντηρησιμότητας του λογισμικού και την παρακολούθηση της προόδου του
5. Το πακετάρισμα του κώδικα σε εκτελέσιμο αρχείο

Το τρίτο βήμα της δημιουργίας, εκτέλεσης και αξιολόγησης δοκιμών θα αναλυθεί με μεγαλύτερη λεπτομέρεια στην επόμενη ενότητα (2.4), μαζί με τις μεθόδους και τους τύπους δοκιμών. Παρομοίως, το πέμπτο βήμα πακεταρίσματος του κώδικα σε εκτελέσιμο αρχείο, θα το δούμε αναλυτικότερα στην μεθεπόμενη ενότητα (2.5), όπου θα αναλύσουμε τις μεθόδους διάθεσης μίας υπηρεσίας μέσω δικτύου στο νέφος (deployment).

2.3.1.1 Εργαλεία συνεργατικής ανάπτυξης

Ξεκινώντας, βλέπουμε πως το πρώτο βήμα αφορά την συνεργατική ανάπτυξη κώδικα. Η συνεργατική ανάπτυξη κατά την διαδικασία δημιουργίας λογισμικού είναι ιδιαιτέρα σημαντική καθώς μέσω αυτής μπορούν να εμπλακούν πολλαπλά μέλη στην ανάπτυξη κώδικα. Η διαδεδομένη τεχνολογία που πλέον χρησιμοποιείται σχεδόν αποκλειστικά, είναι το Git. Το Git είναι ένα σύστημα ανοιχτού κώδικα, μέσω του οποίου καθίσταται δυνατή η συνεργασία και η ταυτόχρονη υλοποίηση μεταξύ πολλών χρηστών, πάνω σε ένα κοινό έργο. Επίσης, μέσω της τεχνολογίας αυτής, είναι δυνατό να παρακολουθούμε τις εκδόσεις του έργου καθώς και την πρόοδο του. Το Git λειτουργεί με την χρήση αποθετηρίων για κάθε ξεχωριστό έργο. Στο αποθετήριο αποθηκεύεται ο πηγαίος κώδικας του έργου, καθώς και το ιστορικό με τις αλλαγές που έχουν γίνει σε αυτό. Όταν ένας χρήστης του αποθετηρίου κάνει τοπικά αλλαγές στον κώδικα, μπορεί να τις καταθέσει και να τις οριστικοποιήσει, σπρώχνοντας τις στο αποθετήριο. Το σύστημα του Git θα αναλύσει τις αλλαγές που έγιναν και θα ενημερώσει τον κώδικα και τα αρχεία ιστορικού του αποθετηρίου. Στις μέρες μας, έχουν δημιουργηθεί πολλές νέες υλοποιήσεις και εργαλεία συνεργατικής ανάπτυξης που χρησιμοποιούν το Git στον πυρήνα τους και προσφέρουν γραφικά περιβάλλοντα με περαιτέρω διευκολύνσεις για τον χρήστη.

GitHub

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Διαδίκτυο
Πρόσβαση	Ιστοσελίδα, Τοπική Εφαρμογή και γραμμή εντολών
Υποστήριξη CI/CD (continuous integration / continuous delivery-deployment)	Ναι
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	Όχι
Χρέωση	Όχι
Ειδικές χρεώσεις	Ναι στην περίπτωση χρήσης ιδιωτικών αποθετηρίων
API	Ναι

GitLab

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Διαδίκτυο
Πρόσβαση	Ιστοσελίδα και γραμμή εντολών
Υποστήριξη CI/CD (continuous integration / continuous delivery-deployment)	Ναι
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό Server	Ναι
Χρέωση	Όχι
Ειδικές χρεώσεις	Ναι στην περίπτωση χρήσης ιδιωτικών αποθετηρίων. Όχι στην περίπτωση χρήσης σε ιδιωτικό Server.
API	Ναι

Bitbucket

Ανοιχτός Κώδικας	Όχι
Διαθεσιμότητα	Διαδίκτυο
Πρόσβαση	Ιστοσελίδα και γραμμή εντολών
Υποστήριξη CI/CD (continuous integration / continuous delivery-deployment)	Ναι
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	Όχι
Χρέωση	Όχι
Ειδικές χρεώσεις	Ναι στην περίπτωση χρήσης από ομάδες πολλών ατόμων
API	Ναι

SourceForge

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Διαδίκτυο
Πρόσβαση	Ιστοσελίδα και γραμμή εντολών
Υποστήριξη CI/CD (continuous integration / continuous delivery-deployment)	Ναι
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	Όχι
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
API	Ναι

2.3.1.2 Εργαλεία συγγραφής κώδικα

Συνεχίζοντας, το δεύτερο βήμα αφορά την συγγραφή του κώδικα, για την οποία είναι αναγκαία η χρήση κάποιου εργαλείου ή περιβάλλοντος ανάπτυξης. Υπάρχει πληθώρα διαθέσιμων εργαλείων τα οποία προσφέρουν διαφορετικές λειτουργίες και διευκολύνσεις.

Καθώς κατά κύριο λόγο μας ενδιαφέρει η συγγραφή κώδικα χρησιμοποιώντας την γλώσσα Java, θα περιορίσουμε τα διαθέσιμα εργαλεία σε αυτά που παρέχουν υποστήριξη για την γλώσσα.

Vim

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Τοπικά
Πρόσβαση	Γραμμή Εντολών
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	-
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	Ναι

Atom

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Τοπικά
Πρόσβαση	Γραφικό Περιβάλλον
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	-
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	Ναι

IntelliJ IDEA

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Τοπικά
Πρόσβαση	Γραφικό Περιβάλλον
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	-
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	Ναι

Visual Studio Code (VSCode)

Ανοιχτός Κώδικας	Όχι
Διαθεσιμότητα	Τοπικά
Πρόσβαση	Γραφικό Περιβάλλον
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	-
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	Ναι

NetBeans

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Τοπικά
Πρόσβαση	Γραφικό Περιβάλλον
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	-
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	Ναι

Eclipse IDE

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Τοπικά
Πρόσβαση	Γραφικό Περιβάλλον
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	-
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	Ναι

Eclipse Theia (Eclipse Che)

Ανοιχτός Κώδικας	Ναι
Διαθεσιμότητα	Νέφος
Πρόσβαση	Γραφικό Περιβάλλον μέσω διαδικτύου
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	Ναι
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	Ναι

Cloud9 IDE

Ανοιχτός Κώδικας	Όχι
Διαθεσιμότητα	Νέφος
Πρόσβαση	Γραφικό Περιβάλλον μέσω διαδικτύου
Συμβατότητα με άλλα εργαλεία	Ναι
Δυνατότητα χρήσης σε ιδιωτικό δίκτυο	-
Χρέωση	Όχι
Ειδικές χρεώσεις	Όχι
Επεκτάσιμο	-

2.3.1.3 Αξιολόγηση της συντηρησιμότητας λογισμικού και παρακολούθηση της προόδου του λογισμικού

Όπως αναφέραμε στην εισαγωγή, οι εταιρίες παράγουν και διαθέτουν λογισμικό σε όλο και πιο απαιτητικά χρονικά περιθώρια. Αυτό συχνά οδηγεί σε μείωση της ποιότητας του κώδικα, καθώς οι αποφάσεις και η ανάπτυξη του πραγματοποιούνται πιο βιαστικά. Δεδομένου ότι ένα λάθος στα αρχικά στάδια υλοποίησης ενός λογισμικού μπορεί επιφέρει πολύ μεγάλο κόστος επιδιόρθωσης στο μέλλον, καθώς έχουν επηρεαστεί τα μετέπειτα στάδια υλοποίησης, ο χρήστης πρέπει να υποβοηθάται όσο το δυνατόν περισσότερο. Επομένως, γνωρίζοντας πως ένα από τα βασικότερα έξοδα κατά την διάρκεια ανάπτυξης και τεχνικού χρέους του λογισμικού είναι η συντήρηση του, καθώς αυτή μπορεί να φτάσει το 50%-75% του συνολικού κόστους [30], είναι σημαντικό να παρακολουθούμε την ποιότητα και συντηρησιμότητα του έργου.

Τεχνικό Χρέος

Η πρώτη εμφάνιση του δανεισμού του όρου τεχνικού χρέους (ΤΧ) στον κλάδο της τεχνολογίας λογισμικού συνέβη το 1992, από τον Ward Cunningham. Με την χρήση αυτού του όρου, αναφερόμαστε στις αλλαγές κώδικα στα αρχικά στάδια υλοποίησης, εισάγοντας τους όρους του χρέους και του επιτοκίου στην ανάπτυξη λογισμικού [31]. Η κύρια λειτουργία του τεχνικού χρέους είναι να δρα ως ένα κοινά αντιληπτό χαρακτηριστικό ανάμεσα σε διαφορετικούς φορείς της πληροφορικής, όπως αυτούς των τεχνικών και των διαχειριστικών [32].

Βασικές Έννοιες Τεχνικού Χρέους

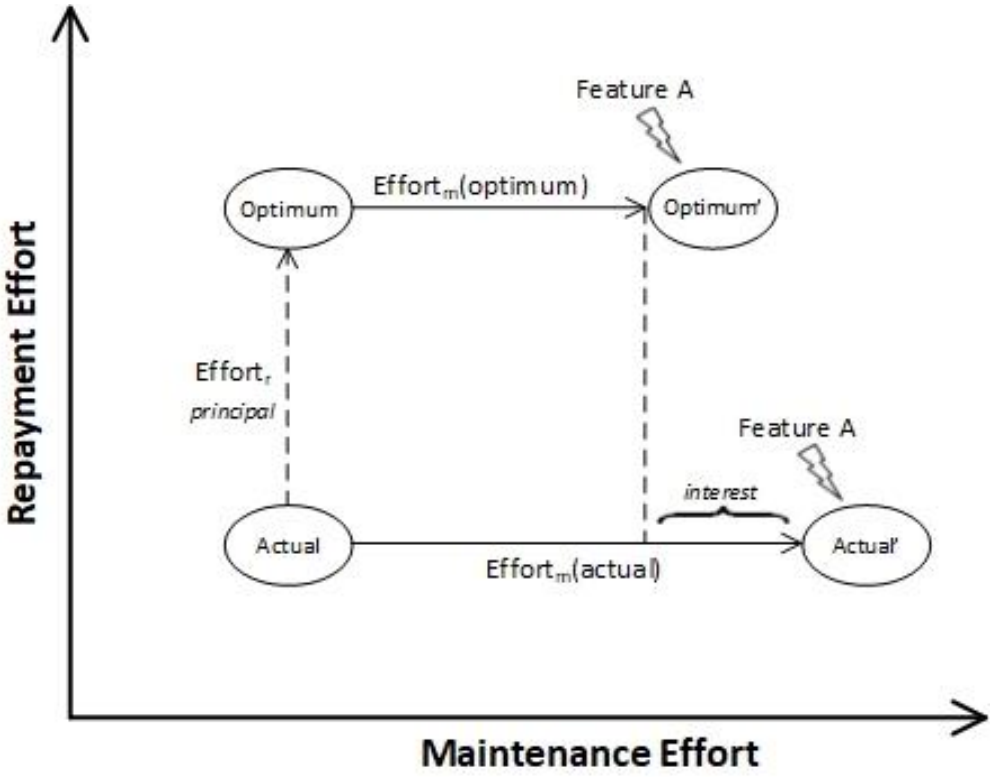
Τα βασικά στοιχεία του Τεχνικού Χρέους προέρχονται από την επιστήμη της οικονομίας και είναι το κεφάλαιο (principal) και ο τόκος (interest). Οι όροι αυτοί παίζουν σημαντικό ρόλο και εμπλέκονται στην διαδικασία ανάπτυξης ενός λογισμικού. Αν υποθέσουμε ότι μία εταιρία σχεδιάζει να φτιάξει ένα λογισμικό, αυτό απαιτεί την χρήση πόρων, π.χ. εργατοώρες. Υπολογίζοντας τους απαιτούμενους πόρους για την σωστή και πλήρη ανάπτυξη του λογισμικού, μπορούμε να καθορίσουμε το κεφάλαιο μας.

Όπως αναφέραμε και παραπάνω, πολλές φορές το προϊόν χρειάζεται να παραδοθεί σε πολύ απαιτητικά χρονικά περιθώρια επομένως μπορεί να είναι απαραίτητο

να παραμεληθούν κομμάτια της υλοποίησης. Όταν συμβαίνει αυτό, μια εταιρία κάνει εξοικονόμηση προσπάθειας και εργατικού δυναμικού, επομένως θεωρείται ότι δανείζεται κεφάλαιο λόγω της παραγωγής ανώριμου λογισμικού. Ο δανεισμός όμως επιφέρει και ποινή, η οποία είναι η πρόσθετη προσπάθεια μελλοντικής συντήρησης του κώδικα (τόκος).

Στην παρακάτω εικόνα, φαίνεται η σχέση μεταξύ των δύο εννοιών κατάστασης του συστήματος, το πραγματικό και το ιδεατό. Ανάλογα με τις τιμές που έχουμε σε κάποια χρονική στιγμή, μπορούμε να δούμε τους άξονες για να κατανοήσουμε πληροφορίες σχετικά με το λογισμικό. Στο άξονα y, αναπαρίσταται το επίπεδο ποιότητας του συστήματος. Επιπρόσθετα, μπορούμε να δούμε και την απόσταση μεταξύ της ιδεατής ποιότητας συγκριτικά με την πραγματική. Η δουλειά που απαιτείται από την ομάδα υλοποίησης του λογισμικού, έτσι ώστε αυτό να φτάσει στην ιδεατή του κατάσταση, αντιπροσωπεύεται από το Κεφάλαιο του Τεχνικού Χρέους, δηλαδή την κατακόρυφη απόσταση μεταξύ ιδεατής και πραγματικής κατάστασης.

Τέλος, ο Τόκος του Τεχνικού Χρέους, είναι η επιβάρυνση που προκύπτει λόγω της μη ταύτισης της πραγματικής κατάστασης με την ιδεατή. Αντιπροσωπεύει την επιπρόσθετη δουλεία που απαιτείται για την συντήρηση του λογισμικού στην πραγματική του κατάσταση, συγκριτικά με αυτήν που θα χρειαζόταν το αντίστοιχο σύστημα εάν ήταν στην ιδεατή του μορφή. [33]



Εικόνα 4: Οπτικοποίηση Ορολογίας TX [33]

2.4 Δοκιμή λογισμικού

Όπως αναφέραμε στην προηγούμενη ενότητα, η δοκιμή ενός λογισμικού αποτελεί βασικό βήμα για την διαδικασία ανάπτυξης. Μέσω των δοκιμών μπορούμε να εντοπίσουμε την εμφάνιση λογικών λαθών και να αποφύγουμε την λανθασμένη λειτουργία του λογισμικού. Η δοκιμή λογισμικού στο πλαίσιο της έρευνας μας μπορεί να διαχωριστεί σε δύο βασικές κατηγορίες, στην δοκιμή κατά την ανάπτυξη μίας υπηρεσίας και στην δοκιμή μετά την διάθεσή της.

2.4.1 Δοκιμή νέφους ή Δοκιμή στο νέφος

Επικρατεί σύγχυση όταν αναφερόμαστε στην δοκιμή σε περιβάλλοντα νέφους, επομένως είναι αναγκαίο να αποσαφηνίσουμε το θέμα.

Η δοκιμή νέφους αναφέρεται στην επικύρωση και επαλήθευση των εφαρμογών, των περιβαλλόντων και των υποδομών που διατίθενται μέσω αυτού, διασφαλίζοντας ότι

όλα ακολουθούν τις κατευθυντήριες γραμμές του επιχειρηματικού του μοντέλου. Αυτό σημαίνει ότι για μια εφαρμογή νέφους, θα πρέπει να εφαρμόζονται και να εκτελούνται όλα τα παραδοσιακά πρότυπα δοκιμών για να διασφαλιστεί ότι πληρούνται όλες οι λειτουργικές απαιτήσεις και ότι η ποιότητα της παρεχόμενης υπηρεσίας είναι υψηλή. Για την παροχή υψηλής ποιότητας, εκτός από τις λειτουργικές απαιτήσεις, πρέπει να εστιάσουμε και στην ικανοποίηση των μη λειτουργικών απαιτήσεων καθώς αποτελούν τον πυρήνα του επιχειρηματικού μοντέλου ενός νέφους.

Από την άλλη, η δοκιμή στο νέφος είναι ένα τύπος δοκιμής λογισμικού, κατά τον οποίο γίνεται η δοκιμή μίας εφαρμογής νέφους και των υπηρεσιών από τις οποίες αποτελείται. Η διαδικασία αυτή μπορεί να αναφερθεί και σαν «Δοκιμή ως υπηρεσία» (TaaS - Testing as a Service), ο σκοπός της οποίας είναι η διαβεβαίωση ότι η υπηρεσία που βρίσκεται υπό δοκιμή ανταποκρίνεται στις ανάγκες και τους περιορισμούς που της έχουν τεθεί. Οι ανάγκες και οι περιορισμοί μιας υπηρεσίας μπορούν να είναι πάνω στην κλιμακωσιμότητα, στην απόδοση, στην αξιοπιστία και στην ασφάλειά της.

2.4.2 Τύποι δοκιμών

Όπως είναι γνωστό, κάθε τύπος λογισμικού είναι απαραίτητο να περνάει από διάφορα επίπεδα και τύπους δοκιμών. Οι δύο βασικές κατηγορίες δοκιμών είναι οι λειτουργικές και οι μη λειτουργικές που ελέγχουν την λειτουργικότητα αλλά και την ποιότητα του λογισμικού. Όπως και με κάθε εφαρμογή, πέρα από τις λειτουργικές δοκιμές, είναι σημαντικό να πραγματοποιούνται και οι μη λειτουργικές, καθώς διαβεβαιώνουν την καλή ποιότητα του προϊόντος και την όσο το δυνατόν καλύτερη εμπειρία του τελικού χρήστη. Οι δύο παραπάνω κατηγορίες δοκιμών μπορούν να πραγματοποιηθούν με αρκετούς τρόπους και διαφορετικές μεθόδους που παρουσιάζονται παρακάτω.

2.4.2.1 Λειτουργικές δοκιμές

Οι λειτουργικές δοκιμές ελέγχουν και διαβεβαιώνουν ότι μία εφαρμογή ή υπηρεσία παρέχει σωστά όλη την λειτουργικότητα της. Τέτοιου είδους δοκιμές εκτελούνται σε μεμονωμένες υπηρεσίες αλλά και σε μία σύνθεση υπηρεσιών.

- Το **Unit Testing** ή αλλιώς δοκιμή μονάδας, είναι ένα είδος δοκιμών που πραγματοποιείται από τους προγραμματιστές που γράφουν τον πηγαίο κώδικα για να αναπτύξουν μία εφαρμογή ή υπηρεσία. Οι δοκιμές αυτές, όπως υποδηλώνει και το όνομα, έχουν σκοπό να ελέγχουν μία μονάδα της εφαρμογής. Καθώς κάθε εφαρμογή αποτελείται από αρκετές διακριτές λειτουργικότητες που αλληλεπιδρούν, κάθε μία από αυτές πρέπει να περάσει από την διαδικασία της δοκιμής. Τελειώνοντας την διαδικασία, μπορούμε να ελέγξουμε τα αποτελέσματα για να διαβεβαιώσουμε ότι όλα λειτουργούν όπως πρέπει, ή να εντοπίσουμε πιθανά σφάλματα κατά την υλοποίηση των λειτουργικοτήτων.
- Το **Integration Testing** διαφέρει από την δοκιμή μονάδας, καθώς πραγματοποιείται σε μία σύνθεση υλοποιημένων λειτουργιών ή υπηρεσιών. Σκοπός του είναι να ελέγχει και να διαβεβαιώνει πως η αλληλεπίδραση και η σύνθεση μεταξύ των λειτουργιών ή υπηρεσιών λειτουργεί ορθά και ότι δεν προκύπτουν σφάλματα κατά την εκτέλεση. Τέτοιες δοκιμές μπορούν να εκτελεστούν για διάφορους τύπους εφαρμογών, όπως τον έλεγχο μιας διεπαφής χρήστη (UI), την κλήση ενός API, τον έλεγχο μίας σύνθεσης υπηρεσιών, κτλ.
- Το **Acceptance Testing** μπορεί να θεωρηθεί ως η επίσημη δοκιμή ολόκληρου του συστήματος. Οι δοκιμές αυτές πραγματοποιούνται από μερικούς επιλεγμένους χρήστες που θα είναι και οι χρήστες του τελικού συστήματος. Οι χρήστες παραλαμβάνουν το σύνολο του συστήματος για να το δοκιμάσουν, χρησιμοποιώντας το σε πραγματικές συνθήκες. Οι χρήστες, στο τέλος της διαδικασίας της δοκιμής και βάσει των απαιτήσεων και των προσφερόμενων λειτουργιών, αποφασίζουν αν το σύστημα είναι ολοκληρωμένο και επιτυχές.

2.4.2.2 Μη λειτουργικές δοκιμές

Οι μη λειτουργικές δοκιμές επικεντρώνονται στον έλεγχο των παροχών και των χαρακτηριστικών του νέφους

- **Δοκιμή Διαθεσιμότητας - Availability Testing**: Ακολουθώντας την λογική της παροχής υπηρεσιών μέσω νέφους, πρέπει να βεβαιωθούμε για

την συνεχόμενη διαθεσιμότητα τους. Τα παραπάνω είναι ευθύνη του παρόχου του περιβάλλοντος νέφους.

