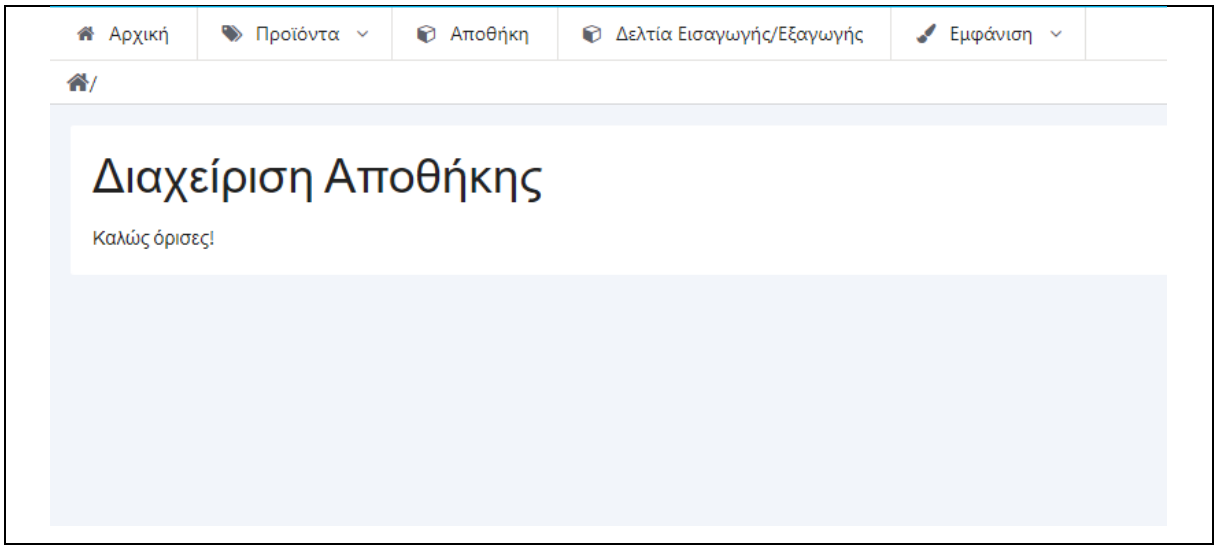
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ανάπτυξη web εφαρμογής για τη διαχείριση αποθήκης με σκοπό την αποτελεσματική διαχείριση της εφοδιαστικής αλυσίδας



Του φοιτητή
Βιτσιώτη Δημήτριου
Αρ. Μητρώου: 134103

Επιβλέπων
Ονοματεπώνυμο Γουλιάνας Κωνσταντίνος
Βαθμίδα Αναπληρωτής Καθηγητής

Ημερομηνία 13-06-2022

Τίτλος Δ.Ε. Ανάπτυξη web εφαρμογής για τη διαχείριση αποθήκης με σκοπό την αποτελεσματική διαχείριση της εφοδιαστικής αλυσίδας

Κωδικός Δ.Ε. 21347

Ονοματεπώνυμο φοιτητή/τών Βιτσιώτης Δημήτριος

Ονοματεπώνυμο εισηγητή Γουλιάνας Κωνσταντίνος

Ημερομηνία ανάληψης Δ.Ε. 13-10-2021

Ημερομηνία περάτωσης Δ.Ε. 13-06-2022

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

«Αφιέρωση»

Πρόλογος

Την παρούσα πτυχιακή εργασία την επέλεξα βλέποντας την ανάγκη που υπάρχει για τη διαχείριση του αποθέματος μιας επιχείρησης ή ενός φορέα του δημοσίου. Έτσι μου κίνησε το ενδιαφέρον για την ανάπτυξη και υλοποίηση μιας εφαρμογής που εύκολα, γρήγορα και αποτελεσματικά να προσφέρει ένα ικανοποιητικό αποτέλεσμα της παραπάνω ανάγκης.

Η παρούσα κατάσταση της πανδημίας που ζούμε σημέρα οστόσο με έκανε να σκεφτώ ένα επιπλέον παράγοντα που θα έφερνε σημαντικές αλλαγές στην υλοποίηση της εφαρμογής. Αυτό την εξ αποστάσεως εργασίας που λίγο πολύ όλοι έχουν συναντήσει σε πολλές μορφές της, είτε είναι στην εργασία μας, είτε σε κάποια υπηρεσία που χρειαζόμασταν. Η εφαρμογή λοιπόν προσφέρει την προσβασιμότητα σε αυτή από οπουδήποτε βρίσκεται ο χρήστης. Εύκολα και γρήγορα από οποιδήποτε περιβάλλον εργάζεται μπορεί να έχει πρόσβαση σε αυτή, έχοντας όμως απαραίτητη προϋπόθεση τη σύνδεση στο διαδίκτυο.

Η εν λόγο διαδικτυακή εφαρμογή θα προσφέρει την αποτελεσματική διαχείριση της εφοδιαστικής αλυσίδας περιορίζοντας όσο είναι δυνατόν τον ανθρώπινο παράγοντα. Προσφέρει αυτοματοποιημένες λειτουργίες όπου είναι απαραίτητο, κρατώντας ενήμερο το χρήστη κάθε στιγμή για την παρούσα κατάσταση του αποθέματος. Ο χρήστης θα μπορεί από ένα περιβάλλον να διαχειριστεί περισσότερες της μίας αποθήκης και σε όλο το απόθεμα.

Περίληψη

Η εφαρμογή που αναπτύχθηκε έχει σκοπό την αποτελεσματική διαχείριση της εφοδιαστικής αλυσίδας μιας επιχείρισης ή φορέα. Κάθε επιχειρίση ή φορέας έχει στην κατωχή του τεράστια αποθέματα σε αποθήκες που χρείαζονται να καταγράφονται κάθε φορά που γίνεται μια κίνηση εισαγωγής ή εξαγωγής.

Τα αποθέματα τις περισσότερες φορές μάλιστα δεν βρίσκονται σε μια αποθήκη αλλά σε περισσότερες ή οποίες μπορεί να έχουν μεγάλες αποστάσεις μεταξύ τους, ακόμα και χιλιομετρικές. Οι αποθήκες χωρίζονται κάποιες φορές και σε τμήματα ανάλογα με τα προϊόντα που αποθηκεύονται σε αυτά. Έτσι διαφορετικά τμήματα περιέχουν διαφορέτικα προϊοντα με συγκεκριμένες ανάγκες, όπως η θερμοκρασία του περιβάλλοντος. Η εφαρμογή ενημερώνει το χρήστη που βρίσκεται το κάθε προϊόν και το απόθεμα του. Επιπλέον του προσφέρει και κάποια στοιχεία του κάθε προϊόντος, όπως η ημερομηνία λήξης για τα φαγώσιμα.

Ο χρήστης εφόσον γνωρίζει ανα πάσα στιγμή το απόθεμα που διαθέτει για κάθε προϊόν της κάθε αποθήκης μπορεί πιο εύκολα να εισάγει ή να εξάγει κάποιο από αυτό όταν υπάρχει ζήτηση. Επίσης με αυτή τη δυνατότητα ο χρήστης μπορεί να προβλέψει και να διαχειριστεί πιο έυκολα το απόθεμα, ώστε να αποτρέψει κάθε πιθανότητα να ξεμείνει από κάποιο προϊοίον με αυξημένη ζήτηση στην αγορά. Τα δελτία εισαγωγής και εξαγωγής των προϊόντων αποτελεί εύκολη διαδικασία η οποία αυτόματα ενημερώνει και το απόθεμα για το εν λόγω προϊον.

Σηματικό πλεονέκτημα της εφαρμογής είναι ότι είναι διαδικτυακή. Ο χρήστης μπορεί να συνδεθεί από αποιοδήποτε σημείο έχει πρόσβαση στο διαδίκτυο.

«Development of a web application for warehouse management in order to effectively manage the supply chain»

«Dimitrios Vitsiotis»

Abstract

The application developed aims to effectively manage the supply chain of a business or organization. Every business or entity has huge stocks at its disposal that need to be recorded each time an import or export transaction is made.

The stocks most of the time are not in a warehouse but in more or which may have long distances between them, even kilometers. Warehouses are sometimes divided into sections depending on the products stored in them. Thus different parts contain different products with specific needs, such as the ambient temperature. The application informs the user where each product is and its stock. It also offers some details of each product, such as the expiration date for the food.

The user, if he knows at any time the stock he has for each product of each warehouse, can more easily import or export any of it when there is a demand. Also with this feature the user can predict and manage the inventory more easily, in order to prevent any possibility of running out of a product with increased demand in the market. The import and export vouchers of the products is an easy process which automatically updates the stock for the product in question.

An important advantage of the application is that it is online. The user can connect from anywhere with internet access.

Περιεχόμενα

Συντομογραφίες

REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol Secure
URI	Uniform Resource Identifier
JVM	Java virtual machine
CRUD	Create, Update, Delete
IDE	Integrated development environment
UML	Unified Modeling Language

Κεφάλαιο 1ο: Ανάπτυξη τεχνολογιών

1.1 Εισαγωγή

Στο κεφάλαιο αυτό γίνεται αναφορά των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Η εφαρμογή σχεδιάστηκε σύμφωνα με την αρχιτεκτονική RESTful API για να «τρέχει» στο διαδίκτυο με σκοπό ο χρήστης να έχει πρόσβαση απ' οπουδήποτε βρίσκεται συνδεδεμένος στο διαδίκτυο. Για να κατανοήσει αυτή την αρχιτεκτονική, ο αναγνώστης πρέπει να γίνει μια εκτενής αναδρομή το πως φτάσαμε ως εκεί.

1.2 Representational State Transfer - REST

1.2.1 Η ιστορία του Διαδικτύου

Ο Ιστός άρχισε να εισέρχεται σε καθημερινή χρήση το 1993-4 [1], όταν άρχισαν να γίνονται διαθέσιμοι ιστότοποι γενικής χρήσης. Εκείνη την εποχή, υπήρχε μόνο μια κατακερματισμένη περιγραφή της αρχιτεκτονικής του Ιστού και υπήρχε πίεση στη βιομηχανία να συμφωνήσει σε κάποιο πρότυπο για τα πρωτόκολλα διεπαφής Ιστού. Για παράδειγμα, αρκετές πειραματικές επεκτάσεις είχαν προστεθεί στο πρωτόκολλο επικοινωνίας (HTTP) για την υποστήριξη διακομιστών μεσολάβησης και προτάθηκαν περισσότερες επεκτάσεις, αλλά υπήρχε ανάγκη για μια επίσημη αρχιτεκτονική Ιστού με την οποία θα αξιολογούνταν ο αντίκτυπος αυτών των αλλαγών.

Μαζί οι ομάδες εργασίας W3C και IETF, άρχισαν να εργάζονται για τη δημιουργία επίσημων περιγραφών των τριών βασικών προτύπων του Ιστού: URI, HTTP και HTML. Ο Roy Fielding συμμετείχε στη δημιουργία αυτών των προτύπων (συγκεκριμένα HTTP 1.0 και 1.1 και URI) και κατά τα επόμενα έξι χρόνια ανέπτυξε το αρχιτεκτονικό στυλ REST, δοκιμάζοντας τους περιορισμούς του στα πρότυπα πρωτοκόλλου του Ιστού και χρησιμοποιώντας το ως μέσο για τον καθορισμό αρχιτεκτονικές βελτιώσεις — και για τον εντοπισμό αρχιτεκτονικών αναντιστοιχιών. Ο Fielding όρισε το REST στη διδακτορική του διατριβή το 2000 «Αρχιτεκτονικά Στυλ και Σχεδιασμός Αρχιτεκτονικών Λογισμικού βασισμένων σε Δικτύωση» στο UC Irvine.

1.3 Διδακτορική διατριβή του Thomas Fielding

1.3.1 Παραγωγή REST

Για να δημιουργήσει το αρχιτεκτονικό στυλ REST, ο Fielding προσδιόρισε τις απαιτήσεις που ισχύουν κατά τη δημιουργία μιας παγκόσμιας εφαρμογής που βασίζεται σε δίκτυο, όπως η ανάγκη για ένα χαμηλό φράγμα εισόδου για να καταστεί δυνατή η παγκόσμια υιοθέτηση. Επίσης, ερεύνησε πολλά υπάρχοντα αρχιτεκτονικά στυλ για εφαρμογές που βασίζονται σε δίκτυο, προσδιορίζοντας ποια χαρακτηριστικά είναι κοινά με άλλα στυλ, όπως η προσωρινή αποθήκευση και τα χαρακτηριστικά πελάτη-διακομιστή, και εκείνα που είναι μοναδικά για το REST, όπως η έννοια των πόρων. Ο Fielding προσπαθούσε τόσο να κατηγοριοποιήσει την υπάρχουσα αρχιτεκτονική της τρέχουσας υλοποίησης όσο και να προσδιορίσει ποιες πτυχές πρέπει να θεωρούνται κεντρικές για τις απαιτήσεις συμπεριφοράς και απόδοσης του Ιστού.

Από τη φύση τους, τα αρχιτεκτονικά στυλ είναι ανεξάρτητα από οποιαδήποτε συγκεκριμένη υλοποίηση, και ενώ το REST δημιουργήθηκε ως μέρος της ανάπτυξης των προτύπων Ιστού, η υλοποίηση του Ιστού δεν υπακούει σε κάθε περιορισμό στο αρχιτεκτονικό στυλ REST. Αναντιστοιχίες μπορεί να προκύψουν λόγω άγνοιας ή παράβλεψης, αλλά η ύπαρξη του αρχιτεκτονικού στυλ REST σημαίνει ότι μπορούν να εντοπιστούν προτού τυποποιηθούν. Για παράδειγμα, ο Fielding αναγνώρισε την ενσωμάτωση πληροφοριών περιόδου λειτουργίας σε URI ως παραβίαση των περιορισμών του REST που μπορεί να επηρεάσει αρνητικά την κοινόχρηστη προσωρινή αποθήκευση και την επεκτασιμότητα διακομιστή. Τα cookie HTTP παραβίασαν επίσης τους περιορισμούς REST επειδή μπορεί να μην συγχρονίζονται με την κατάσταση εφαρμογής του προγράμματος περιήγησης, καθιστώντας τα αναξιόπιστα. περιέχουν επίσης αδιαφανή δεδομένα που μπορεί να είναι ανησυχητικά για το απόρρητο και την ασφάλεια.

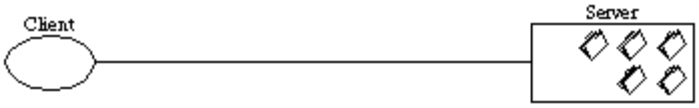
1.4 Περιγραφή περιορισμών που συνθέτουν το Rest

1.4.1 Null Style

Υπάρχουν δύο κοινές οπτικές σχετικά με τη διαδικασία του αρχιτεκτονικού σχεδιασμού, είτε πρόκειται για κτίρια είτε για λογισμικό. Το πρώτο είναι ότι ένας σχεδιαστής ξεκινά από το μηδέν- με μια κενή πλάκα, ένα λευκό πίνακα ή ένα πίνακα σχεδίασης - και χτίζει μια αρχιτεκτονική από γνωστά στοιχεία μέχρι να ικανοποιήσει τις ανάγκες του επιδιωκόμενου συστήματος. Το δεύτερο είναι ότι ένας σχεδιαστής ξεκινά με τις ανάγκες του συστήματος στο σύνολό του, χωρίς περιορισμούς, και στη συνέχεια εντοπίζει και εφαρμόζει σταδιακά περιορισμούς σε στοιχεία του συστήματος προκειμένου να διαφοροποιήσει τον χώρο σχεδιασμού και να επιτρέψει στις δυνάμεις που επηρεάζουν τη συμπεριφορά του συστήματος να ρέουν φυσικά, σε αρμονία με το σύστημα. Ενώ το πρώτο δίνει έμφαση στη δημιουργικότητα και το απεριόριστο όραμα, το δεύτερο δίνει έμφαση στον περιορισμό και την κατανόηση του πλαισίου του συστήματος. Το REST έχει αναπτυχθεί χρησιμοποιώντας την τελευταία μέθοδο.

