ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εφαρμογή διαχείρισης ηλεκτρονικών σημειώσεων με χρήση Spring Boot & Microservices** **E-notes management application with Spring Boot & Microservices**
Ονοματεπώνυμο Φοιτητή	**Σταύρος Λάϊος**
Πατρώνυμο	**Βασίλειος**
Αριθμός Μητρώου	**ΜΠΣΠ19025**
Επιβλέπων	**Ευθύμιος Αλέπης, Καθηγητής**

Ημερομηνία Παράδοσης **Ιούνιος 2024**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Ευχαριστίες

Για την υλοποίηση της παρούσας διπλωματικής διατριβής, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Ευθύμιο Αλέπη για την βοήθεια και την καθοδήγησή του καθ' όλη τη διάρκεια της εκπόνησης της εργασίας. Επίσης, θα ήθελα να ευχαριστήσω όλη την οικογένειά μου για την στήριξή τους κατά την διάρκεια των σπουδών μου καθώς και τον φίλο και συνάδελφο Βασίλη για τις συμβουλές στην εκπόνηση της παρούσας διπλωματικής εργασίας.

Περίληψη

Στη σημερινή εποχή της μηχανογράφησης, οι άνθρωποι και οι οργανισμοί καλούνται να χειριστούν ένα εύρος σημειώσεων που μπορεί να περιέχουν ευαίσθητα δεδομένα και να απαιτούν ισχυρά μέτρα ασφάλειας όπως η απόδοση λογαριασμού μισθοδοσίας αλλά και σημειώσεις που δεν φέρουν ευαίσθητες πληροφορίες όπως μια λίστα με ψώνια. Οι υπάρχουσες ρυθμίσεις μπορεί να χρειάζονται ορισμένες επισημάνσεις ή να θέτουν σε κίνδυνο την ασφάλεια, διευρύνοντας το αίτημα για ένα ολοκληρωμένο σύστημα διαχείρισης σημειώσεων. Στην παρούσα διατριβή παρουσιάζεται τόσο η ανάπτυξη όσο και η χρήση ενός ολοκληρωμένου συστήματος για την διαχείριση μεμονωμένων αλλά και ανοικτών σημειώσεων.

Abstract

In today's era of computerization, people and organizations are required to manage a range of notes that may contain sensitive data and require robust security measures such as payroll account certificate but also notes that do not carry sensitive information such as a shopping list. Existing arrangements may need certain markings or pose a safety risk, expanding the demand for an integrated notes management system. This dissertation presents both the development and use of an integrated system for the management of individual and open notes.

Πίνακας Περιεχομένων

1. Εισαγωγή

Σημαντικός αριθμός φορέων και οργανισμών τόσο στον ιδιωτικό όσο και στον δημόσιο τομέα σε Ελλάδα και εξωτερικό, καλούνται να διαχειριστούν ένα σύνολο εγγράφων, όπου αυτά τα έγγραφα περιέχουν πληροφορία η οποία είναι σημαντική στον εκάστοτε οργανισμό/φορέα.

Ένα τέτοιο έγγραφο αποτελεί ένας είδος σημείωσης στην οποία αποτυπώνεται ένα κόμματί πληροφορίας το όποιο αποτελεί και δίνει αξία στον οργανισμό/φορέας. Συνεπώς, γίνεται κατανοητό η διασφάλιση και η ακεραιότητα ενός τέτοιου εγγράφου όπου σε διαφορετική περίπτωση θα μπορούσε να επιφέρει πλήγμα ή ζημιά στον οργανισμό/φορέας. Ωστόσο, παρά την σημαντική της σημασία, η διαχείριση τέτοιων εγγράφων δεν γίνεται μέσα σε ένα κοινό και οριοθετημένο πλαίσιο. Κατά συνέπεια, αυτό οδηγεί σε σημαντική αύξηση του απαιτούμενου χρόνου που χρειάζεται για την διαχείριση σημειώσεων, δημιουργώντας πολλαπλά προβλήματα καθώς ο όγκος αυτών των εγγράφων δύναται να είναι αρκετά μεγάλος.

Σημαντικό μέρος στη διαδικασία της διαχείρισης αυτών των σημειώσεων αποτελεί και το είδος του εγγράφου. Τα έγγραφα διαφέρουν μεταξύ τους και αποτυπώνουν διαφορετική πληροφορία με αντίκτυπο στον φορέα/οργανισμό. Μερικά παραδείγματα, είναι όπως το έγγραφό μισθοδοσίας των υπάλληλών μιας εταιρείας, η απόδειξη αγοράς προϊόντος ενός πελάτης ή ακόμα και το έγγραφο πληρωμής του ηλεκτρικού ρεύματος του κτιρίου μιας εταιρείας το αποτελεί για την εταιρεία ένα πάγιο έξοδο. Δυστυχώς, έως σήμερα, φορείς και οργανισμοί δεν δίνουν την δέουσα σημασία και προσοχή στην αποτελεσματική διαχείριση τέτοιων εγγραφών με αποτέλεσμα να παρατηρείται διαρροή πληροφορίας προς τα έξω αλλά και απώλειας αυτής λόγω φθοράς/φυσικής καταστροφής καθώς υπάρχουν περιπτώσεις όπου η διαχείριση αυτών των εγγραφών γίνεται με έναν παραγκωνισμένο τρόπο οργάνωσης σε φυσικούς χώρους αποθήκευσης. Επιπλέον, παρατηρείται παντελής απουσία ενός αυτοματοποιημένου συστήματος μέσω του οποίου θα πραγματοποιείται η καταχώρηση, η οργάνωση και η διαχείριση των σημειώσεων. Ως αποτέλεσμα, ελλοχεύει ο κίνδυνος απώλειας και καταστροφής πληροφορίας η οποία αποτελεί σημαντική αξία στον οργανισμό/φορέα.

Στην παρούσα εργασία προσπαθούμε να λύσουμε τα προαναφερθέντα προβλήματα δημιουργώντας μια ηλεκτρονική διαδικτυακή εφαρμογή μέσω της οποίας θα απλοποιείται η διαχείριση των σημειώσεων και θα γίνεται πιο φιλική προς τους χρήστες ολόκληρη η διαδικασία καταχώρησης, οργάνωσης και προβολής των σημειώσεων.

Πιο συγκεκριμένα, ένας χρήστης μέσω κατάλληλης φόρμας θα μπορεί καταχωρεί σημειώσεις. Οι σημειώσεις αυτές θα δίνεται η επιλογή να χαρακτηρισθούν δημόσιες ή ιδιωτικές. Οι δημόσιες σημειώσεις ενός χρήστη θα είναι προσπελάσιμες και από άλλους χρήστες ενώ για κάθε χρήστη θα δίνεται η δυνατότητα να προσθέτει είτε δικές του σημειώσεις είτε σημειώσεις από άλλους χρήστες σε έναν ειδικό χώρο που θα λέγεται “Αγαπημένα”. Επιπλέον, ένας χρήστης θα μπορεί να τροποποιεί μια σημείωση του, να την διαγράφει αλλά και να προβάλει σημειώσεις. Ακόμα θα μπορεί μέσα στην εφαρμογή να αναζήτα σημειώσεις εφαρμόζοντας συγκεκριμένα φίλτρα. Ακόμα, ένας νέος χρήστης θα μπορεί να κάνει εγγραφή στην εφαρμογή μέσω κατάλληλης φόρμας, να προβάλει και να ενημερώνει τις πληροφορίες του προφίλ του καθώς και να διαγράφει τον λογαριασμό του αν το επιθυμεί.

Ο Διαχειριστής της εφαρμογής θα μπορεί να προβάλει τις πληροφορίες του λογαριασμού του καθώς και πληροφορίες/μετρικές που έχουν να κάνουν με την εύρυθμη λειτουργία της εφαρμογής. Τέλος, σέ όλο το μήκος της εφαρμογής υπάρχει η αυθεντικοποιήση του κάθε χρήστη που γίνεται μέσω διαπιστευτηρίων για την εύρυθμη λειτουργία της εφαρμογής.

Στην πρώτη ενότητα, γίνεται ανάλυση των διαδικτυακών εφαρμογών που βοήθησαν σημαντικά στην σύσταση του δικού μας συστήματος. Παράλληλα αναφέρονται οι βασικές τεχνολογίες που χρησιμοποιήσαμε προκειμένου να κατασκευασθεί η εφαρμογή. Στην επόμενη ενότητα, παρουσιάζονται οι βασικές παραδοχές και οι περιορισμοί που τέθηκαν για την ομαλή δημιουργία της εφαρμογής και αναφέρονται οι λειτουργίες και οι ειδικές απαιτήσεις των εμπλεκόμενων μελών. Στην τρίτη ενότητα, γίνεται ανάλυση του προβλήματος που διαχειρίζεται η εφαρμογή, της αρχιτεκτονικής που χρησιμοποιήθηκε για την κατασκευή του, των βασικών λειτουργιών των χρηστών καθώς και των βασικών κομματιών κώδικα. Τέλος, στην τελευταία ενότητα παρουσιάζεται ένα απλοποιημένο εγχειρίδιο χρήσης στο οποίο αναλύονται βήμα προς βήμα οι διαθέσιμες επιλογές που μπορούν να εκτελεσθούν από τους βασικούς χρήστες του συστήματος.

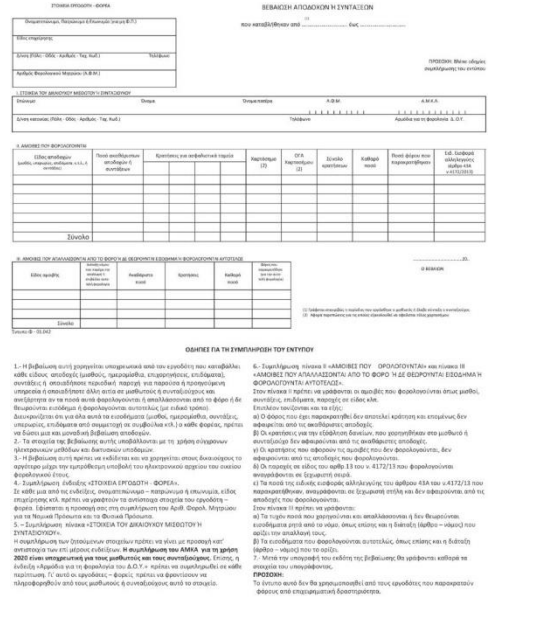
2. Γενική Περιγραφή

Στην συγκεκριμένη ενότητα, παρουσιάζονται η βασική οντότητα της εφαρμογής, ο παρελθοντικός τρόπος διαχείρισης σημειώσεων, τα υπάρχοντα συστήματα που συνδέονται στο πλαίσιο της διαχείρισης σημειώσεων ευρύτερα. Στην συνέχεια γίνεται αναφορά στα εργαλεία και στις σύγχρονες τεχνολογίες που χρησιμοποιήθηκαν ώστε να παραχθεί η συγκεκριμένη εφαρμογή. Επιπλέον, αναφέρονται οι βασικές παραδοχές και οι γενικοί περιορισμοί που λήφθηκαν υπόψη κατά τη δημιουργία της παρούσας εφαρμογής. Τέλος, γίνεται ανάλυση των λειτουργιών καθώς και των ειδικών απαιτήσεων των δύο βασικών οντοτήτων του συστήματος δηλαδή από την πλευρά του Χρήστη αλλά και του Διαχειριστή.

2.1 Σημείωση – Βασική Οντότητα

Σημείωση ή έγγραφο θεωρείται ένα τμήμα χαρτί πάνω στο οποίο αποτυπώνεται ένα τμήμα ή σύνολο δεδομένων. Το σύνολο της πληροφορίας που αποτυπώνεται στο έγγραφο μπορεί να ποικίλει και έχει να κάνει με τον φορέα/οργανισμό που διαχειρίζεται αυτή τη πληροφορία. Με την πάροδο των χρόνων, μια σημείωση ψηφιοποιήθηκε και εκτός της φυσικής μορφής, τμήμα χαρτιού, δύναται να έχει και ψηφιακή μορφή. Παρακάτω, παρουσιάζονται κάποια ενδεικτικά παράδειγμα που έχουν ως σκοπό την περαιτέρω ανασκόπηση της Σημείωσης ως βασικής οντότητας που αποτελεί τον κύριο πυλώνα της συγκεκριμένης διπλωματικής εργασίας.

Εικόνα 1 – Λογαριασμός Δ.Ε.Η.



Εικόνα 2 – Βεβαίωση Απόδοσης Αποδοχών ή Συντάξεων

Στην πρώτη εικόνα, παρουσιάζεται ένας λογαριασμός της Δ.Ε.Η. . Στον λογαριασμό της Δ.Ε.Η. αποτυπώνεται ένα σύνολο πληροφορίας που αφορά το σύνολο της πληρωμής, το κωδικό της πληρωμής, πότε λήγει η προθεσμία της πληρωμής καθώς και άλλες πληροφορίες που σχετίζονται με την κατανάλωση ηλεκτρικού ρεύματος ενός χώρου. Αυτή η πληροφορία είναι σημαντική τόσο για την ίδια την Δ.Ε.Η. αλλά και για τον ιδιοκτήτη του ακινήτου είτε πρόκειται για μεμονωμένο πρόσωπο είτε για οργανισμό/φορέα. Ενώ στην δεύτερη εικόνα παρουσιάζεται ένα έγγραφο που αφορά την βεβαίωση απόδοσης αποδοχών ή συντάξεων. Ένα τέτοιο έγγραφο φέρει πολύ προσωπικές πληροφορίες σχετικά με τις αποδοχές και τις συντάξεις ενός προσώπου και απαιτεί την δέουσα προσοχή όσο αφορά την αρχειοθέτηση και την οργάνωση του.

2.2 Παρελθοντική Διαχείριση Σημειώσεων

Στο παρελθόν, η διαχείριση των εγγραφών ενός οργανισμού/φορέα λάμβανε χώρα με έναν ποιο φυσικό τρόπο, μιας και η ψηφιοποιήση των εγγραφών καθώς και η χρήσης κατάλληλων συστημάτων βρίσκονταν σε προοίμιο στάδιο. Παρατηρήθηκε το φαινόμενο αρχειοθέτησης και οργάνωσης αυτών των εγγραφών σε ειδικούς χώρους – δωμάτια, όπου υπήρχε κάποιος άνθρωπος σε μία μόνιμη θέση για τον έλεγχο και την διαχείριση αυτών των εγγραφών.

Καθώς όμως ο όγκος της πληροφορίας μεγάλωνε αλλά και με την δημιουργία της ανάγκης και αποτελεσματικότερη οργάνωση και πληροφόρηση έφερε στην επιφάνεια την αναγκαιότητα της δημιουργίας πληροφοριακών συστημάτων που θα επιτελέσουν αυτό το έργο. Παράλληλα, η παρελθοντική διαχείριση των εγγραφών ελλόχευε κινδύνους φθοράς και καταστροφής των εγγραφών από φυσικές καταστροφές ή την πάροδο των χρονών με αποτέλεσμα να επισπευστεί η ψηφιοποιηση αυτών των εγγράψαν.

2.3 Υπάρχοντα Συστήματα

Τα τελευταία χρονιά όλο και περισσότεροι οργανισμοί έχουν αντιληφθεί την χρησιμότητα των συστημάτων διαχείρισης της γνώσης/πληροφορίας. Τέτοια συστήματα επιτρέπουν την καλύτερη οργάνωση και διαμοίραση της γνώσης με αποτέλεσμα την καλύτερη πληροφόρηση των ανθρώπων που κάνουν χρήση ενός τέτοιου συστήματος. Συνήθως υπάρχει αρκετός όγκος πληροφορίας και πολλές φορές απαιτείται η ανάκτηση της ή μέρος αυτής. Ωστόσο καθώς ο όγκος της πληροφορίας γίνεται ολοένα και περισσότερος παρατηρούνται στα συστήματα αυτά πολλά προβλήματα που σχετίζονται με την αναζήτηση, την ανάκτηση καθώς και την συντήρηση της γνώσης.

Η φύση αυτών των προβλημάτων βρίσκεται κατά κύριο λόγο στην έλλειψη κεντρικοποιήμενης οργάνωσης και διαχείρισης της αποθηκευμένης πληροφορίας. Όμως τον τελευταίο καιρό έχουν λάβει χώρα διάφορες τεχνολογίες όπου σχετίζονται με έναν ποιο ολιστικό τρόπο διαχείρισης της πληροφορίας εντός ενός οργανισμού. Ενδιαφέρον, παρατηρείται στην ανάπτυξη συστημάτων διαχείρισης εγγράφων/σημειώσεων από εταιρίες στον χώρο της πληροφορικής.

Τέτοιου είδους συστήματα έχουν εφαρμογές και στην εκπαιδευτική κοινότητα. Υπάρχει η ανάγκη της σταδιακής ενσωμάτωσης των συστημάτων διαχείρισης της πληροφορίας στον τομέα της εκπαίδευσης, συμβάλλοντας σημαντικά στη διαχείριση και βελτίωση του εκπαιδευτικού και ακαδημαϊκού υλικού. Η αυτοματοποίηση πολλών διαδικασιών, που κάποτε στερούσαν πολύτιμους ανθρώπινους και υλικούς πόρους, αποτελεί βασικό στόχο πολλών εκπαιδευτικών και όχι μόνο οργανισμών συμβάλλοντας στον εκσυγχρονισμό αυτών. Σημαντικό πλεονέκτημα τέτοιών συστημάτων είναι η εύκολη πρόσβαση σε δεδομένα χωρίς να απαιτείται φυσική παρουσία. Επιπλέον, η γρήγορη διάδοση και ανάπτυξη σχετικών τεχνολογιών, δίνει την δυνατότητα στους αρμόδιους φορείς να δημιουργήσουν μεγάλες πλατφόρμες διαχείρισης γνώσης και πληροφορίας υπό την μορφή εγγράφων/σημειώσεων. Μέσω αυτής παρέχεται πλήθος δεδομένων ελευθέρα στους χρήστες τους, που αποτελούν μέλη της ακαδημαϊκής κοινότητας. Ένα τέτοιο σύστημα μπορεί να επιδράσει θετικά τόσο στην ηλεκτρονική μάθηση (e-learning) αλλά και στην εξ΄ αποστάσεως εκπαίδευση όπου η διείσδυση της τεχνολογίας στις βαθμίδες της εκπαίδευσης δεν αποτελεί απειλή για το εκπαιδευτικό προσωπικό καθώς οι αυτοματοποιήσεις στόχο έχουν την διευκόλυνση του έργου όλων των συμβαλλόμενων που συμμετέχουν στις εκπαιδευτικές διαδικασίες, χωρίς να τους απομονώνει μεταξύ τους. Αντιθέτως, η ζωή όλων απλοποιείται σημαντικά, δίνοντας τους τη δυνατότητα να διαθέτουν τον ήδη περιορισμένο χρόνο τους σε ουσιώδη ζητήματα.