- **Δοκιμή Ασφάλειας - Security Testing**: Η ασφάλεια των δεδομένων είναι καίριας σημασίας για την πλειοψηφία των εφαρμογών. Είναι αναγκαίο να ακολουθούνται οι τρεις βασικές αρχές της ασφάλειας:
    - ***Εμπιστευτικότητα*** (confidentiality): Οι πληροφορίες και τα δεδομένα δεν πρέπει να είναι προσβάσιμες/α σε μη εξουσιοδοτημένους χρήστες.
    - ***Ακεραιότητα*** (integrity): Πρέπει να βεβαιώνεται η ακρίβεια, η ακεραιότητα και η γνησιότητα των πληροφοριών και των δεδομένων, δηλαδή οι πληροφορίες και τα δεδομένα να μην είναι εσφαλμένα, αλλοιωμένα ή μη ενημερωμένα.
    - ***Διαθεσιμότητα*** (availability): Οι πληροφορίες και τα δεδομένα πρέπει να είναι διαθέσιμα στου χρήστες όποτε οι ίδιοι το απαιτούν
- **Δοκιμή Απόδοσης - Performance Testing**: Η μέτρηση της αποδοτικότητας μιας εφαρμογής νέφους είναι διαφορετική από αυτή μίας τοπικής παραδοσιακής εφαρμογής. Η διάθεση των πόρων του νέφους πρέπει να είναι ελαστική, δηλαδή να εξαρτάται από την ζήτηση που έχει μία εφαρμογή. Ξεκινώντας, οι πόροι πρέπει να είναι περιορισμένοι, έτσι ώστε να δεσμεύονται όσο το δυνατόν λιγότεροι για κάθε εφαρμογή. Έπειτα από μία αύξηση ζήτησης, το νέφος πρέπει να κατανέμει περισσότερους πόρους προς την εφαρμογή, έτσι ώστε να μην επηρεάζεται η απόδοση και η απόκρισή της από την αυξημένη ζήτηση. Καθώς η ζήτηση κάθε εφαρμογής είναι μεταβαλλόμενη και παρουσιάζει διακυμάνσεις, οι διαθέσιμοι πόροι πρέπει να αυξομειώνονται παράλληλα και με ανάλογο ρυθμό. Για την δοκιμή της απόδοσης μίας εφαρμογής νέφους, χρησιμοποιούνται δύο μέθοδοι:
    - Δοκιμή φορτίου - Load testing
    - Δοκιμή πίεσης - Stress testing
- **Δοκιμή Διαλειτουργικότητας - Interoperability Testing**: Ένα από τα βασικά μη λειτουργικά χαρακτηριστικά μίας εφαρμογής, είναι η διαλειτουργικότητά της. Για να θεωρείται διαλειτουργική μία εφαρμογή, θα πρέπει να είναι ικανή να εκτελείται και να λειτουργεί κανονικά και

χωρίς σφάλματα σε διαφορετικά περιβάλλοντα, καθώς και σε διαφορετικές πλατφόρμες νέφους. Εκτός αυτού, η μεταφορά και εγκατάσταση της εφαρμογής θα πρέπει να είναι όσο το δυνατόν ευκολότερη.

- **Δοκιμή Αποκατάστασης από Καταστροφές - Disaster Recovery Testing**: Όπως αναφέραμε και παραπάνω, μία εφαρμογή νέφους περνάει από δοκιμές Διαθεσιμότητας, έτσι ώστε να επιτευχθεί η όσο το δυνατόν καλύτερη προσφορά της. Παρόλα αυτά, η προσφορά της διαθεσιμότητας σε ποσοστό εκατό τοις εκατό δεν είναι εφικτή, ακόμα και όταν μιλάμε για παραδοσιακές τοπικές εφαρμογές. Επομένως, στην περίπτωση που μία εφαρμογή νέφους παρουσιάσει πρόβλημα διαθεσιμότητας, πρέπει να την επαναφέρουμε όσο το δυνατόν γρηγορότερα, διαβεβαιώνοντας πως δεν υπήρξαν απρόοπτες επιδράσεις στα δεδομένα και την λειτουργία της.

2.5 Διάθεση λογισμικού στο νέφος (Deployment)

Στο τέλος της διαδικασίας ανάπτυξης ενός λογισμικού, είναι απαραίτητη η προετοιμασία του έτσι ώστε να καταστεί λειτουργικό. Η υλοποίηση μίας υπηρεσίας, έχει ως απώτερο σκοπό την χρήση της από τους τελικούς χρήστες. Για να γίνει αυτό, πρέπει ο κώδικας να πακεταριστεί σε εκτελέσιμο αρχείο (build - package) και να διατεθεί μέσω δικτύου στο νέφος (deployment).

2.5.1 Πακετάρισμα κώδικα

Το πακετάρισμα κώδικα αναφέρεται στην δημιουργία ενός εκτελέσιμου τελικού προϊόντος. Είναι πολύ σημαντικό για την ευχρηστία ενός λογισμικού, να μπορεί εύκολα κάποιος να κατεβάσει ένα ολοκληρωμένο πακέτο έτοιμο για χρήση το οποίο είναι άμεσα λειτουργικό. Επομένως, η διαδικασία δημιουργίας του πακέτου περιλαμβάνει την συμπερίληψη όλων των απαραίτητων εργαλείων και τεχνολογιών που χρειάζονται για την εκτέλεση και τη σωστή λειτουργία του κώδικα.

Αρχικά, είναι απαραίτητο ο κώδικας που έχουμε αναπτύξει να είναι εκτελέσιμος, επομένως στην περίπτωση της Java, πρέπει να χτίσουμε ένα αρχείο jar (Java Archive). Η δημιουργία ενός αρχείου jar, μπορεί να γίνει με διάφορους τρόπους:

- Παραδοσιακός [34]:

Η δημιουργία ενός αρχείου jar, μπορεί να γίνει με την χρήση περιβάλλοντος εντολών. Αυτή η μέθοδος απαιτεί την χειροκίνητη δημιουργία των απαραίτητων

αρχείων, π.χ. Manifest για την σωστή λειτουργία του Java Archive. Επιπρόσθετα, πρέπει να συμπεριληφθούν όλες οι εξαρτήσεις του κώδικα, επίσης με χειρωνακτικό τρόπο. Μπορούμε να καταλάβουμε πως μία τέτοια διαδικασία είναι ιδιαίτερα απαιτητική για τον χρήστη και πολύ ευάλωτη σε λάθη καθώς περιλαμβάνει πολλά βήματα.

- Μέσω εργαλείου:

Τα μοντέρνα εργαλεία συγγραφής κώδικα, όπως το NetBeans ή το Eclipse, συνήθως παρέχουν λειτουργικότητα πακεταρίσματος κώδικα. Στην ουσία, πραγματοποιούν αυτόματα πολλές από τις απαραίτητες ενέργειες, διευκολύνοντας τον χρήστη στην δημιουργία. Αυτή η μέθοδος αποτελεί σημαντική βελτίωση από την προηγούμενη, αλλά πάλι απαιτεί βήματα από τον χρήστη. Το κυριότερο εμπόδιο σε αυτή την διαδικασία είναι η διαχείριση των βιβλιοθηκών, που πρέπει να γίνει από τον χρήστη πριν την εκτέλεση και πακετάρισμα του κώδικα.

- Εργαλεία αυτοματισμού κατασκευής:

Για την περαιτέρω διευκόλυνση του χρήστη, αλλά και την σωστή διαχείριση των βιβλιοθηκών ενός έργου, έχουν δημιουργηθεί πολλά εργαλεία. Τα δύο πιο διαδεδομένα εργαλεία για την χρήση με την γλώσσα Java, είναι το Maven και το Gradle. Αυτά, στην ουσία λειτουργούν με την χρήση ενός αρχείου, στο οποίο ο προγραμματιστής δηλώνει τα χαρακτηριστικά του κώδικα του, δηλαδή το όνομα της κύριας κλάσης, την έκδοση Java του κώδικα, τις εξαρτήσεις, κ.α. Έπειτα, χρησιμοποιώντας το γραφικό περιβάλλον των εργαλείων που προαναφέραμε (Eclipse Theia, NetBeans, κτλ.) ή με την χρήση εντολών, γίνεται αυτόματα η λήψη των εξαρτήσεων, η παραμετροποίηση των απαραίτητων αρχείων και το χτίσιμο της εφαρμογής σε Java Archive.

2.5.2 Διάθεση λογισμικού

Η διάθεση λογισμικού, είναι ένα κύριο κομμάτι της συνολικής διαδικασίας. Στην ουσία πρόκειται για το τελικό στάδιο στην διαδικασία ανάπτυξης ενός έργου, στο οποίο η λειτουργικότητα γίνεται πλέον διαθέσιμη στους χρήστες του. Για την διάθεση του, ένα λογισμικό πρέπει να παρέχεται μέσο ενός εξυπηρετητή (Server). Ο εξυπηρετητής είναι ένα υπολογιστικό μηχάνημα, πάνω στον οποίο τρέχουν διάφορα εργαλεία και υπηρεσίες. Για την διάθεση υπηρεσιών μέσω δικτύου, θα πρέπει το λογισμικό να φιλοξενηθεί μέσα σε έναν διακομιστή εφαρμογών (application server). Ένας διακομιστής εφαρμογών είναι ο μεσάζοντας της πλατφόρμας που επικοινωνεί από την μία με το λειτουργικό σύστημα

και από την άλλη με το διατιθέμενο λογισμικό και τα συστήματα του (βάσεις δεδομένων, εργαλεία, κτλ).

2.5.2.1 Διακομιστές εφαρμογών

Ανάλογα με την τεχνολογία που χρησιμοποιήθηκε για την ανάπτυξη του λογισμικού, παρέχονται και εξειδικευμένα εργαλεία διάθεσης.

Wildfly

Το Wildfly, που παλαιότερα λεγόταν JBoss, είναι ένας διακομιστής εφαρμογών ανοιχτού κώδικα που υποστηρίζεται από την Red Hat. Το Wildfly παρέχει υποστήριξη για την διάθεση Java εφαρμογών καθώς και την ύπαρξη γραφικού περιβάλλοντος. Η λειτουργικότητα του Wildfly ικανοποιεί την περίπτωση χρήσης Java EE.

Apache Tomcat

Το Apache Tomcat παρέχει ένα περιβάλλον εκτέλεσης κώδικα Java και είναι επίσης λογισμικό ανοιχτού κώδικα. Συνήθως συνδυάζεται με την χρήση εφαρμογών που υλοποιούν το ολοκληρωμένο πλαίσιο Spring. Το Apache Tomcat όπως και το Wildfly, παρέχουν υποστήριξη αποκλειστικά για την διάθεση υπηρεσιών Java.

Docker

Το Docker είναι μία ολοκληρωμένη πλατφόρμα πακεταρίσματος (Docker Images) και διάθεσης λογισμικού. Το Docker φιλοξενεί πακεταρισμένο λογισμικό μέσα σε Docker containers. Ένα container περιλαμβάνει το εκτελέσιμο αρχείο κώδικα μαζί με τις απαραίτητες τεχνολογίες για την εκτέλεσή του. Επιπρόσθετα, μέσα στο container, υπάρχει και ένα απομονωμένο περιβάλλον εκτέλεσης, το οποίο στην ουσία είναι η περιοχή εκτέλεσης και λειτουργίας του λογισμικού. Ως αποτέλεσμα, ένα container είναι εντελώς απομονωμένο και ανεξάρτητο από τον εξυπηρετητή, το λειτουργικό του, καθώς και τις λεπτομέρειες υλοποίησης του κώδικα. Εφόσον ένα λογισμικό έχει πακεταριστεί σε ένα Docker Image, μπορεί και να διατεθεί μέσα από ένα Docker container.

Kubernetes

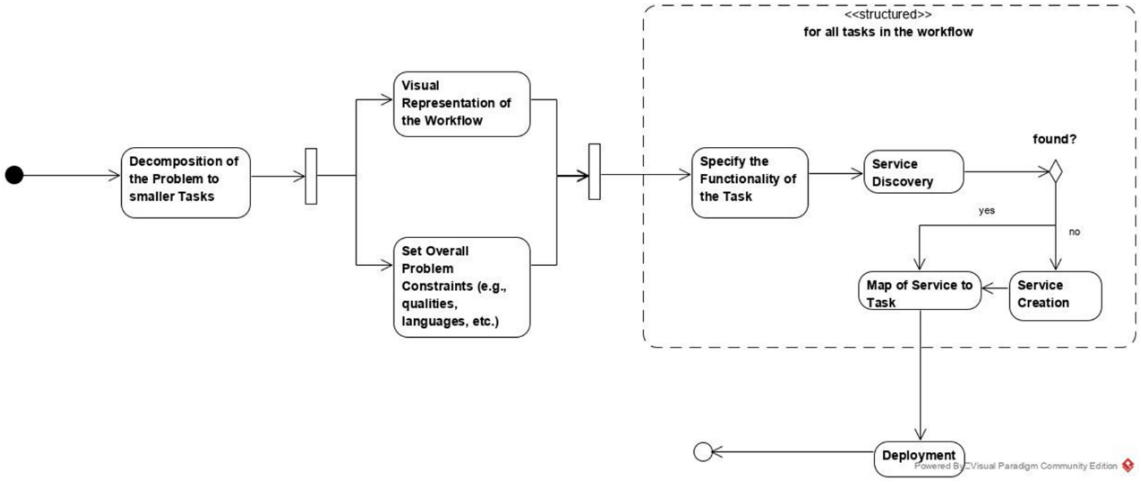
Το Kubernetes χρησιμοποιεί την τεχνολογία του Docker, καθώς διαθέτει τα Docker Images. Επίσης, χτίζει πάνω στην τεχνολογία του Docker και προσφέρει μία πολύ λεπτομερή, δυναμική και δυνατή διαχείριση των Docker container που διαθέτει. Το

Kubernetes είναι ο πλέον διαδεδομένος διακομιστής υπηρεσιών, καθώς οι δυνατότητες που προσφέρει μπορούν να ικανοποιήσουν της ανάγκες διάθεσης μικρών έως και τεράστιων συνόλων υπηρεσιών.

3 Μεθοδολογία

Σε αυτή την ενότητα, παρουσιάζουμε την μεθοδολογία της προσέγγιση μας για την δημιουργία και τη σύνθεση υπηρεσιών λογισμικού, όπως αναπτύχθηκε στα πλαίσια του ευρωπαϊκού έργου SmartCLIDE. Με το τελικό εργαλείο στοχεύουμε στην βελτίωση, στην διευκόλυνση και στην επιτάχυνση της διαδικασίας της εκ του μηδενός δημιουργίας μίας εφαρμογής που χρησιμοποιεί αρχιτεκτονική μικροϋπηρεσιών. Η προσέγγιση στοχεύει στην αυτοματοποίηση όσο το δυνατόν περισσότερων βημάτων και στην προσφορά λειτουργιών στον χρήστη, με τρόπο που δεν αποσπούν την προσοχή του με χρονοβόρες αλληλεπιδράσεις.

Η προσέγγιση βασίζεται στην γενική περιγραφή των βημάτων που παρουσιάστηκαν στην ενότητα της βιβλιογραφικής επισκόπησης. Τα βήματα της προσέγγισης παρουσιάζονται πιο αναλυτικά στο παρακάτω διάγραμμα ροής.



Εικόνα 5: Βήματα προσέγγισης δημιουργίας μιας εφαρμογής SOA

3.1 Σύνθεση Υπηρεσιών

Ακλουθώντας την λογική του προγραμματισμού χρησιμοποιώντας περιορισμένο κώδικα, η απόφαση που ελήφθη είναι η χρήση του προτύπου BPMN 2.0. Βάσει των λειτουργιών που προσφέρει, κρίθηκε ως η ιδανική λύση για την σύνθεση και ενορχήστρωση ετερογενών υπηρεσιών που πιθανώς να προέρχονται από διαφορετικές πηγές. Το αποτέλεσμα θα αποτελεί μία σύνθεση υπηρεσιών διαθέσιμων μέσω δικτύου, η οποία θα καλύπτει μία επιχειρησιακή ανάγκη, προσφέροντας μία συνολική λειτουργικότητα. Το παραπάνω γίνεται δυνατό με την διαχείριση των αυτόνομων

υπηρεσιών ως εκτελεστές λειτουργιών των Service Task ενός BPMN workflow και με την διάταξη και σύνδεση τους μέσω ενός γραφικού περιβάλλοντος, έτσι ώστε με την αλληλεπίδρασή τους να παράγουν το αναμενόμενο αποτέλεσμα

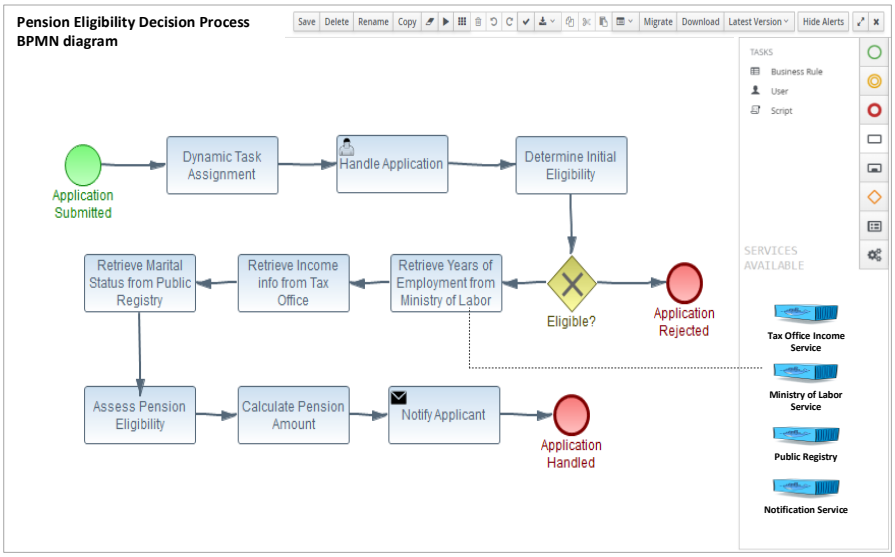
3.1.1 Επιλεγμένη Λύση

Καθώς η γλώσσα BPMN προσφέρει πάρα πολλές λειτουργίες, η υλοποίηση ενός εργαλείου που να την χρησιμοποιεί είναι ιδιαίτερα δύσκολη, επομένως χρειάστηκε να βρούμε μία έτοιμη υλοποίηση. Καθώς στα πλαίσια του SmartCLIDE χρησιμοποιούμε λογισμικά ανοιχτού κώδικα, καταλήξαμε στην χρήση του jBPM για την διαχείριση και ενορχήστρωση υπηρεσιών. Το jBPM είναι ένα εργαλείο που προσφέρει πλήρη έλεγχο και υποστήριξη για την δημιουργία και την εκτέλεση μιας σύνθεσης υπηρεσιών και είναι διαθέσιμο σε δύο μορφές.

Μία επιλογή είναι το Kogito, που αποτελεί μια σχετικά νέα υλοποίηση ακέφαλης (headless) εφαρμογής νέφους (δηλαδή χωρίς διεπαφή χρήστη), που περιλαμβάνει τον μηχανισμό εκτέλεσης συνθέσεων. Καθώς το Kogito δεν έχει διεπαφή χρήστη, ένα ξεχωριστό περιβάλλον σχεδίασης θα πρέπει να διατεθεί για αυτόν τον σκοπό. Για παράδειγμα θα ήταν δυνατό να υποστηριχτεί η σχεδίαση μέσω του Eclipse Theia, εφόσον υπάρχει διαθέσιμη ή αναπτυχθεί κατάλληλη επέκταση για αυτόν τον σκοπό. Μιας και το Kogito είναι ακόμα στην αρχική Άλφα μορφή του, δεν περιέχει όλες τις απαραίτητες λειτουργίες που αναζητούμε για το έργο.

Η δεύτερη επιλογή, η οποία και υιοθετήθηκε, είναι το ολοκληρωμένο πλαίσιο του jBPM Workbench. Το Workbench περιλαμβάνει τον μηχανισμό εκτέλεσης συνθέσεων, και σε αντίθεση με το Kogito, περιέχει και διεπαφή χρήστη για την υποστήριξη σχεδίασης και διαχείρισης των συνθέσεων υπηρεσιών. Η επιλογή αυτή έγινε καθώς το jBPM Workbench διαθέτει άδεια χρήσης Apache License 2.0, ικανοποιεί πολλές από τις απαιτήσεις μας και αποτελεί ένα από τα πιο διαδεδομένα και ώριμα εργαλεία σύνθεσης υπηρεσιών.