1.4.2 Client-Server

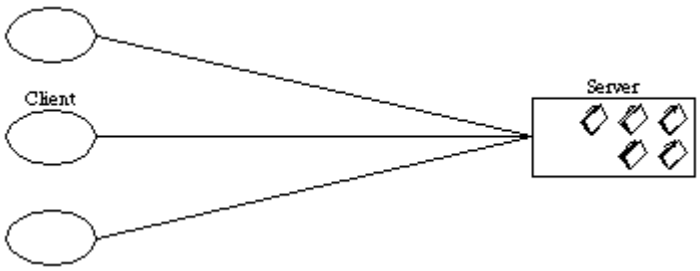
Οι πρώτοι περιορισμοί που προστέθηκαν στο υβριδικό μας στυλ είναι αυτοί του αρχιτεκτονικού στυλ πελάτη-διακομιστή (Εικόνα 5-1). Ο διαχωρισμός των προβληματισμών είναι η αρχή πίσω από τους περιορισμούς πελάτη-διακομιστή. Διαχωρίζοντας τα θέματα της διεπαφής χρήστη από της αποθήκευσης δεδομένων, βελτιώνουμε τη φορητότητα της διεπαφής χρήστη σε πολλές πλατφόρμες και βελτιώνουμε την επεκτασιμότητα απλοποιώντας τα στοιχεία διακομιστή. Ίσως το πιο σημαντικό για τον Ιστό, ωστόσο, είναι ότι ο διαχωρισμός επιτρέπει στα στοιχεία να εξελιχθούν ανεξάρτητα, υποστηρίζοντας έτσι την απαίτηση κλίμακας Διαδικτύου για πολλαπλούς οργανωτικούς τομείς.



Εικόνα 1. Client Server

1.4.3 **Stateless**

Στη συνέχεια προσθέτουμε έναν περιορισμό στην αλληλεπίδραση πελάτη-διακομιστή: η επικοινωνία πρέπει να είναι άδηλης φύσης, όπως στο client-stateless-server (Εικόνα 2), έτσι ώστε κάθε αίτηση από πελάτη σε διακομιστή να περιέχει όλες τις απαραίτητες πληροφορίες για την κατανόηση του αιτήματος και να μην μπορεί να εκμεταλλευτεί οποιοδήποτε αποθηκευμένο περιεχόμενο στον διακομιστή. Συνεπώς, η κατάσταση της συνόδου διατηρείται εξ ολοκλήρου στον client.



Εικόνα 2. Client-Stateless-Server

Αυτός ο περιορισμός προκαλεί τις ιδιότητες της προβολής, της αξιοπιστίας και της επεκτασιμότητας. Η προβολή βελτιώνεται επειδή ένα σύστημα παρακολούθησης δεν χρειάζεται να κοιτάξει πέρα από ένα μεμονωμένο δεδομένο αίτημα για να προσδιορίσει την πλήρη φύση του αιτήματος. Η αξιοπιστία βελτιώνεται επειδή διευκολύνεται το έργο της ανάκαμψης από μερικές αποτυχίες . Η επεκτασιμότητα βελτιώνεται επειδή η μη αποθήκευση της κατάστασης μεταξύ των αιτημάτων επιτρέπει στο στοιχείο του διακομιστή να απελευθερώνει γρήγορα πόρους και απλοποιεί περαιτέρω την υλοποίηση επειδή ο διακομιστής δεν χρειάζεται να διαχειρίζεται τη χρήση των πόρων μεταξύ των αιτημάτων.

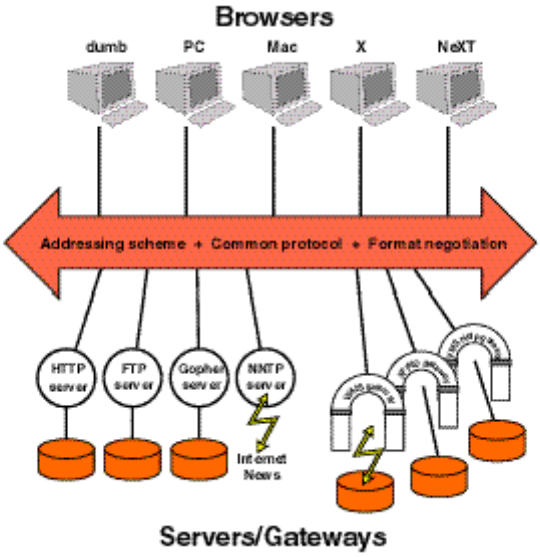
Όπως οι περισσότερες αρχιτεκτονικές εναλλακτικές επιλογές, ο περιορισμός χωρίς κατάσταση(Stateless) αντικατοπτρίζει ένα σχεδιαστικό συμβιβασμό. Το μειονέκτημα είναι ότι μπορεί να μειώσει την απόδοση του δικτύου αυξάνοντας τα επαναλαμβανόμενα δεδομένα (επιβάρυνση ανά διεπαφή) που αποστέλλονται σε μια σειρά αιτημάτων, δεδομένου ότι τα δεδομένα αυτά δεν μπορούν να παραμείνουν στο διακομιστή σε ένα κοινόχρηστο πλαίσιο. Επιπλέον, η τοποθέτηση της κατάστασης της εφαρμογής στην πλευρά του πελάτη μειώνει τον έλεγχο του διακομιστή όσον αφορά τη σταθερή συμπεριφορά της εφαρμογής, καθώς η εφαρμογή εξαρτάται από τη σωστή υλοποίηση της σημασιολογίας σε πολλαπλές εκδόσεις του πελάτη.

1.4.4 **Cache**

Το πλεονέκτημα της προσθήκης περιορισμών στην κρυφή μνήμη είναι ότι έχουν τη δυνατότητα να εξαλείψουν μερικώς ή πλήρως ορισμένες διεπιδράσεις, βελτιώνοντας την αποδοτικότητα, την επεκτασιμότητα και την απόδοση που αντιλαμβάνεται ο χρήστης μειώνοντας τη μέση καθυστέρηση μιας σειράς διεπιδράσεων. Το αντιστάθμισμα, ωστόσο, είναι ότι μια κρυφή μνήμη μπορεί να μειώσει την αξιοπιστία εάν τα παρωχημένα δεδομένα εντός της κρυφής μνήμης διαφέρουν σημαντικά από τα δεδομένα που θα λαμβάνονταν εάν η κλήση είχε σταλεί απευθείας στον διακομιστή.

Η πρώιμη αρχιτεκτονική του Παγκόσμιου Ιστού, όπως απεικονίζεται στο διάγραμμα της Εικόνας 3, καθοριζόταν από το σύνολο περιορισμών πελάτη-κρυφής μνήμης-χωρίς κατάσταση-διακομιστή. Δηλαδή, η σχεδιαστική λογική που παρουσιάστηκε για την αρχιτεκτονική του Ιστού πριν από το 1994

επικεντρώθηκε στην αλληλεπίδραση πελάτη-εξυπηρετητή Stateless για την ανταλλαγή στατικών εγγράφων μέσω του Διαδικτύου.



Εικόνα 3. Πρώιμη αρχιτεκτονική του Παγκόσμιου Ιστού

Η πρώιμη αρχιτεκτονική του Παγκόσμιου Ιστού, όπως απεικονίζεται στο διάγραμμα της Εικόνας 5-5, καθοριζόταν από το σύνολο περιορισμών πελάτη-κρυφής μνήμης-χωρίς κατάσταση-διακομιστή. Δηλαδή, η σχεδιαστική λογική που παρουσιάστηκε για την αρχιτεκτονική του Ιστού πριν από το 1994 επικεντρώθηκε στην αλληλεπίδραση πελάτη-εξυπηρετητή χωρίς κατάσταση για την ανταλλαγή στατικών εγγράφων μέσω του Διαδικτύου. Τα πρωτόκολλα για την επικοινωνία των διεπιδράσεων είχαν υποτυπώδη υποστήριξη για μη κοινόχρηστες κρυφές μνήμες, αλλά δεν περιόριζαν τη διεπαφή σε ένα συνεπές σύνολο σημασιολογίας για όλους τους πόρους. Αντ' αυτού, ο Ιστός βασιζόταν στη χρήση μιας κοινής βιβλιοθήκης υλοποίησης client-server (CERN libwww) για τη διατήρηση της συνοχής σε όλες τις εφαρμογές του Ιστού.

Οι προγραμματιστές των εφαρμογών Ιστού είχαν ήδη ξεπεράσει τον πρώιμο σχεδιασμό. Εκτός από τα στατικά έγγραφα, τα αιτήματα μπορούσαν να προσδιορίζουν υπηρεσίες που παρήγαγαν δυναμικά αποτελέσματα, όπως χαρτογραφικές εικόνες [Kevin Hughes] και server-side scripts [Rob McCool]. Επίσης είχαν αρχίσει να γίνονται εργασίες σε ενδιάμεσα στοιχεία, με τη μορφή πληρεξούσιων αντιπροσώπων και κοινόχρηστων κρυφών αποθηκευτικών χώρων, αλλά χρειάζονταν επεκτάσεις στα πρωτόκολλα προκειμένου να επικοινωνούν αξιόπιστα. Οι ακόλουθες ενότητες περιγράφουν τους περιορισμούς που προστέθηκαν στο αρχιτεκτονικό στυλ του Ιστού, προκειμένου να κατευθύνουν τις επεκτάσεις που διαμορφώνουν τη σύγχρονη αρχιτεκτονική του Ιστού.

1.4.5 **Ενιαία διασύνδεση**

Το κεντρικό χαρακτηριστικό που διακρίνει το αρχιτεκτονικό στυλ REST από άλλα στυλ που βασίζονται σε δίκτυα είναι η έμφαση που δίνει σε μια ενιαία διεπαφή μεταξύ των επιμέρους στοιχείων. Εφαρμόζοντας την αρχή της γενικότητας της μηχανικής λογισμικού στη διεπαφή των συστατικών, η ολική αρχιτεκτονική του συστήματος απλουστεύεται και η ορατότητα των αλληλεπιδράσεων βελτιώνεται. Οι υλοποιήσεις αποδεσμεύονται από τις υπηρεσίες που προσφέρουν, γεγονός που

ενθαρρύνει την ανεξάρτητη εξελιξιμότητα. Το αντίτιμο, ωστόσο, είναι ότι μια ομοιόμορφη διασύνδεση μειώνει την αποδοτικότητα, καθώς οι πληροφορίες μεταφέρονται σε τυποποιημένη μορφή και όχι σε μορφή που είναι προσαρμοσμένη στις ανάγκες μιας εφαρμογής. Η διεπαφή REST έχει σχεδιαστεί για να είναι αποτελεσματική για τη μεταφορά δεδομένων μεγάλου μεγέθους υπερμέσων, βελτιστοποιώντας την κοινή περίπτωση του Παγκόσμιου Ιστού, με αποτέλεσμα όμως μια διεπαφή που δεν είναι ιδανική για άλλες μορφές αρχιτεκτονικής διάδρασης.

Προκειμένου να επιτευχθεί μια ομοιόμορφη διασύνδεση, απαιτούνται πολλαπλοί αρχιτεκτονικοί περιορισμοί για την καθοδήγηση της συμπεριφοράς των επιμέρους στοιχείων. Το REST ορίζεται από τέσσερις περιορισμούς διεπαφής: ταυτοποίηση των πόρων, χειρισμός των πόρων μέσω αναπαραστάσεων, αυτο-περιγραφικά μηνύματα και, υπερμέσα ως η μηχανή της λειτουργίας της εφαρμογής.

1.4.6 **Πολυεπίπεδο σύστημα**

Προκειμένου να βελτιώσουμε περαιτέρω τη συμπεριφορά για απαιτήσεις σε κλίμακα Διαδικτύου, προσθέσαμε πολυεπίπεδους περιορισμούς συστήματος. Το στυλ του πολυεπίπεδου συστήματος επιτρέπει σε μια αρχιτεκτονική να αποτελείται από ιεραρχικά στρώματα, περιορίζοντας τη συμπεριφορά των επιμέρους στοιχείων έτσι ώστε κάθε στοιχείο να μην μπορεί να "δει" πέρα από το άμεσο στρώμα με το οποίο αλληλεπιδρά. Περιορίζοντας τη γνώση του συστήματος σε ένα μόνο στρώμα, θέτουμε ένα όριο στη συνολική πολυπλοκότητα του συστήματος και προωθούμε την ανεξαρτησία υποστρώματος. Τα στρώματα μπορούν να χρησιμοποιηθούν για την ενθυλάκωση παλαιών υπηρεσιών και για την προστασία νέων υπηρεσιών από παλαιούς clients, απλοποιώντας τα στοιχεία με το να μεταφέρεται η λειτουργικότητα που χρησιμοποιείται σπάνια σε έναν κοινό ενδιάμεσο. Οι ενδιάμεσοι μπορούν επίσης να χρησιμοποιηθούν για τη βελτίωση της επεκτασιμότητας του συστήματος, επιτρέποντας την εξισορρόπηση του φορτίου των υπηρεσιών σε πολλαπλά δίκτυα και επεξεργαστές.

Το κύριο μειονέκτημα των πολυεπίπεδων συστημάτων είναι ότι αυξάνουν την επιβάρυνση και την καθυστέρηση στην επεξεργασία των δεδομένων, μειώνοντας την απόδοση που αντιλαμβάνεται ο χρήστης. Για ένα σύστημα που βασίζεται στο δίκτυο και υποστηρίζει περιορισμούς κρυφής μνήμης, αυτό μπορεί να αντισταθμιστεί από τα οφέλη της κοινής κρυφής μνήμης σε ενδιάμεσους σταθμούς. Η τοποθέτηση κοινόχρηστων κρυφών μνημών αποθήκευσης στα όρια ενός οργανωμένου τομέα μπορεί να οδηγήσει σε σημαντικά οφέλη απόδοσης. Τέτοια στρώματα επιτρέπουν επίσης την επιβολή των πολιτικών ασφαλείας στα δεδομένα που διέρχονται από τα διάφορα οργανωτικά επίπεδα, όπως απαιτείται από τα firewalls.

Ο συνδυασμός του πολυεπίπεδου συστήματος και των ενιαίων περιορισμών διεπαφής οδηγεί σε αρχιτεκτονικές ιδιαιτερότητες παρόμοιες με εκείνες του ενιαίου στυλ pipe-and-filter. Ωστόσο, αν και η αλληλεπίδραση REST είναι αμφίδρομη, οι ροές δεδομένων μεγάλου μεγέθους της αλληλεπίδρασης υπερμέσων μπορούν να υποβληθούν σε επεξεργασία σαν ένα δίκτυο ροής δεδομένων, με στοιχεία φίλτρων που εφαρμόζονται επιλεκτικά στη ροή δεδομένων προκειμένου να μετασχηματίζουν το περιεχόμενο κατά τη διέλευσή του. Στο πλαίσιο του REST, τα ενδιάμεσα τμήματα μπορούν να μετασχηματίζουν ενεργά το περιεχόμενο των μηνυμάτων, επειδή τα μηνύματα είναι αυτοπεριγραφικά και η σημασιολογία τους είναι ορατή στους ενδιάμεσους.

1.4.7 **Code-On-Demand**

Η τελευταία προσθήκη στο σύνολο των περιορισμών μας για το REST προέρχεται από το στυλ κώδικα on-demand. Το REST επιτρέπει την επέκταση της λειτουργικότητας του πελάτη με τη λήψη και εκτέλεση κώδικα με τη μορφή applets ή scripts. Αυτό απλοποιεί τους clients μειώνοντας τον αριθμό των λειτουργιών που απαιτείται να είναι υλοποιημένες εκ των προτέρων. Η δυνατότητα λήψης λειτουργιών μετά την υλοποίηση βελτιώνει την επεκτασιμότητα του συστήματος. Ωστόσο, μειώνεται επίσης η ορατότητα με αποτέλεσμα να αποτελεί μόνο έναν προαιρετικό περιορισμό στο πλαίσιο του REST.

Η έννοια του προαιρετικού περιορισμού μπορεί να μοιάζει αντιφατική. Ωστόσο, έχει έναν σκοπό στον αρχιτεκτονικό σχεδιασμό ενός συστήματος που περιλαμβάνει πολλαπλά οργανωτικά πλαίσια. Σημαίνει ότι η αρχιτεκτονική επωφελείται από τα οφέλη (και επωμίζεται τα μειονεκτήματα) των προαιρετικών περιορισμών μόνο όταν είναι γνωστό ότι ισχύουν για κάποιο τμήμα του συνολικού συστήματος. Για παράδειγμα, εάν είναι γνωστό ότι όλο το λογισμικό πελάτη σε έναν οργανισμό υποστηρίζει Java applets , τότε οι υπηρεσίες σε αυτόν τον οργανισμό μπορούν να κατασκευαστούν έτσι ώστε να αποκομίσουν το όφελος της βελτιωμένης λειτουργικότητας μέσω κλάσεων Java που μπορούν να ληφθούν. Ταυτόχρονα, όμως, το firewall του οργανισμού μπορεί να αποτρέπει τη μεταφορά των Java applets από εξωτερικές πηγές, και έτσι στον υπόλοιπο Ιστό θα φαίνεται ότι οι πελάτες αυτοί δεν υποστηρίζουν code on demand. Ένας προαιρετικός περιορισμός μας επιτρέπει να σχεδιάσουμε μια αρχιτεκτονική που υποστηρίζει την επιθυμητή συμπεριφορά στη γενική περίπτωση, αλλά με την προϋπόθεση ότι μπορεί να τεθεί εκτός λειτουργίας σε ορισμένα πλαίσια.