Με αυτό τον τρόπο, η δια δραστικότητα που πρέπει να όχι μόνο δεν απουσιάζει αλλά υποστηρίζεται ακόμη περισσότερο. Η λογική πάνω στην οποία βασίζεται η κατασκευή ενός τέτοιου συστήματος είναι τα διαδικτυακά περιβάλλοντα διαχείρισης σημειώσεων (web-based management notes). Πρόκειται για εφαρμογές βασισμένες στον παγκόσμιο ιστό (π.χ. google drive) που διευκολύνουν την διαμοίραση της πληροφορίας. Αυτά τα εργαλεία παρέχουν την δυνατότητα για ομαδική δουλειά (άμεση σύνδεση, διδακτικά υλικά, συνδέσεις με απομακρυσμένες πηγές πληροφορίας). Με βάση το προαναφερόμενο πλαίσιο, η προσπάθεια επικεντρώνεται στην σχεδίαση μιας εφαρμογής με στόχο την καλύτερη και ποιο αποτελεσματική διαχείριση εγγράφων και σημειώσεων με σκοπό ότι η πληροφορία παραμένει αναλλοίωτη όπου οποιαδήποτε στιγμή μπορεί να χρησιμοποιηθεί αποφέροντας πολύτιμο χρόνο στους χρήστες αυτής της εφαρμογής.

2.4 Σύγχρονες Τεχνολογίες

Στην συγκεκριμένη ενότητα παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν για την δημιουργία της εφαρμογής.

Java

Η Java είναι αντικειμενοστρεφής γλώσσα προγραμματισμού που σχεδιάστηκε από την εταιρεία πληροφορικής Sun Microsystems και στην παρούσα διπλωτική εργασία χρησιμοποιείται η έκδοση 17. Ως αντικειμενοστρεφής γλώσσα, η Java βασίζεται σε κλάσεις και αντικείμενα. Μια κλάση (class) είναι μια φόρμα για τη δημιουργία αντικειμένων (objects ή instances). Αφού πρώτα δημιουργηθεί μια κλάση, στη συνέχεια μπορεί να χρησιμοποιηθεί για την παραγωγή αντικειμένων. Παρομοιάζοντας, θα μπορούσαμε να πούμε ότι η κλάση αποτελεί μια "συνταγή" για τη δημιουργία αντικειμένων.

Στις αρχές του 1991, η Sun αναζητούσε το κατάλληλο εργαλείο για να αποτελέσει την πλατφόρμα ανάπτυξης λογισμικού σε μικρο-συσκευές (έξυπνες οικιακές συσκευές έως πολύπλοκα συστήματα παραγωγής γραφικών). Τα εργαλεία της εποχής ήταν γλώσσες όπως η C++ και η C. Μετά από διάφορους πειραματισμούς προέκυψε το συμπέρασμα ότι οι υπάρχουσες γλώσσες δεν μπορούσαν να καλύψουν τις ανάγκες τους. Ο "πατέρας" της Java, James Gosling, που εργαζόταν εκείνη την εποχή για την Sun, έκανε ήδη πειραματισμούς πάνω στη C++ και είχε παρουσιάσει κατά καιρούς κάποιες πειραματικές γλώσσες (C++ ++, που μετέπειτα ονομάστηκε C#) ως πρότυπα για το νέο εργαλείο που αναζητούσαν στην Sun. Τελικά μετά από λίγο καιρό κατέληξαν με μια πρόταση για το επιτελείο της εταιρίας, η οποία ήταν η γλώσσα Oak. Το όνομά της το πήρε από το ομώνυμο δένδρο (βελανιδιά) το οποίο ο Gosling είχε έξω από το γραφείο του και έβλεπε κάθε μέρα.

Η Oak ήταν μία γλώσσα που διατηρούσε μεγάλη συγγένεια με την C++. Παρόλα αυτά είχε πολύ πιο έντονο αντικειμενοστρεφή (object oriented) χαρακτήρα σε σχέση με την C++ και χαρακτηριζόταν για την απλότητα της. Σύντομα η υπεύθυνοι ανάπτυξης της νέας γλώσσας ανακάλυψαν ότι το όνομα Oak ήταν ήδη κατοχυρωμένο οπότε κατά την διάρκεια μιας εκ των πολλών συναντήσεων σε κάποιο τοπικό καφέ αποφάσισαν να μετονομάσουν το νέο τους δημιούργημα σε Java που εκτός των άλλων ήταν το όνομα της αγαπημένης ποικιλίας καφέ για τους δημιουργούς της. Η επίσημη εμφάνιση της Java αλλά και του HotJava (πλοηγός με υποστήριξη Java) στη βιομηχανία της πληροφορικής έγινε το Μάρτιο του 1995 όταν η Sun την ανακοίνωσε στο συνέδριο Sun World 1995. Ο πρώτος μεταγλωττιστής (compiler) της ήταν γραμμένος στη γλώσσα C από τον James Gosling. Το 1994, ο A.Van Hoff ξαναγράφει τον μεταγλωττιστή της γλώσσας σε Java, ενώ το Δεκέμβριο του 1995 πρώτες οι IBM, Borland, Mitsubishi Electronics, Sybase και Symantec ανακοινώνουν σχέδια να χρησιμοοιήσουν τη Java για την δημιουργία λογισμικού. Από εκεί και πέρα η Java ακολουθεί μία ανοδική πορεία και είναι πλέον μία από τις πιο δημοφιλείς γλώσσες στον χώρο της πληροφορικής. Στις 13 Νοεμβρίου του 2006 η Java έγινε πλέον μια γλώσσα ανοιχτού κώδικα (GPL) όσον αφορά το μεταγλωττιστή (javac) και το πακέτο ανάπτυξης (JDK, Java Development Kit).

Στις 27 Απριλίου 2010 η εταιρία λογισμικού Oracle Corporation ανακοίνωσε ότι μετά από πολύμηνες συζητήσεις ήρθε σε συμφωνία για την εξαγορά της Sun Microsystems και των τεχνολογιών (πνευματικά δικαιώματα/ πατέντες) που η δεύτερη είχε στην κατοχή της ή δημιουργήσει. Η συγκεκριμένη συμφωνία θεωρείται σημαντική για το μέλλον της Java και του γενικότερου οικοσυστήματος τεχνολογιών γύρω από αυτή μιας και ο έμμεσος έλεγχος της τεχνολογίας και η εξέλιξη της περνάει σε άλλα χέρια.

Spring Framework

Το Spring Framework είναι μια πλατφόρμα Java, ανοιχτού κώδικα, η οποία παρέχει υποστήριξη για την γρήγορη δημιουργία εφαρμογών Java. Είναι το πιο δημοφιλές framework ανάπτυξης, enterprise εφαρμογών Java. Χρησιμοποιείται ευρέως για την κατασκευή μικροϋπηρεσιών, εφαρμογών Ιστού και άλλων έργων που βασίζονται σε Java λόγω της ευκολίας χρήσης και της στιβαρότητάς του. Στην παρούσα διπλωματική εργασία γίνεται η χρήση της έκδοσης 6.

Χαρακτηριστικά του Spring Framework

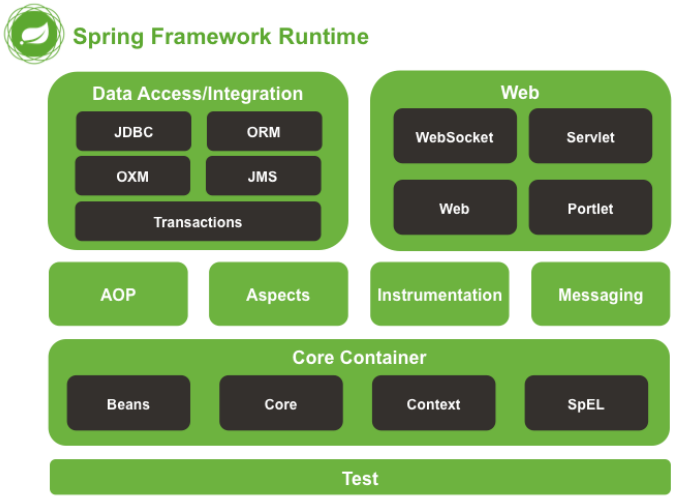
- Ελαφρύ
    - Το Spring είναι ένα ελαφρύ, modular framework, το οποίο επιτρέπει την επιλογή οποιουδήποτε module, πάνω απο το Spring Core.
- Inversion Of Control
    - Οι εξαρτήσεις της εφαρμογής ικανοποιούνται από το ίδιο το framework. Το Framework δημιουργεί τα αντικείμενα κατά την εκτέλεση του εφαρμογής και ικανοποιεί τις εξαρτήσεις της εφαρμογής.
- Aspect Oriented Programming
    - Οι μηχανικοί λογισμικού μπορούν να χρησιμοποιήσουν το AOP, για να αναπτύξουν μια εφαρμογή, όπου το business logic είναι διαχωρισμένο από τα system services.
- Container
    - Παρέχει το δικό του container, για τη διαχείριση των αντικειμένων - beans.
- MVC Framework
    - Το Spring MVC Framework μπορεί να χρησιμοποιηθεί, για την ανάπτυξη web εφαρμογών.
- Transaction Management
    - Παρέχει ένα γενικό επίπεδο Transaction Management.

Πλεονεκτήματα του Spring Framework

- Το Spring επιτρέπει στους μηχανικούς λογισμικού, να αναπτύσσουν enterprise εφαρμογές, χρησιμοποιώντας POJOs. Το πλεονέκτημα της χρησιμοποίησης μόνο POJOs είναι ότι δεν χρειαζόμαστε ένα EJB container, όπως ένα application server, αλλά έχουμε την επιλογή, να χρησιμοποιήσουμε ένα servlet Container όπως Tomcat.
- Η εφαρμογές σε Spring έχουν χαμηλή σύζευξη, εξαιτίας του dependency injection.
- Τα IoC containers τείνουν να είναι ελαφριά, ειδικά σε σύγκριση με τα EJB containers. Αυτό είναι χρήσιμο, για την ανάπτυξη εφαρμογών, που θα φιλοξενηθούν σε υπολογιστές με περιορισμένη μνήμη και CPU.
- Είναι χωρισμένο σε modules. Αν και έχει πολλά πακέτα και κλάσεις, χρησιμοποιούμε μόνο αυτά που χρειαζόμαστε.
- Δεν προσπαθεί, να ανακαλύψει ξανά τον τροχό. Αντίθετα, χρησιμοποιεί κάποιες από τις υπάρχουσες τεχνολογίες, όπως ORM Frameworks, loggin frameworks, JEE και άλλες τεχνολογίες.
- Ο έλεγχος μιας εφαρμογής, γραμμένη σε Spring είναι απλό γιατί ο εξαρτημένος από το περιβάλλον κώδικας, είναι μετακινημένος μέσα στο framework. Επίσης, χρησιμοποιώντας POJOs, είναι ευκολότερο, να χρησιμοποιήσουμε dependency injection, για να κάνουμε inject test δεδομένα.
- Το web framework του Spring είναι ένα καλοσχεδιασμένο MVC Framework, το οποίο αποτελεί μια εξαιρετική εναλλακτική αντί για web frameworks όπως το Strutts.

Spring Modules

Το Spring Framework αποτελείται από features, τα οποία είναι οργανωμένα σε 20 modules. Αυτά τα modules είναι ομαδοποιημένα στο Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging και Test.



MySQL Server Workbench

Το MySQL Workbench είναι ένα οπτικό εργαλείο σχεδίασης βάσεων δεδομένων που ενσωματώνει την ανάπτυξη, τη διαχείριση, το σχεδιασμό, τη δημιουργία και τη συντήρηση της βάσης δεδομένων SQL σε ένα ενιαίο ολοκληρωμένο περιβάλλον ανάπτυξης για το σύστημα βάσεων δεδομένων MySQL.

Στην παρούσα διπλωματική εργασία γίνεται η χρήση της έκδοσης 8.0 . Σχετικά, με αυτήν την έκδοση, στις 5 Απριλίου 2018, η ομάδα του MySQL Workbench ανακοίνωσε την πρώτη δημόσια έκδοση της έκδοσης 8.0.11 ως υποψήφια έκδοση (RC) μαζί με τον MySQL Community Server 8.0.11. Η πρώτη έκδοση γενικής διαθεσιμότητας (GA) εμφανίστηκε στις 27 Ιουλίου 2018 και πάλι μαζί με τον διακομιστή ακολουθώντας τη νέα πολιτική για την ευθυγράμμιση των αριθμών εκδόσεων στα περισσότερα προϊόντα MySQL. Το MySQL Workbench χρησιμοποιεί τώρα το ANTLR4 ως αναλυτή υποστήριξης και διαθέτει μια νέα μηχανή αυτόματης συμπλήρωσης που λειτουργεί με επεξεργαστές αντικειμένων (έναρξη, προβολές, αποθηκευμένες διαδικασίες και λειτουργίες) στον οπτικό επεξεργαστή SQL και σε μοντέλα. Η νέα προεπιλεγμένη προσθήκη ελέγχου ταυτότητας caching_sha2_password στο MySQL 8.0 υποστηρίζεται πλέον από το Workbench, επομένως η επαναφορά των λογαριασμών χρηστών σε άλλους τύπους ελέγχου ταυτότητας δεν είναι πλέον απαραίτητη κατά τη σύνδεση με τους πιο πρόσφατους διακομιστές. Οι καρτέλες διαχείρισης ενημερώνονται με τις πιο πρόσφατες επιλογές διαμόρφωσης και η διεπαφή χρήστη έγινε πιο συνεπής μεταξύ των καρτελών.

RabbitMQ

Το RabbitMQ είναι ένα λογισμικό τύπου message-broker ανοιχτού κώδικα που διευκολύνει την επικοινωνία μεταξύ διαφορετικών συστημάτων. Εφαρμόζει το Advanced Message Queuing Protocol (AMQP), το οποίο είναι ένα τυπικό πρωτόκολλο για την ανταλλαγή μηνυμάτων. Το RabbitMQ έχει σχεδιαστεί για να είναι ισχυρό, επεκτάσιμο και εξαιρετικά διαθέσιμο, καθιστώντας το μια δημοφιλή επιλογή για την κατασκευή κατανεμημένων συστημάτων.

<u>Τα βασικά χαρακτηριστικά της RabbitMQ περιλαμβάνουν:</u>

Message Broker: Η RabbitMQ λειτουργεί ως μεσάζων που λαμβάνει μηνύματα από producers (εφαρμογές που στέλνουν μηνύματα) και τα παραδίδει στους consumers (εφαρμογές που λαμβάνουν και επεξεργάζονται μηνύματα).

Queues: Τα μηνύματα τοποθετούνται σε ουρές, οι οποίες λειτουργούν ως μηχανισμοί αποθήκευσης και διανομής. Οι consumers μπορούν να εγγραφούν σε συγκεκριμένες ουρές για να λαμβάνουν μηνύματα.

Exchanges: Οι producers στέλνουν μηνύματα, τα οποία στη συνέχεια δρομολογούνται σε μία ή περισσότερες ουρές με βάση τους κανόνες δρομολόγησης που ορίζονται από τον τύπο ανταλλαγής.

Bindings: Τα bindings καθορίζουν τη σχέση μεταξύ exchanges και routes. Καθορίζουν τον τρόπο δρομολόγησης των μηνυμάτων από τα exchanges στα queues.

Routing: Η RabbitMQ υποστηρίζει διάφορους μηχανισμούς δρομολόγησης, συμπεριλαμβανομένων των ανταλλαγών απευθείας, fanout, topic, and headers exchanges, επιτρέποντας ευελιξία στη δρομολόγηση μηνυμάτων.

Message Acknowledgment: Η RabbitMQ υποστηρίζει επιβεβαίωση μηνύματος, επιτρέποντας στους consumers να επιβεβαιώσουν την επιτυχή επεξεργασία ενός μηνύματος. Αυτό βοηθά στη διασφάλιση αξιόπιστης παράδοσης μηνυμάτων.

Durability: Η RabbitMQ παρέχει επιλογές για ανθεκτικότητα των μηνυμάτων και των ουρών, διασφαλίζοντας ότι τα μηνύματα δεν χάνονται ακόμη και σε περίπτωση επανεκκίνησης ή αποτυχίας του broker.

Plugins and Extensibility: Η RabbitMQ είναι επεκτάσιμη και υποστηρίζει πρόσθετα που παρέχουν πρόσθετες λειτουργίες. Μπορεί να ενσωματωθεί με άλλες τεχνολογίες, όπως η μετατροπή μηνυμάτων ή οι μηχανισμοί ελέγχου ταυτότητας.

Παραδείγματα περιπτώσεων χρήσης της RabbitMQ

Decoupling Microservices: Η RabbitMQ μπορεί να χρησιμοποιηθεί για την αποσύνδεση μικροϋπηρεσιών επιτρέποντάς τους να επικοινωνούν μέσω ασύγχρονης μετάδοσης μηνυμάτων.