3.1.2 JBPM Workbench - Business Central



Εικόνα 6: Παράδειγμα σύνθεσης υπηρεσιών χρησιμοποιώντας το διαδικτυακό περιβάλλον του JBPM Business Central

Η ανάπτυξη των συνθέσεων υπηρεσιών πραγματοποιείται χρησιμοποιώντας τον διαδικτυακά διαθέσιμο καμβά του Business Central. Ο καμβάς υποστηρίζει drag-n-drop χρήση για την προσθήκη νέων κόμβων όπως User Tasks, Start/End nodes και Decision Gateways στην σύνθεση, βελτιώνοντας αρκετά την εμπειρία χρήσης της εφαρμογής για τους χρήστες. Μετά την ανάπτυξη της σύνθεσης, είναι δυνατόν να ελεγχθούν οι συνδέσεις και οι κόμβοι και το κατά πόσο λειτουργεί σωστά το σύνολό τους

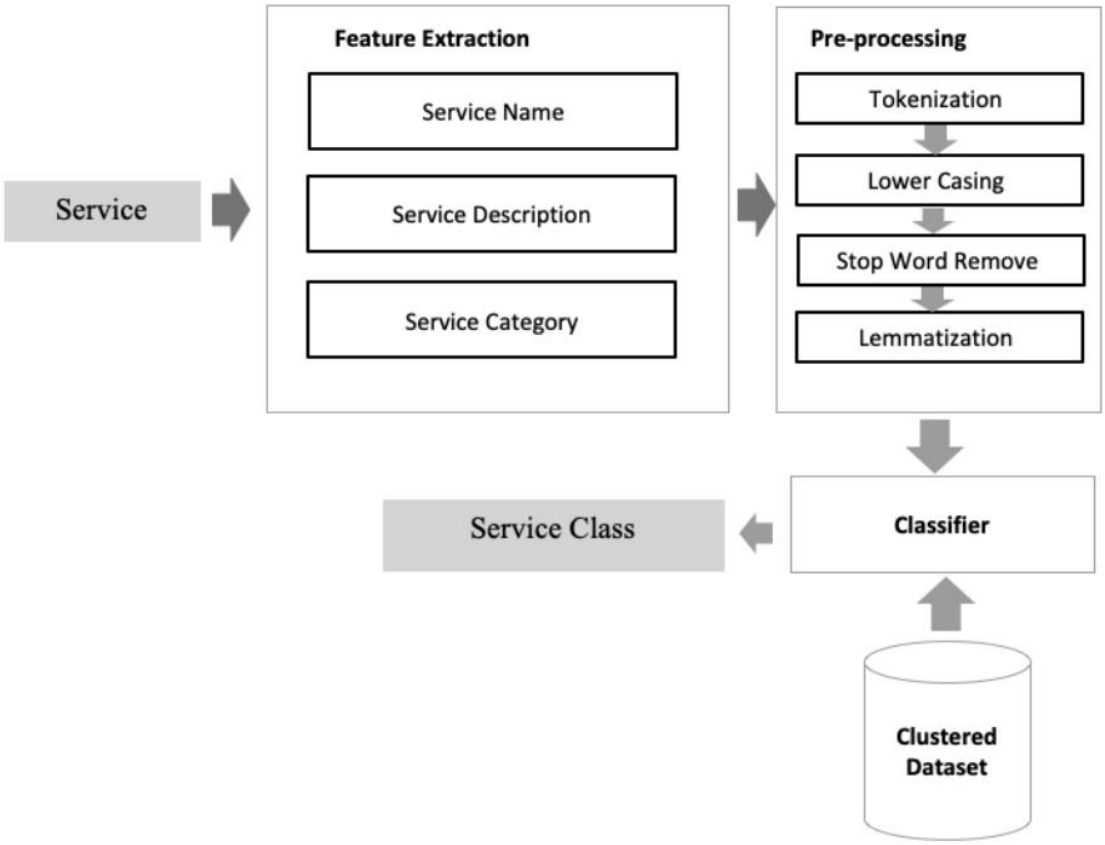
Καθώς στα πλαίσια του SmartCLIDE επιδιώκουμε την επαναχρησιμοποίηση ήδη υπαρχόντων υπηρεσιών, ένας χρήστης θα πρέπει να μπορεί να ψάξει και να εντοπίσει υπηρεσίες μέσω της πλατφόρμας. Επομένως, θα πρέπει να υλοποιήσουμε μία κατάλληλη επέκταση για το περιβάλλον του Business Central, που να υποστηρίζει την διαδικασία εύρεσης υπηρεσιών. Για την περίπτωση που δεν βρεθεί κατάλληλη υπηρεσία, ο χρήστης θα καλείται να δημιουργήσει μία νέα εκ του μηδενός, επομένως είναι αναγκαία η προσθήκη λειτουργίας που να καλύπτει και αυτή την περίπτωση. Η διαδικασία της Εύρεσης και της Δημιουργίας υπηρεσιών, αναλύονται σε επόμενες ενότητες.

3.2 Κατάταξη Υπηρεσιών

Ο σκοπός της κατάταξης ενός πλήθους υπηρεσιών, είναι η κατηγοριοποίησή τους βάσει ενός συνόλου προκαθορισμένων κλάσεων, έτσι ώστε να είναι εφικτή η γρηγορότερη αναζήτηση τους. Οι Alizadehsani κ.ά. [35], για το πρόβλημα της κατάταξης, πρότειναν μία προσέγγιση που περιέχει πολλαπλά βήματα:

1. Η παροχή του συνόλου των δεδομένων
2. Η επεξεργασία φυσικής γλώσσας (NLP) των δεδομένων
3. Η ομαδοποίηση
4. Η κατάταξη των δεδομένων κειμένου
5. Η αξιολόγηση του αποτελέσματος

Στο παρακάτω διάγραμμα, απεικονίζεται η διαδικασία με μεγαλύτερη λεπτομέρεια, καθώς συμπεριλαμβάνονται και κάποιες εσωτερικές λειτουργίες των πέντε βημάτων.



Εικόνα 7: Υβριδική μέθοδος κατάταξης

3.2.1 Εξαγωγή χαρακτηριστικών

Μέσω της επεξεργασίας φυσικής γλώσσας κειμένου[36], μπορούμε να μετατρέψουμε δεδομένα κειμένου σε αριθμητικές τιμές. Η μετατροπή μπορεί να γίνει με διάφορους τρόπους, όπως:

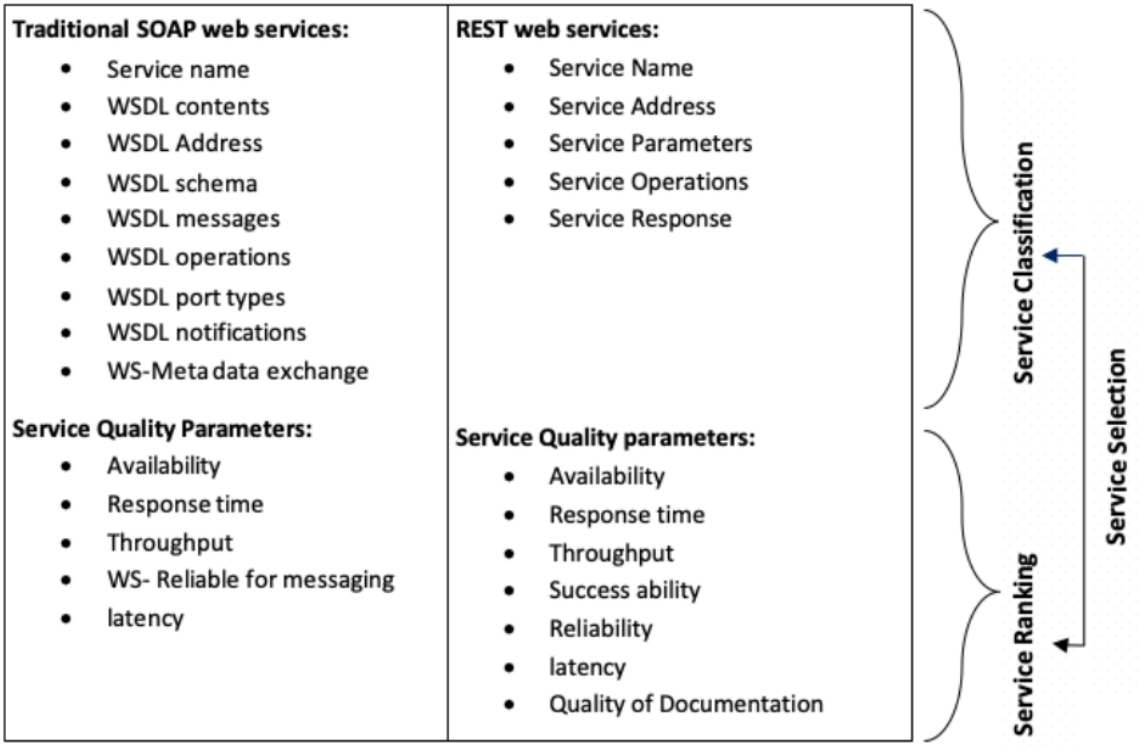
1. Το μοντέλο Bag of Words (BOW) όπου το κάθε κείμενο εισόδου, αναπαριστάται σαν ένα σύνολο αριθμών, ανάλογα με την συχνότητα εμφάνισης λέξεων σε αυτό.
2. Το Word Embedding, που αποτελεί την αναπαράσταση ενός κειμένου χρησιμοποιώντας διανύσματα. Το σημαντικό πλεονέκτημα αυτής της μεθόδου, είναι η αναπαράσταση ενός πολυδιάστατου κειμένου σε μορφή που είναι πιο διαχειρίσιμη.

Ωστόσο, ο όγκος των δεδομένων που συλλέχθηκαν για την μελέτη δεν είναι μεγάλος, επομένως μπορεί να χρησιμοποιηθεί η μέθοδος Bag of Words. Για την υλοποίηση της, χρησιμοποιήθηκαν οι διαδεδομένες βιβλιοθήκες της γλώσσας python, NLTK (Natural Language Toolkit) και Sklearn [37].

Κατά την διάρκεια της επιλογής μίας υπηρεσίας, μπορούν να χρησιμοποιηθούν διάφορα χαρακτηριστικά της. Ως επακόλουθο, αυτά μπορούν να αξιοποιηθούν και κατά την διαδικασία κατάταξής της. Τα σημαντικότερα από αυτά μπορούν να εξαχθούν από τις περιγραφές των REST API και της WSDL.

Η χρήση της WSDL είναι ευκολότερη, καθώς χρησιμοποιείται για την περιγραφή της λειτουργικότητας υπηρεσιών. Η χρήση περιγραφών REST απαιτεί περισσότερη επεξεργασία για την εξαγωγή χρήσιμης πληροφορίας, καθώς το REST υποστηρίζει μεγαλύτερη πληθώρα μορφών για τα δεδομένα

περιγραφής.



Εικόνα 8: Χαρακτηριστικά υπηρεσιών

3.2.2 Επεξεργασία δεδομένων

Τα δεδομένα των υπηρεσιών που θα αντληθούν για την εκπαίδευση του μοντέλου, θα πρέπει πρώτα να περάσουν από διαδικασία επεξεργασίας, έτσι ώστε να μετατραπούν σε χρήσιμη πληροφορία. Ένα σύνολο δεδομένων συλλέχθηκε από το Programmableweb, το οποίο είναι αποθετήριο υπηρεσιών και περιέχει δεδομένα από πραγματικές υπηρεσίες που είναι διαθέσιμες προς κατανάλωση.

Προτού προχωρήσουμε στην διαδικασία της κατάταξης, πρέπει να μελετήσουμε τα δεδομένα που έχουμε για κάθε υπηρεσία. Η μορφή των δεδομένων που συλλέχθηκαν είναι σε απλό κείμενο, επομένως είναι αναγκαία η επεξεργασία τους. Η επεξεργασία πρέπει να γίνει, έτσι ώστε να υπάρχει ομοιογένεια στα δεδομένα, με αποτέλεσμα να είναι εφικτή η ομαδοποίηση τους. Για να επιτευχθεί η επεξεργασία του κειμένου χρησιμοποιούνται μέθοδοι επεξεργασία φυσικής γλώσσας (NLP).

Πίνακας 3 Σύνολο δεδομένων

Δεδομένα	Πλήθος εγγραφών	Κατηγορίες	Ποσοστό δεδομένων εκπαίδευσης	Ποσοστό δεδομένων δοκιμής	Επεξεργασία
Αρχικά	6535	403	-	-	-
Ομαδοποιημένα	4645	24	80	20	

Κατά την συλλογή μεταδιδομένων υπηρεσιών, παρατηρούμε μερικά προβλήματα. Δεδομένης της έλλειψης κανόνων κατά την δημιουργία υπηρεσιών, κάθε προγραμματιστής είναι ελεύθερος να την αναπτύξει όπως θέλει. Αυτό έχει ως επακόλουθο να υπάρχουν διαφορές στην λογική ονομασίας διαφόρων υπηρεσιών. Το ίδιο πρόβλημα συναντάται και κατά την ονομασία κλάσεων και μεθόδων. Επομένως, δεν είναι εφικτή η αυτόματη εξαγωγή πληροφορίας από τα δεδομένα των υπηρεσιών, χρησιμοποιώντας ένα προκαθορισμένο μοτίβο.

Όπως αναφέραμε, τα δεδομένα που συλλέχθηκαν αφορούν 6535 εγγραφές, χωρισμένες σε 403 διαφορετικές κατηγορίες. Για την μείωση του αριθμού των κατηγοριών ακολουθήθηκαν κάποια βήματα:

1. Επιλογή κατηγοριών που περιέχουν περισσότερες από 30 εγγραφές
2. Χρήση μεθόδου Bag of Words, ώστε να διατεθούν αριθμητικά δεδομένα στον αλγόριθμο ομαδοποίησης
3. Αφαίρεση των πιο συχνά εμφανιζόμενων λέξεων
4. Ιεραρχική ομαδοποίηση για την περαιτέρω μείωση των κλάσεων

Τα δεδομένα που έχουν εξαχθεί από τα προηγούμενα βήματα, θα χρησιμοποιηθούν για την εκπαίδευση ταξινομητών, έτσι ώστε να είναι δυνατή η κατάταξη υπηρεσιών χρησιμοποιώντας το μοντέλο που προτάθηκε.

3.2.3 Σύνολο δεδομένων

Το σύνολο δεδομένων που χρησιμοποιείται για την εκπαίδευση ενός μοντέλου μηχανικής μάθησης πρέπει να είναι ισορροπημένο, έτσι ώστε η διαδικασία εκπαίδευσης να είναι αποτελεσματική. Παρόλα αυτά, το σύνολο δεδομένων που συλλέχτηκε για την εκπαίδευση του μοντέλου ταξινόμησης είναι σημαντικά λοξό. Αυτό συμβαίνει καθώς τα δεδομένα αντιπροσωπεύουν πολλές διαφορετικές υπηρεσίες. Κάθε υπηρεσία εκτελεί

διαφορετικές λειτουργίες και εντάσσεται σε πολλούς διαφορετικούς τομείς, όπως για παράδειγμα στα οικονομικά, στον καιρό, στην ιατρική, στην ασφάλεια, κ.α. Δεδομένου ότι κάποιοι τομείς έχουν μεγαλύτερη κάλυψη και προσφορά από υπηρεσίες, όπως π.χ. οι βάσεις δεδομένων, αυτό έχει ως αποτέλεσμα να υπερνικούν των υπολοίπων λιγότερο διαδεδομένων τομέων.

3.3 Εύρεση Υπηρεσιών

Η διαδικασία της εύρεσης υπηρεσιών είναι ιδιαιτέρα σημαντική, καθώς διευκολύνει τον προγραμματιστή κατά την σχεδίαση του διαγράμματος BPMN. Η μέθοδος που ακολουθήθηκε για την εύρεση υπηρεσιών στην πλατφόρμα του SmartCLIDE περιέχει πολλά διαφορετικά δομικά κομμάτια. Η εύρεση υπηρεσιών είναι υπεύθυνη για την συλλογή υπηρεσιών από διάφορες διαδικτυακές πηγές. Μετά την συλλογή, οι υπηρεσίες περνούν από την διαδικασία της κατάταξης που προαναφέραμε και τέλος, αποθηκεύονται σε μία εσωτερική βάση. Η αποθήκευση των πλέον καταταγμένων υπηρεσιών, έχει ως αποτέλεσμα την πιο εύκολη και γρήγορη αναζήτηση τους.

Κατά την αίτηση ενός χρήστη για την εύρεσης μίας υπηρεσίας, η αναζήτηση γίνεται εντός της εσωτερική βάσης του SmartCLIDE. Αν αναζήτηση δεν αποφέρει αποτέλεσμα, τότε πραγματοποιείται αναζήτηση και σε εξωτερικές πηγές.

Η βασικές απαιτήσεις για την δομή είναι οι παρακάτω:

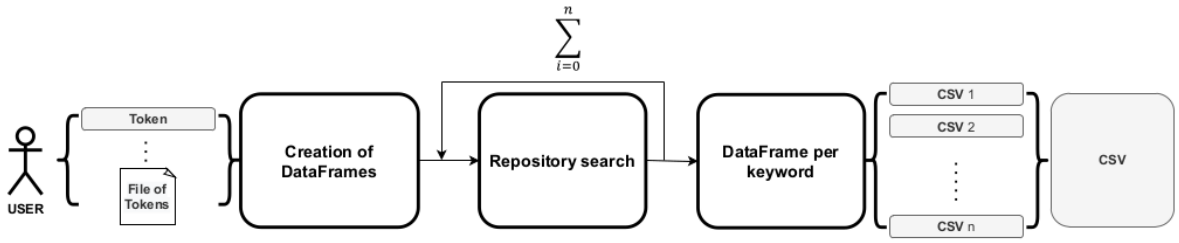
1. Η ύπαρξη ενός εσωτερικού αποθηκευτικού χώρου που υποστηρίζει την ελαστική αναζήτηση για την αποθήκευση υπηρεσιών.
2. Η αναζήτηση να γίνεται βάσει της περιγραφής μιας υπηρεσίας και προαιρετικά χρησιμοποιώντας τα χαρακτηριστικά της και την κατηγορία στην οποία ανήκει.
3. Αν η κατηγορία στην οποία ανήκει η υπηρεσία δεν έχει δοθεί ως είσοδος της αναζήτησης, εξάγεται μία μέσω της επεξεργασίας της περιγραφή με την χρήση της μεθόδου κατάταξης
4. Η αναζήτηση πρώτα πραγματοποιείται στην εσωτερική βάση υπηρεσιών και στην περίπτωση που αποτύχει, αναζητά σε εξωτερικές πηγές.
5. Οι υπηρεσίες κάθε κατηγορίας βαθμολογούνται βάσει των χαρακτηριστικών τους, π.χ. αριθμός λήψεων, αστέρια, κ.α.

6. Η επιστροφή υπηρεσιών μετά το αίτημα εύρεσης, αν υπάρχουν διαθέσιμες, επιστρέφει το όνομα, το κώδικα, την διεύθυνση κτλ. της υπηρεσίας.

3.3.1.1 Ανιχνευτές

Ο κύριος στόχος που εξυπηρετεί η διαδικασία της Εύρεσης, είναι η δημιουργία και το γέμισμα μιας εσωτερικής βάσης δεδομένων υπηρεσιών για την μετέπειτα αναζήτηση και εύρεση τους από αυτή. Επίσης, επιβλέπει την διαδικασία συλλογής πληροφοριών από τους ανιχνευτές, οι οποίοι ερευνούν το διαδίκτυο και τα αποθετήρια υπηρεσιών, είτε χρησιμοποιώντας scraping είτε με την χρήση API, και δημιουργούν λίστες υπηρεσιών.

Η δημιουργία των πλαισίων δεδομένων αναλύεται παρακάτω με βάση τις πληροφορίες που μπορούν να ληφθούν χρησιμοποιώντας τις διαφορετικές διαθέσιμες μεθόδους.



Εικόνα 9: Εξαγωγή πληροφορίας

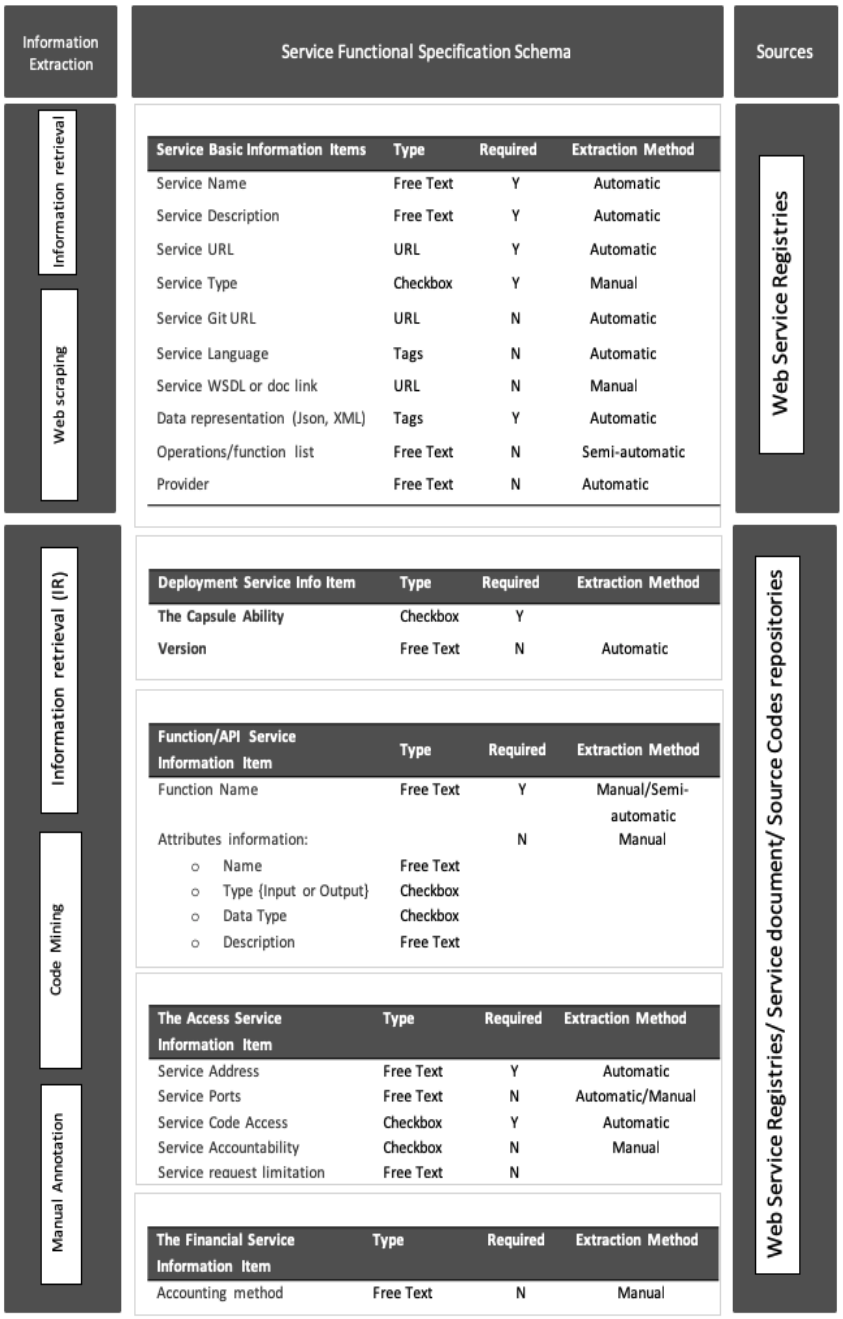
Με την είσοδο τιμών από τον χρήστη, ο ανιχνευτής δημιουργεί ένα πλαίσιο δεδομένων και πραγματοποιεί την επαναληπτική διαδικασία αναζήτησης. Τέλος, οι ανιχνευτές εξάγουν λέξεις κλειδιά, οι οποίες εισάγονται στην εσωτερική βάση δεδομένων.