Βιβλιογραφικές αναφορές

Internet Site

[1] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_1

Επίλογος

Το κεφάλαιο αυτό αποτελεί μια εισαγωγική αναφορά της τεχνολογίας REST και εξηγεί τον αναγνώστη πως μέσα από την ιστορία καταλήξαμε να αναπτύξουμε αυτή την αρχιτεκτονική και να την χρησιμποποιούμε στη ανάπτυξη σύγχρονων συστημάτων σήμερα.

Κεφάλαιο 2ο: Περιγραφή των Framework και τεχνολογιών χρησιμοποιήθηκαν για την υλοποίησης

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο γίνεται αναφορά στα framework που χρησιμοποιήθηκαν για την υλοποίηση του back end συστήματος της εφαρμογή αλλά και του front end. Θα γίνει περιγραφή του ελέγχου των κλήσεων του back end και αντίσοιχα στο front end.

2.2 Java Frameworks

Τα Java Frameworks και γενικότερα ένα framework στον προγραμματισμό είναι ένα σύνολο προ-γραμμένου κώδικα που λειτουργεί ως πρότυπο ή σκελετός, το οποίο ένας προγραμματιστής μπορεί στη συνέχεια να χρησιμοποιήσει και να επαναχρησιμοποιήσει για να δημιουργήσει μια εφαρμογή προσθέτοντας τον δικό του κώδικα όπως χρειάζεται για να λειτουργήσει η εφαρμογή όπως επιθυμεί. Περιλαμβάνουν προκαθορισμένες κλάσεις και λειτουργίες που χρησιμοποιούνται για την επεξεργασία, την εισαγωγή και τη διαχείριση συσκευών υλικού, καθώς και για την αλληλεπίδραση με το λογισμικό του συστήματος. Εξαρτάται από τον τύπο του Framework, το επίπεδο δεξιοτήτων του προγραμματιστή, το τι προσπαθεί να επιτύχει και τις προτιμήσεις του.

2.3 Spring Framework

Το Spring Framework υποστηρίζει ένα ευρύ φάσμα εφαρμογών[1]. Σε μεγάλες επιχειρήσεις θα βρείτε Spring enterprise εφαρμογές που τρέχουν σε κάποιον application server, άλλες που τρέχουν με την μορφή jar και έχουν ενσωματωμένους application servers, και ίσως και standalone εφαρμογές κυρίως για batch δουλειές ή για integration με πολλαπλά συστήματα.

Το Spring Framework διευκολύνει την δημιουργία εταιρικών εφαρμογών Java. Παρέχει όλα τα απαραίτητα εργαλεία και βιβλιοθήκες που χρειαζόμαστε για να γράψουμε Java enterprise εφαρμογές και επίσης υποστηρίζει Groovy και Kotlin ως εναλλακτικές γλώσσες προγραμματισμού για το JVM. Είναι ένα πολύ ευέλικτο framework που μας επιτρέπει να αναπτύσσουμε κώδικα για πολλές και διαφορετικές αρχιτεκτονικές ανάλογα με τις ανάγκες της εκάστοτε εφαρμογής.

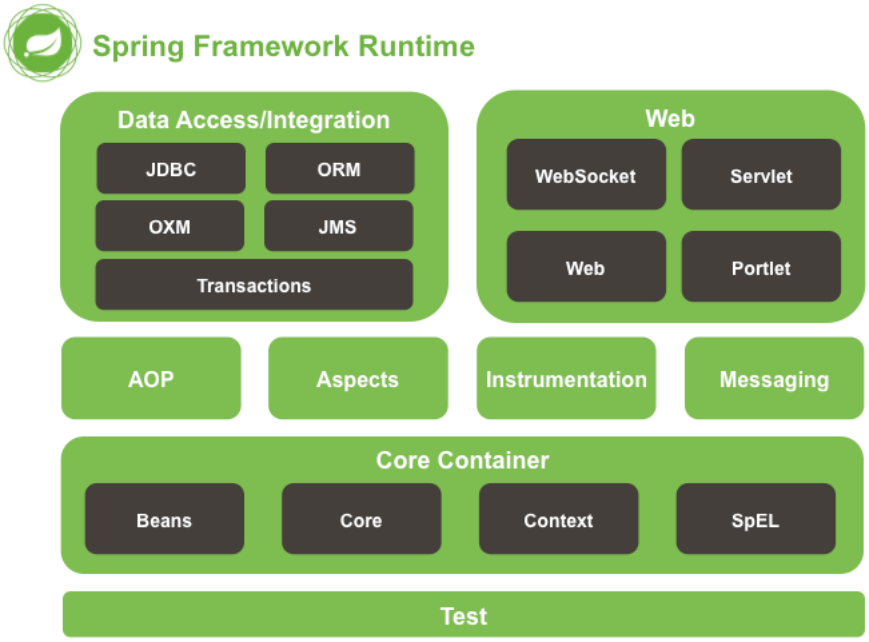
Από το Spring Framework 5.1, το Spring απαιτεί την εγκατάσταση της Java 8 αλλά παρέχει και υποστήριξη για το JDK 11 LTS. Η Java SE 8 update 60 προτείνεται ως η ελάχιστη έκδοση αν χρησιμοποιείται Java 8 σε συνδυασμό με την τελευταία έκδοση του Spring Framework.

Το community του Spring είναι αρκετά μεγάλο. Αυτό σημαίνει ότι το συγκεκριμένο open source framework το έχουν υιοθετήσει αρκετοί προγραμματιστές και υπάρχει μια πληθώρα πληροφοριών και άρθρων στον Ίντερνετ που καλύπτουν σχεδόν όλες σας τις ερωτήσεις.

Το **Spring Framework Inversion of Control (IoC)** αναγνωρίζει αυτή την αδυναμία της Java και έρχεται να προσφέρει ένα δομημένο και εύκολο τρόπο σύνδεσης των διαφόρων αντικειμένων που υπάρχουν σε μια εφαρμογή. Το καταφέρνει αυτό έχοντας ήδη υλοποιήσει στις βιβλιοθήκες του τα περισσότερα από τα design patterns τα οποία ο προγραμματιστής απλά ενσωματώνει στον κώδικα με

την χρήση των annotations. Όλη αυτή η θεωρία θα γίνει πολύ πιο κατανοητή όταν αρχίζουμε να αναλύουμε τα προγράμματα μας.

Όπως ήδη αναφέραμε στην αρχή της ενότητας, θα ασχοληθούμε σε αυτή την σειρά δωρεάν μαθημάτων Spring Framework μόνο με τις κύριες βιβλιοθήκες του Spring Framework. Το Spring Framework αποτελείται από σχεδόν 20 modules. Αυτά τα modules είναι οργανωμένα σε Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging, and Test. Εμείς θα συγκεντρωθούμε στα modules του Core Container.



Εικόνα 2.1 Core Container modules

Μια από τις καινοτομίες, ίσως η πιο σημαντική, του Spring Framework είναι η ιδέα του **Spring Container**. Στην ορολογία ανάπτυξης λογισμικού, η λέξη container χρησιμοποιείται για την περιγραφή οποιουδήποτε στοιχείου που περιέχει άλλα στοιχεία μέσα του. Για παράδειγμα, οι application servers όπως ο Tomcat ή ο TomEE έχουν ενσωματωμένους containers για να μπορούν να διαχειρίζονται WAR αρχεία. Ο ρόλος του container στους application servers είναι να δημιουργεί τα αντικείμενα, να τα συνδέει μεταξύ τους και γενικότερα να διαχειρίζεται ολόκληρο το life-cycle όπως την δημιουργία και την διαγραφή τους. Επίσης αν υπάρχουν σχέσεις μεταξύ των αντικειμένων (object dependencies), τότε αναλαμβάνει να εισάγει το ένα αντικείμενο μέσα σε ένα άλλο με την τεχνική του injection όπως ονομάζεται.

Οπότε στο δικό μας java παράδειγμα που αναλύσαμε πιο πάνω, σκεφτείτε ότι ένα container θα είχε την ευθύνη να δημιουργήσει το EmployeeServiceImpl αντικείμενο για εμάς, και εμείς απλά θα δηλώναμε που ακριβώς θέλουμε να το χρησιμοποιήσουμε. Ίσως το παρακάτω διάγραμμα να σας βοηθήσει να καταλάβετε την γενική ιδέα του container καλύτερα.

Σε Java standalone εφαρμογές δεν υπάρχει η έννοια του container. Αυτό είναι μόνο διαθέσιμο αν και εφόσον γράφουμε Enterprise Java επάνω σε κάποιον Application Server όπως TomEE ή Wildfly. Εδώ ακριβώς έρχεται λοιπόν η ιδέα του Spring Framework. Μας δίνει την δυνατότητα να έχουμε container μέσα σε μια Java standalone εφαρμογή. Όπως μας δείχνει και το παραπάνω διάγραμμα, το Spring Framework μας παρέχει ένα Object Factory από το οποίο μπορούμε να ζητάμε αντικείμενα και εκείνο με την σειρά του,με βάση το configuration που του έχουμε δώσει, να βρίσκει την σωστή υλοποίηση του (implementation) και να μας το παραδίδει.

Ο σκοπός λοιπόν του Spring container είναι να μας παρέχει δύο κύριες λειτουργίες:

1. Να δημιουργεί και να διαχειρίζεται τα αντικείμενα της εφαρμογής μας (**Inversion of Control**)

2. Να εισάγει αντικείμενα στην εφαρμογή όπου χρειάζονται (**Dependency Injection**).

Όπως βλέπετε, ο έλεγχος δημιουργία των αντικειμένων απομακρύνεται πια από τον προγραμματιστή και το αναλαμβάνει εξ ολοκλήρου το spring framework. Αυτό ακριβώς σημαίνει και το **Inversion of Control**. Ότι αντιστρέφουμε την ευθύνη και από τον προγραμματιστή την αναλαμβάνει το Spring Framework.

2.3.1 Δημιουργία και Εισαγωγή του Spring Container στον κωδικά

Έχουμε τρεις επιλογές:

1. XML configuration file – Αυτός ο τρόπος απλά παραμένει ακόμα για να υποστηρίζει legacy εφαρμογές και δεν χρησιμοποιείται πια.

2. Java Annotations – Αυτός είναι ένας από τους σύγχρονους τρόπους για να ενεργοποιήσουμε και να χρησιμοποιήσουμε το Spring Container. Αν και χρησιμοποιούμε annotations, θα χρειαστεί και η ύπαρξη του XML configuration file αλλά με πολύ λίγες πληροφορίες.

3. Java Source code – Είναι και αυτός ένας μοντέρνος τρόπος, τον οποίο μπορούμε να χρησιμοποιήσουμε για να ενεργοποιήσουμε και να χρησιμοποιήσουμε το Spring Container μόνο μέσα από τον κώδικα μας χωρίς να χρειάζεται καθόλου το XML Configuration file.

Για να μετατρέψουμε ένα απλό Java πρόγραμμα σε Spring Java πρόγραμμα ή να δημιουργήσουμε και να γράψουμε ένα Spring πρόγραμμα από την αρχή χρειάζεται κάθε φορά να ακολουθήσουμε τρία απλά βήματα:

1. Να δηλώσουμε τα Spring Beans (δηλαδή τα αντικείμενα μας)

2. Να δημιουργήσουμε το Spring Container

3. Να ανακτήσουμε τα Spring Beans από το Spring Container και να τα χρησιμοποιήσουμε

Ας πάρουμε λοιπόν ένα ένα τα βήματα και ας δημιουργήσουμε μετατρέψουμε το Java πρόγραμμα μας σε Spring. Μαζί με τον κώδικα και το configuration θα προσθέσουμε και την απαραίτητη θεωρία.

Ο όρος Spring σήμερα σημαίνει διαφορετικά πράγματα ανάλογα με την εφαρμογή στην οποία αναφερόμαστε. Αν και ο όρος Spring αρχικά αναφερόταν στο ίδιο το Framework, σήμερα πολλοί αναφέρονται σαν Spring στην σουίτα προϊόντων που έχουν αναπτυχθεί τα τελευταία χρόνια όπως Spring Boot, Spring Batch κτλ.Για τις ανάγκες της Π.Ε χρησιμοποιήθηκε το Spring Boot. Το Spring Boot έρχεται να δώσει νέα πνοή στις Web εφαρμογές προσφέροντας εύκολη ανάπτυξη των υπηρεσιών που είναι και η προτιμώμενη αρχιτεκτονική δομή μοντέρνων Web εφαρμογών. Διευκολύνει τη

δημιουργία αυτόνομων, παραγωγικών εφαρμογών βασισμένων στο Spring, οι οποίες δεν χρειάζεται επιπλέον παραμετροποίηση για να ξεκινήσει να ‘παζεί’.

Μια Spring Boot εφαρμογή μπορεί πολύ εύκολα να διαμορφωθεί στις ανάγκες μας εδώ https://start.spring.io/ . Επιλέγοντας κάποιες από τις παραμέτρους που μας δίνει το εργαλείο εξάγει ένα project που είναι έτοιμο να παίξει χωρίς επιπρόσθετες ρυθμίσεις. Παρακάτω θα γίνει μια αναφορά των παραμέτρων που χρησιμοποιήθηκαν.

2.3.2 Project - Maven Project

Για να χτιστεί το project χρησιμοποιήθηκε το Build Tool Maven[3]. Το Maven Apache είναι ένα ανοιχτού-κώδικα project που έχει ένα δομημένο και απλό τρόπο να δημιουργεί, να διαχειρίζεται και να τεστάρει java projects. Παρέχει όμως και άλλες δυνατότητες όπως εύκολη διαχείριση των βιβλιοθηκών στις οποίες στηρίζεται το project αλλά και εύκολη συμβατότητα και επικοινωνία με άλλα προϊόντα.

Το Apache Maven είναι μια εφαρμογή που στηρίζεται στην Java για να λειτουργήσει σωστά. Οπότε σαν πρώτο βήμα θα πρέπει να εγκαταστήσουμε το Java Development Kit (JDK) και να το δηλώσουμε σωστά στο λειτουργικό μας σύστημα. Το κεντρικό αποθετήριο βιβλιοθηκών για το Maven.

Προσφέρεται και συντηρείται από την κοινότητα Maven. Περιλαμβάνει μεγάλη γκάμα βιβλιοθηκών που χρησιμοποιούνται ευρέως σε projects, για προσαρμογή και επέκταση της λειτουργικότητας του build tool. Οι βιβλιοθήκες είναι στοιχεία ανοιχτού κώδικα από συνεισφέροντες που κυμαίνονται από μεμονωμένους developers έως μεγάλους οργανισμούς.

2.3.2.1 Build Tools

Τα Build Tools είναι λογισμικά που αυτοματοποιούν τη δημιουργία εφαρμογών από πηγαίο κώδικα. Η διαδικασία περιλαμβάνει τη μεταγλώττιση, τη σύνδεση και την προετοιμασία του κώδικα σε εκτελέσιμη μορφή.

Οι διαδικασίες που αυτοματοποιούνται από τη χρήση των build tools περιλαμβάνουν τα εξής:

Compiling	Μεταγλώττιση πηγαίου κώδικα σε κώδικα μηχανής
Dependency management	Προσδιορισμός και επιδιόρθωση απαραίτητων ενσωματώσεων τρίτων
Automated tests	Εκτέλεση δοκιμών και αναφορά αποτυχίας
Packaging app for deployment	Προετοιμάζει τον πηγαίο κώδικα για να γίνει deploy σε Servers

2.1.2.1.1 Γιατί χρειαζόμαστε τα Build Tools

Σε μικρά projects δεν κρίνεται απαραίτητη η χρήση των build tools. Σε μεγάλα projects όμως είναι δύσκολο να υπάρχει ανά πάσα στιγμή έλεγχος της πορείας της υλοποίησης, οπότε και εργαλεία που

αυτοματοποιούν τις διαδικασίες που συνθέτουν τη δημιουργία του κώδικα κρίνονται απαραίτητα. Μερικά build tools της αγοράς φαίνονται στην εικόνα 2.2.