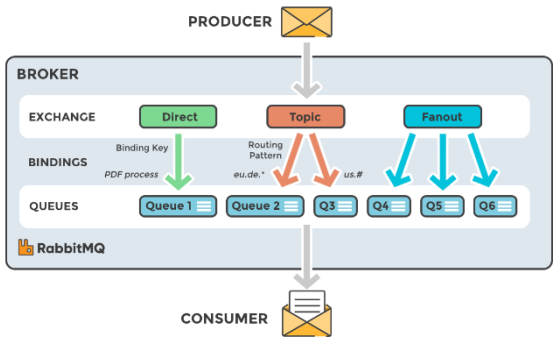
Task Queues: Η RabbitMQ χρησιμοποιείται συχνά σε σενάρια task queues, όπου οι εργασίες (tasks) κατανέμονται μεταξύ πολλών workers για παράλληλη επεξεργασία.

Event-Driven Architectures: Η RabbitMQ είναι κατάλληλο για την κατασκευή συστημάτων που βασίζονται σε συμβάντα, όπου τα στοιχεία αντιδρούν σε συμβάντα ή μηνύματα.

Data Synchronization: Η RabbitMQ μπορεί να χρησιμοποιηθεί για συγχρονισμό δεδομένων μεταξύ κατανεμημένων συστημάτων, διασφαλίζοντας ότι οι ενημερώσεις διαδίδονται αποτελεσματικά.

Load Balancing: Η RabbitMQ βοηθά στην κατανομή του φόρτου εργασίας μεταξύ πολλών καταναλωτών, επιτρέποντας την εξισορρόπηση φορτίου σε κατανεμημένα συστήματα.

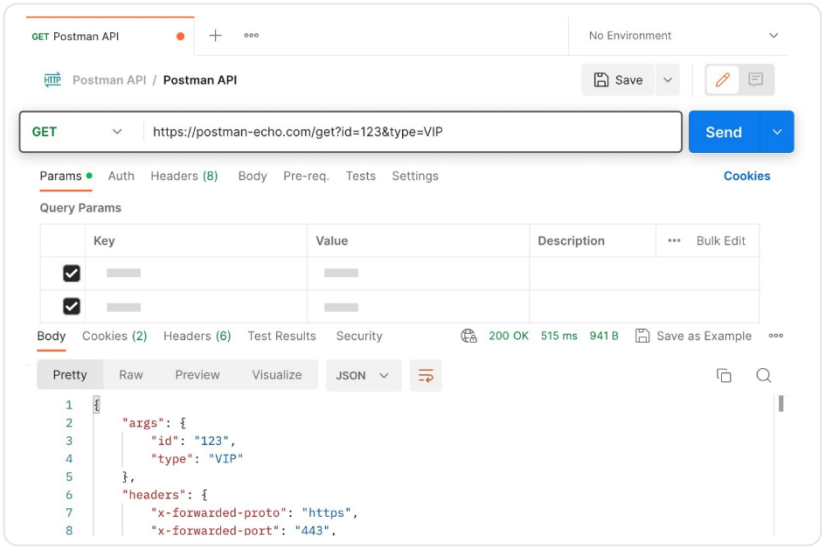
Η RabbitMQ είναι γραμμένο στην γλώσσα προγραμματισμού Erlang και παρέχει βιβλιοθήκες πελατών για διάφορες γλώσσες προγραμματισμού, καθιστώντας το ευέλικτο και προσβάσιμο για μηχανικούς λογισμικού που εργάζονται με διαφορετικές τεχνολογίες. Στην παρούσα διπλωματική εργασία γίνεται χρήση της έκδοσης 15 για την Erlang και χρησιμοποιείται η έκδοση 3 για την RabbitMQ.



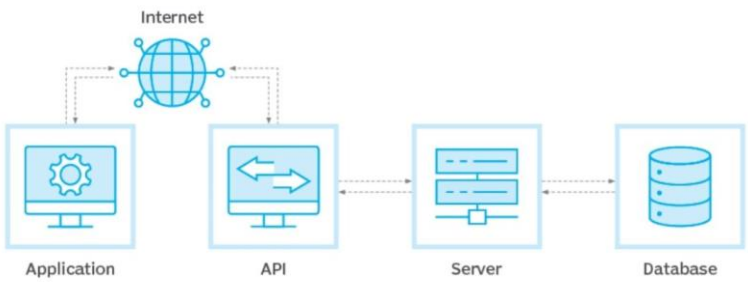
Postman API

Το Postman είναι ένα από τα πιο δημοφιλή εργαλεία δοκιμής λογισμικού που χρησιμοποιείται για δοκιμές API. Με τη βοήθεια αυτού του εργαλείου, οι μηχανικοί λογισμικού μπορούν εύκολα να δημιουργήσουν, να δοκιμάσουν, να μοιραστούν και να τεκμηριώσουν API.

Το Postman είναι μια αυτόνομη πλατφόρμα δοκιμής λογισμικού API (Διασύνδεση Προγραμματισμού Εφαρμογών) για τη δημιουργία, δοκιμή, σχεδιασμό, τροποποίηση και τεκμηρίωση API. Είναι μια απλή γραφική διεπαφή χρήστη για αποστολή και προβολή αιτημάτων και απαντήσεων HTTP. Κατά τη χρήση του Postman, για λόγους δοκιμής, δεν χρειάζεται να γράψετε κανέναν κώδικα http client network. Αντίθετα, δημιουργούμε δοκιμαστικές σουίτες που ονομάζονται collections και το Postman στην συνέχεια αλληλεπιδρά με το API.

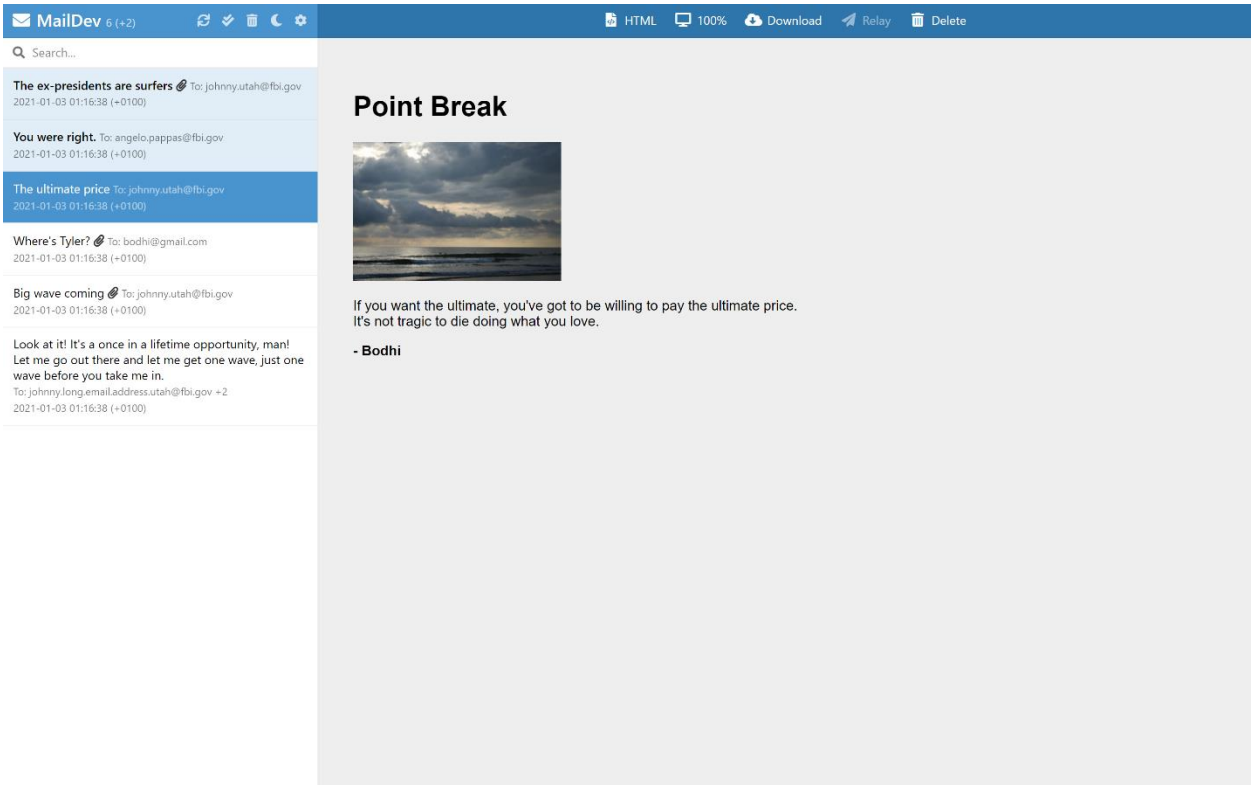


Το API (Application Programming Interface), είναι η διεπαφή μέσω της οποίας τα δεδομένα τα οποία καταχωρείς, σε μία εφαρμογή ή πλατφόρμα, αποστέλλονται σε έναν server και στέλνονται ξανά πίσω σε σένα, με τις απαντήσεις και τα αποτελέσματα που επιθυμείς. Ένα παράδειγμα για καλύτερη κατανόηση του πώς ακριβώς λειτουργεί το API, είναι οι πλατφόρμες που λειτουργούν ως μεσάζοντες θα λέγαμε, των e-shops κάθε επιχείρησης (πχ Skroutz). Ο χρήστης μπαίνει στην πλατφόρμα και καταχωρεί τα δεδομένα της αναζήτησής που επιθυμεί. Στη συνέχεια, η πλατφόρμα “επικοινωνεί” με την ιστοσελίδα της κάθε επιχείρησης ώστε να δώσει πίσω τις πληροφορίες που χρειάζεται κάποιος.



MailDev

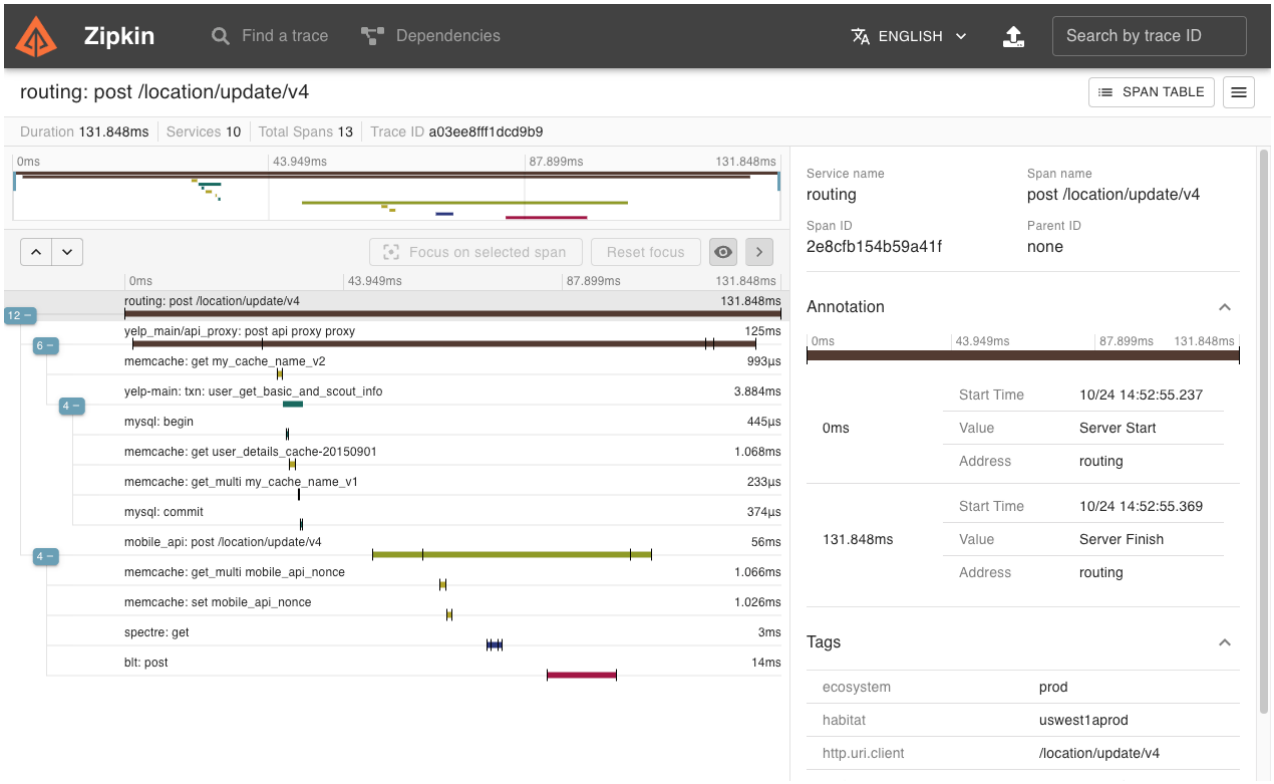
Το MailDev αποτελεί τον απλούστερο τρόπο ελέγχου του παραγόμενου e-mail μιας εφαρμογής που βρίσκεται σε περίοδο development (ανάπτυξη της εφαρμογής) παρέχοντας ένα εύκολο διαδικτυακό περιβάλλον χρήσης (web interface) και το οποίο μπορεί να χρησιμοποιηθεί στον τοπικό υπολογιστή (localhost) του μηχανικού λογισμικού που αναπτύσσει την εφαρμογή. Το MailDev έχει δημιουργηθεί στην γλώσσα προγραμματισμού JavaScript κάνοντας χρήση της βιβλιοθήκης Node.js. Στην παρούσα διπλωματική εργασία γίνεται χρήση της έκδοσης 2.



Zipkin

Το Zipkin είναι ένα έργο που ξεκίνησε στο Twitter το 2010 και βασίζεται στα έγγραφα Google Dapper. Η παρατήρηση του συστήματος από διαφορετικές γωνίες είναι κρίσιμη κατά την αντιμετώπιση προβλημάτων, ειδικά όταν ένα σύστημα είναι πολύπλοκο και κατανεμημένο.

Το Zipkin βοηθά στη συλλογή δεδομένων (με βάση το χρόνο) που απαιτούνται για την αντιμετώπιση προβλημάτων που προκύπτουν στις εφαρμογές. Τα χαρακτηριστικά περιλαμβάνουν τόσο τη συλλογή όσο και την αναζήτηση αυτών των δεδομένων. Βοηθάει επίσης στην προβολή πού ακριβώς ξοδεύεται ο περισσότερος χρόνος ενός αιτήματος προς την εφαρμογή. Είτε πρόκειται για εσωτερική κλήση είτε για εξωτερική κλήση σε άλλη υπηρεσία, μπορεί να οργανωθεί η εφαρμογή για την απεικόνιση αυτής της πληροφορίας. Οι μικροϋπηρεσίες συνήθως μοιράζονται αυτή την πληροφορία συσχετίζοντας αιτήματα με ένα μοναδικό αναγνωριστικό.

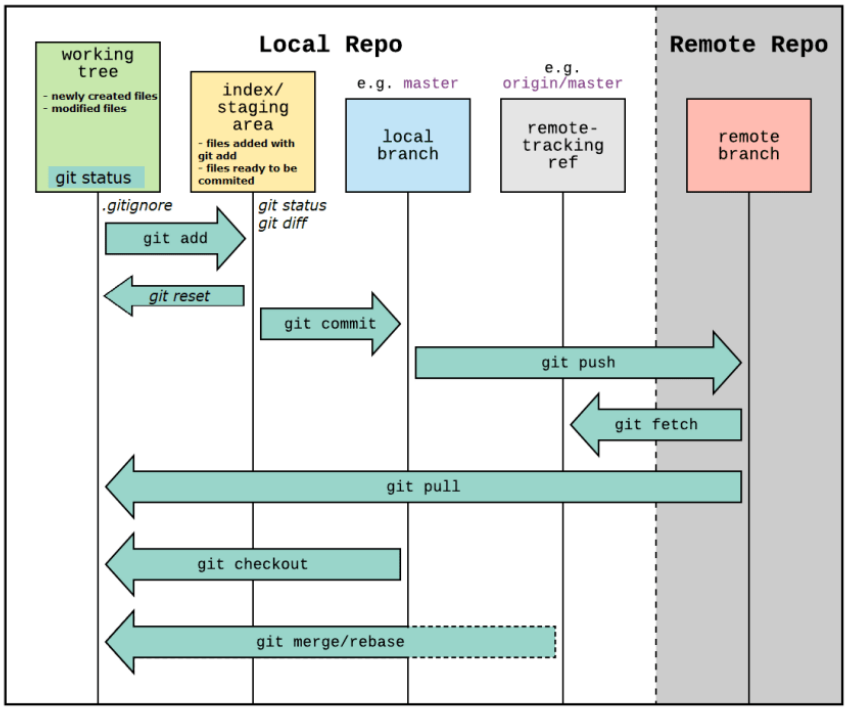


Git

Το version control ή στα ελληνικά ο έλεγχος εκδόσεων, επίσης γνωστό και έλεγχος πηγαίου κώδικα, είναι η πρακτική της παρακολούθησης και διαχείρισης αλλαγών στον κώδικα λογισμικού. Τα συστήματα ελέγχου εκδόσεων είναι εργαλεία λογισμικού που βοηθούν τις ομάδες των μηχανικών λογισμικού να διαχειρίζονται αλλαγές στον πηγαίο κώδικα κατά το πέρασμα του χρόνου. Χρησιμοποιώντας ένα τέτοιο σύστημα υπάρχει η δυνατότητα να αποθηκεύετε το ιστορικό των αλλαγών του κώδικα της εφαρμογής και γενικότερα να γίνεται tracking στο τι συμβαίνει στα αρχεία του κώδικα μιας εφαρμογής τόσο στη φάση ανάπτυξης αλλά και στη φάση του live production.