Η αναζήτηση πραγματοποιείται ανά συγκεκριμένες χρονικές στιγμές πάνω στις ιστοσελίδες του Programmableweb, GitHub και GitLab, καθώς αυτές θεωρούνται οι πιο διαδεδομένες πηγές υπηρεσιών

3.3.1.2 Εσωτερικό Ευρετήριο Υπηρεσιών

Για την καταχώριση των υπηρεσιών στο εσωτερικό ευρετήριο, αποφασίστηκε να χρησιμοποιηθεί το Elascticsearch. Το ευρετήριο περιλαμβάνει το σύνολο των δεδομένων υπηρεσιών σε ένα ενδιάμεσο μητρώο που περιγράφεται παρακάτω. Τα πεδία που επισημαίνονται ως απαιτούμενα είναι η ελάχιστη απαιτούμενη πληροφορία ώστε μια υπηρεσία να μπορεί να ταξινομηθεί.

Το Elascticsearch είναι μια κατανεμημένη μηχανή αναζήτησης ανοιχτού κώδικα, γραμμένη χρησιμοποιώντας την γλώσσα Java. Επιτρέπει την αποθήκευση, την αναζήτηση και την ανάλυση μεγάλου όγκου δεδομένων σε πολύ σύντομο χρονικό διάστημα καθώς και την εκτέλεση γρήγορων αναζητήσεων με την χρήση ευρετηρίων αναζήτησης. Σε αντίθεση με την SQL, το Elascticsearch βασίζεται σε ένα σύστημα μηχανής αναζήτησης, καθιστώντας ευκολότερη την πρόσληψη νέων δεδομένων με οριζόντια κλιμάκωση. Το πιο σημαντικό πλεονέκτημά του είναι ότι λειτουργεί και ως ένα REST API για ενημερώσεις και αναζητήσεις δεδομένων. Επίσης, επιτρέπει την εκτέλεση εξαιρετικά προσαρμοσμένων αναζητήσεων, καθώς και την εκτέλεση ασαφών αναζητήσεων. Μέσω της Εύρεσης Υπηρεσιών, το ευρετήριο Elascticsearch θα λαμβάνει ερωτήματα για εύρεση υπηρεσίας καθώς και αιτήματα εγγραφής μίας νέας Υπηρεσίας, έτσι ώστε αυτή να αποθηκευτεί στο ευρετήριο.



Εικόνα 10: Απαιτούμενα πεδία υπηρεσίας

Για την χρήση ως βάση δεδομένων που εξυπηρετεί τις λειτουργίες των ανιχνευτών, διατηρείται ένα εσωτερικό μητρώο SQL. Αυτή η βάση δεδομένων συγχρονίζεται με το Elasticsearch για να παράσχει τους τύπους αναζητήσεων που περιγράφονται παραπάνω. Ως εκ τούτου, το εσωτερικό ευρετήριο έχει δύο βάσεις δεδομένων, μία για την υποστήριξη των ανιχνευτών (SQL) και μία για την αναζήτηση υπηρεσιών από τους χρήστες (Elasticsearch).

3.4 Δημιουργία Υπηρεσιών

Όπως είχαμε αναφέρει, κατά την διάρκεια σύνθεσης υπηρεσιών οι κόμβοι του διαγράμματος μπορούν να συνδεθούν με υπηρεσίες. Αυτό μπορεί να γίνει είτε με την εύρεση μιας ήδη υπάρχουσας υπηρεσίας, είτε με την δημιουργία μιας νέας. Στην περίπτωση που η διαδικασία της εύρεσης αποτύχει, ή δεν επιφέρει ικανοποιητικά αποτελέσματα, ο χρήστης είναι αναγκασμένος να δημιουργήσει μία νέα υπηρεσία εκ του μηδενός. Όπως αναφέραμε προηγουμένως στην βιβλιογραφική επισκόπηση, για να είναι σωστή η διαδικασία ανάπτυξης μίας νέας υπηρεσίας πρέπει να ακολουθήσουμε κάποια βήματα και να χρησιμοποιηθούν οι απαραίτητες τεχνολογίες για:

1. Την συνεργατική ανάπτυξη του κώδικα
2. Την συγγραφή κώδικα
3. Την δημιουργία, εκτέλεση και αξιολόγηση δοκιμών
4. Την αξιολόγηση της συντηρησιμότητας του λογισμικού και την παρακολούθηση της προόδου του
5. Το πακετάρισμα του κώδικα σε εκτελέσιμο αρχείο

Ο στόχος λοιπόν που θέτουμε για την επιλογή του περιβάλλοντος ανάπτυξης και των υπολοίπων τεχνολογιών που θα χρησιμοποιήσουμε, είναι αυτά να ικανοποιούν όσο το δυνατόν περισσότερες από τις απαραίτητες βασικές ανάγκες της διαδικασίας ανάπτυξης.

3.4.1 Εργαλείο συγγραφής κώδικα

Δεδομένου ότι η συγγραφή κώδικα είναι το κεντρικό κομμάτι της διαδικασίας ανάπτυξης μιας νέας υπηρεσίας, θεωρήσαμε αντίστοιχα, πως η διεπαφή χρήστη θα πρέπει να επικεντρώνεται σε αυτόν το σκοπό. Επομένως, θέσαμε σαν κύριο περιβάλλον χρήστη, το περιβάλλον του εργαλείου ανάπτυξης κώδικα. Βάσει των παραπάνω και λαμβάνοντας υπόψη τις απαιτήσεις του έργου σχετικά με την ανάγκη του εργαλείου να είναι διαθέσιμο μέσω δικτύου, αξιολογήσαμε πολλά διαθέσιμα εργαλεία και τελικά καταλήξαμε στην χρήση του Eclipse Theia. Το Eclipse Theia αποτελεί ένα εργαλείο ανοιχτού κώδικα, το οποίο είναι επεκτάσιμο και αρκετά διαδεδομένο. Επίσης, έχει την δυνατότητα χρήσης σε ιδιωτικό δίκτυο, πράγμα απαραίτητο, καθώς ο κώδικας που

αναπτύσσεται και αποθηκεύεται σε αυτό πολλές φορές δεν μπορεί να είναι δημόσια διαθέσιμος.

3.4.2 Εργαλείο συνεργατικής ανάπτυξης κώδικα

Για να καλύψουμε την ανάγκη συνεργατικής ανάπτυξης κώδικα, είχαμε τέσσερις βασικές επιλογές, που όλες είναι ιδιαιτέρα ώριμες και προσφέρουν πληθώρα λειτουργιών. Καθώς όμως ο κώδικας, που θα αναπτύσσεται και αποθηκεύεται μέσω της πλατφόρμας του SmartCLIDE, πολλές φορές θα είναι εμπιστευτικός, το επιλεγμένο εργαλείο πρέπει να έχει την δυνατότητα χρήσης σε ιδιωτικό δίκτυο. Το μόνο εργαλείο από τα διαθέσιμα που ερευνήσαμε, που υποστηρίζει αυτού του είδους την λειτουργία, είναι το GitLab.

3.4.3 Μέθοδος δημιουργίας, εκτέλεσης και αξιολόγησης δοκιμών

Όσον αφορά στην δημιουργία, στην εκτέλεση και στην αξιολόγηση δοκιμών, αυτό μπορεί να επιτευχθεί μέσω των εργαλείων που επιλέξαμε. Οι δοκιμές μπορούν να δημιουργηθούν χρησιμοποιώντας την διεπαφή και τις λειτουργίες του Eclipse Theia και να εκτελεστούν και να αξιολογηθούν χρησιμοποιώντας το GitLab. Το GitLab παρέχει δυνατότητα εκτέλεσης δοκιμών μέσω της υποστήριξής του για CI/CD (continuous integration / continuous delivery-deployment).

3.4.4 Μέθοδος αξιολόγησης της συντηρησιμότητας του λογισμικού

Συνεχίζοντας, για την αξιολόγηση της συντηρησιμότητας του λογισμικού και την παρακολούθηση της προόδου του, αναπτύξαμε υπηρεσίες ελέγχου τεχνικού χρέους. Μέσω αυτών των υπηρεσιών, θα υπολογίζονται τα βασικά συστατικά του Τεχνικού Χρέους, έτσι ώστε να μπορούμε να τα ποσοτικοποιήσουμε και να εξάγουμε την γενική συντηρησιμότητα του έργου. Παρόμοια διαδικασία θα πραγματοποιείται για όλες τις γενιές του κώδικα, χρησιμοποιώντας το ιστορικό που μας παρέχει το εργαλείο συνεργατικής ανάπτυξης GitLab. Αυτό θα έχει ως αποτέλεσμα, να παρακολουθούμε την πρόοδο του έργου και να εντοπίζουμε πιθανά προβλήματα κατά την ανάπτυξή του.

3.4.5 Περιβάλλον

Ακολουθώντας την προαναφερθείσα λογική, πως το κύριο περιβάλλον χρήστη θα είναι το περιβάλλον του εργαλείου ανάπτυξης κώδικα, καλούμαστε να ενσωματώσουμε

όλες τις λειτουργίες σε αυτό. Καθώς το περιβάλλον του Eclipse Theia είναι επεκτάσιμο, μπορούμε να δημιουργήσουμε τις απαραίτητες επεκτάσεις και προσθήκες, έτσι ώστε να καλούμε υπηρεσίες παρασκηνίου που προσφέρουν περίπλοκη λειτουργικότητα. Βάσει των παραπάνω, δημιουργήσαμε σειρά εργαλείων με σκοπό την υποστήριξη και την αυτοματοποίηση των αναγκαίων βημάτων. Με την αυτοματοποίηση, στοχεύουμε στην μείωση της πολυπλοκότητας της διαδικασίας, αποκρύπτοντας τις παραμετροποιήσεις και τις αλληλεπιδράσεις με εξωτερικά εργαλεία. Η παραμετροποίηση συμβαίνει στο υπόβαθρο και η αλληλεπίδραση και η επικοινωνία με τα εργαλεία γίνεται μέσω κλήσεων API από το Eclipse Theia, χωρίς ο χρήστης να χρειάζεται να αλλάξει περιβάλλον.

3.5 Δοκιμή Λογισμικού

3.5.1 Δοκιμή Απόδοσης

Στην περίπτωσή μας, θα επικεντρωθούμε κυρίως στη δοκιμή απόδοσης και στον τρόπο αυτοματοποίησης και ενσωμάτωσής στη ροή ανάπτυξης. Η παρακολούθηση της απόδοσης είναι μια συμπληρωματική πρακτική που παρέχει μια επισκόπηση της τρέχουσας κατάστασης του συστήματος, μετέπειτα της ανάπτυξής του. Αντίθετα, ο στόχος της δοκιμής απόδοσης είναι να διασφαλίσει ότι το σύστημα θα είναι σε θέση να χειριστεί έναν απαιτούμενο φόρτο εργασίας και ότι τελικά θα είναι ικανό να λειτουργεί και να διαχειρίζεται τον υψηλότερο φόρτο εργασίας.

3.5.2 Ενσωμάτωση δοκιμών απόδοσης στον κύκλο ανάπτυξης κώδικα

Δεδομένου ότι το μεγαλύτερο μέρος των λειτουργικών δοκιμών είναι ήδη ενσωματωμένο στη ροή ανάπτυξης και ότι οι περισσότερες από τις μη λειτουργικές δοκιμές χρειάζονται μια συγκεκριμένη προσέγγιση, το SmartCLIDE Cloud Testing θα επικεντρωθεί στην ενσωμάτωση των δοκιμών διαθεσιμότητας και απόδοσης κατά την διάρκεια του κύκλου ανάπτυξης. Με αυτόν τον τρόπο, ο χρήστης θα είναι σε θέση να εκτιμήσει νωρίς εάν η εφαρμογή πληροί τις απαιτήσεις, αντί να περιμένει μέχρι το τέλος του κύκλου. Έτσι, ο χρήστης δεν θα χρειάζεται να κάνει έντονες προγραμματιστικές αλλαγές για να επιδιορθώσει τον πηγαίο κώδικα του έργου ώστε να διορθώσει σημεία συμφόρησης που προκαλούν αποτυχίες όταν αυτό τίθεται υπό βαρύ φορτίο.

Για την ενσωμάτωση αυτού του είδους δοκιμών στην ροή ανάπτυξης, προτείνουμε τη χρήση μεθόδου ανάπτυξης με γνώμονα την επιθυμητή συμπεριφορά του

λογισμικού (Behavior driven development - BDD) που επιτρέπει σε άτομα με ελάχιστη έως καθόλου γνώση συγγραφής κώδικα να γράψουν τις προδιαγραφές και την αναμενόμενη συμπεριφορά του. Αξιοποιώντας τα σωστά εργαλεία, μπορούμε να δημιουργήσουμε πηγαίο κώδικα δοκιμών, ο οποίος μπορεί στη συνέχεια να ενσωματωθεί στην λειτουργία CI/CD. Αυτό θα έχει ως αποτέλεσμα οι δοκιμές να μπορούν να εκτελούνται από την αρχή του κύκλου ανάπτυξης του έργου, αυτόματα και συχνά, αποκομίζοντας όλα τα πλεονεκτήματα που αυτό επιφέρει, όπως αποτελεσματικότητα ανάπτυξης, γρήγορη ανατροφοδότηση και βελτιωμένη ποιότητα λογισμικού.

3.5.3 Αυτοματοποιημένη δοκιμή απόδοσης

Στην ανάπτυξη που χρησιμοποιεί την τεχνική Waterfall, η δοκιμή απόδοσης συνήθως γίνεται ακριβώς πριν από την διάθεση του προϊόντος. Στην περίπτωση που η ανάπτυξη και οι λειτουργικές δοκιμές διαρκέσουν περισσότερο από το αναμενόμενο, απομένει λιγότερος χρόνος για την εκτέλεση δοκιμών απόδοσης. Το παραπάνω, είναι κάτι που συμβαίνει αρκετά συχνά, επομένως τα προβλήματα απόδοσης του λογισμικού μπορεί να ανακαλυφθούν αρκετά αργά στον κύκλο ανάπτυξης, Ακόμα χειρότερο είναι το γεγονός πως τα ζητήματα απόδοσης του κώδικα τείνουν να είναι δυσκολότερο να επιλυθούν όταν ανακαλύπτονται αργά, καθώς οι αλλαγές που απαιτούνται επηρεάζουν μεγαλύτερο κομμάτι κώδικα. Στην περίπτωση που η εύρεση ζητημάτων που σχετίζονται με τον κώδικα ή την αρχιτεκτονική του λογισμικού καθυστερήσει, συχνά δεν υπάρχει χρόνος για αντίδραση και επιδιόρθωση.

Η μετάβαση στην χρήση μεθοδολογίας ανάπτυξης Agile και η μετατόπιση των δοκιμών απόδοσης πιο κοντά στην περίοδο ανάπτυξης συμβάλλει στην επίλυση αυτών των ζητημάτων νωρίτερα, όταν ακόμα υπάρχει χρόνος για την διερεύνηση και την επίλυσή τους.

Για παράδειγμα, μια αρχιτεκτονική μικροϋπηρεσιών (SOA) αποτελείται από μικρά στοιχεία. Ένα από τα πλεονεκτήματα της είναι ότι όταν υπάρχει αυξημένο φόρτο, αρκεί απλώς να αναπτύξουμε περισσότερους κόμβους των εμπλεκόμενων υπηρεσιών. Το παραπάνω όμως μπορεί να δημιουργήσει προβλήματα απόδοσης. Εάν η εφαρμογή καταναλώνει αρκετούς πόρους λόγω κάποιου σημείου συμφόρησης ή διαρροής πόρων, θα συνεχίσει να απαιτεί όλο και περισσότερους κόμβους.

Ως αποτέλεσμα των παραπάνω, παρόλο που η εφαρμογή θα εμφανίζει αποδεκτούς χρόνους απόκρισης και καλή εμπειρία χρήστη, το κόστος των υποδομών θα

έχει αυξηθεί σημαντικά καθώς θα έχει δημιουργηθεί μεγάλο πλήθος κόμβων υπηρεσιών. Η δοκιμή απόδοσης μπορεί να προσομοιώσει αυτό το σενάριο πριν συμβεί. Μέσω της αυτοματοποιημένης δοκιμής απόδοσης, μπορούμε να ελέγξουμε την ταχύτητα, τον χρόνο απόκρισης, την αξιοπιστία, τη χρήση πόρων και την επεκτασιμότητα του λογισμικού κάτω από ένα αναμενόμενο φόρτο εργασίας. Ο στόχος των δοκιμών απόδοσης είναι να εξαλειφθούν τα σημεία συμφόρησης και επιδείνωσης της απόδοσης του λογισμικού.

Υπάρχουν πολλοί τύποι δοκιμών απόδοσης, με τον καθένα να εξυπηρετεί διαφορετικό σκοπό.

- Το Smoke Test που διαβεβαιώνει ότι το σύστημά μπορεί να χειριστεί το ελάχιστο φορτίο, χωρίς κανένα πρόβλημα.
- Το Load Test που ασχολείται πρωτίστως με την αξιολόγηση της απόδοσης του συστήματος σε σχέση με τους ταυτόχρονους χρήστες και τα αιτήματα ανά δευτερόλεπτο.
- Το Stress Test και το Spike Test που αφορούν την αξιολόγηση των ορίων του συστήματος και της σταθερότητάς του υπό ακραίες συνθήκες.
- Το Soak Test που βοηθά στον έλεγχο της αξιοπιστίας και της απόδοσης ενός συστήματος για την πάροδο μεγάλου χρονικού διαστήματος.

Κάθε προαναφερθείσα δοκιμή μπορεί να εκτελεστεί βάσει του ίδιου σεναρίου. Ένας προγραμματιστής μπορεί να γράψει ένα σενάριο και να εκτελέσει οποιαδήποτε από τις παραπάνω δοκιμές επιθυμεί. Το μόνο που αλλάζει, είναι η ρύθμιση παραμέτρων της δοκιμής, αλλά η λογική παραμένει η ίδια. Υπάρχουν δύο τρόποι δημιουργίας αυτών των ειδών δοκιμών, τα σενάρια για API hammering ή σενάρια ροής χρήστη (user flow scenarios).

Το API hammering αναφέρεται στην εκμετάλλευση των εγγράφων που περιγράφουν τις προδιαγραφές API, όπως για παράδειγμα οι προδιαγραφές OpenAPI/Swagger. Όταν είναι διαθέσιμη μια τέτοια προδιαγραφή, είναι δυνατή η αυτόματη δημιουργία προσαρμοσμένων αιτημάτων ώστε να καλύπτουν κάθε διαθέσιμο API. Με μερική διαμόρφωση, είναι εύκολο να αυξηθούν οι ταυτόχρονες κλήσεις των API έτσι ώστε να δημιουργηθεί φόρτος σε αυτά. Παρόλα αυτά, αυτά τα αιτήματα μπορεί να αποδειχτούν άσχετα και ενδέχεται να μην εντοπίσουν προβλήματα που θα εμφανιζόντουσαν κατά την διάρκεια λειτουργίας του λογισμικού στον πραγματικό κόσμο. Αυτά τα προβλήματα, πιθανώς να μπορούν να προκύψουν μόνο μετά από

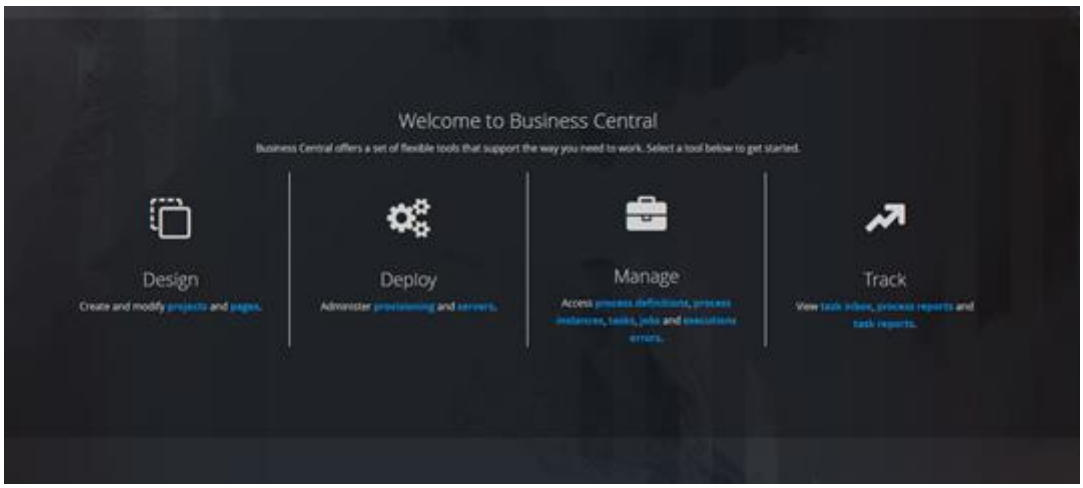
συνεχόμενη και έντονη χρήση της εφαρμογής. Αυτά τα προβλήματα μπορούν να εκτεθούν μόνο με τη δημιουργία σεναρίων ροής χρήστη.