Εικόνα 2.2. Δημοφιλή Build Tools

2.3.2.2 Grandle Build Tool

Ακόμα ένα δημοφιλές build tool είναι το Grandle. Το Grandle είναι ένα Open source build tool με έτος κυκλοφορίας το 2008, ως διάδοχος του Maven, βασισμένο στις αρχές και τη λογική του. Υποστηρίζει projects γραμμένα σε Java, Kotlin, Groovy, Scala, C/C++, JavaScript και χρησιμοποιείται σε multi-language ανάπτυξη λογισμικού. Εισήγαγε και χρησιμοποιεί για το configuration την domain-specific language (DSL), βασισμένη στις γλώσσες προγραμματισμού Groovy και Kotlin. Υποστηρίζει τα Maven και Ivy repositories για τη δήλωση project configurations και σχεδιάστηκε με γνώμονα τις multi-project κατασκευές.

2.3.2.3 Maven vs Gradle (Ομοιότητες)

- Δωρέαν λογισμικά ανοιχτού κώδικα, διανέμονται υπό την άδεια Apache 2.0
- Εξαιρετικά προσαρμόσιμα, υποστηρίζονται από διάφορα Java IDE
- Χρησιμοποιούν το GAV coordinate standard (groupId, artifactId, version) για τη διαχείριση των
- artifacts
- Τα plugins προσθέτουν λειτουργικότητα, συμπεριλαβανομένης της προσθήκης εργασιών και διαμορφώσεων εξάρτησης σε έργα
- Ίδια δομή καταλόγου
- Επιλύουν εξαρτήσεις από configurable repositories

2.3.2.4 Maven vs Gradle

- Maven: Βασισμένο σε script γλώσσα (XML) ≠ Gradle: Βασισμένο σε γλώσσα προγραμματισμού (Groovy)
- Το build script του Gradle είναι πιο ευέλικτο και ισχυρό από αυτό του Maven, αλλά πιο ευάλωτο σε σφάλματα, λόγω της φύσης του
- Το Gradle εφαρμόζει στρατηγικές για αυξημένη ταχύτητα απόδοσης (δημιουργία κρυφής μνήμης, incremental compilations κλπ.) θεωρητικά το Gradle τρέχει αρκετές φορές πιο γρήγορα από το Maven, στην πράξη υπάρχουν χρήστες που υποστηρίζουν το αντίθετο
- Το αρχείο προσαρμογής του Gradle προσφέρει περισσότερη ευελιξία από αυτό του Maven
- Το Maven διαθέτει περισσότερες προσθήκες από το Gradle, διότι τεχνολογικά είναι αρκετά παλαιότερο και οι προμηθευτές υποστηρίζουν περισσότερο το Maven σε σχέση με το Gradle
- Ως προς την επίλυση διενέξεων μεταξύ εξαρτήσεων, το Maven ακολουθεί μια σειρά δηλώσεων, ενώ το Gradle αναφέρεται σε δέντρο εξαρτήσεων

2.3.2.2 Language – Γλώσσα προγραμματισμού

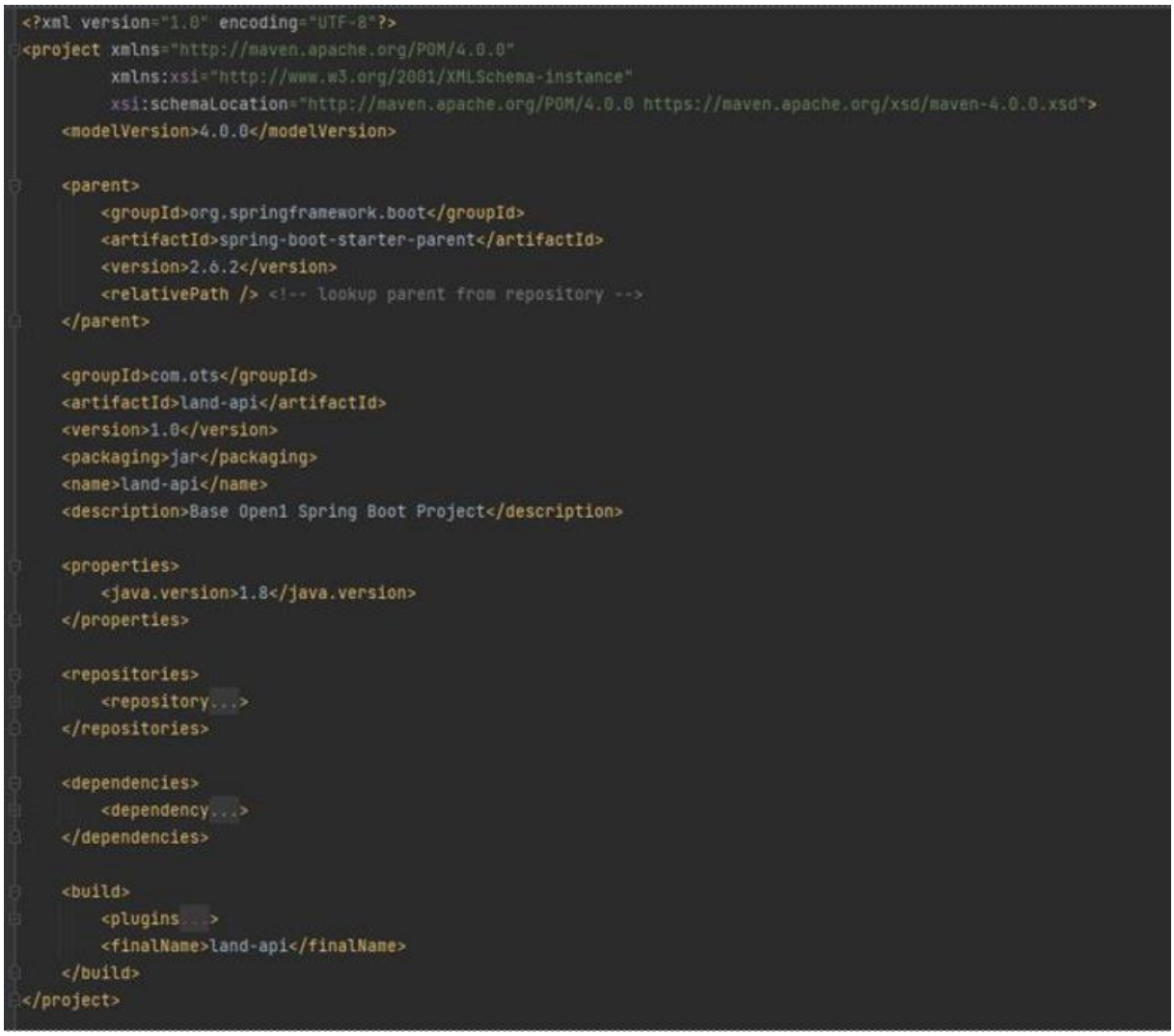
Εδώ επιλέγουμε τη γλώσσα προγραμματισμού που θα χρησιμοποιήσουμε στην εφαρμογή μας. Ανάλογα τη γλώσσα που θα επιλέξουμε, το project με τον αρχικό κώδικα είναι γραμμένο σε αυτή. Έχουμε 3 επιλογές Java, Kotlin και Groovy. Εγώ έχω επιλέξει τη Java.

2.3.2.2 Spring Boot

Σε αυτό το σημείο επιλέγουμε την έκδοση της Sping Boot που θα χρησιμοποιήσουμε. Σημαντικό είναι να επιλέξουμε έκδοση που δεν είναι SNAPSHOT. Οι εκδόσεις αυτές είναι για τις ανάγκες testιng και ανάπτυξης λογισμικού που σε καμία περίπτωση δεν πρέπει να επιλεχθεί για Enterprise ανάπτυξη καθώς δεν είναι σταθερές και έχουν κενά ασφαλείας.

Η έκδοση που δημιουργήθηκε το project είναι η 2.5.3 και μπορούμε να το δούμε πολύ εύκολα από το αρχείο pom.xml. Μέσα σε αυτό το αρχείο, βρίσκονται (ή θα προσθέτουμε) όλες τις απαραίτητες πληροφορίες που χρειάζεται το project μας για να δημιουργηθεί συμπεριλαμβανομένων και των εκτός java βιβλιοθηκών.

Το **pom.xml** ορίζεται από το **project** element () το οποίο ορίζει και το xml schema επάνω στο οποίο θα στηριχτεί η δομή του pom.xml (**xsi:schemaLocation**). Αυτό σημαίνει ότι όλα τα XML tags που θα γράψουμε μέσα στο POM.xml(εικόνα 2.3) πρέπει να είναι δομημένα σωστά - συντακτικά και με την σωστή σειρά.



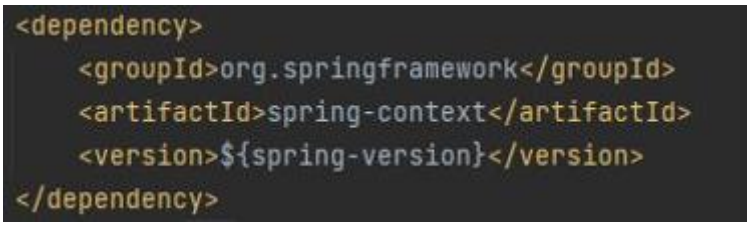
Εικόνα 2.3 POM.xml

2.3.2.3 Project Metadata

Εδώ συμπληρώνουμε κάποια στοιχεία όπως το όνομα του project, σε τι μορφή θα είναι το packaging δηλαδή το εκτελέσιμο που θα βγάλουμε αν είναι .jar ή war[2] και την έκδοση της Java που θα χρησιμοποιήσουμε. Το project υλοποιήθηκε με .jar packaging και έκδοση java 11.

2.3.2.4 Dependencies

Έδω μπορούμε να εισάγωγουμε τις βιβλιοθήκες με τις οποίες θα εξάγουμε το αρχικό project. Οστόσο αυτό μπορούμε να το κάνουμε και στη συνέχεια κατά την υλοποίηση της εφαρμογής.



Εικόνα 2.4 Depedencies

2.3.3 Γιατί Spring

Τα πλεονεκτήματα που μας προσφέρει το Spring Framework είναι τα παρακάτω:

- Μας επιτρέπει να αναπτύξουμε enterprise εφαρμογές χρησιμοποιώντας POJOs. Το πλεονέκτημα της χρησιμοποίησης μόνο POJOs είναι ότι δεν χρειάζομαστε ένα EJB container, όπως ένα application server, αλλά έχουμε την επιλογή, να χρησιμοποιήσουμε ένα servlet Container όπως Tomcat.
- Τα **IoC containers τείνουν να είναι ελαφριά**, ειδικά σε σύγκριση με τα EJB containers. Αυτό είναι χρήσιμο, για την ανάπτυξη εφαρμογών, που θα φιλεξονηθούν 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.
- Το **testing** μιας εφαρμογής, γραμμένη σε Spring είναι **απλό,** γιατί ο εξαρτημένος από το περιβάλλον κώδικας, είναι βρίσκεται 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 ένα καλοσχεδιασμένο MVC Framework, το οποίο αποτελεί μια εξαιρετική εναλλακτική αντί για web frameworks όπως το Strutts.

2.3.4 Διαφορές μεταξύ Java και Spring

Java	Spring
Η Java είναι μία από τις διαδεδομένες γλώσσες προγραμματισμού στην αγορά.	Η Spring είναι ένα βασισμένο σε Java framework εφαρμογών ανοικτού κώδικα.
Η Java παρέχει μια πλήρως ενημερωμένη δομή Enterprise Application Framework που ονομάζεται Java EE για την ανάπτυξη εφαρμογών ιστού.	Το Spring Framework περιλαμβάνει διάφορες ενότητες όπως Spring MVC, Spring Boot, Spring Security, οι οποίες παρέχουν διάφορα έτοιμα προς χρήση εργαλεία για την ανάπτυξη εφαρμογών ιστού.
Η Java EE βασίζεται σε ένα τρισδιάστατο αρχιτεκτονικό πλαίσιο, το οποίο αποτελείται από τα Logical Tiers, τα Client Tiers και τα Presentation Tiers.	Η Spring βασίζεται σε μια πολυεπίπεδη αρχιτεκτονική που αποτελείται από διάφορες ενότητες που χτίζονται πάνω στον πυρήνα του container της.

2.4 Angular 2 Framework

Για την υλοποίηση του view της εφαρμογής, δηλαδή το περιβάλλον που θα βλέπει ο απλός χρήστης και θα αλληλεπιδρά με το back end της εφαρμογής χρησιμοποιήθηκε το framework **Angular 2**. Η Angular 2 είναι ένα **JavaScript Framework** για τη δημιουργία διαδραστικών client side εφαρμογών που τρέχουν στον browser και κυκλοφόρησε επισήμως στα τέλη του 2016 δημιουργόντας μεγάλο ενθουσιασμό στα communities των developers.

2.4.1 Δυνατότητες του Framework

Το Web έχει αλλάξει δραματικά τα τελευταία πέντε χρόνια. Με την **ECMAScript (ES) 2015**, έχουμε **mudules**, **κλάσεις** και **arrow functions**. Η Angular υλοποιεί αυτές τις δυνατότητες και βοηθάει στο να τα κάνει πιο εύκολο να τα χρησιμοποιήσουμε. Αν προσθέσουμε και την TypeScript, έχουμε πολύ σημαντικές δυνατότητες όπως έλεγχο, ασφάλεια και δυναμικότητα.

Έτσι η παραγωγικότητα βρίσκεται σε πολύ υψηλό επίπεδο καθώς είναι για εμάς πιο εύκολο να κάνουμε debug αλλά και να καταλάβουμε τον κώδικά μας.

Ένα από τα σημαντικότερα πράγματα που μπορούν να αυξήσουν την παραγωγικότητα είναι τα **συνεπή coding patterns**, και η Angular το κάνει πολύ καλά

Για παράδειγμα, η Angular 2 χρησιμοποιεί τα ES classes και ένα set από annotations που ονομάζονται decorators ώστε να χτιστεί η λειτουργικότητα. Άρα, για να δημιουργήσουμε ένα Angular **component** χρειαζόμαστε μία κλάση και τον κατάλληλο decorator. Με την ίδια λογική αν θέλουμε να δημιουργήσουμε ένα **module**, χρειαζόμαστε μία κλάση και έναν decorator. Το ίδιο για να φτιάξουμε **φίλτρα**, **υπηρεσίες** κ.λ.π.

Ένα ακόμα παράδειγμα συνεπών coding patterns είναι η δύναμη του **dependency injection (DI)**. Για τη χρησιμοποίηση μιας υπηρεσίας όπως π.χ. HTTP ή Router, απλά κάνουμε inject την υπηρεσία σε κάποια κλάση που τη χρειάζεται. Η Angular κάνει το Implement του constructor-based dependency injection, άρα για να κάνουμε inject μία υπηρεσία απλά περνάμε την υπηρεσία αυτή σαν argument στον constructor της κλάσης

Επιπλέον, η Angular χρησιμοποιεί τα γνωστά μας pattern αρχιτεκτονικής κώδικα, και κυρίως το **model-view-view/model ή αλλιώς το MVVM pattern.**

Πολλές εφαρμογές client side λειτουργούν με δεδομένα. Η εφαρμογή "τραβάει" τα δεδομένα από έναν back-end server και τα παρουσιάζει χρησιμοποιώντας ένα **view template.** Και αν ο χρήστης κάνει Edit αυτά τα δεδομένα, τότε οι αλλαγές θα πρέπει να σταλθούν πίσω στον back-end server προς αποθήκευση.

Ο κώδικας για αυτές τις διαδικασίες μπορεί να είναι πολύπλοκος και πολύπλευρος. Με το **data binding** της Angular η διαδικασία είναι αρκετά εύκολη. Απλά κάντε bind τα HTML elements του template στα properties του μοντέλου στην κλάση και τα δεδομένα **αυτόματα** θα εμφανίζονται στην view. Και αν ο χρήστης τα αλλάξει, η Angular υποστηρίζει το λεγόμενο **two-way data binding,** οπότε αυτά αυτόματα γίνονται update στην κλάση.

Επιπλέον του data binding, η Angular υποστηρίζει και **property binding.** Αυτό επιτρέπει τον έλεγχο του DOM με το binding HTML properties με τα component class properties. Για παράδειγμα, μπορούμε

να κάνουμε bind μία εικόνα σε ένα component class property, π.χ. το hideImage. Όταν hideImage είναι true, η εικόνα είναι αυτόματα invisible. Μόλις το hideImage γίνει false, αυτόματα η εικόνα εμφανίζεται.