Είναι ιδιαίτερα χρήσιμο όταν αναπτύσσετε κάποιο έργο λογισμικού στο πλαίσιο μιας ομάδας όπου αποτελείται από δύο ή περισσότερα άτομα, γιατί μπορούν να ενώσουν πολύ εύκολα τα κομμάτια του κώδικα που γράφει ο κάθε μηχανικός λογισμικού ξεχωριστά. Η καρδιά ενός τέτοιος συστήματος είναι να κρατάει τις αλλαγές που γίνονται σε ένα έργο σε όλη τη διάρκεια της ύπαρξης του. Οπότε οι μηχανικοί λογισμικού σε τακτικά διαστήματα εισάγουν αλλαγές ή καινούργια κομμάτια κώδικα στην εφαρμογή και ανά πάσα στιγμή μπορούν να πάνε πίσω στην ιστορία και να κάνουν check-out σε παρελθοντικά χρονικά σημεία.

Ακόμα και αν κάποιος δουλεύει μόνος του και όχι στα πλαίσια κάποιας ομάδας, αν δημιουργηθεί κάποιο πρόβλημα στο λογισμικό ή έχει κάποιο σημαντικό πρόβλημα (bug) μπορεί πολύ εύκολα να επαναφέρει σε μία προηγούμενη έκδοση την εφαρμογή μέχρι να λυθεί το εκάστοτε πρόβλημα.



2.7 Λειτουργικότητες Χρήστη

Κατόπιν της ταυτοποίησης του από το σύστημα, ένας χρήστης έχει την δυνατότητα να εκτελέσει τις λειτουργίες που ορίζονται παρακάτω,

- προβολή πληροφορίας του λογαριασμού
- ενημέρωση πληροφορίας του λογαριασμού
- διαγραφή του λογαριασμού
- εισαγωγή μιας νέας σημείωσης του χρήστη
- τροποποίησης μιας σημείωσης του χρήστη
- διαγραφή μιας σημείωσης του χρήστη
- προβολή μιας σημείωσης του χρήστη
- προβολή όλων των σημειώσεων του χρήστη
- αναζήτηση σημειώσεων άλλων χρηστών μέσα στην εφαρμογή βάσει του συγγραφέα
- αναζήτηση σημειώσεων άλλων χρηστών μέσα στην εφαρμογή βάσει του τίτλου
- εισαγωγή σημειώσεων στη λίστα “Αγαπημένα”
- προβολή σημειώσεων που βρίσκονται στην λίστα “Αγαπημένα”

Ένας γενικός κανόνας που λήφθηκε υπόψη κατά την ανάπτυξη της εφαρμογής είναι ότι μία σημείωση μπορεί να χαρακτηρισθεί ως δημοσιά ή ιδιωτική. Μια δημοσιά σημείωση ενός χρήστη μπορεί να προβληθεί στην αναζήτηση σημειώσεων από άλλους χρήστες ενώ μια ιδιωτική σημείωση μπορεί να προβληθεί μόνο από τον χρήστη που την καταχώρησε.

Επιπρόσθετα, η εισαγωγή σημειώσεων στην λίστα “Αγαπημένα” μπορεί να γίνει μόνο για σημειώσεις που έχει καταχωρήσει ο ίδιος ο χρήστης και όχι προτεθούν σε αυτή την λίστα σημειώσεις άλλων χρηστών.

Οι επιτρεπόμενοι τύποι σημειώσεων που γίνεται να καταχωρηθούν στην εφαρμογή είναι οι:

- .txt, .csv, .doc, .docx
- .xlx, .xlxs, .ppt, .pdf
- .java, .js, .csharp, .c
- .c++, .sql, .jpeg, .png

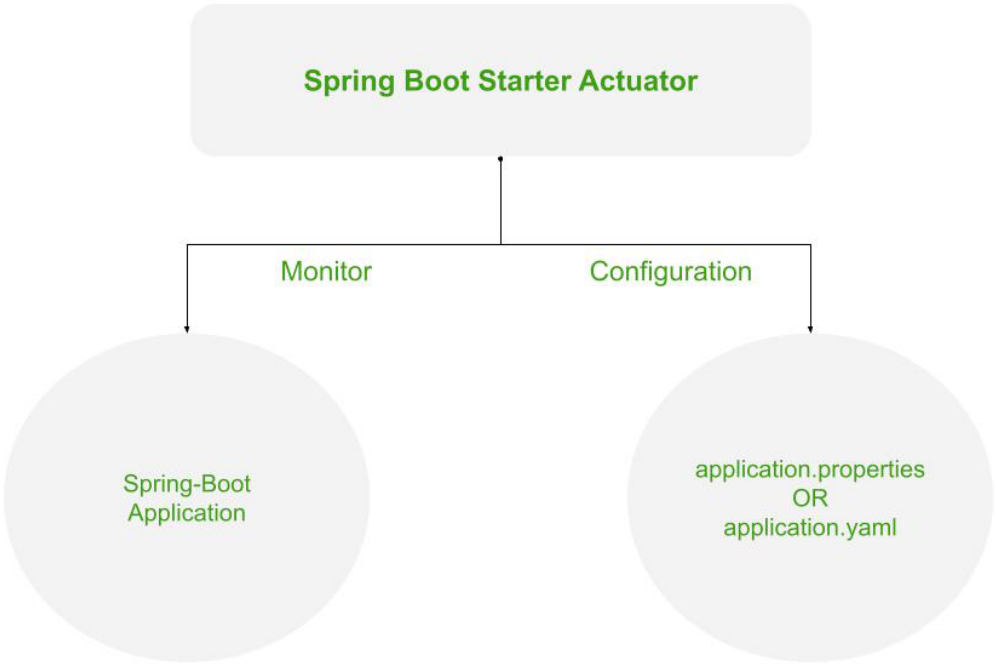
2.8 Λειτουργικότητες Διαχειριστή

Κατόπιν της ταυτοποίησης του από το σύστημα, ο διαχειριστής έχει την δυνατότητα να εκτελέσει τις λειτουργίες που ορίζονται παρακάτω,

- προβολή πληροφορίας του λογαριασμού
- προβολή πληροφορίας για την υγεία των μικοϋπηρεσιών
- προβολή πληροφορίας για το σύνολο των εγγεγραμμένων χρηστών
- προβολή πληροφορίας για το σύνολο των ενεργοποιημένων χρηστών
- προβολή πληροφορίας για το σύνολο των σημειώσεων
- προβολή πληροφορίας για το σύνολο των δημόσιων σημειώσεων
- προβολή πληροφορίας για το σύνολο των ιδιωτικών σημειώσεων
- προβολή πληροφορίας για το σύνολο των “αγαπημένων” σημειώσεων
- προβολή πληροφορίας για το σύνολο των δημόσιων “αγαπημένων” σημειώσεων
- προβολή πληροφορίας για το σύνολο των ιδιωτικών “αγαπημένων” σημειώσεων
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .txt
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .csv
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .doc
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .docx
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .xlx
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .xlxs
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .ppt
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .pdf
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .java
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .js
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .csharp
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .c
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .cplusplus
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .sql
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .jpeg
- προβολή πληροφορίας για το σύνολο των σημειώσεων που ο τύπος τους είναι .png

Για την προβολή πληροφορίας για την υγεία των μικροϋπηρεσιών, χρησιμοποιείται ένα module της πλατφόρμας του Spring, το οποίο ονομάζεται Actuator. Η ανάπτυξη και η διαχείριση μιας εφαρμογής είναι οι δύο πιο σημαντικές πτυχές του κύκλου ζωής της εφαρμογής. Είναι πολύ σημαντικό να υπάρχει η πληροφορία για το τι συμβαίνει κάτω από την εφαρμογή.

Επίσης, όταν η εφαρμογή “βγαίνει” στην παραγωγή, η διαχείρισή της γίνεται σταδιακά εξαιρετικά σημαντική. Επομένως, συνιστάται πάντα η παρακολούθηση της εφαρμογής τόσο κατά τη φάση ανάπτυξης όσο και κατά τη φάση παραγωγής. Για την ίδια περίπτωση χρήσης, το Spring Boot παρέχει έναν Actuator που μπορεί να χρησιμοποιηθεί για την παρακολούθηση και τη διαχείριση της εφαρμογής. Με τα endpoints /actuator και /actuator/health μπορείτε να επιτευχθεί ο σκοπό της παρακολούθησης.



3. Αναλυτική Περιγραφή Αρχιτεκτονικής Λογισμικού

3.1 Βασικός Στόχος Δημιουργίας Λογισμικού

Η διαδικασία οργάνωσης, αρχειοθέτησης και διαχείρισης σημειώσεων στη σημερινή εποχή με τον όλο και αυξανόμενο όγκο πληροφορίας που καλούνται οι άνθρωποι να διαχειριστούν είναι προβληματισμοί που απασχολούν όχι μόνο μεμονωμένα τον κάθε άνθρωπο αλλά και ένα σύνολο οργανισμών/φορέων. Δυστυχώς, έως σήμερα, δεν υπάρχει ένα σύστημα πληροφορικής που να μπορεί να ενορχηστρώσει και να διαχειριστεί ένα σύνολο σημειώσεων, διαφορετικών στο είδος τους, με ένα αποτελεσματικό και ποιο κεντρικοποιήμενο τρόπο. Ως αποτέλεσμα, τα υπάρχοντα συστήματα που υπάρχουν δυσκολεύουν σημαντικά αυτή την διαδικασία, αυξάνοντας παράλληλα σε σημαντικό βαθμό το φόρτο εργασίας τόσο σε οργανισμούς/φορείς όσο και σε ένα μεμονωμένο άτομο.

Με αφορμή τα παραπάνω, στην παρούσα εργασία γίνεται η προσπάθεια να δοθεί μια λύση στα προαναφερθέντα προβλήματα δημιουργώντας μια εφαρμογή μέσω της οποίας θα απλοποιείται η καταχώρηση, η οργάνωση και η διαχείριση σημειώσεων και θα γίνεται πιο φιλική προς τους χρήστες ολόκληρη η διαδικασία της ψηφιακής ενοποίησης διαφορετικών σημειώσεων σε ένα και μόνο μέρος.

Μέσα από την παρούσα εργασία, γίνεται προσπάθεια να δημιουργηθεί ένα σύστημα που θα διαχειρίζεται τις διαφορετικές σημειώσεις του κάθε χρήστη ξεχωριστά. Τέλος επιδιώκεται η μείωση χρόνου στην οργάνωση και αρχειοθέτηση των διαφορετικών σημειώσεων που ο κάθε χρήστης χρειάζεται να επιτύχει μέσα στην καθημερινότητα του άλλα και σε οργανισμούς/φορείς που καλούνται καθημερινά να διαχειριστούν αποτελεσματικά ένα μεγάλο όγκο εγγραφών/σημειώσεων με πληροφορίες οι οποίες κουβαλάνε προστιθέμενη αξία.

3.2 Θέματα Απόδοσης/Ασφάλειας

Η εφαρμογή διαχείρισης σημειώσεων παρέχει μια φόρμα σύνδεσης στην εφαρμογή για την αυθεντικοποιήση του χρήστη. Μόνο οι χρήστες οι οποίοι έχουν αναγνωριστεί και ταυτοποιηθεί από το την εφαρμογή τους δίνεται το δικαίωμα να εκτελούν λειτουργίες μέσα στο περιβάλλον της εφαρμογής. Η φόρμα ή οποία χρησιμοποιείται ώστε να ταυτοποιηθεί ο χρήστης περιέχει κάποια διαπιστευτήρια. Αυτά τα διαπιστευτήρια αποτελούνται από το όνομα χρήστη και τον κωδικό πρόσβασης.

Για λόγους ασφαλείας, ο κωδικός πρόσβασης του κάθε χρήστη δεν αποθηκεύεται γυμνός αλλά κωδικοποιημένος. Η διαδικασία της κωδικοποίησης του κώδικου πρόσβασης ονομάζεται hashing.

Ο κατακερματισμός (hashing) λύνει το πρόβλημα της άμεσης πρόσβασης στο σύστημα με εκτεθειμένους κωδικούς πρόσβασης. Ο κατακερματισμός είναι μια μονόδρομη (one-way) συνάρτηση που μετατρέπει την είσοδο (input) σε μια γραμμή συμβόλων. Κανονικά το μήκος αυτής της γραμμής είναι σταθερό. Εάν τα δεδομένα είναι κατακερματισμένα, είναι πολύ δύσκολο να μετατραπεί ο κατακερματισμός στην αρχική είσοδο και είναι επίσης πολύ δύσκολο να βρεθεί η είσοδος για να ληφθεί η επιθυμητή έξοδος.

Πρέπει να κατακερματίσουμε τον κωδικό πρόσβασης σε δύο περιπτώσεις:

- Όταν ο χρήστης εγγραφεί στην εφαρμογή κατακερματίζουμε τον κωδικό πρόσβασης και τον αποθηκεύουμε στη βάση δεδομένων.
- Όταν ο χρήστης θέλει να πραγματοποιήσει έλεγχο ταυτότητας, κατακερματίζουμε τον παρεχόμενο κωδικό πρόσβασης και τον συγκρίνουμε με τον κατακερματισμό κωδικού πρόσβασης από τη βάση δεδομένων.

Τώρα, όταν οι εισβολείς λαμβάνουν τον κατακερματισμό ενός κωδικού πρόσβασης, δεν μπορούν να τον χρησιμοποιήσουν για πρόσβαση στο σύστημα. Οποιαδήποτε προσπάθεια εύρεσης του απλού κειμένου από την τιμή κατακερματισμού απαιτεί τεράστια προσπάθεια από τον εισβολέα. Μια επίθεση brute-force μπορεί να είναι πολύ δαπανηρή εάν ο κατακερματισμός είναι αρκετά μεγάλος. Ωστόσο, χρησιμοποιώντας πίνακες rainbow tables, οι επιτιθέμενοι μπορούν να έχουν επιτυχία. Ένας rainbow table είναι ένας πίνακας με προϋπολογισμένους κατακερματισμούς για πολλούς κωδικούς πρόσβασης. Υπάρχουν πολλοί rainbow tables διαθέσιμοι στο διαδίκτυο και μερικοί από αυτούς περιέχουν εκατομμύρια κωδικούς πρόσβασης.

Για να αποτρέψουμε μια επίθεση βασισμένη σε rainbow tables μπορούμε να χρησιμοποιήσουμε salted κωδικούς πρόσβασης. Το salt είναι μια ακολουθία από byte που δημιουργούνται τυχαία και κατακερματίζονται μαζί με τον κωδικό πρόσβασης. Το λεγόμενο αλάτι (salt) αποθηκεύεται στο χώρο αποθήκευσης και δεν χρειάζεται προστασία. Κάθε φορά που ο χρήστης προσπαθεί να ελέγξει την ταυτότητα, ο κωδικός πρόσβασης του χρήστη κατακερματίζεται με το αποθηκευμένο “αλάτι” και το αποτέλεσμα θα πρέπει να ταιριάζει με τον αποθηκευμένο κωδικό πρόσβασης.

Η πιθανότητα ο συνδυασμός του κωδικού πρόσβασης και του “αλατιού” να έχει υπολογιστεί εκ των προτέρων σε ένα rainbow table είναι πολύ μικρή. Εάν το “αλάτι” είναι αρκετά μακρύ και τυχαίο, είναι αδύνατο να βρεθεί ο κατακερματισμός σε ένα rainbow table. Δεδομένου ότι το “αλάτι” δεν είναι μυστικό, οι επιτιθέμενοι εξακολουθούν να είναι σε θέση να ξεκινήσουν μια επίθεση brute-force. Ένα “αλάτι” μπορεί να κάνει την επίθεση δύσκολη για τον εισβολέα, αλλά το υλικό γίνεται πιο αποτελεσματικό. Πρέπει να υποθέσουμε το ταχέως εξελισσόμενο υλικό με το οποίο ο εισβολέας μπορεί να υπολογίσει δισεκατομμύρια hashes ανά δευτερόλεπτο. Έτσι, το hashing και το salting είναι απαραίτητα - αλλά όχι αρκετά.

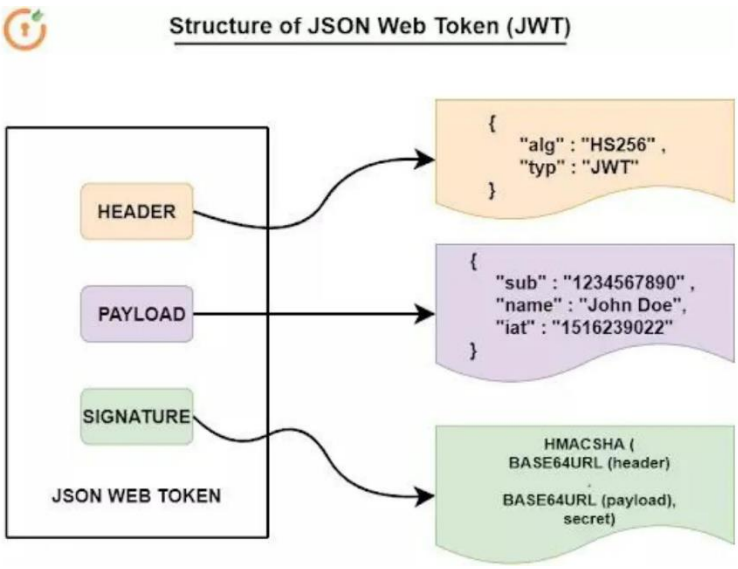
Οι συναρτήσεις κατακερματισμού δεν δημιουργήθηκαν για κατακερματισμό μόνο κωδικών πρόσβασης. Ο εφευρέτης των συναρτήσεων κατακερματισμού έκανε πολύ καλή δουλειά και έκανε τη λειτουργία κατακερματισμού πολύ γρήγορα. Ωστόσο, εάν μπορούμε να κατακερματίσουμε τους κωδικούς πρόσβασης πολύ γρήγορα, τότε ένας εισβολέας μπορεί επίσης να εκτελέσει την επίθεση brute force πολύ γρήγορα.