Τα σενάρια ροής χρήστη μιμούνται μια σειρά ενεργειών που εκτελούνται από έναν πραγματικό χρήστη. Με αυτόν τον τρόπο, η ακολουθία των κλήσεων σε διαφορετικά API έχει λογική, και μπορούν να συσχετιστούν από την εφαρμογή. Έτσι, είναι εύκολο να διερευνηθεί ποια αλληλουχία ενεργειών οδηγεί σε συμφόρηση ή σε αποτυχία. Και πάλι, με μερική διαμόρφωση, είναι δυνατή η προσομοίωση πολλών χρηστών (έως και χιλιάδες από αυτούς) που χρησιμοποιούν την εφαρμογή ώστε να πραγματοποιηθεί η μέτρηση της απόκρισης του συστήματος.

4 Υλοποίηση

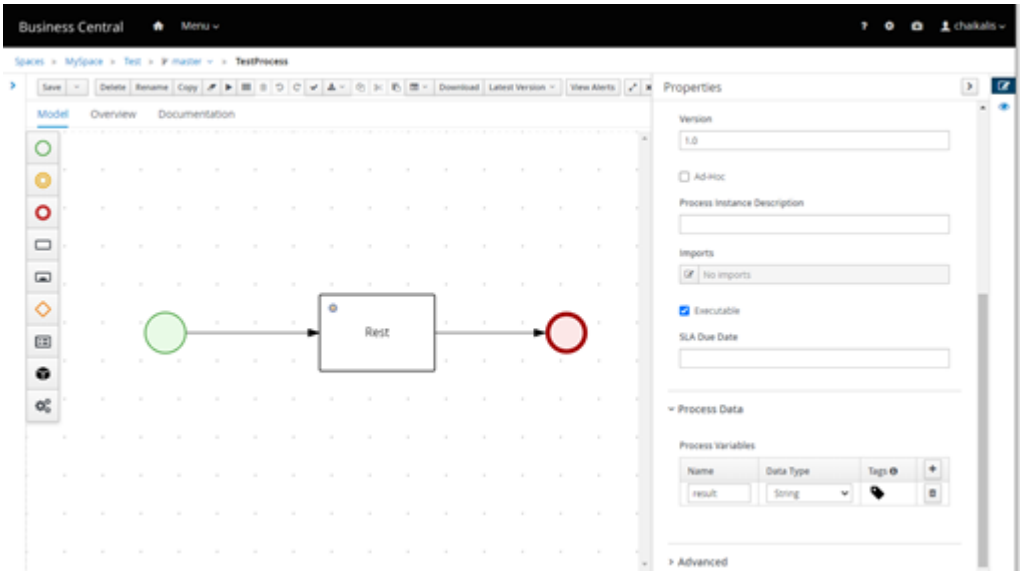
4.1 Σύνθεση Υπηρεσιών

Διάφορες υπηρεσίες μπορούν να συνδυαστούν για την δημιουργία μίας σύνθεσης υπηρεσιών ώστε να ικανοποιηθεί μια συγκεκριμένη επιχειρηματική διαδικασία. Για τον σκοπό αυτό, χρησιμοποιούμε το εργαλείο jBPM Workbench που υλοποιεί το ολοκληρωμένο πλαίσιο του BPMN 2.0, για τη δημιουργία, τη δοκιμή και την εκτέλεση BPMN διαγραμμάτων.



Εικόνα 11: Κύριο μενού του Business Central

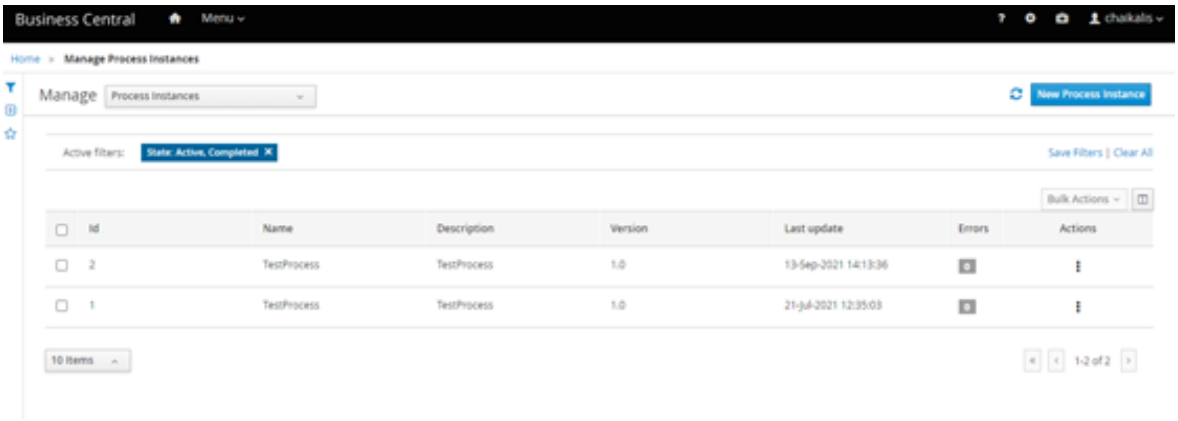
Η εκκίνηση της διαδικασίας γίνεται από το κύριο μενού του εργαλείου, όπως φαίνεται και στην παραπάνω εικόνα. Από εκεί, ο χρήστης μπορεί να επιλέξει την επιλογή «Design», για να δημιουργήσει μία νέα σύνθεση. Στην παρακάτω εικόνα, έχουμε ένα πολύ απλό παράδειγμα διαγράμματος.



Εικόνα 12: Ενδεικτικό διάγραμμα

Ο καμβάς σχεδίασης διαγραμμάτων είναι ιδιαίτερα εύκολος στην χρήση, καθώς ακολουθεί απλές και συνηθισμένες τεχνικές προσθήκης κόμβων, όπως το γνωστό drag-n-drop. Με την προσθήκη ενός νέου κόμβου, ο χρήστης μπορεί να τον παραμετροποιήσει χρησιμοποιώντας το πλαϊνό μενού.

Μετά την δημιουργία του διαγράμματος, ο χρήστης μπορεί να το δοκιμάσει και τέλος, να το εκτελέσει μέσω της πλατφόρμας. Με την χρήση του μενού «Manage Process» που βλέπουμε στην παρακάτω εικόνα, ο χρήστης μπορεί να δει όλες της εκτελέσεις συνθέσεων που έχουν γίνει, είτε αυτές είναι ακόμα ενεργές, είτε έχουν ολοκληρωθεί.



Εικόνα 13: Μενού για την διαχείριση διεργασιών

Ο εμπλουτισμός του εργαλείου που κάναμε, έγινε σε ένα κομμάτι της συνολικής πλατφόρμας, και πιο συγκεκριμένα, μέσω αλλαγών στο λογισμικό kiegroup/kie-wb-common[7], καθώς το kie-wb-common, περιλαμβάνει τον κώδικα λειτουργίας του καμβά σχεδίασης. Πέραν του kie-wb-common, ιδιαίτερη σημασία έχει και το kiegroup/kie-wb-distributions[8], το οποίο είναι το κεντρικό έργο που συλλέγει όλες τις εξαρτήσεις και τα δομικά κομμάτια για το χτίσιμο της πλατφόρμας του jBPM Workbench. Καθώς ο κώδικας της πλατφόρμας χρησιμοποιεί την τεχνολογία Maven που είχαμε προαναφέρει, η συλλογή των βιβλιοθηκών και εξαρτήσεων γίνεται αυτόματα, επομένως το χτίσιμο της εφαρμογής μετά τις στοχευόμενες αλλαγές μας ήταν εύκολο. Καθώς η λειτουργικότητα που προσθέτουμε στην πλατφόρμα του jBPM Workbench απαιτεί την υλοποίηση αλλαγών και στο γραφικό κομμάτι, πρέπει να κάνουμε προσθήκες στον κώδικα εμφάνισης HTML, CSS, JavaScript(JS) του έργου. Η πλατφόρμα και γενικά το έργο, χρησιμοποιούν για το γραφικό τους περιβάλλον την βιβλιοθήκη gwtbootstrap3. Μέσω της βιβλιοθήκης, η οποία αποτελεί ένα περικάλυμμα λειτουργιών, μας δίνεται η δυνατότητα να δημιουργήσουμε responsive HTML με την βοήθεια του Google Web Toolkit (GWT), γράφοντας κώδικα Java.

Το δομικό κομμάτι που έχει ενδιαφέρον για εμάς είναι το «Kie Workbench - Common - Stunner - BPMN Definition Set – Client» με artifact id «kie-wb-common-stunner-bpmn-client». Εντός αυτού του δομικού κομματιού, και συγκεκριμένα μέσα στο πακέτο «org.kie.workbench.common.stunner.bpmn.client.forms.fields», υπάρχει το αρχείο κώδικα «AssignmentsEditor.java». Για την παροχή της δικιάς μας λειτουργικότητας, πρέπει να εμπλουτίσουμε τον κώδικα του «AssignmentsEditor.java». Οι αλλαγές όπως αναφέραμε, γίνονται με την χρήση της γλώσσας Java, που έμμεσα δημιουργούν δυναμική HTML στο γραφικό περιβάλλον. Για παράδειγμα, στον παρακάτω πίνακα επιδεικνύουμε την εισαγωγή ενός label με προσαρμοσμένες ιδιότητες CSS.

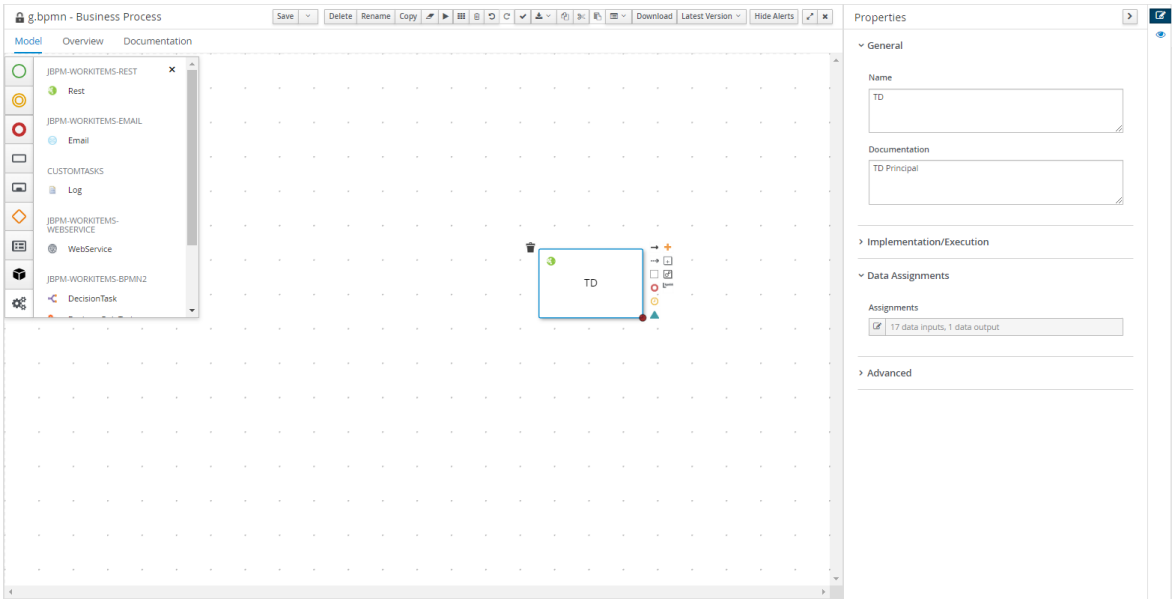
[7] https://github.com/kiegroup/kie-wb-common

[8] https://github.com/kiegroup/kie-wb-distributions

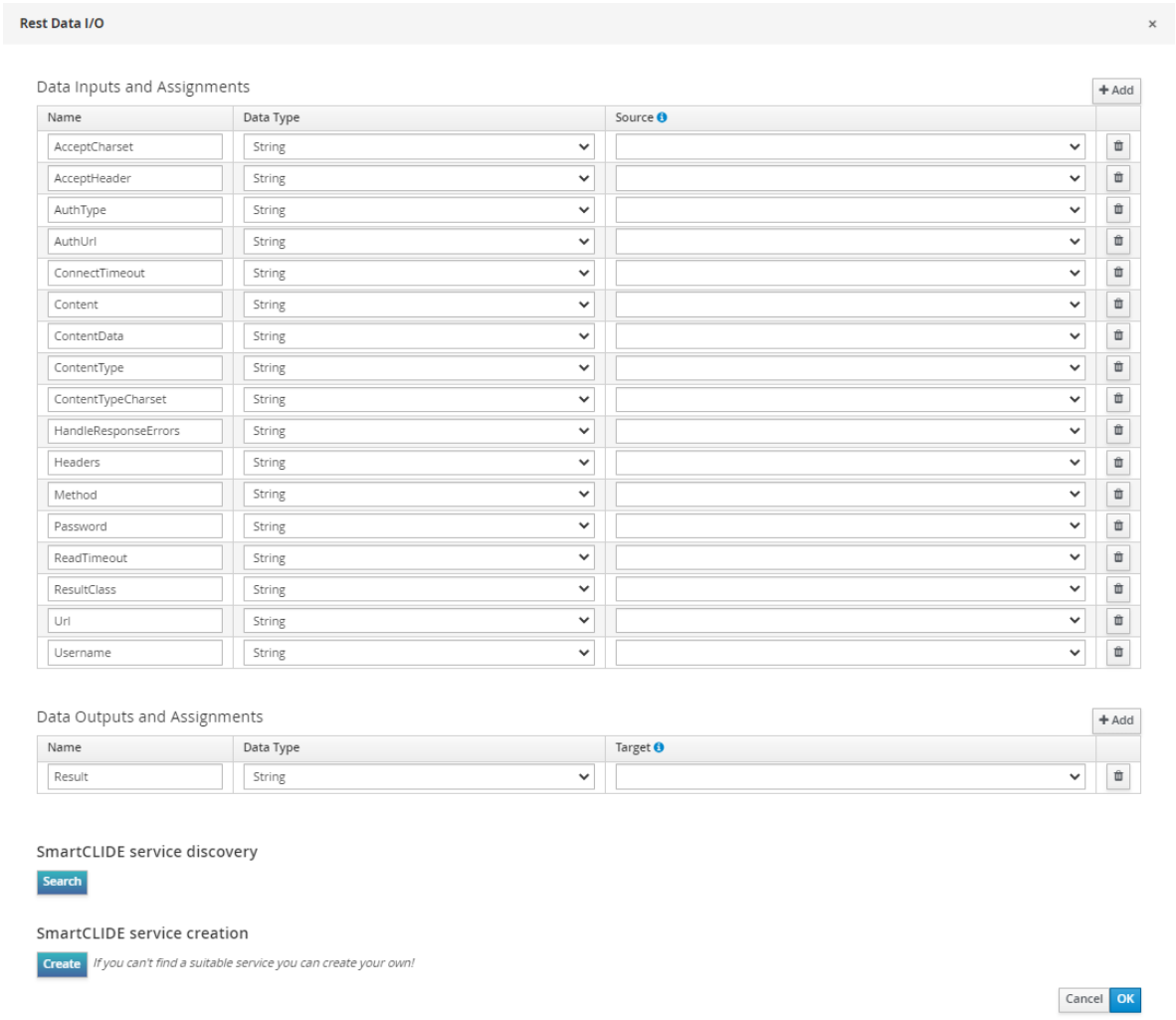
Πίνακας 4 Κώδικας Java για την εισαγωγή ενός label

```
Row SmartCLIDERowCreation = new Row();
SmartCLIDERowCreation.getElement().getStyle().setMarginTop(10,
Style.Unit.PX);
Column SmartCLIDEColumnCreation = new Column(ColumnSize.MD_12);
SmartCLIDERowCreation.add(SmartCLIDEColumnCreation);

Label labelC = new Label("SmartCLIDE service creation");
labelC.setPull(Pull.LEFT);
labelC.getElement().setAttribute("style",    "font-size:    16px;
margin-top: 20px;" +
        "  margin-bottom:  10px;  font-weight:  600;  line-height:
1.1; color: inherit;" +
        "  display:  block;  background:  transparent;  padding:
0px;");

SmartCLIDEColumnCreation.add(labelC);
container.add(SmartCLIDERowCreation);
```

Ακολουθώντας την παραπάνω λογική, μπορούμε να αλλάξουμε την εμφάνιση της πλατφόρμας και να την εμπλουτίσουμε με δικά μας στοιχεία. Η αλλαγές που κάναμε είναι η προσθήκη δύο κουμπιών για την Εύρεση (Service Discovery) και Δημιουργία (Service Creation) Υπηρεσιών. Όταν ο χρήστης επιλέγει να πραμετροποιήσει έναν Rest κόμβο, μπορεί να του αναθέσει δεδομένα (Data Assignments). Στο μενού της ανάθεσης δεδομένων, ο χρήστης πλέον έχει πρόσβαση και στα δύο νέα κουμπιά Εύρεσης και Δημιουργίας Υπηρεσιών.



Εικόνα 14: Εισαγωγή κόμβου υπηρεσίας



Εικόνα 15: Μενού ανάθεσης δεδομένων κόμβου

Η λειτουργικότητα που προσφέρει το κουμπί της Δημιουργίας Υπηρεσιών είναι αρκετά απλή, καθώς ανοίγει ένα νέο παράθυρο και ανακατευθύνει τον χρήστη στο πρόγραμμα συγγραφής κώδικα Eclipse Theia. Ο κώδικας αυτής της λειτουργίας παρατίθεται παρακάτω.

Πίνακας 5 Κώδικας κουμπιού Δημιουργίας Υπηρεσιών

```
Div divServiceCreation = new Div();
Button btnCreate = new Button("Create");
btnCreate.getElement().getStyle().setBackgroundImage("linear-
gradient(to bottom,rgb(53 181 191) 0,rgb(67 103 162) 100%)");
btnCreate.getElement().getStyle().setColor("#ffffff");
btnCreate.addClickHandler(clickEvent1 -> {
    Window.open(urlTheia,"_blank","");
});
divServiceCreation.add(btnCreate);
```

Ωστόσο, η λειτουργία του κουμπιού Εύρεσης Υπηρεσίας δεν είναι τόσο απλή. Για την παροχή της, πρέπει να πραγματοποιήσουμε ένα αίτημα POST προς την υπηρεσία υποστήριξης της Εύρεσης, η οποία μας επιστρέφει ως απόκριση μία λίστα υπηρεσιών. Το αίτημα θα αναλυθεί αναλυτικότερα στην μεθεπόμενη ενότητα που έχει να κάνει με την υλοποίηση της Εύρεσης Υπηρεσιών. Ο κώδικας της υλοποίησης του κουμπιού, καθώς και της επεξεργασίας της απόκρισης παρατίθεται παρακάτω.

Πίνακας 6 Κώδικας κουμπιού Εύρεσης Υπηρεσίας

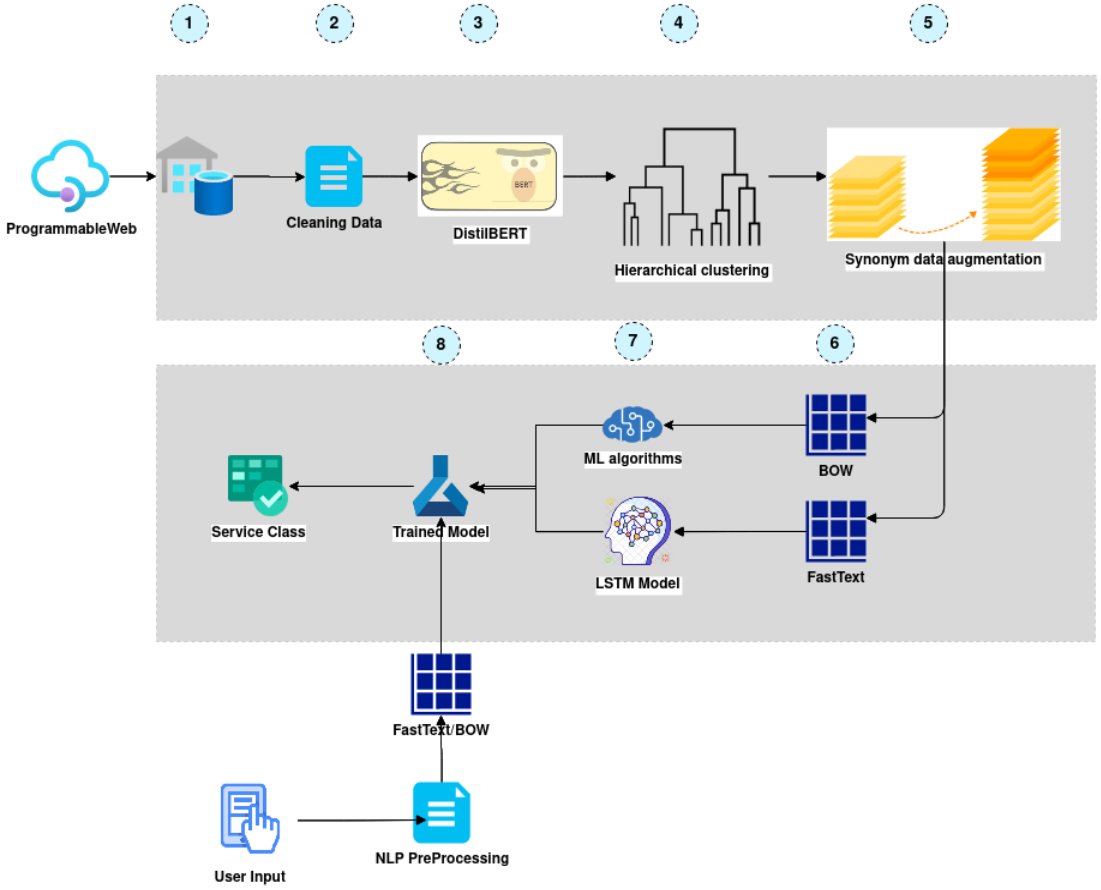
```
RequestBuilder builder = new RequestBuilder(RequestBuilder.POST,
urlServiceDiscovery);
builder.setHeader("Content-Type", "application/json");
String    jsonInputString3    =    "{\"full_name\":{\"0\":\""    +
this.taskTitle +
        "\"},        \"description\":        {\"0\":\""        +
this.taskDocumentation +"\"}}";
Request   response   =   builder.sendRequest(jsonInputString3,   new
RequestCallback() {
    public void onError(Request request, Throwable exception) { }
    public   void   onResponseReceived(Request   request,   Response
```

```
response) {
       JSONArray         jsonArray         =         (JSONArray)
JSONParser.parse(response);
       ListGroup listGroup= new ListGroup();
       for(int i=0; i<jsonArray.size(); i++) {
          ListGroupItem listGroupItem1= new ListGroupItem();
          Span spanName = new Span();
          spanName.setText(fullName);
          listGroupItem1.add(spanName);
          listGroup.add(listGroupItem1);
       }
    }
});
```

.