Τέλος, η Angular υποστηρίζει και **event binding**, που σημαίνει ότι μπορεί να αντιδράσει σε κάποιο event που συμβαίνει στην view, όπως ένα κλικ, ένα event από ένα third party component ή τα δικά μας custom events. Απλά κάνουμε bind τα event σε μία method στην κλάση του component. Όταν συμβαίνει το event, η method καλείται.

Μία web εφαρμογή συνήθως δεν στηρίζεται σε μία μόνο view. Οι περισσότερες θα έχουν **διάφορες views**, όπως π.χ. την εισαγωγική, μία για listing, μία για details, μία για edit κ.λ.π. Πρέπει να εμφανίσουμε την κατάλληλη view κάθε φορά. Γι αυτό χρειαζόμαστε το routing.
Η Angular παρέχει πλήρως λειτουργικό routing. Ορίζουμε το route σε κάθε σελίδα της εφαρμογής. Έπειτα ενεργοποιούμε το κατάλληλο route βασιζόμενοι στις ενέργειες του χρήστη. Μπορούμε να περάσουμε δεδομένα στα routes, ώστε να ξέρουμε ότι π.χ. το details view θα παρουσιάζει το αντικείμενο με id 15.

Επίσης μπορούμε να θέσουμε **permissions** για τους χρήστες και τα ποιά views θα μπορούν αν δουν με κωδικό ή χωρίς. Μπορούμε να αποτρέψουμε το κλείσιμο μιας σελίδας που βρίσκεται σε κατάσταση edit data μέχρι ο χρήστης να το επιλέξει.
Τέλος, υπάρχει και η υποστήριξη **lazy load routes** στην οποία δεν κατεβαίνουν στον browser όλα τα routes αν ο χρήστης δεν το ζητήσει. Για παράδειγμα, αν έχουμε σε μία εφαρμογή admin διαχειριστικό περιβάλλον, αυτό μπορεί να μην κατέβει στον browser μέχρι ο ίδιος ο χρήστης να το ζητήσει.

Το μέγεθος και οι αποδόσεις συνδέονται στις web εφαρμογές. Ένα μικρότερο component χρειάζεται μικρότερο χρόνο για να κατέβει στον browser. Ένας από τους στόχους της Angular είναι να **μειώσει το μέγεθος και να αυξήσει την απόδοση.**
Η μείωση του μεγέθους πετυχαίνεται με διάφορους τρόπους. Αρχικά μπορούμε να κάνουμε το μέγεθος του κάθε component όσο πιο μικρό γίνεται. Έπειτα, πρέπει να οργανώσουμε τα components σε modules με τέτοιο τρόπο ώστε να μπορούν να γίνουν download μαζί. Και τέλος, μπορούμε να χρησιμοποιήσουμε το **lazy loading** των routes που αναφέραμε παραπάνω.

Ακόμα θα πρέπει να αναφέρουμε και το **Ahead of Time (AoT) compiler.** Με τη χρήση του AoT, ο compiler τρέχει μια φορά κατά τη διάρκεια του build. Ο browser κατεβάζει την pre-compiled version της εφαρμογής και την κάνει render αμεσα, χωρίς να χρειάζεται να την κάνει compile πρώτα στον browser.

Έτσι η Angular εκτελείται γρήγορα, έχει λιγότερη δουλειά και χαμηλότερο memory footprint.

2.4.2 PrimeNG User Interface Suite

Το User Interface αναπτύχθηκε πάνω σε ένα αρχικό demo template της primeng και όλα τα στοιχεία που χρησιμοποιήθηκαν(components) είναι αυτής της σουίτας.

Το PrimeNG είναι μια συλλογή από UI components για το framework Angular. Όλα τα widgets είναι ανοιχτού κώδικα και ελεύθερα προς χρήση υπό την άδεια MIT License. Το PrimeNG αναπτύσσεται από την PrimeTek Informatics, εταιρεία με πολυετή εμπειρία στην ανάπτυξη λύσεων UI ανοιχτού κώδικα.

Τα components που διαθέτει το PrimeNG καλύπτουν σχεδόν το 100% των σχεδιαστικών και λειτουργικών αναγκών που μπορεί να προκύψουν. Στην εφαρμογή χρησιμοποιήθηκαν πολλά από αυτά. Ένα σημαντικό πλεονέκτημα των component είναι ότι προσφέρει και συγκεκριμένες λειρουργίες που είναι έτοιμες και γλυτώνει από τον προγραμματιστή να τις υλοποιήσει από την αρχή. Υπαρχει ένα τεράστιο πλήθος από components που μπορούμε να χρησιμοποιήσουμε (εδώ: https://www.primefaces.org/primeng/showcase/#/), προσέχοντας φυσικά την έκδοση του primeng που έχουμε εγκαταστήσει στο angular project μας.

Βιβλιογραφικές αναφορές

Internet Site

[1] https://kassapoglou.github.io/spring/spring-unit1-introduction-to-spring-framework.html

[2] https://www.baeldung.com/java-jar-war-packaging

[3] https://kassapoglou.github.io/maven/maven.html

Επίλογος

Στο κεφάλαιο αυτό αναφέρονται τα frameworks που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Συγεκριμένα πως δημιουργούμε ένα spring container, που είναι η βάση για να ξεκινήσουμε το back end κώδικά μας και τι μπορούμε να επιλέξουμε από τις επιλογές που μας δίνονται. Επίσης γίνετε λόγος των δυνατοτήτων που μας προσφέρει το framework της Angular με το οποίο δημιουργήθηκε το UI της εφαρμογής.

Κεφάλαιο 3ο: Ανάλυση απαιτήσεων

Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει η βασική ανάλυση της εφαρμογής και θα γίνει μια αρχική σχεδίαση για τα βήματα που πρέπει να γίνουν για την υλοποίηση της εφαρμογής. Θα γίνει αναφορά των διαγραμμάτων που χρειάζεται να γίνουν πάνω στα οποία θα στηριχτούμε για να υλοποιήσουμε την εφαρμογή καθώς και τα σενάρια ελέγχου, βάζοντας κάποια όρια, ώστε να μη ξεφύγουμε από αυτό που θέλει πραγματικά ο τελικός χρήστης. Είναι σημαντικό το αποτέλεσμα του έργου μας, να μην έχει μεγάλη απόκλιση από που ζητήθηκε και συζητήθηκε με τον πελάτη.

3.1 Μοντέλο Οντοτήτων – Συσχετίσεων

Το μοντέλο οντοτήτων-συσχετίσεων (μοντέλο Ο/Σ - ER model) είναι ένα αφαιρετικό ιδεατό μοντέλο δεδομένων, τα οποία έχουν καθορισμένη δομή[1]. Στη μηχανική λογισμικού χρησιμοποιείται για να παρέχει ένα εννοιολογικό σχήμα κατά τη σχεδίαση βάσεων δεδομένων, ως μοντέλο δεδομένων ενός συστήματος και των απαιτήσεών του με top-down προσέγγιση. Ένα διάγραμμα που δημιουργείται με αυτή τη διαδικασία σχεδίασης καλείται διάγραμμα οντοτήτων-συσχετίσεων, ή διάγραμμα Ο/Σ ή ΟΣΔ εν συντομία. Προτάθηκε αρχικά το 1976 από τον Peter Chen, ωστόσο στη συνέχεια επινοήθηκαν πολλές παραλλαγές της διαδικασίας.

Χρησιμοποιείται στο πρώτο στάδιο σχεδίασης ενός συστήματος πληροφοριών, κατά την ανάλυση των απαιτήσεών του. Σκοπός του είναι να περιγράφει τις αναγκαίες πληροφορίες οι οποίες πρόκειται να αποθηκευτούν στη βάση δεδομένων ή τον τύπο τους. Η μοντελοποίηση δεδομένων γίνεται για την περιγραφή των χρησιμοποιούμενων όρων και των σχέσεών τους σε έναν ορισμένο τομέα ενδιαφέροντος. Στην περίπτωση σχεδιασμού ενός συστήματος πληροφοριών, που στηρίζεται σε μια βάση δεδομένων, το εννοιολογικό μοντέλο δεδομένων χαρτογραφείται σε προχωρημένο στάδιο σε ένα λογικό μοντέλο δεδομένων, όπως το σχεσιακό μοντέλο δεδομένων.[4] Το στάδιο αυτό ονομάζεται συνήθως στάδιο λογικού σχεδιασμού. Ύστερα, κατά τη διάρκεια του φυσικού σχεδιασμού το λογικό μοντέλο χαρτογραφείται σε κάποιο φυσικό μοντέλο. Ορισμένες φορές και οι δύο φάσεις αναφέρονται ως "φυσικός σχεδιασμός".

Υπάρχουν διάφορες συμβάσεις για τη σημειογραφία που χρησιμοποιείται στα διαγράμματα οντότητων-συσχετίσεων (ERDs). Αρκετές φορές η σημειογραφία που υιοθετείται κατά το λογικό και φυσικό σχεδιασμό μιας βάσης δεδομένων διαφέρει ως προς τη σαφήνεια, τις δυνατότητες που έχει η γραφική γλώσσα, την υποστήριξη από πρότυπα και τα εργαλεία. Συνήθως για τις συσχετίσεις χρησιμοποιούνται ευθείες με διαφορετικές άκρες, ή με σημειώσεις που παριστάνουν την πληθικότητα του τύπου συσχέτισης.

Ιδιαίτερη - κυρίως ιστορική - σημασία έχουν οι παρακάτω σημειογραφίες:

- Η **σημειογραφία Chen**, του Peter Chen που δημιούργησε τα διαγράμματα το 1976.
- Η **σημειογραφία IDEF1X** ως de facto πρότυπο, χρησιμοποιούμενο για χρόνια από τις αρχές των ΗΠΑ.
- Η **σημειογραφία Martin** (πόδι του κόρακα) διαδεδομένη σε εργαλεία για διαγράμματα.
- Η **σημειογραφία του Charles Bachman**, σε εργαλεία διαγραμμάτων.
- Η **σημειογραφία (Min, max)**, του Jean Raymond Abrial το 1974.

- Η **γλώσσα UML**, πρότυπο που χρησιμοποιείται ως αντικαταστάτης των διαγραμμάτων Ο/Σ.

Βάση για των μοντέλων Ο/Σ είναι η κατηγοριοποίηση αντικειμένων και των σχέσεών τους μεταξύ τους.

3.1.1 Οντότητα

Οντότητα (entity) είναι ένα αντικείμενο ενδιαφέροντος στον πραγματικό κόσμο το οποίο ξεχωρίζει από τα υπόλοιπα. Μια οντότητα λειτουργεί αφαιρετικά σε έναν πολύπλοκο τομέα. Οντότητες μπορεί να είναι άνθρωποι, μέρη, αντικείμενα, γεγονότα, έννοιες κλπ. Στιγμιότυπο (instance) μιας οντότητας είναι μια συγκεκριμένη περίπτωση ενός τύπου οντότητας.

3.1.2 Χαρακτηριστικό

Κάθε οντότητα έχει διάφορα στοιχεία που την προσδιορίζουν. Ένα τέτοιο στοιχείο ονομάζεται ιδιότητα (attribute), χαρακτηριστικό ή πεδίο της οντότητας. Τα χαρακτηριστικά χωρίζονται σε μονότιμα (single valued), τα οποία έχουν μόνο μια τιμή και πλειότιμα (multi-valued), τα οποία έχουν σύνολο από τιμές.

Στο διάγραμμα Ο/Σ οι ιδιότητες που έχει μια οντότητα παριστάνονται μέσα σε έλλειψη, με υπογραμμισμένο το πρωτεύον κλειδί. Τα πλειότιμα χαρακτηριστικά μιας οντότητας παριστάνονται μέσα σε έλλειψη με διπλό περίγραμμα.

3.1.3 Συσχέτιση

Συσχέτιση (relationship) είναι η σύνδεση δύο ή περισσότερων τύπων οντοτήτων που παρουσιάζει ενδιαφέρον για σχεδιασμό. Με συσχετίσεις μπορούν να συνδέονται και χαρακτηριστικά οντοτήτων. Ένας τύπος συσχέτισης (σύνολο συσχετίσεων) παριστάνεται με ρόμβο. Στο εσωτερικό αναγράφεται το όνομα με μικρά γράμματα.Υποδεικνύουμε τα όρια της συσχέτισης με ένα δείκτη.

Ως όρια μπούμε να συναντήσουμε:

- **0 έως άπειρο** (κατώτατο όριο 0, ανώτατο όριο άπειρο)
- **τουλάχιστον 1** (κατώτατο όριο 1, ανώτατο όριο άπειρο)
- **ακριβώς 1** (κατώτατο όριο 1, ανώτατο όριο 1)
- **το πολύ 1** (κατώτατο όριο 0, ανώτατο όριο 1)

3.1.4 Βαθμός ή πολυπλοκότητα ενός τύπου συσχετίσεων

Ο βαθμός μιας συσχέτισης είναι ο αριθμός των τύπων οντοτήτων που παίρνουν μέρος στη συσχέτιση. Οι πιο συνηθισμένες συσχετίσεις είναι

- μοναδικές, ο βαθμός τους τότε είναι 1
- δυαδικές, ο βαθμός τους τότε είναι 2
- τριαδικές, ο βαθμός τους τότε είναι 3

3.1.5 Πληθικός λόγος

Ο πληθικός λόγος, περιγράφει τον αριθμό στιγμιοτύπων ενός τύπου οντοτήτων που μπορούν να αντιστοιχίζονται με μία οντότητα ενός άλλου τύπου σε μια συσχέτιση.

Ο λόγος πληθικότητας ή πληθικός λόγος (cardinality ratio), είναι ο λόγος των πληθικοτήτων μιας συσχέτισης.

Μπορούμε να έχουμε συσχετίσεις με λόγο πληθικότητας:

- 1-1(ένα-προς-ένα) Αντιστοιχίζεται μια οντότητα ενός τύπου με το πολύ ή ακριβώς μια οντότητα ενός άλλου τύπου.
- 1-N (ένα-προς-πολλά) Αντιστοιχίζεται μια οντότητα ενός τύπου με κανένα, ένα ή πολλά στιγμιότυπα ενός άλλου τύπου.
- M-N (πολλά-προς-πολλά) Αντιστοιχίζεται κάθε στιγμιότυπο του ενός τύπου με ένα, κανένα ή πολλά στιγμιότυπα του άλλου τύπου.

3.1.6 Ασθενής Τύπος Οντότητας

Αδύναμη ή ασθενής οντότητα λέγεται μια οντότητα που εξαρτάται από την ύπαρξη κάποιας άλλης. Οι αδύναμες οντότητες συμμετέχουν σε συσχετίσεις μέσω ταυτοποιητικών συσχετίσεων με ισχυρή οντότητα. Ταυτοποιητική συσχέτιση ονομάζεται η συσχέτιση στην οποία το πρωτεύον κλειδί της ισχυρής οντότητας χρησιμοποιείται ως μέρος του πρωτεύοντος κλειδιού της αδύναμης οντότητας. Διακριτικό ή μερικό κλειδί ονομάζεται το χαρακτηριστικό της αδύναμης οντότητας το οποίο μαζί με το πρωτεύον κλειδί της ισχυρής οντότητας είναι το πρωτεύον κλειδί της αδύναμης.

Κατά την αναπαράσταση αδύναμων οντοτήτων:

- η οντότητα παριστάνεται με διπλό ορθογώνιο
- ταυτοποιητική συσχέτιση με διπλό ρόμβο
- το μερικό κλειδί με διακεκομμένη γραμμή

3.1.7 Επαναλαμβανόμενες ομάδες

Μια επαναλαμβανόμενη ομάδα είναι ένα σύνολο δύο ή περισσότερων πλειότιμων γνωρισμάτων που έχουν μια λογική σχέση μεταξύ τους.

3.1.8 Πλειότιμα χαρακτηριστικά

Είδαμε ότι τα πλειότιμα χαρακτηριστικά παριστάνονται με μια διπλή έλλειψη. Ωστόσο, στην **ιδανική** περίπτωση πλειότιμα χαρακτηριστικά πρέπει να αφαιρούνται από το μοντέλο δεδομένων κατά τη φάση σχεδίασης. Με τον τρόπο αυτό, η σχέση βρίσκεται σε δεύτερη κανονική μορφή.
Για να επιτύχουμε κάτι τέτοιο στο μοντέλο Ο/Σ προσθέτουμε μια ακόμη συσχέτιση.