Η λύση είναι να κάνετε τον κατακερματισμό κωδικού πρόσβασης αργό. Ωστόσο, δεν θα πρέπει να είναι τόσο αργό ώστε να είναι απαράδεκτο για τον χρήστη, αλλά αρκετά αργό ώστε μια επίθεση ωμής βίας να πάρει άπειρο χρόνο. Δεν χρειάζεται να αναπτύξουμε τον αργό κατακερματισμό μόνοι μας. Έχουν αναπτυχθεί αρκετοί αλγόριθμοι ειδικά για κατακερματισμό κωδικού πρόσβασης, όπως ο bcrypt που χρησιμοποιεται στην παρούσα διπλωματική εργασία, που χρησιμοποιούν έναν περίπλοκο κρυπτογραφικό αλγόριθμο και εκχωρούν πόρους όπως η CPU.

Η παρούσα εφαρμογή κάνει χρήση του module spring security, στο οποίο παρέχεται η συνάρτηση του αλγόριθμου κατακερματισμού bcrypt μέσω του bean BcryptPasswordEncoder, ο οποίος χρησιμοποιείται από το 1999. Αυτός ο αλγόριθμος ενημερώνεται συχνά για να ταιριάζει με τις σύγχρονες υπολογιστικές εξελίξεις. Ο κατακερματισμός με το BcryptPasswordEncoder απαιτεί σημαντικούς υπολογισμούς CPU, καθιστώντας τον ανθεκτικό σε γρήγορες επιθέσεις brute-force. Ο χρόνος που απαιτείται για τον κατακερματισμό αυξάνεται με τον αριθμό των διαμορφωμένων γύρων (rounds), ενισχύοντας την ασφάλεια. Σε αντίθεση με απλούς αλγόριθμους κατακερματισμού όπως ο SHA-256 ή ο MD5, η έξοδος του bcrypt περιέχει μετα-πληροφορίες σχετικά με την έκδοση του αλγορίθμου, το work factor και το salt. Δεν χρειάζεται να αποθηκεύσουμε αυτές τις πληροφορίες χωριστά.

Στην συνέχεια, αφού ταυτοποιηθεί ένας χρήστης βάσει των διαπιστευτηρίων που έδωσε, παράγεται από την εφαρμογή ένα μοναδικό αλφαριθμητικό αναγνωριστικό που αντιστοιχίζεται με τον χρήστη και το οποίο ο χρήστης το χρησιμοποιεί για κάθε λειτουργία που κάνει μέσα στην εφαρμογή ώστε να γνωρίζει η εφαρμογή ότι ο χρήστης είναι έγκυρος και έχει ταυτοποιήθει. Αυτό το αναγνωριστικό ονομάζεται JWT Token. Το JWT, ή JSON Web Token, είναι ένα ανοιχτό πρότυπο που χρησιμοποιείται για την ασφαλή ανταλλαγή πληροφοριών μεταξύ δύο μερών — πελάτη και διακομιστή. Στις περισσότερες περιπτώσεις, είναι ένα κωδικοποιημένο JSON που περιέχει ένα σύνολο διακριτικών (claims) και μια υπογραφή. Συνήθως χρησιμοποιείται στο πλαίσιο άλλων μηχανισμών ελέγχου ταυτότητας όπως το OAuth, το OpenID για την κοινή χρήση πληροφοριών που σχετίζονται με τον χρήστη. Είναι επίσης ένας δημοφιλής τρόπος ελέγχου ταυτότητας/εξουσιοδότησης χρηστών σε μια αρχιτεκτονική microservice.

Η δομή JWT χωρίζεται σε τρία μέρη: κεφαλίδα (header), ωφέλιμο φορτίο (payload), υπογραφή (signature) και χωρίζεται το ένα από το άλλο με τελεία (.) και θα ακολουθεί την παρακάτω δομή:



Επικεφαλίδα

Η επικεφαλίδα αποτελείται από δύο μέρη:

- Ο αλγόριθμος υπογραφής που χρησιμοποιείται
- Ο τύπος του διακριτικού, που σε αυτήν την περίπτωση είναι κυρίως "JWT"

Ωφέλιμο φορτίο

Το ωφέλιμο φορτίο περιέχει συνήθως τα διακριτικά (χαρακτηριστικά χρήστη) και πρόσθετα δεδομένα όπως ο εκδότης και ο χρόνος λήξης.

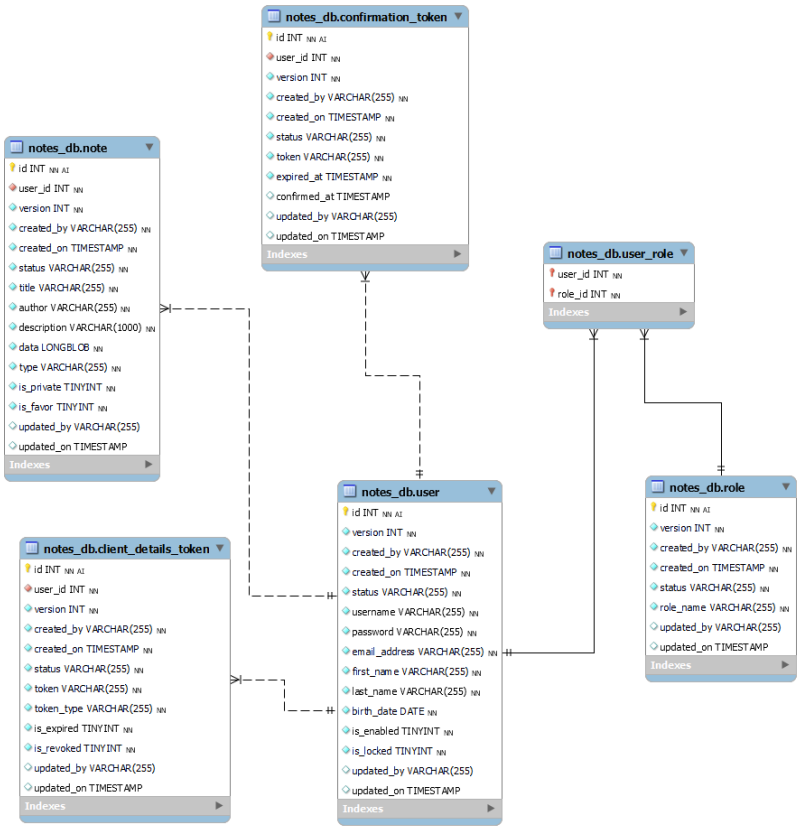
Υπογραφή

Αυτό είναι συνήθως ένας κατακερματισμός της επικεφαλίδας και του ωφέλιμου φορτίου του JWT. Ο αλγόριθμος που χρησιμοποιείται για τη δημιουργία της υπογραφής είναι ο ίδιος αλγόριθμος που αναφέρεται στην επικεφαλίδας του JWT. Η υπογραφή χρησιμοποιείται για να επιβεβαιωθεί ότι το διακριτικό JWT δεν τροποποιήθηκε ή δεν άλλαξε κατά τη μεταφορά. Μπορεί επίσης να χρησιμοποιηθεί για την επικύρωση του αποστολέα.

Η επικεφαλίδα και το ωφέλιμο φορτίο του JWT είναι πάντα κωδικοποιημένη στο Base64. Στην παρούσα διπλωματική εργασία ο αλγόριθμος κατακερματισμού που χρησιμοποιείται για την δημιουργία του JWT είναι ο RSA512. Πρόκειται για έναν ασύμμετρο αλγόριθμο, που σημαίνει ότι υπάρχουν δύο κλειδιά: ένα δημόσιο κλειδί και ένα ιδιωτικό κλειδί που πρέπει να κρατηθεί μυστικό. Η εφαρμογή έχει το ιδιωτικό κλειδί που χρησιμοποιείται για τη δημιουργία της υπογραφής και ο client του JWT ανακτά ένα δημόσιο κλειδί και το χρησιμοποιεί για να επικυρώσει την υπογραφή JWT.

3.3 Σχεδιαστικές Αποφάσεις Λογισμικού

Οι σχεδιαστικές αποφάσεις που πάρθηκαν για την εφαρμογή αντικατοπτρίζουν το πώς είναι οργανωμένη συνολικά η πληροφορία στο σύστημα. Σύμφωνα με την ανάλυση που έγινε για την οργάνωση της βάσης δεδομένων προκύπτει ότι έχουμε δύο βασικούς χρήστες, τον χρήστη της εφαρμογής και τον διαχειριστή της εφαρμογής. Πρόκειται για δύο διαφορετικά εννοιολογικές έννοιες όπου η κάθε μια έχει διαφορετικό σκοπό μέσα στο σύστημα. Επιπλέον, προστέθηκε η έννοια της σημείωσης όπου επιτελεί ένα ξεχωριστό έργο μέσα στην εφαρμογή. Παρακάτω φαίνεται ενδεικτικά ο σχεδιασμός της βάσης δεδομένων καθώς και η αλληλεπίδραση μεταξύ των στοιχείων της.



Εικόνα 3 – Απεικόνιση Βάσης Δεδομένων

Πίνακας user

Ο πίνακας user περιέχει όλη την πληροφορία που σχετίζεται με την οντότητα χρήστης. Σε αυτή την οντότητα περιέχονται και οι πληροφορίες του διαχειριστή. Ο Διαχειριστής της εφαρμογής είναι και αυτός ένας χρήστης της εφαρμογής με διαφοροποιημένα δικαιώματα.

Πεδίο	Τύπος
id	integer
version	integer
created_by	character
created_on	timestamp
status	character
username	character
password	character
email_address	character
first_name	character
last_name	character
birth_date	date
is_enabled	integer
is_locked	integer
updated_by	character
updated_on	timestamp

Πίνακας item

Ο πίνακας item περιέχει όλη την πληροφορία που σχετίζεται με την οντότητα της σημείωσης.

Πεδίο	Τύπος
id	integer
version	integer
created_by	character
created_on	timestamp
status	character
user_id	integer
item_name	character
item_descryption	character
item_type	character
data	blob
is_private	integer
is_favorite	integer
updated_by	character
updated_on	timestamp

Πίνακας bearer_token

Ο πίνακας bearer_token περιέχει όλη την πληροφορία που σχετίζεται με το jwt token.

Πεδίο	Τύπος
id	integer
user_id	integer
version	integer
created_by	character
created_on	timestamp
status	character
token	character
token_type	character
is_expired	integer
is_revoked	integer
updated_by	character
updated_on	timestamp

Πίνακας email_confirmation_token

Ο πίνακας email_confirmation_token περιέχει όλη την πληροφορία που σχετίζεται με το token της εγγραφής ενός χρήστη στην εφαρμογή. Κατά την εγγραφή ενός χρήστη παράγεται ένα μοναδικό αναγνωριστικό (token) βάσει του οποίου γίνεται η ενεργοποίηση του χρήστη στην εφαρμογή.

Πεδίο	Τύπος
id	integer
user_id	integer
version	integer
created_by	character
created_on	timestamp
status	character
token	character
expired_at	timestamp
confirmed_at	timestamp
updated_by	character
updated_on	timestamp

Πίνακας role

Ο πίνακας role περιέχει όλη την πληροφορία που σχετίζεται με τους διαθέσιμους ρόλους που μπορεί να έχει ένας χρήστης μέσα στην εφαρμογή.

Πεδίο	Τύπος
id	integer
version	integer
created_by	character
created_on	timestamp
status	character
role_name	character
updated_by	character
updated_on	timestamp

Πίνακας user_role

Ο πίνακας user_role περιέχει όλη την πληροφορία σχετικά με ποιόν ρόλο ή ποιους ρόλους έχει ένας χρήστης. Πρόκειται για έναν ενδιάμεσο πίνακα ανάμεσα στον πίνακα user και τον πίνακα role.

Πεδίο	Τύπος
user_id	integer
role_id	integer

Στην Βάση Δεδομένων της εφαρμογής υπάρχουν **οι παρακάτω εξαρτήσεις**,

- σχέση 1-1 του πίνακα user με τον πίνακα email_confirmation_token
  - στον πίνακα confirmation_token υπάρχει ένα πεδίο user_id που αποτελεί το foreign key της σχέσης
- σχέση 1-n του πίνακα user με τον πίνακα bearer_token
  - στον πίνακα client_details_token υπάρχει ένα πεδίο user_id που αποτελεί το foreign key της σχέσης
- σχέση 1-n του πίνακα user με τον πίνακα item
  - στον πίνακα note υπάρχει ένα πεδίο user_id που αποτελεί το foreign key της σχέσης
- σχέση n-m του πίνακα user με τον πίνακα role. Για αυτό υπάρχει ο ενδιάμεσος πίνακας user_role που γεφυρώνει ποιος χρήστης έχει ποιους ρόλους.
  - Στον πίνακα user_role υπάρχουν δύο πεδία το user_id και το role_id. Τα δύο αυτά πεδία αποτελούν το primary key του πίνακα, όποτε το primary key είναι composite αλλά ταυτόχρονα είναι και το κάθε πεδίο foreign key προς τους πίνακες user και role αντίστοιχα.

3.4 Αρχιτεκτονική Μικροϋπηρεσιών

Μια αρχιτεκτονική μικροϋπηρεσιών (microservices architecture) είναι ένα αρχιτεκτονικό μοτίβο (pattern) που οργανώνει μια εφαρμογή ως μια συλλογή από χαλαρά συζευγμένες, λεπτομερείς υπηρεσίες, που επικοινωνούν μέσω πρωτοκόλλων. Ένας από τους στόχους του είναι ότι οι ομάδες μπορούν να αναπτύξουν μια μικροϋπηρεσία ανεξάρτητα από κάποια άλλη. Αυτό επιτυγχάνεται με τη μείωση πολλών εξαρτήσεων στη βάση του πηγαίου κώδικα, επιτρέποντας στους μηχανικούς λογισμικού να εξελίσσουν τις μικροϋπηρέσιες με ελάχιστους περιορισμούς και να κρύβεται πρόσθετη πολυπλοκότητα από τους χρήστες. Οι απαιτήσεις επικοινωνίας μειώνονται. Έτσι, η αρχιτεκτονική των μικροϋπηρεσιών μπορεί να είναι μια καλή επιλογή μόνο εάν η εφαρμογή είναι πολύ περίπλοκη για τη διαχείριση της ως “μονόλιθο”.

Βαθμός ανάλυσης των μικροϋπηρεσίων

Ένα βασικό βήμα στον καθορισμό μιας αρχιτεκτονικής μικροϋπηρεσιών είναι να υπολογιστεί πόσο μεγάλη πρέπει να είναι μια μεμονωμένη μικρουπηρεσία. Δεν υπάρχει σωστή ή λάθος προσέγγιση για αυτό, καθώς η σωστή απάντηση εξαρτάται από άλλους παράγοντες, όπως το πλαίσιο που οριοθετείτε από τον οργανισμό/φορέα ή την εταιρεία παραγωγής λογισμικού. Για να βρεθεί το σωστό επίπεδο ανάλυσης των μικρϋπηρεσιών, οι αρχιτέκτονες λογισμικού χρειάζεται να επικοινωνούν συχνά τα σχεδιαστικά πρότυπα με τους μηχανικούς λογισμικού. Οι αρχιτέκτονες λογισμικού χρειάζεται να λαμβάνουν υπόψη τις απαιτήσεις των χρηστών, τις ευθύνες και τα αρχιτεκτονικά χαρακτηριστικά/πρότυπα.

Γενικά, η ορολογία έχει ως εξής: οι μικροϋπηρεσίες που είναι αφιερωμένες σε μια μεμονωμένη εργασία, όπως η κλήση ενός back-end συστήματος ή η πραγματοποίηση ενός συγκεκριμένου υπολογισμού, ονομάζονται ατομικές μικροϋπηρεσίες. Ομοίως, οι μικροϋπηρεσίες που καλούν τέτοιες ατομικές μικροϋπηρεσίες προκειμένου να ενοποιήσουν (orchestration) ένα αποτέλεσμα, ονομάζονται σύνθετες μικροϋπηρεσίες.

Θεωρείται κακή πρακτική να κάνετε την μικροϋπηρεσία πολύ μικρή, καθώς τότε ο γενικός χρόνος εκτέλεσης και η λειτουργική πολυπλοκότητα μπορεί να υπερκαλύψουν τα οφέλη της προσέγγισης. Όταν τα πράγματα γίνονται πολύ μικρά, πρέπει να ληφθούν υπόψη εναλλακτικές προσεγγίσεις - όπως η συσκευασία (packaging) της λειτουργικότητας αυτής σε μία βιβλιοθήκη ή η μεταφορά της λειτουργικότητας αυτής σε άλλες μικροϋπηρεσίες.

Εάν η σχεδίαση είναι καθοδηγούμενη από το domain (πεδίο) για την μοντελοποίηση του για το οποίο κατασκευάζεται μια εφαρμογή ή ένα σύστημα ότε μια μικρουπηρεσία θα μπορούσε να είναι τόσο μικρή όσο ένα σύνολο ή τόσο μεγάλη όσο ένα περιορισμένο πλαίσιο. Στη συζήτηση για τις μικροϋπηρεσίες, υπάρχει ένα φάσμα. Στο ένα άκρο βρίσκονται οι Anemic Services, οι οποίες δεν έχουν μεγάλο αριθμό ευθυνών και στο άλλο άκρο βρίσκονται οι Modular Monolith, που είναι μεγάλες ενότητες ενός συστήματος.