4.2 Κατάταξη Υπηρεσιών

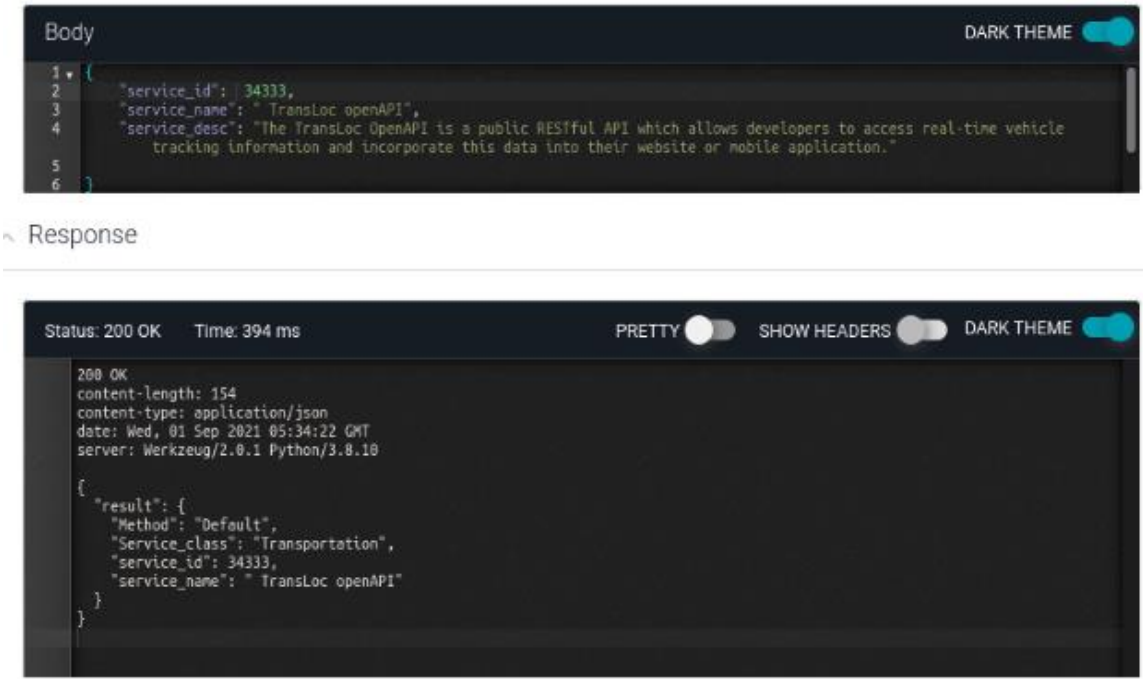
Για την κατάταξη υπηρεσιών, χρησιμοποιήθηκε μία υβριδική προσέγγιση εποπτευόμενης και μη εποπτευόμενης εκπαίδευσης μοντέλου. Η εκπαίδευση του μοντέλου έγινε πάνω στα δεδομένα του Programableweb, τα οποία περιέχουν χιλιάδες εισόδους υπηρεσιών όπως για παράδειγμα και πολλά Google API.



Εικόνα 16: Υβριδική προσέγγιση κατάταξης

Για την κατάταξη, εκπαιδεύτηκε ένα Δίκτυο Μακράς Βραχύχρονης Μνήμης (long short-term network - LSTM) με την πληροφορία που προέκυψε μετά την επεξεργασία των δεδομένων, χρησιμοποιώντας Fasttext[]. Επιπρόσθετα, για την επεξεργασία των δεδομένων χρησιμοποιήθηκαν αλγόριθμοι μηχανικής μάθηση όπως οι MultinomialNB classifier, GradientBoosting και Support Vector Classifier (SVC).

Το τελικό εκπαιδευμένο μοντέλο προσφέρει την λειτουργικότητα του μέσω API, και αποδέχεται αιτήματα όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 17: Αίτημα API για την κατάταξη μίας υπηρεσίας

4.3 Εύρεση Υπηρεσιών

Η Εύρεση Υπηρεσιών παρέχει την λειτουργικότητα της μέσω REST API, γραμμένο χρησιμοποιώντας το πλαίσιο λογισμικού Flask της γλώσσας Python. Η υπηρεσία της Εύρεσης υποστηρίζει την ταυτόχρονη αναζήτηση στα τρία πιο διαδεδομένα αποθετήρια κώδικα, το GitHub, το GitLab, και το Bitbucket. Η κύρια λειτουργία αυτή της υπηρεσίας είναι η ανάκτηση υπηρεσιών από τα τρία προαναφερθέντα αποθετήρια και η αποθήκευση τους στο εσωτερικό αποθετήριο του SmartCLIDE.

Παρακάτω ακολουθεί ο κώδικας από την αναζήτηση στο αποθετήριο κώδικα GitHub.

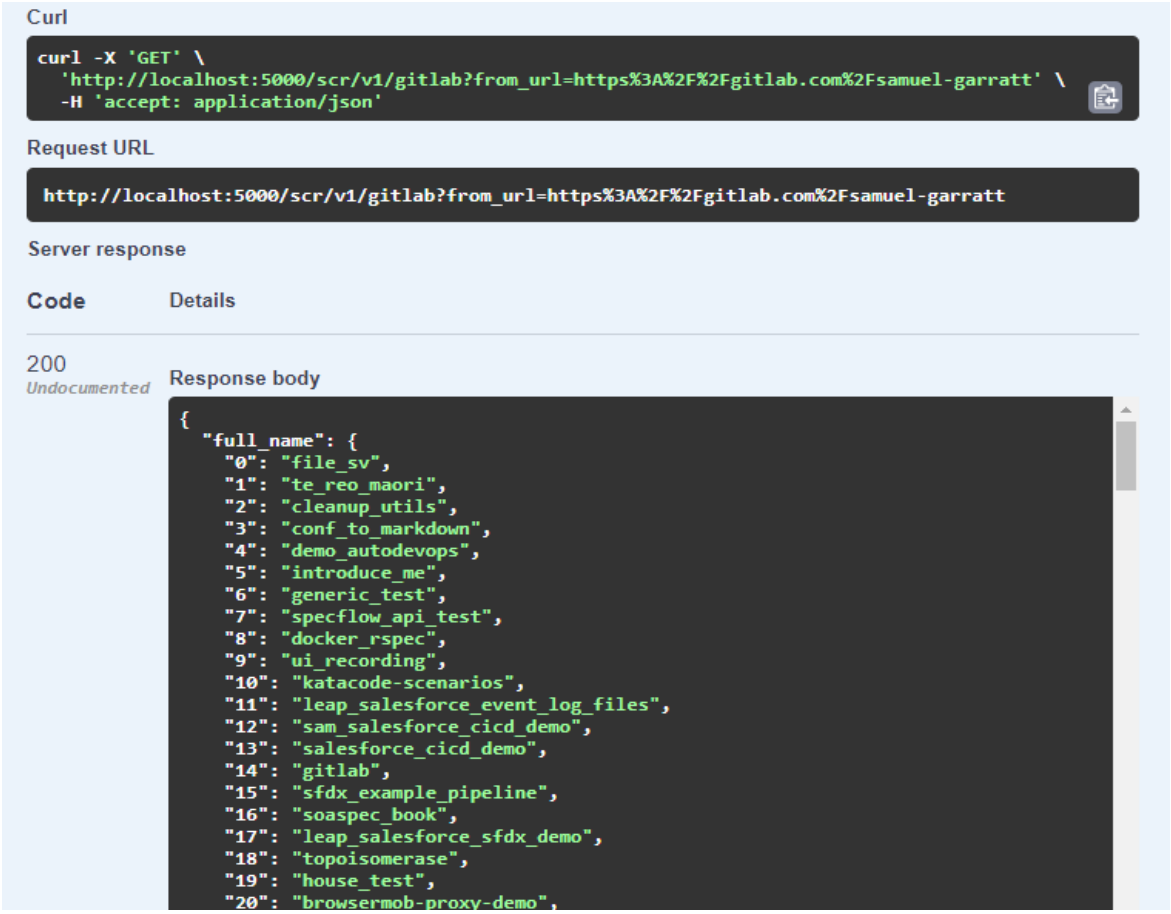
Πίνακας 7 Κώδικας για αναζήτηση υπηρεσιών στο GitHub

```
      def search_github_repos(keywords):
          global g

          # Split keywords in case of multiple
          keywords    =    [keyword.strip()    for    keyword    in
keywords.split(',')]

          # The query is set
          # We look at the readme and the project description
          query          =          '+'.join(keywords)          +
'+in:readme+in:description'

          result = g.search_repositories(query, 'stars', 'desc')

          # Limited to 1k for search
          print(f'Found  {result.totalCount}  repo(s)  with  the
keywords',
                ','.join(keywords))

          df_temp = pd.DataFrame()
          df_temp['Url'] = ""
          df_temp['Description'] = ""
          df_temp['Topics'] = ""
          df_temp['Stars'] = ""

          iter_obj = iter(result)
          while True:
              try:
                  for repo in result:

                      time.sleep(0.1)

                      cloneUrl = str(repo.clone_url)
                      description = str(repo.description)
                      stars = str(repo.stargazers_count)
```

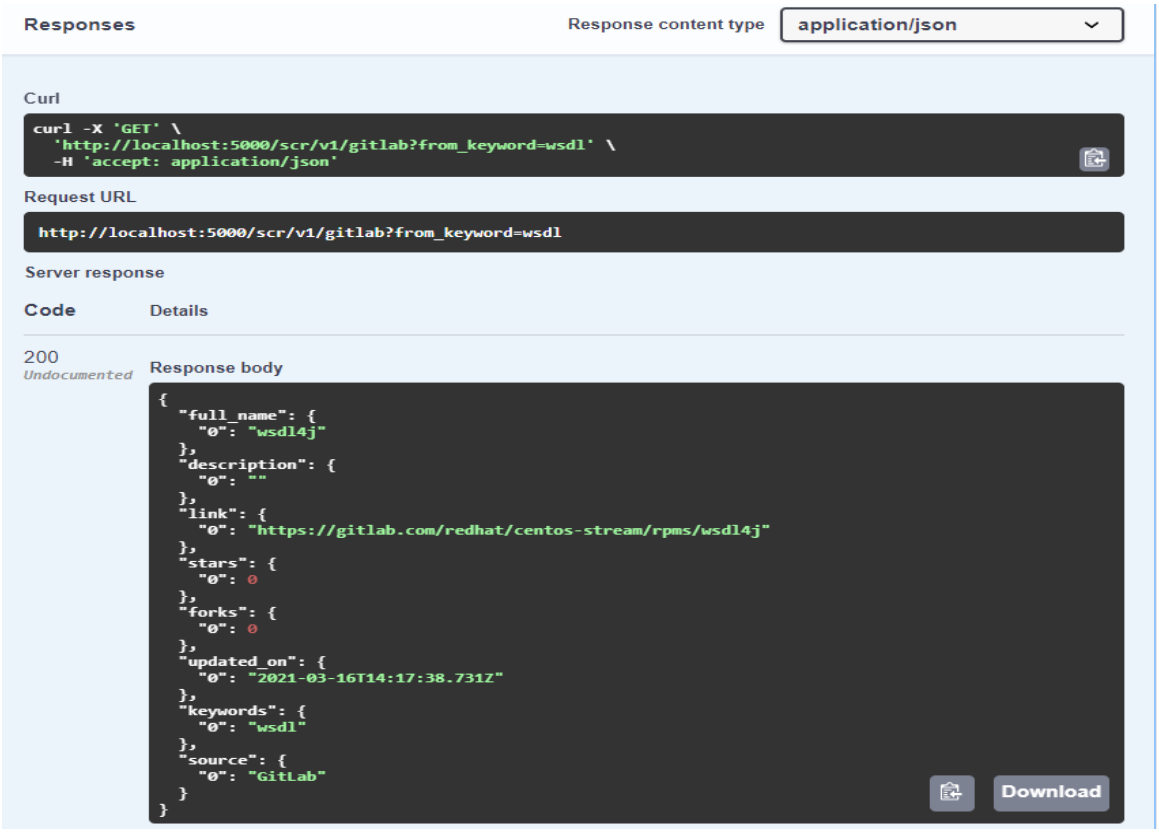
```
                    topics = ','.join(repo.get_topics())

                    df_repo = pd.DataFrame(
                        {'Url':     cloneUrl,     'Description':
description, 'Topics': topics, 'Stars': stars}, index=[0])

                    df_temp = df_temp.append(df_repo)

                    # Dw readme?
                    #README contents                                =
repo.get_contents("readme.md")

                df_temp = df_temp
                # One file per keyword
                df_temp.to_csv(r'./Github/'    +    'GitHub_'    +
','.join(keywords) +
                                '_'                                +
datetime.now().strftime('%d_%m_%Y')    +    '.csv',    index=True,
header=True)

                raise StopIteration

            except StopIteration:
                break

            except requests.exceptions.Timeout:
        …

            except RateLimitExceededException:
               …

    return df_temp
```

Υπάρχουν υλοποιημένες παρόμοιες μέθοδοι για τα άλλα δύο αποθετήρια του GitLab και του BitBucket.

Η ακόλουθη εικόνα δείχνει την μορφή αίτησης για την ανάκτηση αποθετηρίων από το GitLab, παρέχοντας την διαδικτυακή διεύθυνση τους.



Εικόνα 18: Ανάκτηση έργου από το GitLab χρησιμοποιώντας API



Εικόνα 19: Ανάκτηση έργου από το GitLab χρησιμοποιώντας λέξη κλειδί

Τέλος, η αναζήτηση υπηρεσιών για χρήση στην διαδικασία σύνθεσης υπηρεσιών, γίνεται στο εσωτερικό αποθετήριο υπηρεσιών. Η μέθοδος αναζήτησης χρησιμοποιεί το Elasticsearch API (στο Service Registry), που αναζητά στις προηγουμένως καταταγμένες υπηρεσίες, με την χρήση πολλαπλών φιλτραρισμάτων για την αποτελεσματικότερη παραγωγή αποτελεσμάτων.

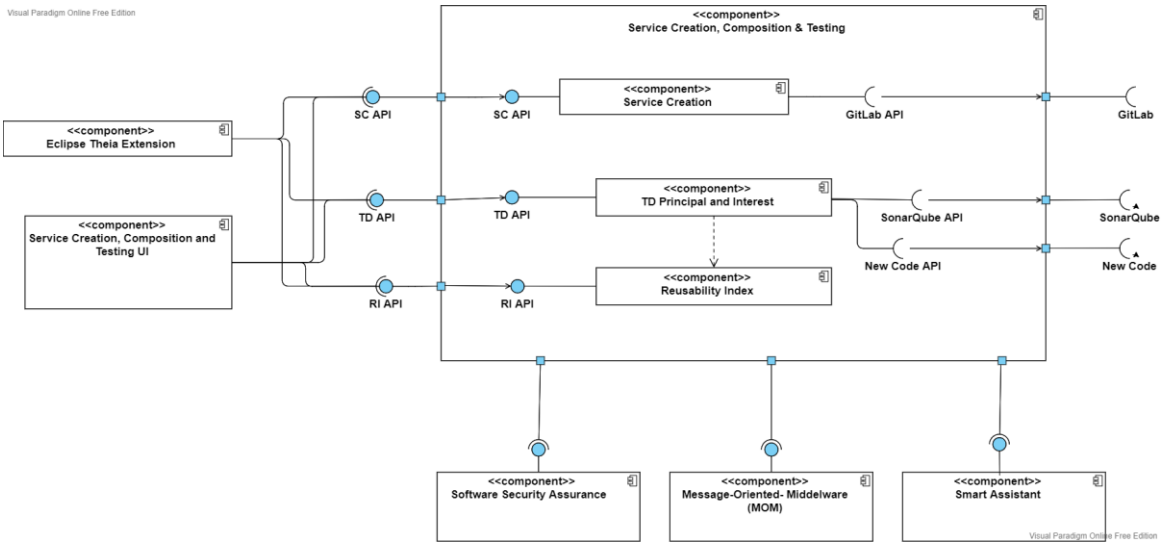
4.4 Δημιουργία Υπηρεσιών

Η λειτουργικότητα που προσφέρει η Δημιουργία Υπηρεσιών είναι η διαχείριση και η όσο το δυνατόν μεγαλύτερη αυτοματοποίηση των απαιτούμενων ενεργειών για την εκ του μηδενός ανάπτυξη κώδικα. Η συνολική λειτουργικότητα, χωρίζεται σε τρία επιμέρους βασικά κομμάτια:

1. Δημιουργία δομής (Service Creation)
2. Υπολογισμός Τεχνικού Χρέους (TD Principal and Interest)
    a. Κεφάλαιο (Principal)
    b. Τόκος (Interest)

3. Υπολογισμός δυνατότητας επαναχρησιμοποίησης λογισμικού (Reusability Index)

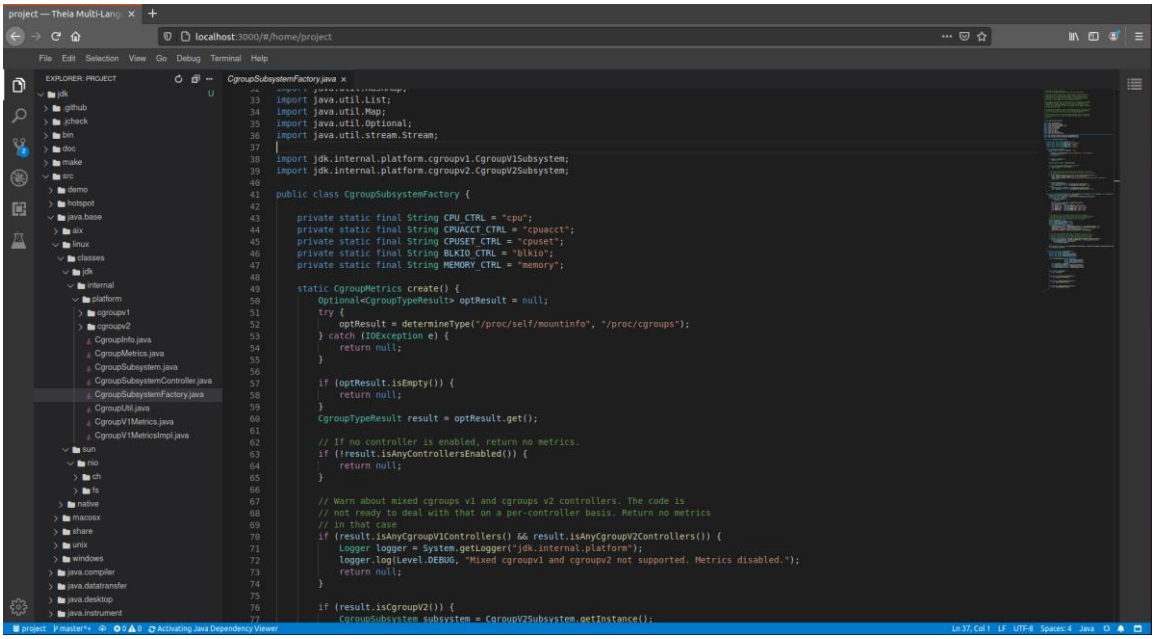
Τα παραπάνω μέρη αλληλεπιδρούν μεταξύ τους και με τα εξωτερικά εργαλεία και τεχνολογίες που αναφέραμε, για να παρέχουν αυτοματοποίηση και πληροφορία στον χρήστη.



Εικόνα 20: Διάγραμμα αλληλεπίδρασης λειτουργιών για την δημιουργία μιας νέας υπηρεσίας

Παραπάνω βλέπουμε την δομή και τους τρόπους αλληλεπίδρασης μεταξύ των δομικών μερών του λογισμικού.

Ξεκινώντας, για την εκ του μηδενός ανάπτυξη λογισμικού, η εκκίνηση της διαδικασίας γίνεται από το περιβάλλον συγγραφής κώδικα. Το περιβάλλον που επιλέξαμε για αυτό τον σκοπό είναι το Eclipse Theia.