3.1.9 Υποκλάσεις και υπερκλάσεις

3.1.9.1 Γενίκευση / Εξειδίκευση

Με την έννοια **γενίκευση** (generalization) εννοούμε τον εντοπισμό ενός συνόλου οντοτήτων (κλάση) που έχουν κοινά χαρακτηριστικά με πιο γενικευμένα αντικείμενα (υπέρκλαση). Η **εξειδίκευση**

(specialization) είναι το ακριβώς αντίθετο της γενίκευσης, δηλαδή ο εντοπισμός υποσυνόλων ενός τύπου οντοτήτων με κοινά χαρακτηριστικά, τα οποία τα διαφοροποιούν από τα υπόλοιπα μέλη του.
Η συσχέτιση μεταξύ κάθε υπόκλασης και υπέρκλασης ονομάζεται ISA συσχέτιση.

3.1.10 Κληρονομικότητα

Σε κάθε επίπεδο της ιεραρχίας οι τύποι οντοτήτων κληρονομούν τα χαρακτηριστικά των τύπων του αμέσως ανώτερου επιπέδου.

3.1.11 Περιορισμός Επικάλυψης

Όταν υπάρχει περιορισμός επικάλυψης μια οντότητα δεν επιτρέπεται να ανήκει ταυτόχρονα σε δύο υποκλάσεις. (exclusive subtypes). Ο περιορισμός επικάλυψης συμβολίζεται με μια καμπύλη γραμμή στο διάγραμμα Ο/Σ, που τέμνει την ακμή του τύπου οντοτήτων με κάθε ISA συσχέτιση. Υπάρχει όμως περίπτωση να μην ισχύει κανένας περιορισμός. Στην περίπτωση αυτή ένα στιγμιότυπο μπορεί να ανήκει σε περισσότερες από μια υποκλάσεις. (non-exclusive subtypes).

3.1.12 Η έννοια του κλειδιού κλειδιού

Ένα χαρακτηριστικό (ή σύνολο χαρακτηριστικών) ενός τύπου οντοτήτων / συσχετίσεων για τον οποίο κάθε οντότητα / συσχέτιση στο σύνολο πρέπει να έχει μοναδική τιμή ονομάζεται κλειδί (key).

- **Υποψήφιο κλειδί** (candidate key) ονομάζεται ένα ελάχιστο (minimal) κλειδί (δηλαδή, κανένα υποσύνολο των χαρακτηριστικών του δεν είναι κλειδί)
- **Το κύριο κλειδί** (primary key) είναι ένα από τα υποψήφια κλειδιά που ορίζεται σαν αναγνωριστής (identifier) για τον τύπο οντοτήτων / συσχετίσεων

3.1.13 Εργαλεία ελεύθερου λογισμικού για διαγράμματα Ο/Σ

Εργαλεία που δέχονται SQL και μπορούν να παράγουν μοντέλα Ο/Σ ή και να κάνουν ανάλυση σε βάσεις δεδομένων.

BrModelo, DBDesigner-Fork, ER2SQL, Ferret, Gliffy, ModelRight, Mogwai ERDesigner NG, MySQL Workbench, Open System Architect, Power*Architect, StarUML

Τα παρακάτω εργαλεία δεν δημιουργούν διαγράμματα Ο/Σ, αλλά απλώς σχεδιάζουν τα σχήματα χωρίς να αντιλαμβάνονται τη σημασία τους ή να παράγουν SQL.

Kivio, Dia

3.2 Μετατροπή Μετατροπή σχήματος σχήματος E-R σε σχεσιακό σχεσιακό σχήμα

- Τα πρωτεύοντα κλειδιά επιτρέπουν να εκφραστούν τα σύνολα οντοτήτων και τα σύνολα συσχετίσεων ως πίνακες που αναπαριστούν τα περιεχόμενα μιας βάσης δεδομένων.
- Μια βάση δεδομένων συμβατή με ένα διάγραμμα E-R μπορεί να αναπαρασταθεί με μια συλλογή πινάκων.
- Η μετατροπή ενός διαγράμματος E-R σε συλλογή πινάκων αποτελεί την αφετηρία για να προκύψει μια σχεσιακή βάση δεδομένων από μια εννοιολογική σχεδίαση στο μοντέλο E-R.

3.3 Σενάρια Ελέγχου

Μόλις ολοκληρωθεί ο σχεδιασμό των απαιτήσεων της εφαρμογής, συνέχεια έχει η δημιουργία των σεναρίων ελέγχου[2]. Τα σενάρια ελέγχου δημιουργούνται πριν την υλοποίηση της εφαρμογής και περιλαμβάνει σε απλή μορφή τα βήματα που πρέπει να γλινουν για να ελεγχούν οι λειτουργίες που θα υλοποίηθουν στον κώδικα. Τα σενάρια που θα δημιουργηθούν πρέπει να καλύπτουν τουλάχιστον όλες τις λειτουργίες που συζητήθηκαν με τον πελάτη, αλλά και άλλων λειτουργιών που μπορεί να προκύψουν στη συνέχεια. Τα σενάρια περιλαμβάνουν κάποιες γενικές πληροφορίες, προαπαιτούμενες ενέργειες-εξαρτήσεις, την εκτέλεση του σεναρίου ελέγχου και το αποτέλεσμα.

1.4.7.1 **Γενικές πληροφορίες**

Περιλαμβάνεται ο κωδικός του σεναρίου ελέγχου, με τη βοήθεια του οποίου ομαδοποιούμε τις λειτουργίες για να μπορέσουμε να ανατρέξουμε σε αυτές πιο γρήγορα όταν χρειαστεί. Την περιγραφή που αναφέρει τη λειτουργία για την οποία γράφονται τα σενάρια.

1.4.7.2 **Προαπαιτούμενα-Εξαρτήσεις**

Εδώ αναφέρονται οι απαιτήσεις του συστήματος πριν την εκτέλεση του σεναρίου ελέγχου. Αναφέρονται οι ενέργειες που πρέπει να κάνει ο χρήστης που θα κάνει τον έλεγχο του σεναρίου. Κάποια από τις προαπαιτούμενες ενέργειες θα μπορούσε να είναι η αυθεντικοποίηση του χρήστη στο σύστημα, η κατοχή των απαραίτητων δικαιωμάτων ή ακόμα και να πρέπει να υπάρχουν δεδομένα σε μία άλλη οντότητα προκειμένου να πάμε στην παρούσα.

1.4.7.3 **Εκτέλεση Σεναρίου**

Στην εκτέλεση του σεναρίου αναγράφονται τα βήματα που πρέπει να ακολουθήσει αυτός που θα κάνει τον έλεγχο κατά γράμμα για επιτύχει το αποτέλεσμα που θα αναφερθεί παρακάτω. Είναι ίσως το πιο βασικό τμήμα ενός σεναρίου αφού εδώ θα φανεί αν αποτύχει ή όχι η λειτουργία που γίνετε ο έλεγχος. Αν αποτύχει είναι φυσικό επακόλουθο να μην έχουμε τα αναμενώμενα αποτελέσματα για την οποία δημιουργήθηκε η λειτουργία και ο χρήστης να μην μείνει ευχαριστημένος. Έτσι το σενάριο παραμένει ανοιχτό μέχρι να γίνει η διόρθωση από μεριάς του κώδικα και να ελεγχει ξάνα.

1.4.7.4 **Αποτελέσμα**

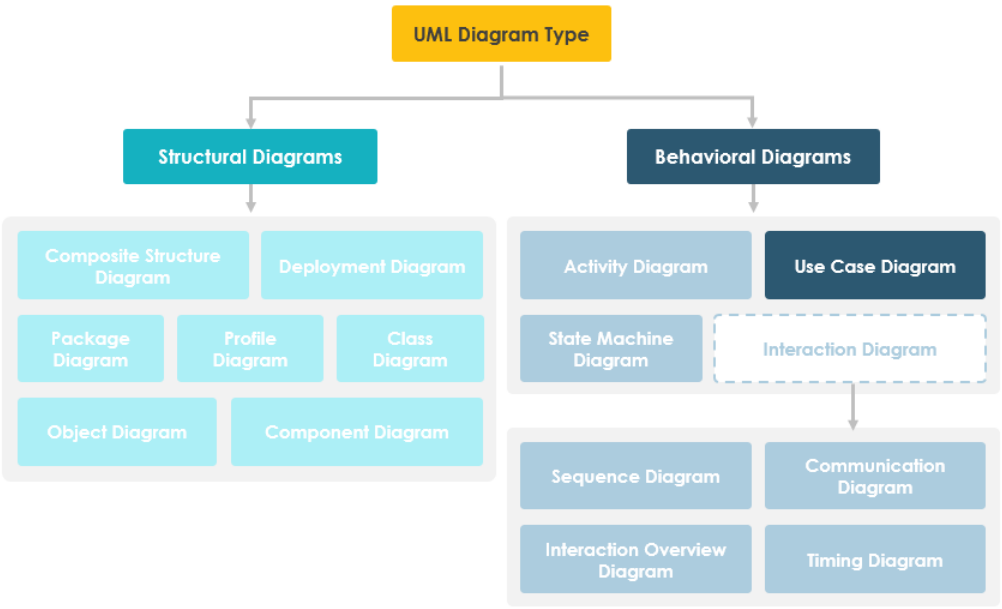
Εδώ αναφέρονται τα αναμενόμενα αποτελέσματα μετά την επιτυχή εκτέλεση των σεναρίων ελέγχου. Αυτό το τμήμα είναι πολύ βοηθητικό για αυτόν που κάνει τον έλεγχο αφού του γνωρίζει τι να περιμένει μετά την εκτέλεση των σεναρίων. Αν δεν εμφανίζεται το ίδιο αποτέλσμα με αυτό που αναφέρεται, το σενάριο θα ελεχθεί ξανά όταν διορθωθεί το κομμάτι του κώδικα που απατείται.

3.1.1 Σενάρια Χρήσης-Διάγραμμα περιπτώσεων

3.1.1.1 Τι είναι ένα διάγραμμα περιπτώσεων

Ένα διάγραμμα περιπτώσεων χρήσης UML(Εικόνα 3.1) είναι η βασική μορφή απαιτήσεων συστήματος/λογισμικού για ένα νέο υπό ανάπτυξη σύστημα λογισμικού. Οι περιπτώσεις χρήσης καθορίζουν την αναμενόμενη συμπεριφορά (τι) και όχι την ακριβή μεθοδολογία για την πραγματοποίησή της (πώς). Οι περιπτώσεις χρήσης, αφού καθοριστούν, μπορούν να αποτυπωθούν είτε με κείμενο είτε με οπτική αναπαράσταση (π.χ. διάγραμμα περιπτώσεων χρήσης). Μια βασική έννοια της δημιουργίας μοντέλων περιπτώσεων χρήσης είναι ότι μας βοηθά να σχεδιάσουμε ένα σύστημα από

την οπτική γωνία του τελικού χρήστη. Είναι μια αποτελεσματική τεχνική για την αναπαράσταση της συμπεριφοράς του συστήματος με τους κανόνες του τελικού χρήστη, προσδιορίζοντας όλη την εξωτερικά ορατή συμπεριφορά του συστήματος.



Εικόνα 3.1 Τύποι UML διαγράμματος

Ένα διάγραμμα περιπτώσεων χρήσης είναι συνήθως απλό. Δεν δείχνει τις λεπτομέρειες των περιπτώσεων χρήσης:

- Συγκεντρώνει μόνο ορισμένες από τις σχέσεις μεταξύ περιπτώσεων χρήσης, παραγόντων και συστημάτων.
- Δεν δείχνει τη σειρά με την οποία εκτελούνται τα βήματα για την επίτευξη των αποτελεσμάτων κάθε περίπτωσης χρήσης.

3.1.2 Πως προήλθαν τα σενάρια χρήσης

Η μοντελοποίηση περιπτώσεων χρήσης συνδέεται συχνά με την UML, αν και καθιερώθηκε πριν από την ύπαρξη της. Η σύντομη ιστορία της έχει ως εξής:

- Το 1986, ο Ivar Jacobson διατύπωσε για πρώτη φορά τεχνικές μοντελοποίησης κειμένου και οπτικής μοντελοποίησης για τον προσδιορισμό περιπτώσεων χρήσης.
- Το 1992, το από κοινού συγγραφικό βιβλίο Object-Oriented Software Engineering - A Use Case Driven Approach βοήθησε στη διάδοση της τεχνικής για την καταγραφή λειτουργικών απαιτήσεων, ειδικά στην ανάπτυξη λογισμικού.

3.1.3 Σκοπός

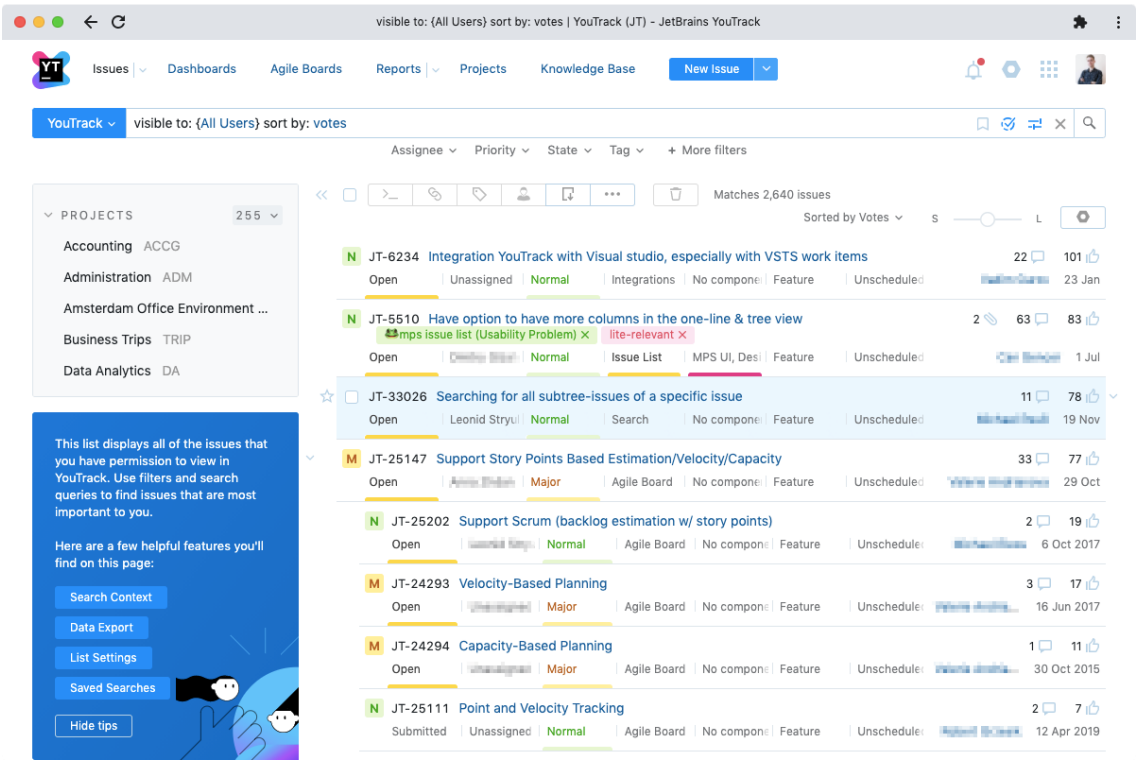
Τα διαγράμματα περιπτώσεων χρήσης αναπτύσσονται συνήθως στο αρχικό στάδιο της ανάπτυξης και εφαρμόζονται κυρίως για τους ακόλουθους σκοπούς:

- Καθορίζουν το πλαίσιο ενός συστήματος
- Καταγραφή των απαιτήσεων ενός συστήματος
- Επικυρώνουν την αρχιτεκτονική ενός συστήματος
- Προώθηση της υλοποίησης και δημιουργία περιπτώσεων ελέγχου

- Αναπτύσσονται από αναλυτές μαζί με εμπειρογνώμονες του τομέα

3.4 Εργαλείο δημιουργίας και παρακολούθησης σεναρίων ελέγχου

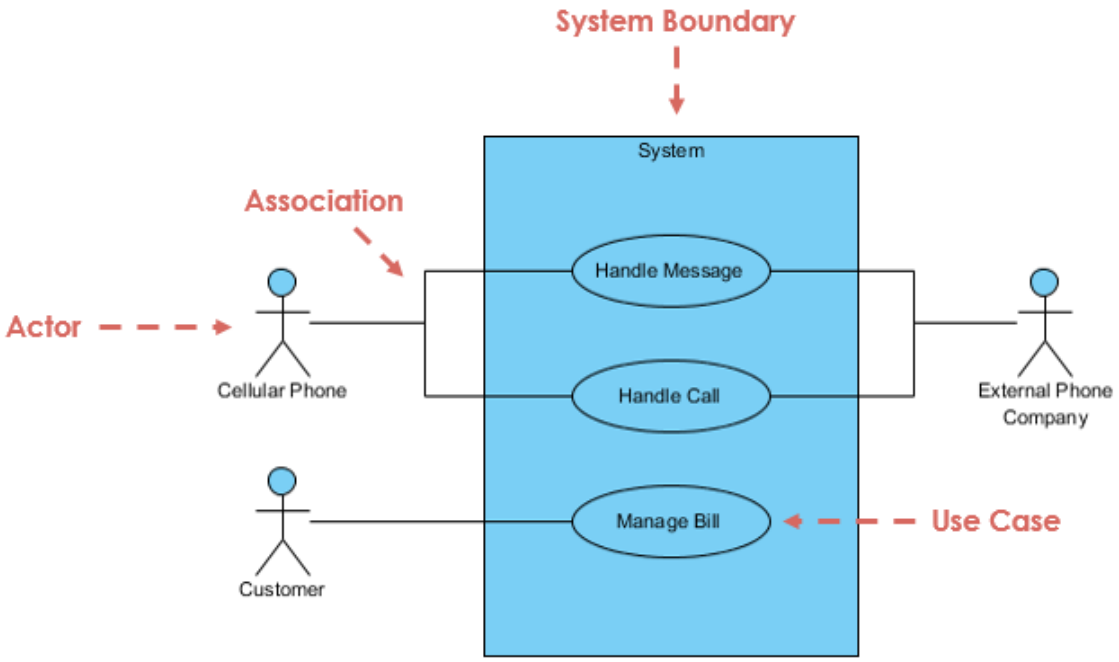
Για την πιο εύκολη δημιουργία και παρακολούθηση των σεναρίων ελέγχου και χρήσης μια εφαρμογής συχνά χρησιμοποιούνται κάποια εργαλεία, όπως είναι το Youtrack(Εικόνα 3.2). Τα εργαλεία αυτά έχουν σχεδιαστεί για να να εισάγουμε εύκολα και γρήγορα τα σενάρια χρήσης σε ένα περιβάλλον διαχείρισης, ξεχωριστά για κάθε εφαρμογή, που είναι ύπο ανάπτυξη. Παρακάτω θα γίνει αναφορά κάποιον λειτουργιών όπως αυτές γίνονται στο εργαλειο Youtrack.