Οφέλη μικροϋπηρεσίων

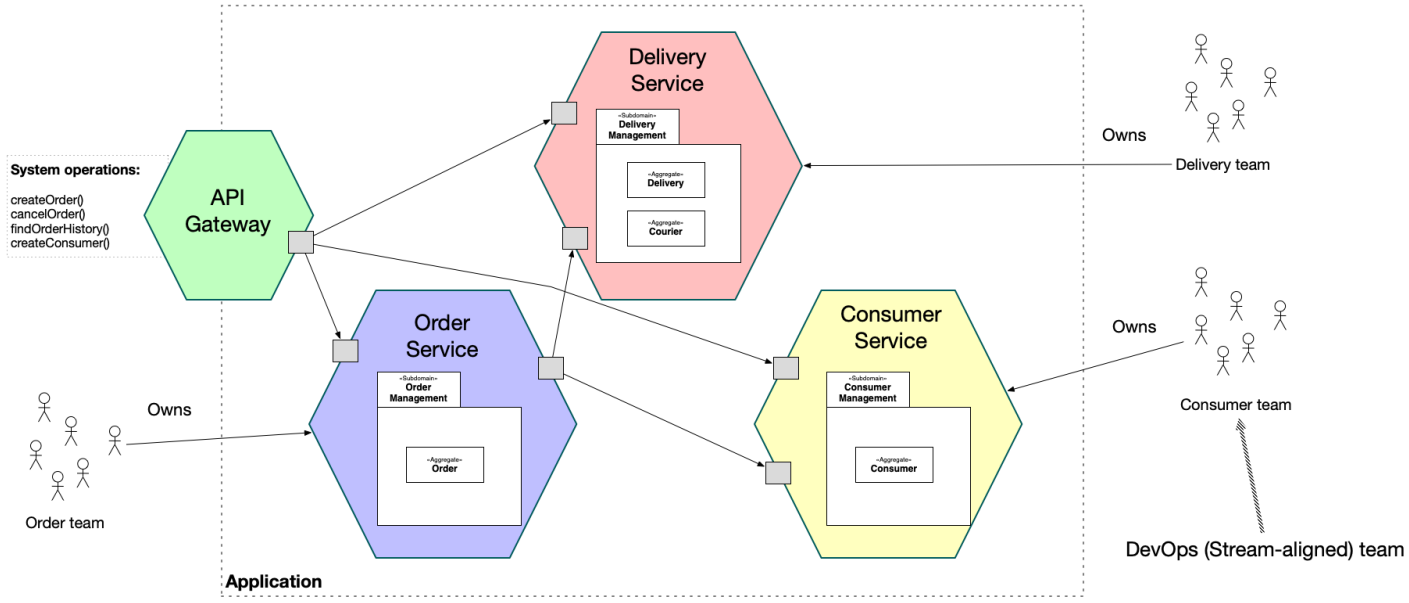
Τα οφέλη από την αποσύνθεση μιας εφαρμογής σε διαφορετικές μικρότερες υπηρεσίες είναι πολλά.

Modularity: Αυτό καθιστά την εφαρμογή ευκολότερη στην κατανόηση, ανάπτυξη, δοκιμή και μεγαλύτερη ανθεκτικότητα στη διάβρωση της αρχιτεκτονικής. Αυτό το όφελος συχνά υποστηρίζεται σε σύγκριση με την πολυπλοκότητα των μονολιθικών αρχιτεκτονικών.

Scalability: Εφόσον οι μικροϋπηρεσίες υλοποιούνται και αναπτύσσονται ανεξάρτητα η μία από την άλλη, δηλαδή εκτελούνται εντός ανεξάρτητων διεργασιών, μπορούν να παρακολουθούνται και να κλιμακώνονται ανεξάρτητα.

Integration of heterogeneous and legacy systems: οι μικροϋπηρεσίες θεωρούνται βιώσιμο μέσο για τον εκσυγχρονισμό της υπάρχουσας μονολιθικής εφαρμογής λογισμικού. Υπάρχουν αναφορές πολλών εταιρειών που έχουν αντικαταστήσει επιτυχώς μέρη του υπάρχοντος λογισμικού τους με μικροϋπηρεσίες ή βρίσκονται στη διαδικασία να το κάνουν. Η διαδικασία για τον εκσυγχρονισμό λογισμικού παλαιού τύπου εφαρμογών γίνεται χρησιμοποιώντας μια σταδιακή και επαυξημένη προσέγγιση.

Distributed development: παραλληλίζει την ανάπτυξη δίνοντας τη δυνατότητα σε μικρές αυτόνομες ομάδες να αναπτύξουν και να κλιμακώσουν τις αντίστοιχες μικροϋπηρεσίες τους ανεξάρτητα. Επιτρέπει επίσης την εμφάνιση της αρχιτεκτονικής μιας μεμονωμένης υπηρεσίας μέσω συνεχούς ανακατασκευής. Οι αρχιτεκτονικές που βασίζονται σε μικροϋπηρεσίες διευκολύνουν τη συνεχή ενοποίηση, τη συνεχή παράδοση και ανάπτυξη.



Σχεδιαστικό πρότυπο “API Gateway”

Το σχεδιαστικό πρότυπο “API Gateway” χρησιμοποιείται σε συστήματα που βασίζονται σε μικροϋπηρεσίες για τη δημιουργία ενός κεντρικού σημείου εισόδου για τις αλληλεπιδράσεις με τους χρήστες της εφαρμογής. Ο Gateway είναι υπεύθυνος για τη δρομολόγηση αιτημάτων στις αντίστοιχες μικροϋπηρεσίες, την διαχείριση της ταυτότητας και της εξουσιοδότησης των χρηστών, τη συγκέντρωση απαντήσεων και τη διαχείριση πολλών προβλημάτων. Μέσω αυτού, παρέχεται στις εφαρμογές δυνατότητες επεκτασιμότητας, ασφάλειας και διαχειρισιμότητας σε περιβάλλον μικροϋπηρεσιων.

Το σχεδιαστικό πρότυπο “API Gateway” διαθέτει διάφορες λειτουργίες για την εφαρμογή όπως,

Centralized Entry Point

Το σχεδιαστικό πρότυπο “API Gateway” χρησιμεύει ως ένα ενιαίο σημείο εισόδου σε ολόκληρη την εφαρμογή (όπως εφαρμογές ιστού ή φορητές συσκευές) για την πρόσβαση του χρήστη στο σύστημα. Μέσω του σημείου εισόδου, ο χρήστης μπορεί να επικοινωνήσει με όλες τις μεμονωμένες μικροϋπηρεσίες, καθώς ο gateway χειρίζεται τις διαδρομές για τις μικροϋπηρεσίες αντί για τον χρήστη. Αυτό καθιστά την επικοινωνία μεταξύ του χρήστη και της εφαρμογής απλή μέσω μιας ενιαίας διαδρομής, ανεξάρτητα από την πολυπλοκότητα της αρχιτεκτονικής των μικροϋπηρεσιων.

Request Routing

Δεδομένου ότι όλα τα αιτήματα έρχονται στο μοναδικό σημείο εισόδου όπου βρίσκεται ο gateway, εξυπηρετεί την δρομολόγηση προς τις μικρούπηρεσίες. Αυτό συμβαίνει, μέσω της αναγνώρισης ενός συγκεκριμένου τερματικού σημείου μιας μικροϋπηρεσίας που ζητείται από τον χρήστη ανάλογα με τις λεπτομέρειες του αιτήματος, όπως URL αιτήματος, μεθόδους HTTP, HTTP κεφαλίδες ή άλλες παραμέτρους, και προωθεί το αίτημα στην αντίστοιχη μικροϋπηρεσία.

Protocol Translation

Ο gateway επιτρέπει την επικοινωνία σε μία μόνο μορφή με τον πελάτη, ανεξάρτητα από την ποικιλία των τεχνολογιών που χρησιμοποιούνται στις διάφορες μικροϋπηρεσίες. Για παράδειγμα, η εφαρμογή μπορεί να περιλαμβάνει διαφορετικές προσεγγίσεις επικοινωνίας όπως gRPC, WebSocket και RESTful, ωστόσο η επικοινωνία μεταξύ του πελάτη και του gateway θα εξακολουθεί να χρησιμοποιεί ένα είδος επικοινωνίας, καθώς ο gateway μπορεί να χειριστεί τη μετάφραση του πρωτοκόλλου σύμφωνα με την μικροϋπηρεσια που καλείται. Αυτό διατηρεί την αρχιτεκτονική ανοιχτή για την εφαρμογή μιας ποικιλίας τεχνολογιών ταυτόχρονα.

Authentication and Authorization

Ο χειρισμός του ελέγχου ταυτότητας και της εξουσιοδότησης σε επίπεδο gateway παρέχει έναν κεντρικό και αποτελεσματικό τρόπο διαχείρισης προβλημάτων ασφάλειας. Ελέγχει τα εισερχόμενα αιτήματα από τον πελάτη χρησιμοποιώντας τεχνικές όπως API Keys ή JWT Tokens κ.λπ. Επιπλέον, μπορεί να διασφαλίσει την υλοποίηση των αρχών που βασίζονται σε ρόλους του πελάτη παρέχοντας πρόσβαση στις μικροϋπηρεσίες μόνο στις οποίες ο πελάτης έχει εξουσιοδοτημένη πρόσβασης.

Ωστόσο, ενδέχεται να μην εξαλείψει την ανάγκη εφαρμογής ελέγχου ταυτότητας και εξουσιοδότησης σε άλλες μικροϋπηρεσίες, ανάλογα με τις απαιτήσεις της εφαρμογής, την πολυπλοκότητα και τις ανάγκες ασφαλείας. Ωστόσο, ο χειρισμός του ελέγχου ταυτότητας και της εξουσιοδότησης κυρίως σε επίπεδο gateway αρκεί για ορισμένες εφαρμογές ή περιπτώσεις χρήσης.

Request Aggregation

Σε ορισμένες περιπτώσεις, οι πελάτες ενδέχεται να χρειάζεται δεδομένα που σχετίζονται με διαφορετικές μικροϋπηρεσίες σε ένα μόνο αίτημα. Σε τέτοια σενάρια, ο gateway μπορεί να συγκεντρώσει τα δεδομένα σε ένα μόνο αίτημα πριν τα επιστρέψει στον πελάτη. Αυτό θα βοηθούσε στη μείωση των αριθμών αιτημάτων για τον πελάτη, βελτιώνοντας την απόδοση και μειώνοντας τον λανθάνοντα χρόνο. Σε αυτήν την περίπτωση, ο gateway θα προσδιορίσει ποιες μικροϋπηρεσίες συμμετέχουν στη συλλογή των απαραίτητων δεδομένων με βάση το αίτημα του πελάτη, θα στείλει αιτήματα σε όλες τις αναγνωρισμένες μικροϋπηρεσίες, θα περίμενε μέχρι να λάβει όλες τις απαντήσεις από όλες τις μικροϋπηρεσίες, στη συνέχεια θα συγκεντρώσει τα δεδομένα και θα στείλει την απάντηση στο πελάτης.

Monitoring and Analytics

Ο gateway μπορεί να συγκεντρώσει αρχεία καταγραφής και μετρήσεις σχετικά με τα εισερχόμενα αιτήματα πελατών και τις απαντήσεις εφαρμογών. Στη συνέχεια παρέχει πολύτιμες πληροφορίες σχετικά με την απόδοση του συστήματος, επιτρέποντας τον εντοπισμό της χρήσης προτύπων και πιθανών ζητημάτων στην εφαρμογή. Μέσω αυτού μπορούν να εντοπιστούν και να διαγνωστούν τα υπάρχοντα προβλήματα σε πραγματικό χρόνο διασφαλίζοντας την αξιοπιστία και τη διαθεσιμότητα του συστήματος. Επιπλέον, βοηθά στην ανεξάρτητη παρακολούθηση κάθε μικροϋπηρεσιας για την απόδοσή της από ένα μόνο σημείο.

Caching

Η προσωρινή αποθήκευση (caching) παίζει πάντα βασικό ρόλο στη βελτίωση της απόδοσης. Ειδικά, στην αρχιτεκτονική μικροϋπηρεσιών όπου η καθυστέρηση τείνει να είναι συγκριτικά υψηλότερη, η εφαρμογή προσωρινής αποθήκευσης είναι πολύ χρήσιμη σε επίπεδο gateway. Ο gateway μπορεί να αποθηκεύσει προσωρινά τις απαντήσεις από τις μικροϋπηρεσίες και να τις εξυπηρετήσει απευθείας στον πελάτη για πανομοιότυπα αιτήματα, μειώνοντας τον λανθάνοντα χρόνο ως αποτέλεσμα και βελτιώνοντας την επεκτασιμότητα.

Ωστόσο, το σχεδιαστικό πρότυπο “API Gateway” έχει επίσης ορισμένα μειονεκτήματα όπως,

Single Point of Failure

Δεδομένου ότι ο gateway χρησιμεύει ως η μόνη είσοδος στην εφαρμογή για τα αιτήματα του πελάτη, ο χρόνος διακοπής λειτουργίας του μπορεί να σημαίνει δυνητικά χρόνο διακοπής λειτουργίας ολόκληρης της εφαρμογής για τον πελάτη. Η αντιμετώπιση αυτού του πιθανού ζητήματος είναι ζωτικής σημασίας για τον μετριασμό του κινδύνου.

Increased Complexity

Το σχεδιαστικό πρότυπο “API Gateway” εισάγει ένα πρόσθετο επίπεδο στην εφαρμογή καθιστώντας την απλούστερη την επικοινωνία για τον πελάτη, καθιστώντας την ωστόσο πιο περίπλοκη για την αρχιτεκτονική. Θα πρέπει πάντα να διαχειρίζεται σωστά, να συντηρείται και να διαμορφώνεται με αντίστοιχες διαδρομές και κανόνες, να χειρίζεται οριακές περιπτώσεις που μπορεί να οδηγήσουν την εφαρμογή σε μη διαθεσιμότητα και να διασφαλίζει τη συμβατότητα με διάφορα πρωτόκολλα και υπηρεσίες.

Performance Bottlenecks

Όντας ένα ενιαίο σημείο εισόδου για ολόκληρη την εφαρμογή, σημαίνει ότι όλα τα αιτήματα πελάτη έρχονται στο μεμονωμένο σημείο. Αυτό μπορεί επίσης να προκαλέσει συμφόρηση απόδοσης όταν η λογική επεξεργασίας αιτημάτων είναι πολύπλοκη και ο όγκος επισκεψιμότητας είναι πολύ υψηλός. Επομένως, ο σχεδιασμός θα πρέπει να βελτιστοποιηθεί προσεκτικά για να διασφαλιστεί η βέλτιστη απόδοση και επεκτασιμότητα.

Service Discovery and Configuration Management

Ένα από τα βασικά χαρακτηριστικά που έχει χτιστεί ολόκληρη η αρχιτεκτονική των μικροϋπηρεσίων είναι ότι είναι επιρρεπής σε συχνές αλλαγές. Ωστόσο, όταν συμβαίνουν αλλαγές στη δρομολόγηση ή στην προσθήκη νέων μικροϋπηρεσιών, ο gateway απαιτεί την ενημέρωση της διαχείρισης του discovery service, δυναμικής δρομολόγησης και άλλες ενημερώσεις διαμόρφωσης. Μπορεί να δημιουργήσει μερικές πρόσθετες προκλήσεις κατά τη διάρκεια αυτής της περιόδου για τους μηχανικούς λογισμικού.

Increased Latency for Some Requests

Φυσικά, η καθυστέρηση μπορεί να αντιμετωπιστεί με τη σωστή διαμόρφωση προσωρινής αποθήκευσης. Ωστόσο, αυτή η διαδικασία μπορεί μερικές φορές να είναι λεπτή, καθώς αν δεν χρησιμοποιείται σωστά, μπορεί να βελτιώσει την απόδοση για ορισμένους τύπους αιτημάτων, αλλά να εισάγει λανθάνουσα κατάσταση για άλλους. Επομένως, οι κατάλληλες στρατηγικές προσωρινής αποθήκευσης και οι μηχανισμοί ακύρωσης της προσωρινής μνήμης για την ελαχιστοποίηση του λανθάνοντος χρόνου για τη διασφάλιση της συνέπειας των δεδομένων θα πρέπει να αναπτυχθούν προσεκτικά.

Vendor Lock-In

Αυτό σχετίζεται με ολόκληρο το αρχιτεκτονικό περιβάλλον που βασίζεται στις μικροϋπηρεσιες. Το vendor lock-in αναφέρεται στην κατάσταση όπου ο χρήστης κάποιας τεχνολογίας ή μοτίβου καταλήγει να εξαρτάται από μια ενιαία προσέγγιση καθώς η αλλαγή της τεχνολογίας/αρχιτεκτονικής/μοτίβου γίνεται πολύ δαπανηρή. Το σχεδιαστικό πρότυπο “API Gateway” μπορεί επίσης να οδηγήσει σε vendor lock-in, καθιστώντας δύσκολη τη μετάβαση σε εναλλακτικές λύσεις στο μέλλον.

Συνολικά, ανεξάρτητα από τα μειονεκτήματα που έχει το σχεδιαστικό πρότυπο “API Gateway”, παραμένει μια δημοφιλής επιλογή για το σχεδιασμό αρχιτεκτονικών μικροϋπηρεσιών με την παροχή μιας ευέλικτης και επεκτάσιμης προσέγγισης για τη διαχείριση της επικοινωνίας πελάτη-διακομιστή.

3.5 Κωδικοποίηση

Στην συγκεκριμένη ενότητα γίνεται μια παρουσίαση της δομής πάνω στην οποία αναπτύχθηκε η εφαρμογή της διπλωματικής εργασίας καθώς και σε κάποια βασικά κομμάτια κώδικα της εφαρμογής. Αρχικά η εφαρμογή υλοποιήθηκε βάσει της αρχιτεκτονικής των μικροϋπγρεσίων και επιλέχθηκε το σχεδιαστικό πρότυπο “API Gateway”. Η εφαρμογή αποτελείται από τα παρακάτω modules,

API Gateway

Ο API Gateway είναι ένα κεντρικό σημείο εισόδου για τους χρήστες της εφαρμογής που αλληλεπιδρούν με στις μικροϋπηρεσίες. Διαχειρίζεται αιτήματα, χειρίζεται τον έλεγχο ταυτότητας και δρομολογεί αιτήματα στις κατάλληλες μικροϋπηρεσίες.