Εικόνα 21: Περιβάλλον Eclipse Theia

4.4.1 Δημιουργία Δομής

Πριν ξεκινήσει η διαδικασία ανάπτυξης, είναι απαραίτητο να δημιουργηθεί η κατάλληλη δομή. Για την αυτοματοποίηση των βημάτων, αναπτύξαμε την υπηρεσία “Service Creation” που δέχεται τα απαραίτητα ορίσματα και πραγματοποιεί παρασκηνιακά τα βήματα παραμετροποίησης των εξωτερικών εργαλείων.

Το Service Creation παρέχει δύο σημεία επικοινωνίας (endpoints) μέσω των οποίων πραγματοποιούνται αιτήματα στην υπηρεσία.

- ***createStructure***
    - **projectName**: Το όνομα του λογισμικού που θα υλοποιήσουμε
    - **projVisibility**: Η ορατότητα του GitLab αποθετηρίου του λογισμικού
    - **projDescription**: Η λεκτική περιγραφή του λογισμικού.
    - **gitLabServerURL**: Η διαδικτυακή διεύθυνση του εξυπηρετητή GitLab.
    - **gitlabToken**: Τα διακριτικά πρόσβασης του χρήστη που θα ανήκει το αποθετήριο GitLab.
- ***createStructureJenkins***
    - **projectName**: Το όνομα του λογισμικού που θα υλοποιήσουμε
    - **projVisibility**: Η ορατότητα του GitLab αποθετηρίου του λογισμικού
    - **projDescription**: Η λεκτική περιγραφή του λογισμικού.
    - **gitLabServerURL**: Η διαδικτυακή διεύθυνση του εξυπηρετητή GitLab.

- **gitlabToken**: Τα διακριτικά πρόσβασης του χρήστη που θα ανήκει το αποθετήριο GitLab.
- **jenkinsServerUrl**: Η διαδικτυακή διεύθυνση του εξυπηρετητή Jenkins.
- **jenkinsUsername**: Το όνομα χρήστη του ατόμου στο οποίο θα ανήκει το CI/CD pipeline
- **jenkinsToken**: Το διακριτικό πρόσβασης του χρήστη που θα ανήκει το CI/CD pipeline

Στέλνοντας ένα αίτημα για την αυτόματη δημιουργία δομής, ανάλογα με το σημείο επικοινωνίας, εκκινείται μία διαδικασία.

Πίνακας 8 Βασική ροή δημιουργίας της δομής ανάπτυξης

```
public static ResultObject createStructure(…) {

	GitLab gitApi = new GitLab(gitToken, gitURL);
	boolean result = true;
	try {
		if(gitApi.isProjectNameAvailable(projectName)) {
			try {
				result=gitApi.createProjectWithGitlabCI(…);
			} catch (GitLabApiException e) {
				e.printStackTrace();
				return new ResultObject(…);
			}
		}else {
			return new ResultObject(…);
		}
	} catch (GitLabApiException e) {
		e.printStackTrace();
		return new ResultObject(1, "Problem with GitLab api");
	}

	if(!result) return new ResultObject(…);

	String projectHttpURL;
```

```
      try {
            projectHttpURL = gitApi.getProjectUrl(projectName);
      } catch (GitLabApiException e2) {
            e2.printStackTrace();
            return deleteRepository( args… );
      }

      String gitRepoURL;
      try {
            gitRepoURL = gitApi.getProjectUrl(projectName);
      } catch (GitLabApiException e) {
            e.printStackTrace();
            return deleteRepository(…);
      }

      gitApi.close();
      return new ResultObject(0, gitRepoURL);
}
```

Η παραπάνω διαδικασία για την δημιουργία της δομής που χρησιμοποιεί GitLab CI/CD, μπορεί να παρουσιαστεί συνοπτικά με τα παρακάτω βασικά βήματα.

- Σύνδεση με τον εξυπηρετητή GitLab
- Έλεγχος αν υπάρχει αποθετήριο με το όνομα που έχουμε δώσει
- Αν όχι, δημιουργία του κατάλληλου προτύπου αποθετηρίου, ανάλογα με τις παραμέτρους που δόθηκαν
- Δημιουργία του νέου αποθετηρίου
- Επιστροφή της διαδικτυακής διεύθυνσης του νέου αποθετηρίου

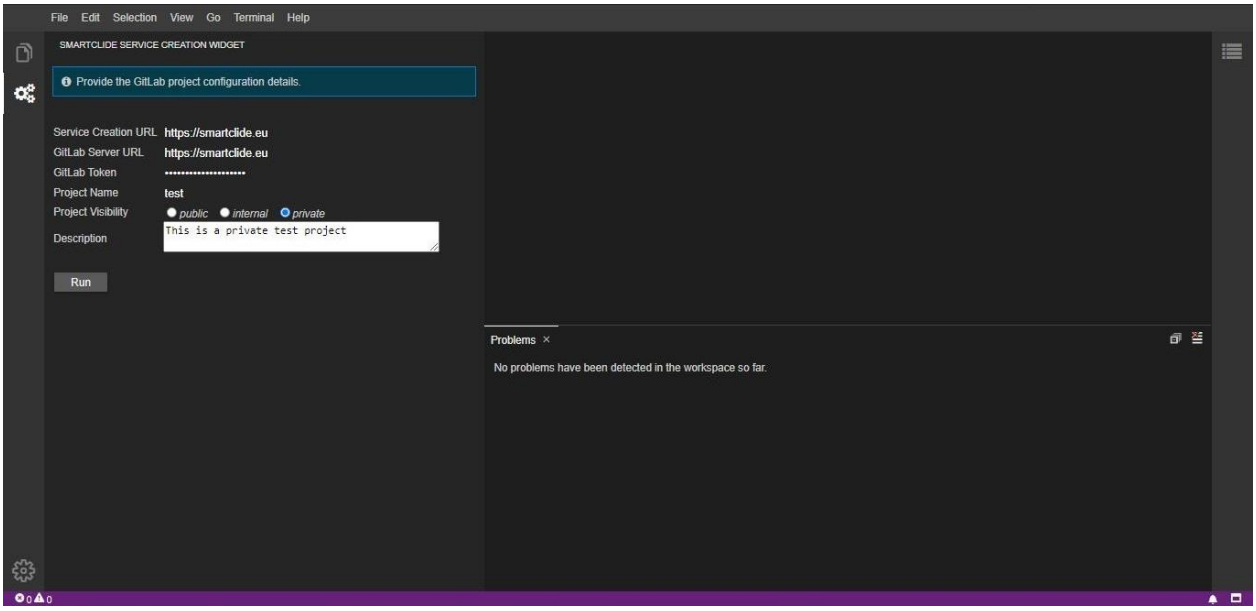
Η παραπάνω διαδικασία για την δημιουργία δομής που χρησιμοποιεί Jenkins CI/CD, μπορεί να παρουσιαστεί συνοπτικά με τα παρακάτω βασικά βήματα.

- Σύνδεση με τον εξυπηρετητή GitLab
- Σύνδεση με τον εξυπηρετητή Jenkins
- Έλεγχος αν υπάρχει αποθετήριο με το όνομα που έχουμε δώσει

- Αν όχι, δημιουργία του κατάλληλου προτύπου αποθετηρίου, ανάλογα με τις παραμέτρους που δόθηκαν
- Δημιουργία του νέου αποθετηρίου
- Ανάκτηση της διαδικτυακής διεύθυνσης του νέου αποθετηρίου
- Δημιουργία και παραμετροποίηση ενός νέου Jenkins pipeline
- Παραμετροποίηση του νέου αποθετηρίου έτσι ώστε να επικοινωνεί με το νέο Jenkins pipeline
- Επιστροφή της διαδικτυακής διεύθυνσης του νέου αποθετηρίου

Ανάμεσα στα παραπάνω βήματα, υπάρχουν οι απαραίτητοι έλεγχοι για την περίπτωση σφαλμάτων. Σε περίπτωση σφάλματος, η υπηρεσία αποκρίνεται με κατάλληλο μήνυμα και πραγματοποιεί τις κατάλληλες ενέργειες, πχ. την διαγραφή του νέου αποθετηρίου που δημιουργήθηκε.

Για την προσφορά της υπηρεσίας στον χρήστη, αναπτύχθηκε μία επέκταση για το περιβάλλον του Eclipse Theia.



Εικόνα 22: Επέκταση για την δημιουργία της δομής ανάπτυξης

Μέσω της επέκτασης, ο χρήστης μπορεί να δώσει τα απαραίτητα στοιχεία για την νέα υπηρεσία που θέλει να δημιουργήσει. Μετά την εκτέλεση του αιτήματος, το νέο

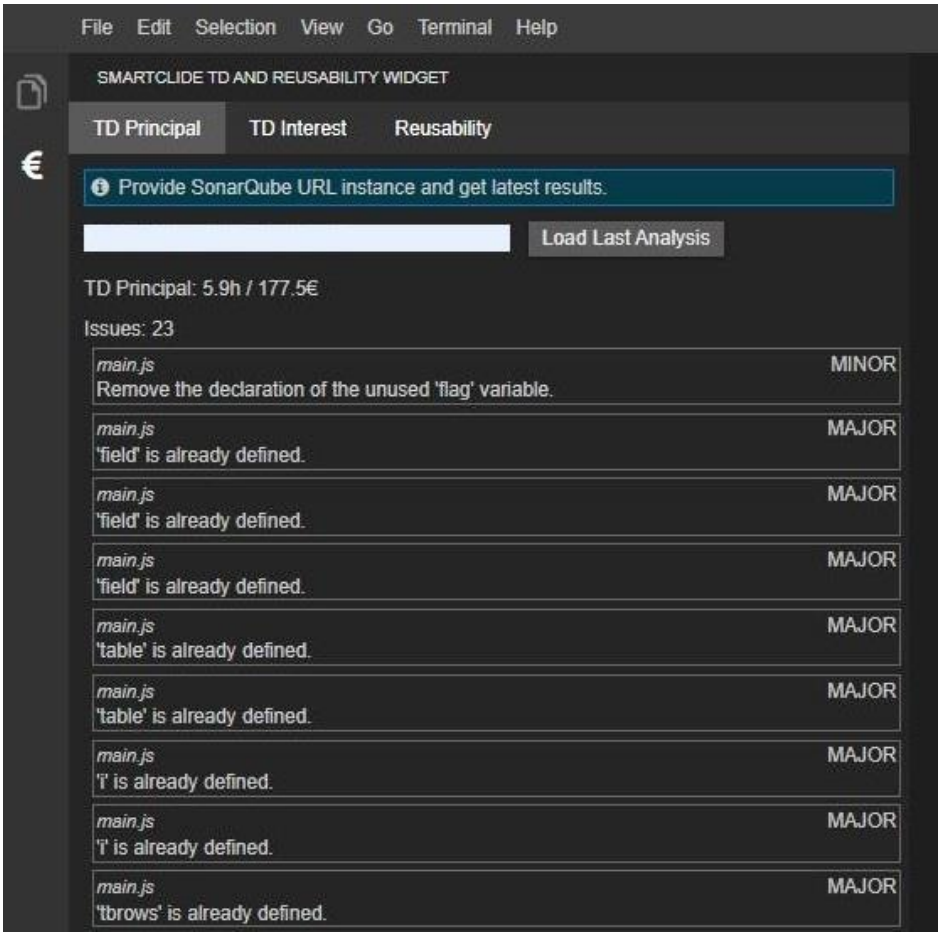
παραμετροποιημένο αποθετήριο που δημιουργήθηκε, συνδέεται μέσω Git στο Eclipse Theia.

Σε αυτό το στάδιο, ο χρήστης μπορεί να ξεκινήσει την συγγραφή του κώδικα. Επίσης, μπορεί να αξιοποιήσει τις δυνατότητες που προσφέρει το εργαλείο συνεργατικής ανάπτυξης GitLab, καθώς το Eclipse Theia παρέχει υποστήριξη και περιβάλλον για την χρήση της τεχνολογίας Git. Ο χρήστης μπορεί να ετοιμάσει (stage), να οριστικοποιήσει (commit) και να σπρώξει (push) τον κώδικά του στο απομακρυσμένο αποθετήριο, όπου λόγο της δομής που έχει φτιαχτεί, είναι δυνατόν να εκτελεστεί μία εργασία CI/CD.

4.4.2 Τεχνικό Χρέος

Κατά την διάρκεια συγγραφής κώδικα, ο χρήστης πρέπει να παρακολουθεί την εξέλιξη του κώδικα και να αξιολογεί την ποιότητα και τη δυνατότητα συντήρησής του. Για αυτόν τον σκοπό χρησιμοποιείται μία νέα επέκταση, η οποία επικοινωνεί με την κατάλληλη υπηρεσία υποστήριξης.

Παρακολουθώντας την παρακάτω εικόνα, βλέπουμε τις βασικές έννοιες του τεχνικού χρέους που περιγράψαμε προηγουμένως.



Εικόνα 23: Επέκταση για την παρακολούθηση του Κεφαλαίου και των προβλημάτων του κώδικα

Στην παραπάνω εικόνα, εμφανίζονται τα αρχεία κώδικα του έργου. Για κάθε αρχείο, παρουσιάζονται διάφορα προβλήματα που υπάρχουν σε αυτό μαζί με τον βαθμό σημαντικότητάς τους. Επίσης, κάτω από την μπάρα κειμένου, βλέπουμε το συνολικό κεφάλαιο που απαιτείται για την επιδιόρθωση των λαθών που εντοπίστηκαν. Ο υπολογισμός του κεφαλαίου γίνεται με την χρήση μιας υπηρεσίας υποστήριξης. Στην ουσία, η υπηρεσία υπολογίζει το χρηματικό ποσό ή τις ώρες που απαιτούνται για την διόρθωση όλων των προβλημάτων του λογισμικού, φέρνοντας το λογισμικό στην ιδεατή του μορφή. Ο εντοπισμός των προβλημάτων και η εκτίμηση του κόστους τους, γίνεται μέσω του εργαλείου SonarQube.

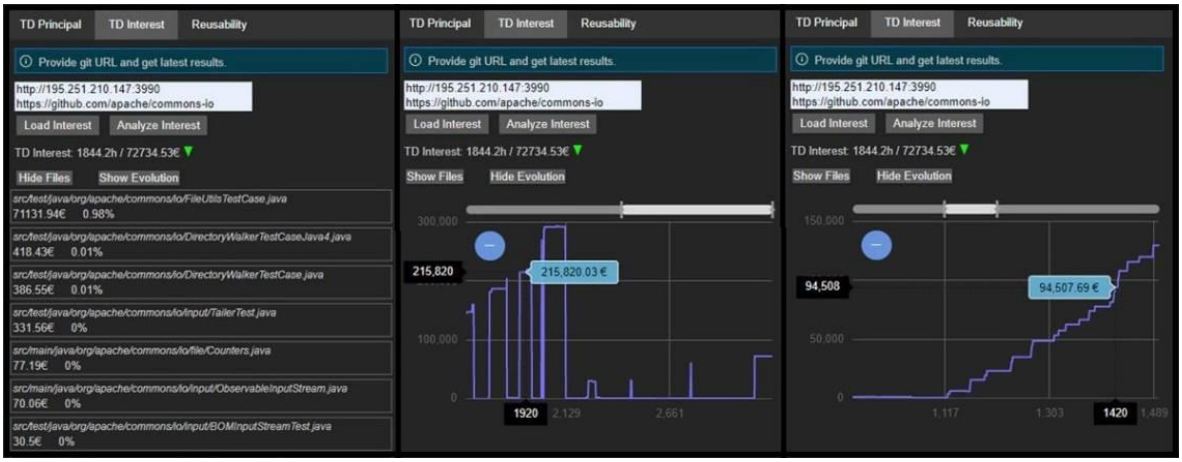
Η υπηρεσία υποστήριξης υπολογισμού κεφαλαίου, προσφέρει τρία σημεία επικοινωνίας:

1. Για την λήψη των μετρήσεων του κεφαλαίου

2. Για την λήψη των προβλημάτων του έργου
3. Για την αίτηση πραγματοποίησης νέας ανάλυσης

Πίνακας 9 Σημεία επικοινωνίας της υπηρεσίας υπολογισμού κεφαλαίου

```
@GetMapping(path="{projectKey}/measures")
      public Metric[] getMeasures(@PathVariable(…) {
            return analysisService.getMeasures(projectKey);
      }

      @GetMapping(path="{projectKey}/issues")
      public List<Issue> getIssues(…) {
            return analysisService.getIssues(projectKey);
      }

      @PostMapping
      @ResponseStatus(HttpStatus.OK)
      public ResponseEntity<String> makeNewAnalysis (…) {
            analysisService.startNewAnalysis(requestBodyAnalysis);
            return new ResponseEntity<>("finished successful",
HttpStatus.OK);
}
```



Εικόνα 24: Επέκταση για την παρακολούθηση του τόκου τεχνικού χρέους

Στην παραπάνω εικόνα, βλέπουμε δύο παράθυρα πληροφορίας. Στα αριστερά, βλέπουμε τον τόκο που έχει κάθε αρχείο κώδικα του έργου και το ποσοστό που αυτό συνεισφέρει στον συνολικό τόκο του έργου. Αυτό έχει σημασία, καθώς μπορούμε να δούμε πιο αρχείο επιδρά πιο αρνητικά στο συνολικό τεχνικό χρέος του έργου μας. Στο δεξί παράθυρο, βλέπουμε πως ο χρήστης μπορεί να αναλύσει την πρόοδο του έργου του βάσει διαγράμματος, όπου η κάθε ακμή αντιστοιχίζεται με μία οριστικοποίηση κώδικα (commit). Στόχος του προγραμματιστή είναι η μείωση του ρυθμού της αύξησης του χρέους και εν τέλει η μείωση του, οδηγώντας σε αποπληρωμή μέρους του τεχνικού χρέους.

Η υπηρεσία υποστήριξης για τον υπολογισμό του επιτοκίου, είναι υπεύθυνη για τον υπολογισμό του επιτοκίου του συνόλου του λογισμικού, αλλά και των μεμονωμένων αρχείων από τα οποία αποτελείται. Ο τόκος υπολογίζεται με δύο διαφορετικές μονάδες μέτρησης, τα χρήματα και τον χρόνο. Τέλος, η υπηρεσία πέραν της τιμής του επιτοκίου, υπολογίζει και διάφορες μετρικές ποιότητας του συστήματος που βοηθούν στην κατανόηση του μεγέθους, της πολυπλοκότητας, της συνεκτικότητας και της συνοχής του έργου. Αυτές οι μετρικές είναι χρήσιμες για τον υπολογισμό της δυνατότητας επαναχρησιμοποίησης λογισμικού.

Η υπηρεσία προσφέρει πολλά σημεία επικοινωνίας, μέσω των οποίων είναι δυνατόν να πάρουμε διαφορετικού είδους πληροφορία.

Πίνακας 10 Σημεία επικοινωνίας της υπηρεσίας υπολογισμού του τόκου τεχνικού χρέους

```
@GetMapping(value = "/cumulativeInterest")
Collection<CumulativeInterest>    getCumulativeInterestPerCommit(
... )
     if (Objects.isNull(sha))
          return
metricsService.findCumulativeInterestPerCommit(url);
     return    metricsService.findCumulativeInterestByCommit(url,
sha);
}
```

```
@GetMapping(value = "/interestPerCommitFile")
Collection<InterestPerCommitFile> getInterestPerCommitFile( ... )
     return   metricsService.findInterestByCommitFile(url,   sha,
filePath);
}

@GetMapping(value = "/interestChange")
Collection<InterestChange> getLastCommitInterestChange( ... )
     return metricsService.findInterestChangeByCommit(url, sha);
}

@GetMapping(value = "/fileInterestChange")
FileInterestChange getFileInterestChange( ... )
     return
metricsService.findInterestChangeByCommitAndFile(url,        sha,
filePath);
}

@GetMapping(value = "/normalizedInterest")
Collection<NormalizedInterest> getNormalizedInterest( ... )
     return ...
}

@GetMapping(value = "/highInterestFiles")
Collection<HighInterestFile> getHighInterestFiles( ... )
     return Objects.isNull(limit) ? ...
}

@GetMapping(value = "/reusabilityMetrics")
Collection<ProjectReusabilityMetrics>  getReusabilityMetrics(  ...
)
     return Objects.isNull(limit) ? ...
}

@GetMapping(value = "/reusabilityMetricsByCommit")
```

```
Collection<FileReusabilityMetrics> getReusabilityMetricsByCommit(
... )
      return Objects.isNull(limit) ? ...
}

@GetMapping(value = "/reusabilityMetricsByCommitAndFile")
Collection<FileReusabilityMetrics>
getReusabilityMetricsByCommitAndFile( ... )
      return Objects.isNull(limit) ? ...
}

@GetMapping(value = "/analyzedCommits")
Collection<AnalyzedCommit> getAnalyzedCommitIds( ... )
      return Objects.isNull(limit) ? ...
}

@PostMapping( … )
ResponseEntity<Project> startInterestAnalysis( ... )
      try {
            return new …
      } catch (IOException | InterruptedException e) {
            throw new ServerException("IOException");
      }
}
```

5 Εμπειρική Μελέτη

Αν και η προτεινόμενη εργαλειοθήκη δεν έχει αξιολογηθεί επίσημα λόγω των χρονικών περιορισμών του έργου και της σχετικής ανωριμότητας ορισμένων υλοποιήσεων, αναζητήσαμε σχόλια για την πρωτότυπη υλοποίηση από τρεις (3) επαγγελματίες προγραμματιστές λογισμικού με εμπειρία εργασίας σε εταιρικές εφαρμογές που επικεντρώνονται στην χρήση αρχιτεκτονικής βασισμένης σε μικροϋπηρεσίες.