Εικόνα 3.2 Περιβάλλον Youtrack

3.4.1 Εισαγωγή σεναρίων χρήσης

Η εισαγωγή των σεναρίων χρήσης(Εικόνα 3.3) στο Youtrack απαιτεί να υπάρχει το project για το οποίο αναφέρονται και να υπάρχουν τα απαραίτητα δικαιώματα του χρήστη σε αυτό. Τα σενάρια χρήσης όπως προαναφέραμε είναι πολύ απλά και αναφέρουν σε μια γραμμή τη σενάριο χρήσης κάθε λειτουργίας που συζητήθηκε από τον πελάτη. Έτσι το μόνο που χρειάζεται να γραψουμε κατά τη δημιουργία ενώς σεναρίου χρήσης είναι ένας τίτλος μία περιγραφή. Κάθε σενάριο στην συνέχεια αποτέλει ένα αίτημα για τον προγραμματιστή στον οποίο θα ανατεθεί, αν μιλάμε για περιβάλλον εταιρίας, από τον project owner.



Εικόνα 3.3 Σενάρια χρήσης

3.4.2 Σενάρια χρήσης-Αίτηματα

Όταν το αίτημα δημιουργηθεί και ανατεθεί στον προγραμματιστή προς υλοποίηση θα πρέπει στο κλείσιμο του να σχεδιαστούν σενάρια ελέγχου τα οποία θα πρέπει να είναι ίδια με αυτά που σχεδιάσαμε στην ανάλυση των απατήσεων. Το αίτημα μπαίνει σε κατάσταση Προς έλεγχο και όταν ελεγχθεί και δουλεύει με επιτυχία τότε πηγαίνουμε στο επόμενο, αλλίως κάνουμε τις απαραίτητες ενέργειες ώστε να φέρει αποτέλεσμα επιτυχίας.

Βιβλιογραφικές αναφορές

Internet Site

[1] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

https://euclid.ee.duth.gr/courses/old/2007-08/Databases/DraftSlides/LecDB02-ER-Large.pdf

[2]https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

Επίλογος

Στο κεφάλαιο αυτό αναφέρθηκαν τα αρχικά βήματα που πρέπει να γίνουν για την υλοποίηση της εφαρμογής. Έγινε ανάλυση των διαγραμμάτων που χρειάζεται να γίνουν πάνω στα οποία θα στηριχτούμε για να υλοποιήσουμε την εφαρμογή καθώς και αναπτύξουμε τα σενάρια ελέγχου γαι αυτήν.

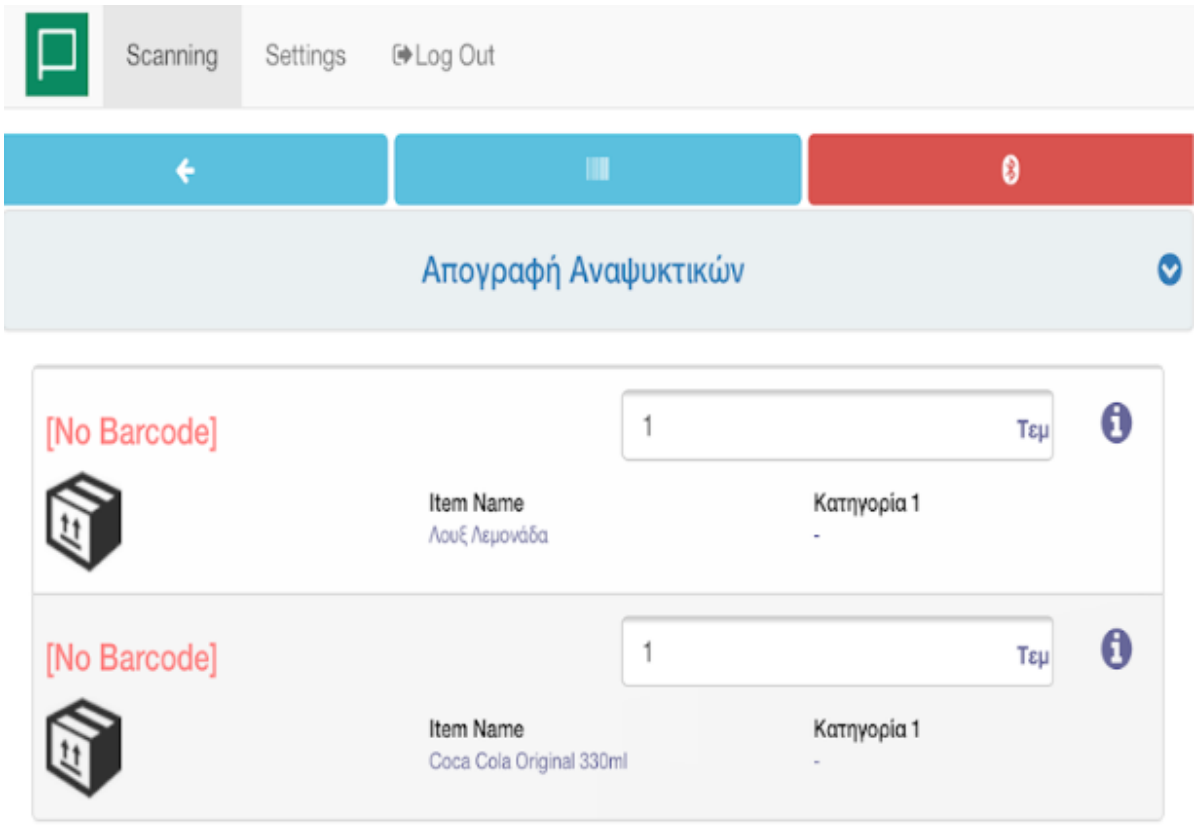
Κεφάλαιο 4ο: Υπάρχουσες εφαρμογές στην αγορά

Εισαγωγή

Στο κεφάλαιο αυτό θα γίνει αναφορά κάποιον εφαρμογών που βρίσκονται στην αγορά αυτή τη στιγμή και θα περιγραφούν κάποιες από τις πιο σημαντικές δυνατότητες που προσφέρει η κάθε μία από αυτές και πώς κατέληξα να δημιουργήσω την υπάρχουσα. Ποιες είναι οι βελτιώσεις αλλά και δυνατότητες που θα προσφέρει σε σχέση με αυτές.

4.1 SoftNet - Διαχείριση αποθήκης

Η εταιρία **SoftNet** [1] έχει δημιουργήσει την εφαρμογή **PYLON WMA (**Warehouse Management Applications) για τη διαχείριση αποθηκών και αποθεμάτων με την χρήση φορητών συσκευών με barcodes scanner και άμεση ενημέρωση της βασικής πλατφόρμας. Το λογισμικό της παρέχει την πλήρη παρακολούθηση των φυσικών αποθηκών με πραγματοποίηση Φυσικής Απογραφής(εικόνα 4.1) πολλαπλών σεναρίων, καταγραφή θέσεων αποθήκευσης και μετακίνησης των ειδών με αυτόματη δημιουργία παραστατικών, έλεγχος και διαχείριση Παραλαβών – Αποστολών (picking / packing) με καταγραφή και παρακολούθηση των όποιων διαφορών είτε σε online είτε offline mode επικοινωνία με την πλατφόρμα.



Εικόνα 4.1 Φυσική απογραφή ειδών

Οι δυνατότητες που προσφέρει είναι:

- Η εφαρμογή προσφέρει mobile διαχείριση με συμβατότητα σε android και iOS.
- Εκτέλεση φυσικής απογραφής ειδών αποθήκης με χρήση φορητών συσκευών και scanner και αποστολή των καταγραφών στην κεντρική βάση δεδομένων με στόχο την ολοκλήρωση της διαδικασίας απογραφής και των σχετικών κινήσεων, μειώνοντας τα χειριστικά λάθη και το χρόνο ολοκλήρωσης της διαδικασίας.
- Καταμέτρηση των εμπορευμάτων κατά την παραλαβή και την επιβεβαίωση της ποσότητας σύμφωνα με τα σχετικά παραστατικά. Η συγκεκριμένη λειτουργικότητα ολοκληρώνει την διαχείριση picking – packing που εκτελείται μέσω της εφαρμογής και διευκολύνει το βήμα καταμέτρησης, το οποίο εκτελείται με χρήση φορητής συσκευής, και πιστοποίησης ότι η ποσότητα που παραλαμβάνεται συμφωνεί με την ποσότητα που αναγράφεται στο συνοδευτικό παραστατικό.
- Παρακολούθηση διακίνησης ειδών σε αποθήκη με διαχείριση θέσεων αποθήκευσης. Η συγκεκριμένη λειτουργία ολοκληρώνει την λειτουργικότητα διαχείρισης θέσεων αποθήκευσης (διάδρομος, θέση, ράφι) διευκολύνοντας την παρακολούθηση των κινήσεων εξαγωγής και τοποθέτησης αποθεμάτων από και προς τα κατάλληλα σημεία με χρήση φορητών συσκευών, μειώνοντας τους χρόνους των αποθηκάριων.
- Σύνδεση με άλλα προϊόντα της σειράς PYLON.

Τα ωφέλη που προσφέρει η εφαρμογή είναι:

- Βελτιστοποίηση της διαδικασίας επιθεώρησης των εργασιών του αποθηκάριου.
- Ορθολογική διαχείριση των αποθεμάτων της επιχείρησης.
- Άμεση και έγκυρη ενημέρωση της εταιρίας για πωλήσεις, παραλαβές, αποθέματα και επιστροφές συνεπάγοντας στην ταχύτερη και ορθότερη λήψη αποφάσεων.

4.2 B.I. Αλμπάνης - Διαχείριση αποθήκης

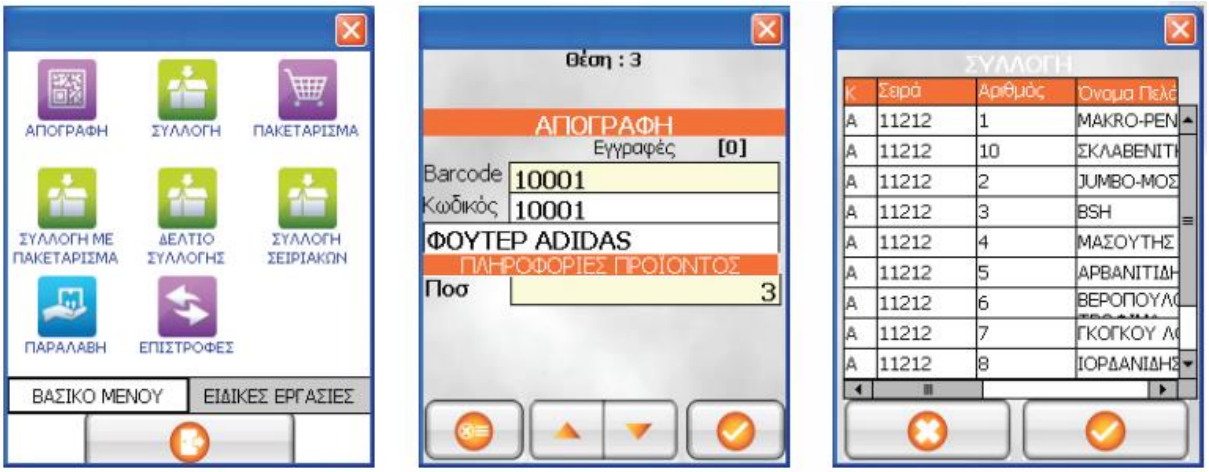
Η λύση που έχει να προτείνει η εφαρμογή της εταιρίας **Αλμπάνης** [2] στη διαχείριση της απόθηκης, ουσιαστικά μηχανογραφεί τις δραστηριότητες. Υποστηρίζει τις διαδικασίες picking, put away, απογραφής και packing(εικόνα 4.2). Ο αποθηκάριος διαθέτει φορητό τερματικό βιομηχανικών προδιαγραφών για να αντέχει στις δύσκολες συνθήκες λειτουργίας της αποθήκης και έχει πρόσβαση στα στοιχεία πελατών, προμηθευτών, παραγγελιών, αναμονής παραλαβών, παρτίδων και serial numbers.

Συνεπώς ο αποθηκάριος μπορεί εύκολα, γρήγορα και χωρίς λάθη να καταγράψει τις παραλαβές από προμηθευτές, να τοποθετήσει τα προϊόντα σε συγκεκριμένες θέσεις αποθήκευσης, να εκτελέσει παραγγελίες πελατών, να παραλάβει επιστροφές και να κάνει απογραφές. Παράλληλα η λύση δίνει την δυνατότητα κιβωτοποίησης των παραγγελιών με έκδοση ετικέτας κιβωτίου με τα στοιχεία του παραλήπτη καθώς και την εκτύπωση ετικετών με ή χωρίς barcode για τεμάχιο ή συσκευασία, παρακολούθηση παρτίδων ή σειριακών αριθμών προϊόντων ενώ συλλέγεται πλήθος στατιστικών στοιχείων για την εκτέλεση των παραγγελιών.

Μηχανογραφώντας την λειτουργία της αποθήκης εξαλείφονται τα λάθη της χειρόγραφης καταχώρησης, μειώνεται ο συνολικός χρόνος εκτέλεσης παραγγελιών, αυξάνεται το πλήθος των παραγγελιών που εκτελούνται στην μονάδα του χρόνου.

Η δυνατότητα διενέργειας συνεχών απογραφών ταυτίζει το πραγματικό απόθεμα με το λογιστικό έτσι η εταιρία αποφεύγει σχετικά θέματα που προκύπτουν από τις φορολογικές αρχές ενώ μειώνονται

δραματικά οι περιπτώσεις έλλειψης αποθέματος (under stocking) ή πλεονάζοντος αποθέματος (over stocking).



Εικόνα 4.2 Διαδικασίες της εφαρμογής

Η εφαρμογή καλύπτει ανάγκες όπως:

- Την παραλαβή προϊόντων από προμηθευτές.
- Την τοποθέτηση προϊόντων στην αποθήκη.
- Ενδοδιακίνηση προϊόντων.
- Διαχείριση επιστροφών.
- Την συλλογή παραγγελιών προς εκτέλεση.
- Απογραφή αποθήκης.
- Πακετοποίηση παραγγελιών.
- Την εκτύπωση ετικετών προϊόντων.
- Παρακολούθηση σειριακού αριθμού προϊόντων.
- Παρακολούθηση στατιστικών στοιχείων εκτέλεσης παραγγελιών / παραλαβών.