Η παρακάτω εικόνα παρουσιάζει κομμάτι της αυθεντικοποιήσης και της ταυτοποιήσης του χρήστη που λαμβάνει χώρα στον gateway. Το συγκεκριμένο κομμάτι αποτελεί μέρος της κλάσης AuthenticationFilter. Αρχικά, εξάγεται το JWT Token από την κεφαλίδα του αιτήματος. Αν απουσιάζει το αίτημα δεν προχωράει προς κάποια μικροϋπηρεσια και επιστέφεται κωδικός σφάλματος 401 – Unauthorized. Αν από το JWT Token απουσιάζει το λεκτικό “Bearer ” το αίτημα δεν προχωράει προς κάποια μικροϋπηρεσια και επιστέφεται κωδικός σφάλματος 401 – Unauthorized. Στην συνέχεια, γίνεται επαλήθευση του token και αν δεν προκύψει κάποιο σφάλμα το αίτημα δρομολογείται προς την αιτούμενη μικροϋπηρεσία.

```
1 usage  Stavros Laios
private Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
    String authorization = exchange.getRequest().getHeaders().getFirst(HttpHeaders.AUTHORIZATION);
    String gatewayRequestUrl = exchange.getRequest().getURI().toString();
    if (Objects.isNull(authorization)) {
        log.error("received request to URI {} but not found authorization key-value pair in Headers!", gatewayRequestUrl);
        return this.onError(exchange, HttpStatus.UNAUTHORIZED);
    } else if (!authorization.substring(0, 7).contains(BEARER)) {
        log.error("received request to URI {} but found wrong token type found in Headers!", gatewayRequestUrl);
        return this.onError(exchange, HttpStatus.UNAUTHORIZED);
    } else {
        try {
            String jwtToken = authorization.substring( beginIndex: 7);
            helper.isJwtExpired(jwtToken);
        } catch (ExpiredJwtException | MalformedJwtException | SignatureException |
                 NoSuchAlgorithmException | InvalidKeySpecException | IOException ex) {
            log.error("received request to URI {} but an error occurred while decoding token ==> {}", gatewayRequestUrl, ex.getMessage());
            return this.onError(exchange, HttpStatus.UNAUTHORIZED);
        }
    }
    return chain.filter(exchange);
}
```

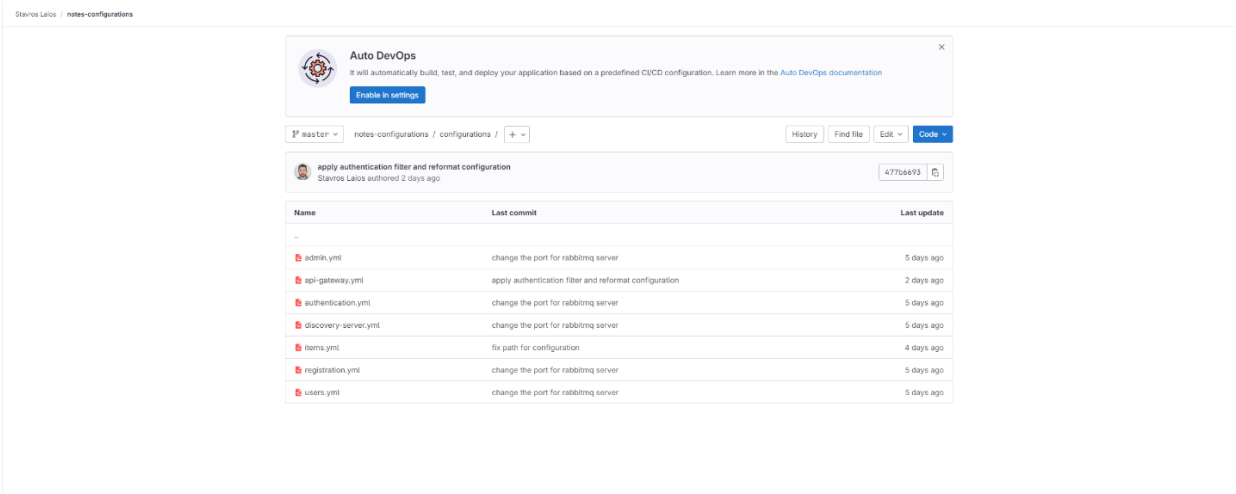
Στην παρακάτω εικόνα παρουσιάζεται κομμάτι του logging του API Gateway για την καταγραφή των αιτήσεων που γίνεται στην εφαρμογή. Το συγκεκριμένο κομμάτι αποτελεί μέρος της κλάσης LoggingFilter. Όπως προκύπτει από την εικόνα γίνεται αποκωδικοποίηση του JWT Token για την εξαγωγή χρήσιμων πληροφοριών, όπως το username του χρήστη που κάνει το αίτημα προς την εφαρμογή.

```
Stavros Laios
@Override
public GatewayFilter apply(Config config) {
    return (exchange, chain) -> {
        StringBuilder logMessage = new StringBuilder();
        long startTime = System.currentTimeMillis();
        return chain.filter(exchange).then(fromRunnable(() -> {
            logMessage.append("received request to URI: ")
                    .append(exchange.getRequest().getURI());
            String authorization = exchange.getRequest().getHeaders().getFirst(HttpHeaders.AUTHORIZATION);
            if(Objects.nonNull(authorization)) {
                try {
                    String jwtToken = authorization.substring( beginIndex: 7);
                    String username = JWT.decode(jwtToken).getSubject();
                    logMessage.append(" with username: ").append(username);
                } catch (JWTDecodeException ex) {
                    log.error("error during decoding jwt token with message: {}", ex.getMessage());
                    throw new ResponseStatusException(HttpStatus.NOT_ACCEPTABLE, ex.getMessage());
                }
            }
            HttpStatus status = HttpStatus.valueOf(Objects.requireNonNull(exchange.getResponse().getStatusCode()).value());
            String gatewayRequestUrl = exchange.getAttribute(ServerWebExchangeUtils.GATEWAY_REQUEST_URL_ATTR) != null
                    ? Objects.requireNonNull(exchange.getAttribute(ServerWebExchangeUtils.GATEWAY_REQUEST_URL_ATTR)).toString()
                    : "N/A";
            long endTime = System.currentTimeMillis();
            logMessage.append(" -- received response from URI: ").append(gatewayRequestUrl)
                    .append(" with status ").append(status)
                    .append(" (duration call time: ")
                    .append(endTime-startTime).append("ms)");
            log.info(logMessage.toString());
        }));
    };
}
```

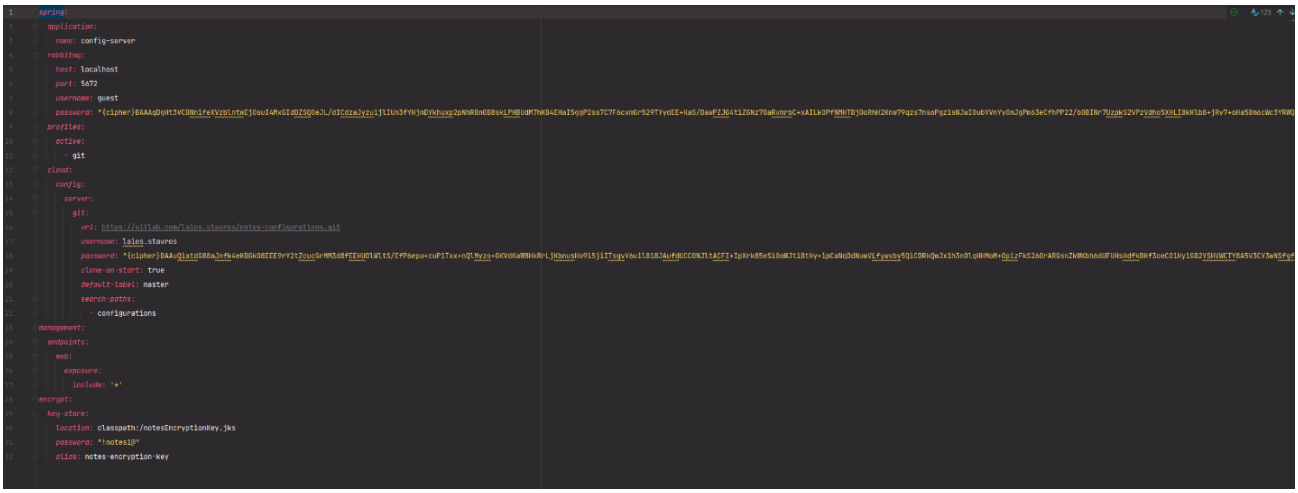
ConfigServer

Ο Spring Boot Config Server είναι ένα ισχυρό εργαλείο για τη διαχείριση της παραμετροποιήσης των μικροϋπηρεσιών. Επιτρέπει την οργάνωση και την διαχείριση των configurations των μικροϋπηρεσιών σε μια κεντρική τοποθεσία και την διαθέσιμότητα τους προς όλες τις μικροϋπηρεσίες.

Στην παρακάτω εικόνα παρουσιάζεται κεντρικοποιημένα αρχεία που έχουν να κάνουν με την παραμετροποίηση των modules της εφαρμογής. Ο ConfigServer θα συνδεθεί στο GitLab, στο συγκεκριμένο αποθετήριο για να τραβήξει αυτά τα αρχεία και να τα κάνει διαθέσιμα στα modules της εφαρμογής.



Στην παρακάτω εικόνα παρουσιάζεται η παραμετροποιήση του ConfigServer, ώστε να μπορεί να τραβήξει τις παραμετροποιήσεις των modules και να τα κάνει διαθέσιμα σε αυτά.



Service Registry and Discovery

Αυτό το τμήμα της εφαρμογής επιτρέπει την παρακολούθηση και τον εντοπισμό των τοποθεσιών και των διευθύνσεων δικτύου όλων των μικροϋπηρεσιών στο σύστημα. Το service discovery διασφαλίζει ότι οι μικροϋπηρεσίες μπορούν να εντοπίζουν και να επικοινωνούν μεταξύ τους δυναμικά.

Στην παρακάτω εικόνες παρουσιάζεται η παραμετροποιήση του module που σχετίζεται με το service registry and discovery. Το annotation @EnableEurekaServer υποδηλώνει ότι το συγκεκριμένο module ενεργεί ως service registry and discovery.

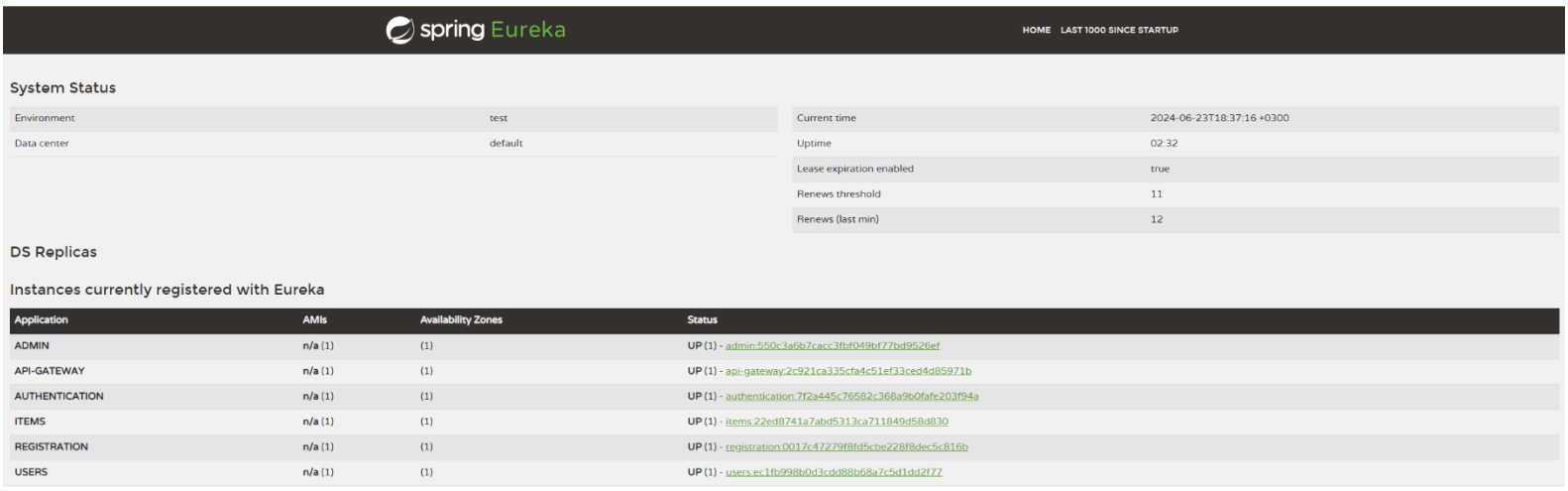
```java
package gr.notes.discovery.server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

1 usage  Stavros Laios
@EnableEurekaServer
@SpringBootApplication
public class DiscoveryServerApplication {

    no usages  Stavros Laios
    public static void main(String[] args) { SpringApplication.run(DiscoveryServerApplication.class, args); }

}
```



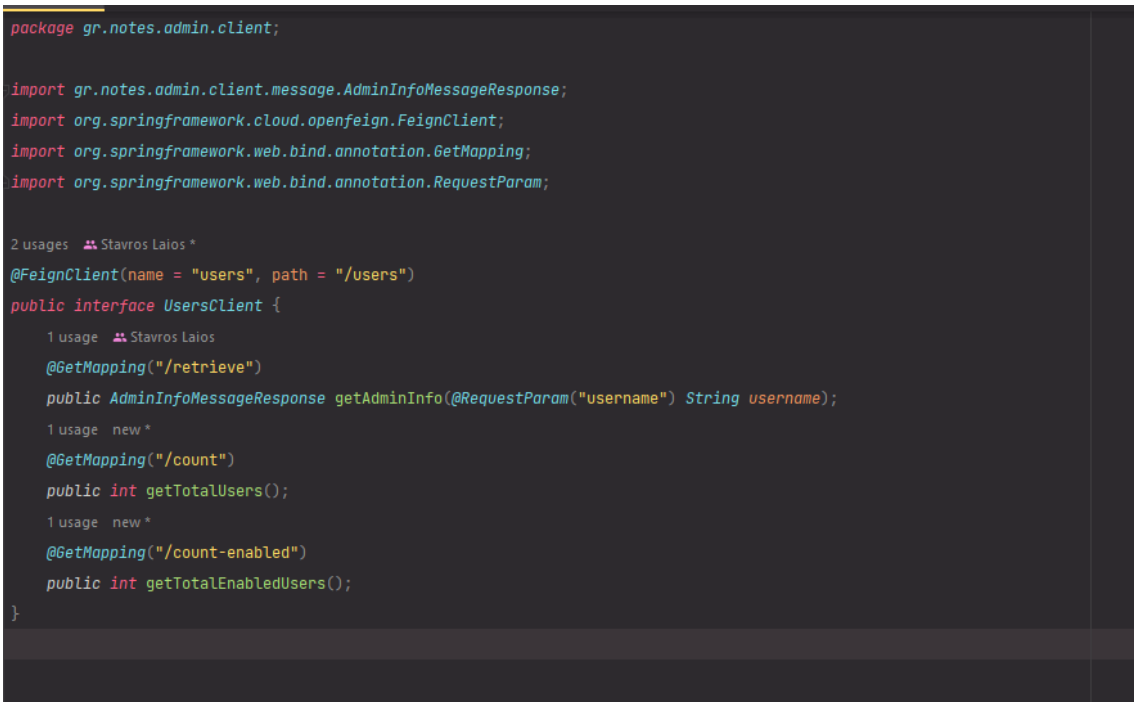
Users Microservice

Αποτελεί την μικροϋπηρεσία η οποία αναλαμβάνει όλες τις λειτουργίες που σχετίζονται με τον χρήστη της εφαρμογής. Στην παρακάτω εικόνα παρουσιάζεται τμήμα της μικροϋπηρεασιας που σχετίζεται με την ανάκτηση των πληροφοριών ενός χρήστη. Το παρακάτω αποτελεί κομμάτι της κλάσης AppUserServiceImpl.

```
1 usage  Stavros Laios
@Override
@Transactional(rollbackOn = Exception.class)
public AppUserResponse retrieveAppUser(String username) {
    log.info("going to retrieve user information...");
    long startTime = System.currentTimeMillis();
    AppUser appUser = appUserRepository.findAppUserByUsername(username).orElseThrow(() -> new AppUserNotFoundException(ERROR_MESSAGE_APP_USER, username));
    SimpleDateFormat formatDate = new SimpleDateFormat( pattern: "dd/MM/yyyy");
    long endTime = System.currentTimeMillis();
    long totalTime = endTime - startTime;
    log.info("user information retrieved in {} ms", totalTime);
    return AppUserResponse.builder()
            .username(appUser.getUsername())
            .emailAddress(appUser.getEmailAddress())
            .firstName(appUser.getFirstName())
            .lastName(appUser.getLastName())
            .roles(appUser.getRoles().stream().map(Role::getRoleName).collect(Collectors.toList()))
            .birthDate(formatDate.format(appUser.getBirthDate()))
            .build();
}
```

Admin Microservice

Αποτελεί την μικροϋπηρεσία η οποία αναλαμβάνει όλες τις λειτουργίες που σχετίζονται με τον διαχειριστή της εφαρμογής. Στην παρούσα εικόνα παρουσιάζεται τμήμα της μικροϋπηρεσιας που κάνει χρήση του OpenFeign. Το OpenFeign είναι ένα έργο ανοιχτού κώδικα που αναπτύχθηκε αρχικά από το Netflix και στη συνέχεια μεταφέρθηκε στην κοινότητα ανοιχτού κώδικα. Το Feign είναι ένας declarative rest πελάτης που δημιουργεί μια δυναμική υλοποίηση της διεπαφής που έχει δηλωθεί ως FeignClient. Το FeignClient χρησιμοποιείται κυρίως για την κατανάλωση REST API που εκτίθενται από τρίτες εφαρμογές ή μικροϋπηρεσίες.



```
package gr.notes.admin.client;

import gr.notes.admin.client.message.AdminInfoMessageResponse;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

2 usages  Stavros Laios *
@FeignClient(name = "users", path = "/users")
public interface UsersClient {
    1 usage  Stavros Laios
    @GetMapping("/retrieve")
    public AdminInfoMessageResponse getAdminInfo(@RequestParam("username") String username);
    1 usage  new *
    @GetMapping("/count")
    public int getTotalUsers();
    1 usage  new *
    @GetMapping("/count-enabled")
    public int getTotalEnabledUsers();
}
```