Μετά από μια δίωρη παρουσίαση των αντίστοιχων εργαλείων και της συνεδρίας Q&A, ζητήσαμε από τους προγραμματιστές να πειραματιστούν με την πρωτότυπη εργαλειοθήκη, μιμούμενοι τη διαδικασία ανάπτυξης, δοκιμής και διάθεσης μιας νέας υπηρεσίας. Ζητήσαμε από τους προγραμματιστές να παράσχουν μια εκτίμηση του επιπλέον ή λιγότερου χρόνου που απαιτείται για την ολοκλήρωση των σχετικών εργασιών (σε σύγκριση με το τυπικό σενάριο εργασίας τους) και να καταγράψουν τυχόν ελαττώματα, περιορισμούς ή προτάσεις για βελτίωση.

Τα σχόλια που λάβαμε είναι αρκετά ενθαρρυντικά με την έννοια ότι οι προγραμματιστές αναγνωρίζουν την ανάγκη για τέτοιου είδους αυτοματισμούς και προβλέπουν σημαντικά μειωμένο χρόνο ανάπτυξης, σε περίπτωση που όλα τα βήματα αυτοματοποιηθούν. Τόνισαν την ανάγκη εξοικείωσης με τη νέα εργαλειοθήκη και τη σημασία της παροχής υλικού και βίντεο υποβοήθησης για την εκμάθηση της πλατφόρμας. Η δυνατότητα αναζήτησης και ανακάλυψης υπηρεσιών μέσω σημασιολογικών πληροφοριών είναι ιδιαίτερα ευπρόσδεκτη, αλλά θα ήταν χρήσιμη και η καθοδήγησης σχετικά με τον τρόπο διάθεσης των υπηρεσιών, όπου στην περίπτωση του Kubernetes η καμπύλη εκμάθησης είναι αρκετά μεγάλη. Η συνολική εμπειρία χρήστη βαθμολογήθηκε πολύ υψηλά από τους προγραμματιστές. Ωστόσο, υπάρχει περιθώριο βελτίωσης στην παρουσίαση και στην περαιτέρω αυτοματοποίηση βημάτων.

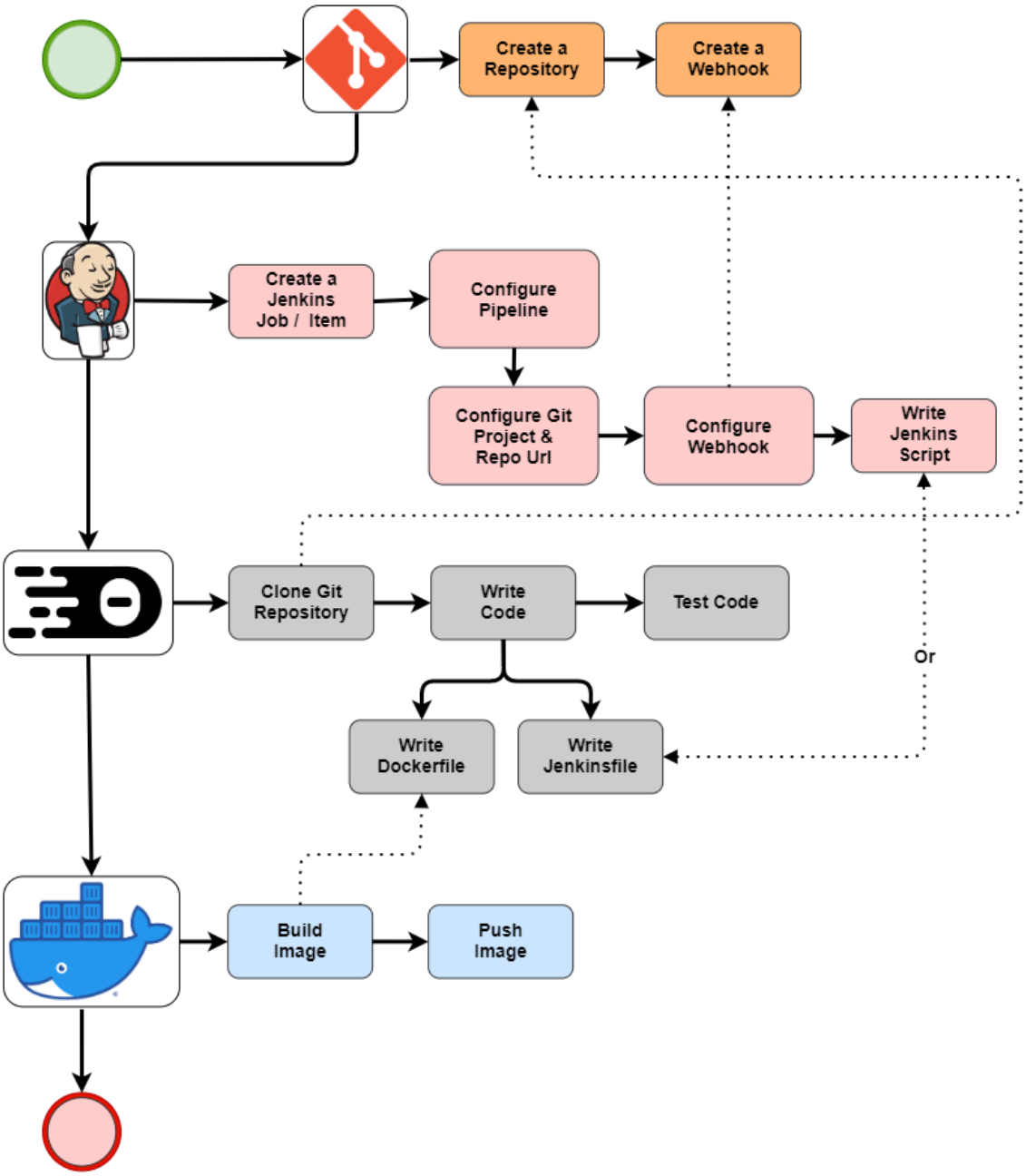
6 Επίλογος

6.1 Σύνοψη και συμπεράσματα

Οι αρχιτεκτονικές λογισμικού που βασίζονται σε υπηρεσίες θεωρούνται ως μια ιδανική εναλλακτική λύση για τον εκσυγχρονισμό μονολιθικών εφαρμογών παλαιού τύπου. Είναι ιδανικές για τη δημιουργία νέων, ευέλικτων και επεκτάσιμων επιχειρηματικών συστημάτων η λειτουργία των οποίων βασίζεται στην εκτέλεση στο νέφος. Ενώ τα αρχιτεκτονικά πλεονεκτήματα των υπηρεσιών έχουν μελετηθεί ευρέως, ο προγραμματισμός, η σύνθεση, η δημιουργία, η δοκιμή και η ανάπτυξη υπηρεσιών παραμένουν μια περίπλοκη και απαιτητική εργασία που απαιτεί εξειδίκευση, πολύ πειραματισμό και εξοικείωση με σειρά από διαφορετικές τεχνολογίες και εργαλεία. Σε αυτή την εργασία προτείνουμε μια προσέγγιση για τη διευκόλυνση της ανάπτυξης λογισμικού που βασίζεται σε εκτέλεση στο νέφος, επεκτείνοντας ένα πολύ γνωστό περιβάλλον ανάπτυξης κώδικα από την Eclipse.

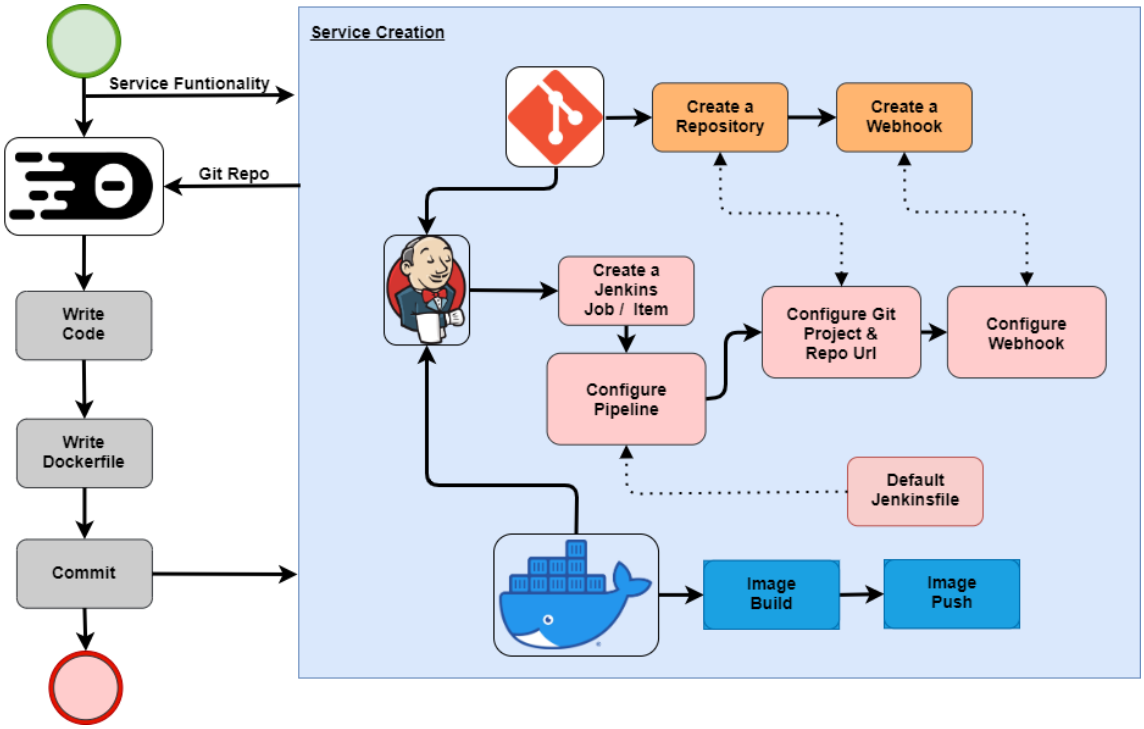
Η λύση που προτάθηκε προσφέρει ένα πλήρες σύνολο εργαλείων για τη σύνθεση, τη δημιουργία, τη δοκιμή και την ανάπτυξη υπηρεσιών, συνδυάζοντας τεχνολογίες και εργαλεία τελευταίας τεχνολογίας. Τα αναμενόμενα οφέλη παρουσιάζονται χρησιμοποιώντας μια οπτική αναπαράσταση που απεικονίζει τις διαδικασίες «πριν» και «μετά» της ανάπτυξης υπηρεσιών.

Από την παρακάτω εικόνα γίνεται αντιληπτό ότι χωρίς την χρήση της πλατφόρμας SmartCLIDE, για την δημιουργία μιας νέας υπηρεσίας, ο προγραμματιστής έπρεπε να αλληλεπιδράσει χειροκίνητα και να παραμετροποιήσει τέσσερα διαφορετικά εργαλεία (git, Jenkins, IDE και Docker). Επιπλέον, εξετάζοντας επίσης τη σύνθεση των υπηρεσιών, την εύρεση υπηρεσιών και την ανάπτυξη υπηρεσιών, μπορούμε να δούμε ότι ο χρόνος, η γνώση και η προσπάθεια που απαιτείται να καταβάλει ο προγραμματιστής θα μπορούσαν να αυξηθούν δραματικά.



Εικόνα 25: Παραδοσιακή ροή ανάπτυξης υπηρεσιών

Από την άλλη, με την χρήση της πλατφόρμας του SmartCLIDE, ο χρήστης αλληλεπιδρά μόνο με το περιβάλλον συγγραφής κώδικα του Eclipse Theia. Όλες οι περίπλοκες λειτουργίες και παραμετροποιήσεις εργαλείων είναι αυτοματοποιημένες και κρυμμένες από τον χρήστη, ο οποίος αλληλεπιδρά με αυτές μέσω των επεκτάσεων του Eclipse Theia.



Εικόνα 26: Ροή ανάπτυξης υπηρεσιών, χρησιμοποιώντας την πλατφόρμα του SmartCLIDE

Κατά συνέπεια, η εστίαση του χρήστη παραμένει στην διαδικασία ανάπτυξης κώδικα και όχι στη διαχείριση και αλληλεπίδραση με όλα τα απαραίτητα εργαλεία. Επιπλέον, οι χρονοβόρες και επιρρεπείς σε σφάλματα παραμετροποιήσεις εργαλείων αφαιρούνται και αποκρύπτονται από τη διαδικασία ανάπτυξης.

Τέλος, όλες οι βασικές πληροφορίες, για την πρόοδο και την ποιότητα του λογισμικού, ανακτώνται από την πλατφόρμα και παρουσιάζονται στον χρήστη μέσω του γραφικού περιβάλλοντος του Eclipse Theia. Ως αποτέλεσμα, ολόκληρη η διαδικασία ανάπτυξης είναι λιγότερο επιρρεπής σε σφάλματα, αλλά και πιο αποτελεσματική και φιλική προς τους προγραμματιστές.

6.2 Μελλοντικές Επεκτάσεις

Οι μελλοντικές βελτιώσεις προέκυψαν από την πραγματοποίηση αξιολόγησης της εργαλειοθήκης.

Σαν πρώτη, προτεραιότητα τέθηκε η αυτοματοποίηση παραγωγής περιπτώσεων δοκιμών. Οι προγραμματιστές που δοκίμασαν την πλατφόρμα, τόνισαν πως η διαδικασία δοκιμής μίας υπηρεσίας είναι αρκετά χρονοβόρα και ανιαρή, επομένως η συνολική εμπειρία που προσφέρει η πλατφόρμα θα υποβοηθούνταν σημαντικά από την ύπαρξη αυτή της λειτουργικότητας.

Επίσης, μία ακόμα λειτουργική βελτίωση θα ήταν η υλοποίηση μεθόδου για την διάθεση υπηρεσιών. Καθώς η διαδικασία διάθεσης λογισμικού σε ένα διακομιστή εφαρμογών είναι δυνατόν να αυτοματοποιηθεί, θα προσφέρει ένα ακόμα επίπεδο βοήθειας προς τον χρήστη. Για να συμβεί αυτό, θα πρέπει πρώτα να αξιολογηθούν όλοι οι δυνατοί τρόποι διάθεσης, και να επιλεγεί ο κατάλληλος για τις ανάγκες του έργου.

Τέλος, όπως αναφέρθηκε κατά την διαδικασία δοκιμής της πλατφόρμας, είναι σημαντική η δημιουργία και η παροχή λεπτομερούς υλικού για την υποβοήθηση της εκμάθησης και χρήσης της πλατφόρμας σε μορφή βίντεο και εγχειριδίων.

Ως μελλοντική ερευνητική εργασία, πέραν των προαναφερθέντων τεχνικών βελτιώσεων, θα ήταν ενδιαφέρον να μελετηθούν τυχόν προβλήματα και βελτιώσεις που προκύπτουν από την χρήση της πλατφόρμας, συγκριτικά με την παλιά μέθοδο ανάπτυξης SOA λογισμικού.

Βιβλιογραφία

[1] Ksenia Striapunina, Software market revenue in the World from 2016 to 2021. Statista, 2019

[2] Zaigham Mahmood. 2007. Service oriented architecture: potential benefits and challenges. In Proceedings of the 11th WSEAS International Conference on Computers (ICCOMP'07). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 497–501.

[3] Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service-Oriented Analysis and Design - An interdisciplinary modeling approach for SOA projects. Technical article, IBM (2 June 2004). Online: http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/

[4] Groves D, Successfully planning for SOA, BEA Systems Worldwide, 11 Sept 2005

[5] Alshafaey, Mohamed & Saleh, Ahmed & Alrahamawy, Mohamed. (2021). A new cloud-based classification methodology (CBCM) for efficient semantic web service discovery. Cluster Computing. 24. 10.1007/s10586-021-03245-z.

[6] Jianxiao Liu, Zonglin Tian, Panbiao Liu, Jiawei Jiang, and Zhao Li. An approach of semantic web service classification based on naive bayes. In 2016 IEEE International Conference on Services Computing (SCC), pages 356–362. IEEE, 2016

[7] Yilong Yang, Nafees Qamar, Peng Liu, Katarina Grolinger, Weiru Wang, Zhi Li, and Zhifang Liao. Servenet: A deep neural network for web services classification. In 2020 IEEE International Conference on Web Services (ICWS), pages 168–175. IEEE, 2020.

[8] Li Yuan-jie and Cao Jian. Web service classification based on automatic semantic annotation and ensemble learning. In 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pages 2274–2279. IEEE, 2012

[9] M Swami Das, A Govardhan, and D Vijaya Lakshmi. Classification of web services using data mining algorithms and improved learning model. Telkomnika, 17(6):3191–3202, 2019

[10] Weishi Shi, Xumin Liu, and Qi Yu. Correlation-aware multi-label active learning for web service tag recommendation. In 2017 IEEE International Conference on Web Services (ICWS), pages 229–236. IEEE, 2017

[11] Kamath, S.S., Ananthanarayana, V.S. Semantics-based Web service classification using morphological analysis and ensemble learning techniques. Int J Data Sci Anal 2, 61–74 (2016). https://doi.org/10.1007/s41060-016-0026-x

[12] Sasikanth Avancha, Anupam Joshi, and Timotby Finin. Enhanced service discovery in bluetooth. Computer, 35(6):96–99, 2002

[13] Crasso, Marco & Zunino, Alejandro & Campo, Marcelo. (2008). AWSC: An approach to Web service classification based on machine learning techniques. Ibm Journal of Research and Development. 12. 25-36. 10.4114/ia.v12i37.955.

[14] Christian Sanchez Sanchez, Esau Villatoro Tello, Adriana Gabriela Ramirez De La Rosa, Hector Jimenez Salazar, and David Eduardo Pinto Avenda˜no. Wsdl information selection for improving web service classification. 2017

[15] Sidra Shafi and Usman Qamar. [wip] web services classification using an improved text mining technique. In 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA), pages 210–215. IEEE, 2018

[16] Yilong Yang, Wei Ke, Weiru Wang, and Yongxin Zhao. Deep learning for web services classification. In 2019 IEEE International Conference on Web Services (ICWS), pages 440–442. IEEE, 2019

[17] QWSdata, Dataset. https://qwsdata.github.io/citations.html, 2007. [On-line; accessed 2020]

[18] Kamath S, Sowmya & Ahmed, Atif. (2015). A composite classification model for web services based on semantic & syntactic information integration. Souvenir of the 2015 IEEE International Advance Computing Conference, IACC 2015. 1169-1173. 10.1109/IADCC.2015.7154887

[19] Manish Kumar, Rajesh Bhatia, and Dhavleesh Rattan. A survey of web crawlers for information retrieval. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 7(6):e1218, 2017

[20] Shilpa Dang and Peerzada Hamid Ahmad. Text mining: Techniques and its application. International Journal of Engineering & Technology Innovations, 1(4):866–2348, 2014

[21] Alberto Rivas, Alfonso Gonz´alez-Briones, Juan J Cea-Mor´an, Arnau Prat-Perez, and Juan M Corchado. My-trac: System for recommendation of points of interest on the basis of twitter profiles. Electronics, 10(11):1263, 2021.

[22] Alberto Rivas, Alfonso Gonzalez-Briones, Guillermo Hernandez, Javier Prieto, and Pablo Chamoso. Artificial neural network analysis of the academic performance of students in virtual learning environments. Neurocomputing, 423:713–720, 2021

[23] Vadim Markovtsev and Waren Long. Public git archive: a big code dataset for all. In Proceedings of the 15th International Conference on Mining Software Repositories, pages 34–37, 2018

[24] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. Big code!= big vocabulary: Open-vocabulary models for source code. In 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pages 1073–1085. IEEE, 2020

[25] Daniel E Krutz, Mehdi Mirakhorli, Samuel A Malachowsky, Andres Ruiz, Jacob Peterson, Andrew Filipski, and Jared Smith. A dataset of open-source android

applications. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pages 522–525. IEEE, 2015

[26] Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. Aroma: Code recommendation via structural code search. Proceedings of the ACM on Programming Languages, 3(OOPSLA):1–28, 2019

[27] Programmableweb, Dataset. [16] https://www.programmableweb.com/api/, 2014. [Online; accessed 2020].

[28] QWSdata, Dataset. https://qwsdata.github.io/citations.html, 2007. [On-line; accessed 2018]

[29] Github API. https://developer.github.com/v3/, 2019. [Online; accessed 2020].

[30] Hans van Vliet, Software Engineering Principles and Practice: Third Edition. 2008

[31] Cunningham, W., 1992. The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger, 4(2), pp.29-30

[32] Kruchten, P., Nord, R.L. and Ozkaya, I., 2012. Technical debt: From metaphor to theory and practice. Ieee software, 29(6), pp.18-21

[33] Chatzigeorgiou, A., Ampatzoglou, A., Ampatzoglou, A. and Amanatidis, T., 2015, October. Estimating the breaking point for technical debt. In 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD) (pp. 53-56). IEEE

[34] https://docs.oracle.com/javase/tutorial/deployment/jar/build.html

[35] Alizadeh-Sani Z., Martínez P.P., González G.H., González-Briones A., Chamoso P., Corchado J.M. (2021) A Hybrid Supervised/Unsupervised Machine Learning Approach to Classify Web Services. In: De La Prieta F., El Bolock A., Durães D., Carneiro J., Lopes F., Julian V. (eds) Highlights in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection. PAAMS Workshops 2021. Communications in Computer and Information Science, vol 1472. Springer, Cham. https://doi.org/10.1007/978-3-030-85710-3_8

[36] Ammar Ismael Kadhim. Survey on supervised machine learning techniques for automatic text classification. Artificial Intelligence Review, 52(1):273–292, 2019

[37] Haibo He and Edwardo A Garcia. Learning from imbalanced data. IEEE Transactions on knowledge and data engineering, 21(9):1263–1284, 2009