Item Microservice

Αποτελεί την μικροϋπηρεσία η οποία αναλαμβάνει όλες τις λειτουργίες που σχετίζονται με την σημείωση ως βασική οντότητα της εφαρμογής. Στην παρακάτω εικόνα παρουσιάζεται τμήμα της μικροϋπρεσίας που σχετίζεται με την καταχώρηση μιας σημείωσης στην εφαρμογή. Το παρακάτω αποτέλει τμήμα της κλάσης ItemController και είναι το entry point σχετικά με την λειτουργία της καταχώρησης μιας σημείωσης στην εφαμρογή.

```
no usages  new *
@PostMapping(value = "/create", consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.MULTIPART_FORM_DATA_VALUE})
@Operation(summary = "create new item", description = "It will create a new item of an existing user",
        requestBody = @io.swagger.v3.oas.annotations.parameters.RequestBody(description = "payload which contains all the necessary information to create a new item of an existing user",
                content = {
                @Content(mediaType = MediaType.APPLICATION_JSON_VALUE, schema = @Schema(implementation = ItemRequest.class)),
                @Content(mediaType = MediaType.MULTIPART_FORM_DATA_VALUE, schema = @Schema(implementation = MultipartFile.class))
                }, required = true))
@ApiResponses(value = @ApiResponse(description = "NOT FOUND", responseCode = "404", content = @Content(mediaType = MediaType.APPLICATION_JSON_VALUE, schema = @Schema(implementation = ApiError.class))))
public void createItem(@Valid @RequestPart(name = "request") ItemRequest request, @NotNull @RequestPart(name = "data") MultipartFile data) throws IOException {
    itemService.insertItem(request, data);
}
```

Registration Microservice

Αποτελεί την μικροϋπηρεσία η οποία αναλαμβάνει όλες τις λειτουργίες που σχετίζονται με την εγγραφή ενός νέου χρήστη στην εφαρμογή. Στην παρακάτω εικόνα παρουσιάζεται τμήμα της μικροϋπρεσίας που σχετίζεται με την εγγραφή ενός νέου χρήστη στην εφαρμογή. Συνοδεύεται με την παραγωγή του confirmation token που χρησιμοποιείται για την ενεργοποίηση του λογαριασμού του καθώς και την αποστολή ενημερωτικού e-mail προς τον χρήστη. Το παρακάτω αποτελεί τμήμα της κλάσης RegistrationServiceImpl.

```
1 usage  Stavros Laios
@Override
@Transactional(rollbackOn = Exception.class)
public RegisterAppUserResponse registerAppUser(RegisterAppUserRequest request) {
    log.info("going to create new user information...");
    long startTime = System.currentTimeMillis();
    String emailToken = UUID.randomUUID().toString();
    AppUser appUser = buildAppUser(request);
    EmailConfirmationToken emailConfirmationToken = buildConfirmationToken(emailToken);
    appUser.setConfirmationToken(emailConfirmationToken);
    emailConfirmationToken.setAppUser(appUser);
    appUserRepository.save(appUser);
    emailService.send(request.getEmailAddress(), emailTemplateHelper.verificationTemplate(request.getFirstName(), properties.getEmailProperties().getConfirmTokenUri(), emailToken));
    long endTime = System.currentTimeMillis();
    long totalTime = endTime - startTime;
    log.info("user information created in {} ms", totalTime);
    return RegisterAppUserResponse.builder()
            .username(request.getUsername())
            .emailAddress(request.getEmailAddress())
            .emailToken(emailToken)
            .build();
}
```

Authentication Microservice

Αποτελεί την μικροϋπηρεσία η οποία αναλαμβάνει όλες τις λειτουργίες που σχετίζονται με την αυθεντικοποιήση και την ταυτοποίηση του χρήστη της εφαρμογής. Στην παρακάτω εικόνα παρουσιάζεται η λειτουργικότητα της παραγωγής JWT Token ενώ στην επόμενη παρουσιάζεται ο τρόπος παραγωγής του public και private key που χρησιμοποιούνται από τον αλγόριθμο RSA 512. Το παρακάτω αποτελεί τμήμα της κλάσης JwtHelperImpl.

```
2 usages  Stavros Laios
@Override
public String generateJwtToken(UserDetails userDetails) throws NoSuchAlgorithmException, InvalidKeySpecException, IOException {
    boolean appUserIsEnabled = userDetails.isEnabled();
    String jwtId = UUID.randomUUID().toString();
    List<String> roles = userDetails.getAuthorities() Collection<capture of ? extends GrantedAuthority>
            .stream() Stream<capture of ? extends GrantedAuthority>
            .map(GrantedAuthority::getAuthority) Stream<String>
            .collect(Collectors.toList());
    return Jwts.builder()
            .claim( name: "userIsEnabled", appUserIsEnabled)
            .claim( name: "roles", roles)
            .setId(jwtId)
            .setSubject(userDetails.getUsername())
            .setIssuer(jwtProperties.getJwtIssuer())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 1000*60*10))
            .signWith(generateJwtKeyEncryption(), SignatureAlgorithm.RS512)
            .compact();
}
```

```
1 usage  Stavros Laios
private PublicKey generateJwtKeyDecryption() throws NoSuchAlgorithmException, InvalidKeySpecException, IOException {
    final File file = resourceLoader.getResource( location: "classpath:keys/rsa-public-key.pem").getFile();
    String rsaPublicKeyStr = new String(Files.readAllBytes(file.toPath()));
    String decoratedRsaPublicKey = rsaPublicKeyStr
            .replace( target: "-----BEGIN PUBLIC KEY-----", replacement: "")
            .replaceAll(System.lineSeparator(), replacement: "")
            .replace( target: "-----END PUBLIC KEY-----", replacement: "");
    KeyFactory keyFactory = KeyFactory.getInstance( algorithm: "RSA");
    byte[] keyBytes = Base64.decodeBase64(decoratedRsaPublicKey);
    X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(keyBytes);
    return keyFactory.generatePublic(x509EncodedKeySpec);
}

1 usage  Stavros Laios
private PrivateKey generateJwtKeyEncryption() throws NoSuchAlgorithmException, InvalidKeySpecException, IOException {
    final File file = resourceLoader.getResource( location: "classpath:keys/rsa-private-key.pem").getFile();
    String rsaPrivateKeyStr = new String(Files.readAllBytes(file.toPath()));
    String decoratedRsaPrivateKey = rsaPrivateKeyStr
            .replace( target: "-----BEGIN PRIVATE KEY-----", replacement: "")
            .replaceAll(System.lineSeparator(), replacement: "")
            .replace( target: "-----END PRIVATE KEY-----", replacement: "");
    KeyFactory keyFactory = KeyFactory.getInstance( algorithm: "RSA");
    byte[] keyBytes = Base64.decodeBase64(decoratedRsaPrivateKey);
    PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(keyBytes);
    return keyFactory.generatePrivate(pkcs8EncodedKeySpec);
}
```

5. Επίλογος - Συμπεράσματα

Βασική επιδίωξη της παρούσας διπλωματικής εργασίας ήταν η δημιουργία ενός φιλικού και λειτουργικού συστήματος χωρίς να δημιουργούνται πολυπλοκότητες και δυσκολίες στους άμεσα ενδιαφερόμενους. Ωστόσο, δεν παραγκωνίζεται ο κυριότερος σκοπός του συστήματος δηλαδή αυτός της δημιουργίας ενός συστήματος διαχείρισης σημειώσεων και ταυτόχρονα δεν παραποιείται η ταυτότητα και η φύση του προγράμματος. Αρχικό βήμα αποτέλεσε τόσο η ανίχνευση των χρηστών που θα χρησιμοποιήσουν την ηλεκτρονική πλατφόρμα όσο και η εύρεση των βασικών υπηρεσιών που μπορεί το σύστημα να παρέχει. Συγχρόνως, μέσω πολλαπλών δοκιμών, αποφασίστηκε η τελική δομή της εφαρμογής. Έπειτα, λαμβάνοντας υπόψη βασικούς αποκλεισμούς, δημιουργήθηκαν σενάρια για κάθε πιθανή περίπτωση που μπορεί να συμβεί στο πλαίσιο του συστήματος.

Βασικός πυλώνας διαφοροποίησης της υπάρχουσας εργασίας είναι η προσπάθεια καταχώρησης, οργάνωσης και αποτελεσματικής διαχείρισης ενός πλήθους διαφορετικών σημειώσεων μεταξύ τους. Επισημαίνεται ότι η συσχέτιση πληροφοριών συνέβαλλε σημαντικά στην καλύτερη προσέγγιση των ενδεχομένων προβλημάτων των χρηστών και στην ευρύτερη χρήση του συστήματος από τους χρήστες της εφαρμογής. Καθώς και ότι επιλέχθηκε μια σειρά τεχνολογιών για spring boot – based application.

Το παρόν σύστημα παρέχει μια σειρά υπηρεσιών που διευκολύνει σημαντικά τη διαδικασία της διαχείρισης σημειώσεων όπου ένας χρήστης έχει στην κατοχή του καθώς και απλοποιεί και λιγοστεύει τις τυπικές ενέργειες που απαιτούνταν να κάνει δίχως την ύπαρξη μιας τέτοιας εφαρμογής. Η προφανής χρήση του παρόντος συστήματος είναι η υιοθέτηση του από τις σχολές των πανεπιστημιακών και τεχνολογικών ιδρυμάτων οι οποίες περιέχουν στην καθημερινότητα τους μια πληθώρα από διαφορετικές σημειώσεις. Με κατάλληλες τροποποιήσεις, μπορεί να εμπλουτιστεί περισσότερο και να χρησιμοποιηθεί και ευρύτερα. Με αυτό τον τρόπο, η χρήση του συστήματος μπορεί να γίνει πιο καθολική και μαζική, ανάγοντας το σε σημαντικό εκπαιδευτικό εργαλείο.

Αρχικά η εφαρμογή ελέγχθηκε σε τοπικό επίπεδο αλλά με τις κατάλληλες τροποποιήσεις μπορεί να συνδεθεί με την βάση δεδομένων κάθε πανεπιστημίου με σκοπό να λαμβάνει και να διαχειρίζεται τις πληροφορίες (σημειώσεις/έγγραφα) .

Κάποιες από τις μελλοντικές επεκτάσεις που μπορεί να λάβει η εφαρμογή, παρουσιάζονται παρακάτω,

Δημιουργία Group: Θα μπορεί ένας χρήστης του συστήματος να δημιουργήσει ένα κλειστό group (ομάδα) όπου μεταξύ αυτού και των μελών της ομάδας να διαμοιράζονται σημειώσεις. Οι σημειώσεις αυτές θα κατηγοριοποιούνται ξεχωριστά και θα είναι ορατές μόνο από τους χρήστες της συγκεκριμένης ομάδας. Ο διαχειριστής της ομάδας θα γίνεται αυτόματα αυτός που δημιούργησε το group και θα έχει μόνο αυτός επιπλέον δικαιώματα από τους υπόλοιπους χρήστες της ομάδας, όπως προσθήκη νέων χρηστών στην ομάδα. Τόσο κατά την προσθήκη όσο και κατά την αφαίρεση χρηστών, οι χρήστες θα ενημερώνονται με e-mail.

Αύξηση των τύπων των σημειώσεων: Θα μπορούσαν να προστεθούν και επιπλέον τύποι σημειώσεων που θα διαχειρίζεται η εφαρμογή. Τέτοιοι τύποι είναι, tasks list και to do list. Οι συγκεκριμένες σημειώσεις θα εμφανίζονται σε ένα visual board όπου με χρήση κάποιου UI θα απαρτίζεται από 5 στήλες: Waiting, To Do, Doing, Hold, Done.

Ημερολόγιο: Δημιουργία Ημερολογίου όπου θα μπορούν να προστίθενται συναντήσεις ή σημαντικά γεγονότα που ενδιαφέρουν τον χρήστη. Για συναντήσεις που πλησιάζουν θα αποστέλλεται στον χρήστη ειδοποίηση. Η ειδοποίηση θα είναι παραμετροποιήσιμη μέσα από την εφαρμογή μέσω κατάλληλων ρυθμίσεων.

Analytics: Εξαγωγή reports που θα περιέχουν πληροφορίες σχετικά με τις μετρικές της εφαρμογής από τον Διαχειριστή της εφαρμογής.

Γνωρίζοντας ότι αυτό το σύστημα δημιουργείται στο πλαίσιο μιας διπλωματικής εργασίας μεταπτυχιακού επιπέδου, υπήρξε αδυναμία στην συλλογή δεδομένων όσο αφορά για την προσομοίωση σε πραγματικές συνθήκες live production. Για αυτό το λόγο, χρησιμοποιήθηκαν ενδεικτικά δεδομένα που αντλήθηκαν από το διαδίκτυο. Επιπλέον η απόφαση να γίνεται η αυθεντικοποιήση του χρήστη σε επίπεδο gateway και να μην γίνεται επιπλέον ταυτοποίηση χρήστη στις επιμέρους μικροϋπηρεσιες πάρθηκε για την διευκόλυνση, για την υλοποίηση του κώδικα καθώς και ότι το σύστημα μας λειτουργεί τοπικά. Επιπρόσθετα, κατά την εγγραφή ενός χρήστη στην εφαρμογή παράγεται έναν μοναδικό token βάσει του όποιου χρησιμοποιείται για την ενεργοποίηση του λογαριασμού του χρήστη ενώ θα μπορούσε παράγονται επιπλέον του ενός token αν σε ένα εύλογο διάστημα ο χρήστης δεν έχει προβεί σε ενεργοποίηση του λογαριασμού.

6. Βιβλιογραφία

1. https://el.wikipedia.org/wiki/Java
2. https://en.wikipedia.org/wiki/Spring_Boot
3. https://koskarajohn.github.io/SpringTutorial/html/spring.html
4. https://en.wikipedia.org/wiki/MySQL_Workbench
5. https://medium.com/@hiteshtiwari21990/what-is-rabbit-mq-3ec095a3c84b
6. https://en.wikipedia.org/wiki/RabbitMQ
7. https://www.javatpoint.com/postman
8. https://bigblue.academy/gr/ti-einai-to-api
9. https://github.com/maildev/maildev/releases/tag/v2.1.0
10. https://el.wikipedia.org/wiki/Git_(%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CE%BC%CE%B9%CE%BA%CF%8C)
11. https://vontikakis.com/el/blog/version-control-systems
12. https://greyamp.medium.com/what-is-zipkin-and-how-does-it-work-86a628e56a2f
13. https://zipkin.io/
14. https://www.geeksforgeeks.org/spring-boot-actuator/
15. https://reflectoring.io/spring-security-password-handling/
16. https://saurabhchaudhary01.medium.com/comparing-password-encoders-in-spring-security-42f88c83cb26
17. https://www.miniorange.com/blog/what-is-jwt-json-web-token-how-does-jwt-authentication-work/
18. https://auth0.com/docs/get-started/applications/signing-algorithms
19. https://microservices.io/patterns/microservices
20. https://en.wikipedia.org/wiki/Microservices
21. https://www.geeksforgeeks.org/api-gateway-patterns-in-microservices/
22. https://medium.com/@apulatjonov/microservices-design-patterns-api-gateway-design-pattern-ddba36700d84
23. https://www.tutorialspoint.com/spring_boot/spring_boot_eureka_server.htm#:~:text=The%20%40EnableEurekaServer%20annotation%20is%20used,acts%20as%20a%20Eureka%20Server.&text=Make%20sure%20Spring%20cloud%20Eureka,in%20your%20build%20configuration%20file.&text=By%20default%2C%20the%20Eureka%20Server%20registers%20itself%20into%20the%20discovery.
24. https://www.geeksforgeeks.org/spring-cloud-openfeign-with-example-project/
25. https://medium.com/@erayaraz10/building-a-microservice-with-spring-boot-config-server-a-step-by-step-guide-with-code-explanation-769d548cd71d
26. https://ieeexplore.ieee.org/document/8633605 (A Web-Based Application for Innovative Hospital Appointment Scheduling Using Neural Network)
27. https://content.iospress.com/articles/intelligent-decision-technologies/idt00078 (Object oriented architecture for affective multimodal e-learning interfaces)
28. https://www.informatica.si/index.php/informatica/article/view/1496 (M-LEARNING PROGRAMMING PLATFORM: EVALUATION IN ELEMENTARY SCHOOLS)

29. https://ieeexplore.ieee.org/document/8633694 (Software Measures for Common Design Patterns Using Visual Studio Code Metrics)
30. https://ieeexplore.ieee.org/document/10098026 (A microservices-based iterative development approach for usable, reliable and explainable A.I.-infused medical applications using R.U.P)
31. https://ieeexplore.ieee.org/document/4492683 (User Modelling in a Collaborative Learning Environment for UML)
32. https://ieeexplore.ieee.org/document/1661516 (A Knowledge-Based Software Life-Cycle Framework for the Incorporation of Multicriteria Analysis in Intelligent User Interfaces)
33. https://ieeexplore.ieee.org/document/1652382 (A Web-based Educational Application for Teaching of Programming: Student Modeling via Stereotypes)
34. https://onlinelibrary.wiley.com/doi/10.1046/j.0266-4909.2001.00172.x (An object-oriented software life cycle of an intelligent tutoring system)
35. https://ieeexplore.ieee.org/document/8272037 (Security and Privacy Analysis of Mobile Health Applications: The Alarming State of Practice)
36. https://ieeexplore.ieee.org/document/7223197 (Recovering RSA private keys on implementations with tampered LSBs)
37. https://eprint.iacr.org/2013/026 (RSA private key reconstruction from random bits using SAT solvers)
38. https://eprint.iacr.org/2013/489 (An Efficient Scheme for Centralized Group Key Management in Collaborative Environments)
39. https://www.didaktorika.gr/eadd/handle/10442/17270 (Κρυπτανάλυση αλγορίθμων και εφαρμογές της κρυπτογραφίας σε κακόβουλο λογισμικό)