Τα ωφέλη που προσφέρει η εφαρμογή είναι:

- Μείωση δαπανών που προκύπτουν από λάθη χειρόγραφης καταχώρησης.
- Βελτιστοποίηση της διαδικασίας επιθεώρησης των εργασιών του αποθηκάριου.
- Αύξηση αποδοτικότητας των αποθηκάριων και βελτίωση του τρόπου εξυπηρέτησης των πελατών.
- Ορθολογική διαχείριση των αποθεμάτων της επιχείρησης.
- Δυνατότητα απασχόλησης ανειδίκευτου χρήστη.
- Άμεση και έγκυρη ενημέρωση της εταιρίας για πωλήσεις, παραλαβές, αποθέματα και επιστροφές συνεπάγοντας στην ταχύτερη και ορθότερη λήψη αποφάσεων.

4.3 SoftOne - Διαχείριση αποθήκης

Η **SoftOne** [3] έχει δημιουργήσει το λογισμικό **Soft1 ERP Series 6** αποτελεί μία ολοκληρωμένη λύση που βοηθάει κάθε επιχείρηση που δραστηριοποιείται στο χονδρικό εμπόριο και τη διανομή προσφέροντας ισχυρά εργαλεία reporting και λειτουργικότητα που εκτείνεται από τις παραγγελίες, τις πωλήσεις και τις αγορές έως τη διαχείριση αποθηκών και αποθεμάτων.

Η εφαρμογή βοηθάει στη καλύτερη παρακολούθηση των αποθηκευτικών χώρων μιας επιχείρησής. Βελτιώνει τη διαχείριση των αποθεμάτων της και μειώνει το χρόνο παράδοσης των προϊόντων της, αυξάνοντας την κερδοφορία.

Αυτοματοποιεί τις διαδικασίες διανομής των προϊόντων και απλοποιηεί όλες τις λειτουργίες της επιχείρησής, από τις πωλήσεις και τα αποθέματα έως τη διανομή και τη χρηματοοικονομική διαχείριση, εξοικονομώντας πολύτιμο χρόνο.

Προσφέρει ολοκληρωμένη εικόνα για την πορεία των δραστηριοτήτων της επιχείρησης από οπουδήποτε, μέσω οποιασδήποτε συσκευής.

Η εφαρμογή καλύπτει ανάγκες όπως:

- Διαχείριση αποθεμάτων ανά εγκατάσταση, αποθήκη, θέση, παρτίδα και serial number.
- Διαχείριση πολλαπλών αποθηκευτικών χώρων.
- Real-time διαχείριση αποθεμάτων
- Παρακολουθήση των προϊόντων σας, σε όλες τις διακινήσεις τους.
- Υπολογισμό του κόστους των εισαγόμενων (ή εξαγόμενων) αποθεμάτων εισαγωγής (ή εξαγωγής).

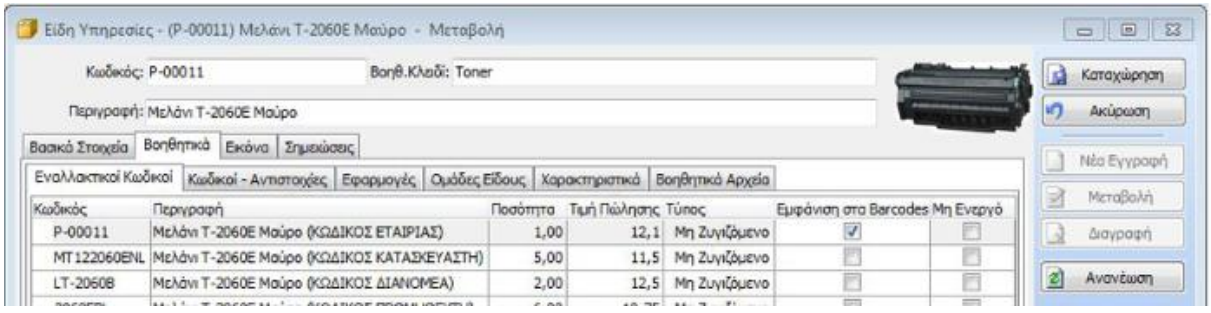
Τα ωφέλη που προσφέρει η εφαρμογή είναι:

- Μείωση δαπανών που προκύπτουν από λάθη χειρόγραφης καταχώρησης.
- Βελτιστοποίηση της διαδικασίας επιθεώρησης των εργασιών του αποθηκάριου.
- Ορθολογική διαχείριση των αποθεμάτων της επιχείρησης.
- Άμεση και έγκυρη ενημέρωση της εταιρίας για πωλήσεις, παραλαβές, αποθέματα και επιστροφές συνεπάγοντας στην ταχύτερη και ορθότερη λήψη αποφάσεων.

4.4 Megasoft - Διαχείριση αποθήκης

Η εφαρμογή **PRISMA Win** Διαχείριση Αποθήκης της **Megasoft** [4] προσφέρει την ιδανική διαχείριση για επιχειρήσεις ανταλλακτικών, βιομηχανικών εργαλείων, χρωματοπωλεία, βιβλιοπωλεία, ένδυση και υπόδηση, τρόφιμα και γενικότερα όλες τις επιχειρήσεις που καλούνται να διαχειριστούν καθημερινά πολλούς κωδικούς.

Η δυνατότητα που προσφέρει η εφαρμογή αυτή είναι διαχείριση μεγάλου όγκου πλήθους κωδικών προϊόντων και η εύκολη αναζήτηση τους μέσα σε αυτή(εικόνα 4.3). Μπορούμε να κάνουμε εύκολα αναζήτηση αυτό που χρειαζόμαστε και να ταξινομήσουμε τα αποτελέσματα βάση της πληροφορίας που επιθυμούμε.



Εικόνα 4.3 Είδη Υπηρεσίες

Οι δυνατότητες που προσφέρει είναι:

- Διαχείριση πολλών κωδικών προϊόντων
- Εύκολη και γρήγορη αναζήτηση
- Παρακολούθηση διαθεσιμότητας αποθέματος

Τα ωφέλη που προσφέρει η εφαρμογή είναι:

- Μείωση δαπανών που προκύπτουν από λάθη χειρόγραφης καταχώρησης.
- Ορθολογική διαχείριση των αποθεμάτων της επιχείρησης.
- Αύξηση αποδοτικότητας των αποθηκάριων και βελτίωση του τρόπου εξυπηρέτησης των πελατών.

4.4 Παρούσα εφαρμογή Διαχείριση αποθήκης

Η παρούσα εφαρμογή έχει ως στόχο να παντρέψει πιο σημαντικές δυνατότητες που προσφέρουν οι παραπάνω εφαρμογές για την αποτελεσματική διαχείριση του αποθέματος. Υλοποιήθηκε με σύγχρονες τεχνολογίες που την επιτρέπει να είναι προσπελάσιμη από οπουδήποτε έχουμε πρόσβαση στο διαδίκτυο αλλά και σε οποιαδήποτε συσκευή.

Το φιλικό προς το χρήστη περιβάλλον προσφέρει εύκολη περιήγηση μέσα από το μενού. Επίσης καθιστά εύκολη τη διαχείριση του συστήματος με τις λειτουργίες αναζήτησης στις λίστες ευρετηρίου και τη δημιουργία και επεξεργασία μέσα από απλές φόρμες προς το χρήστη.

Ο χρήστης αποθηκάριος μπορεί να εκτελέσει πολλές από τις βασικές καθημερινές εργασίες του και να παρακολουθεί σε πραγματικό χρόνο το απόθεμα κάθε προϊόντος.

Η εφαρμογή προσφέρει τη δυνατότητα να παρακολουθήσουμε προϊόντα που βρίσκονται σε διαφορετικές αποθήκες, οι οποίες πολλές φορές μπορεί να έχουν και μέγαλη χιλιομετρική απόσταση μεταξύ τους

Μπορούμε να καταγράψουμε την κίνηση εισαγωγής και εξαγωγής για κάθε προϊον και να κρατάμε μια λίστα αυτών για ιστορική αναφορά. Η κίνηση αυτή γίνετε μέσα από την εφαρμογή και αυτόματα ενημερώνεται και το απόθεμα του καθε προϊόντος.

Η εφαρμογή καλύπτει ανάγκες όπως:

- Διαχείριση αποθεμάτων ανά εγκατάσταση, αποθήκη, θέση, και barcode number.
- Διαχείριση πολλαπλών αποθηκευτικών χώρων.
- Real-time διαχείριση αποθεμάτων
- Παρακολουθήση των προϊόντων σας, σε όλες τις διακινήσεις τους.
- Ιστορικό παρακολούθησης των εισαγόμενων (ή εξαγόμενων) αποθεμάτων εισαγωγής (ή εξαγωγής).

Τα ωφέλη που προσφέρει η εφαρμογή είναι:

- Μείωση δαπανών που προκύπτουν από λάθη χειρόγραφης καταχώρησης.
- Βελτιστοποίηση της διαδικασίας επιθεώρησης των εργασιών του αποθηκάριου.
- Ορθολογική διαχείριση των αποθεμάτων της επιχείρησης.
- Άμεση και έγκυρη ενημέρωση της για πωλήσεις, παραλαβές, αποθέματα συνεπάγοντας στην ταχύτερη και ορθότερη λήψη αποφάσεων.

Τα πραγματικά ωφέλη όμως είναι αυτά που προκύπτουν λόγο το ότι η εφαρμογή στην ουσία είναι ένα api το οποίο μπορούν να χτυπήσουν τρίτες εφαρμογές στις οποίες θα δώσουμε πρόσβαση για να παρέχουμε κάποιες πληροφορίες.

Με αυτόν τον τρόπο μπορούμε να εκμεταλευτούμε και από την αντίθετη όψη συστήματα που μπορούν να επεκτείνουν τις δυνατότητες τις εφαρμογής και να καλύψουν άλλες ανάγκες που δεν προσφέρει η ίδια.

Οστόσο η αρχική σκέψη και ο βασικός στόχος της εφαρμογής ήταν να δουλεύει στο Cloud. Η σκέψη αυτή δημιουργήθηκε μέσα από τη πανδημία του covid-19 που βιώσαμε αυτά τα τελευταία δύο χρόνια. Με αφορμή λοιπόν την εξ αποστάσεως εργασίας σκέφτηκα να υλοποιήσω και να παρουσίασω ένα προϊόν που θα ακολουθούσε αυτή τη λογική και θα επιτρέπει τον εργαζόμενο να τη διαχειριστεί από οπουδήποτε βρίσκεται.

Βιβλιογραφικές αναφορές

Internet Site

[1] https://www.soft-net.gr/logismiko-epixiriseon/diaxeirisi-apothikis/#1578155116627-34717489-3e43

[2]http://www.albanis.gr/el/efarmoges/inventory_control/efarmogi_diaxeirisis_apo8ikon_aapo8emaa.html

[3] https://www.softone.gr/wholesale-distribution/

[4] https://www.megasoft.gr/proionta/emporikes-efarmoges-2

Επίλογος

Στο κεφάλαιο αυτό έγινε λόγος κάποιες από τις πιο γνωστές εφαρμογές που προσφέρουν ολοκληρωμένες λύσεις για την Διαχείρηση Αποθήκης και αναφέρθηκαν οι πιο σημαντικές δυνατότητες που προσφέρει η κάθε μια σε μια επιχείρηση. Επίσης γίνετε ανάφορα στα ωφέλη που προσφέρει στο χρήστη και κατ' επεκταση στο σύνολο της επιχείρησης και που τοποθετείται η εφαρμογή της παρούσας πτυχικής εργασίας μεταξύ αυτών.

Κεφάλαιο 5ο: Υλοποίηση της εφαρμογής

Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει αναφορά στα εργαλεία που χρησιμοποιήθηκαν για να υλοποιηθεί η εφαρμογή και τις κύριες λειτουργίες που συνθέτουν τη βασική δομή της. Τα εργαλεία αυτά χρησιμοποιήθηκαν για τη συγγραφή του κώδικα για το back και front end της εφαρμογής, τη διαχείριση της βάσης δεδομένων και τον έλεγχο των HTTP κλήσεων που δημιουργήθηκαν. Θα γίνει περιγραφή βασικών εννοιών όπως των REST Controller, τα βήματα που πρέπει να ακολουθήσουμε για τη δημιουργία, έλεγχο και σύνδεση τους με το κομμάτι διεπαφής του χρήστη.

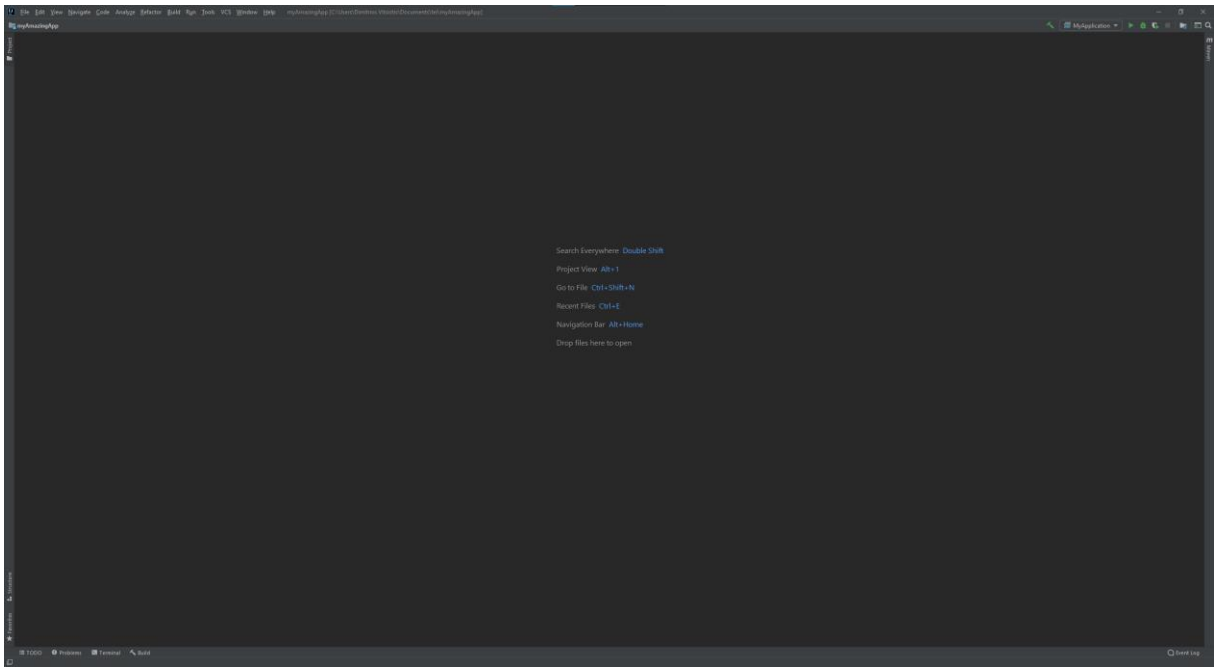
5.1 Java IDE - Ολοκληρωμένο Περιβάλλον Ανάπτυξης

Ένα ολοκληρωμένο περιβάλλον ανάπτυξης(Εικόνα 5.1) (integrated development environment, IDE)[1] είναι μία σουίτα λογισμικού που βοηθάει στην ανάπτυξη προγραμμάτων υπολογιστή. Συνήθως ένα IDE περιλαμβάνει κάποιον επεξεργαστή πηγαίου κώδικα, έναν μεταγλωττιστή, εργαλεία αυτόματης παραγωγής κώδικα, αποσφαλματωτή, συνδέτη, σύστημα ελέγχου εκδόσεων και εργαλεία κατασκευής γραφικών διασυνδέσεων χρήστη για τις υπό ανάπτυξη εφαρμογές.

Για την παρούσα πτυχιακή εργασία χρησιμοποιήθηκαν το Intellij IDEA για τη σύνταξη κώδικα για το back end με τη γλώσσα προγραμματισμού Java και χρήση του Framework Java Spring ενώ το WebStorm για τη σύνταξη του κώδικα για front end με χρήση Javascript και Html, κάνοντας χρήση του framework Angular 2. Τα δύο αυτά IDEA ανήκουν στην σουίτα προγραμμάτων ανάπτυξης κώδικα της JetBrains.

5.1.1 Intellij IDEA

Το IntelliJ IDEA είναι ένα Java IDE που αναπτύχθηκε όπως αναφέρθηκε και παραπάνω από την JetBrains. Η πρώτη έκδοση του IntelliJ κυκλοφόρησε το 2001. Εκείνη την εποχή, ήταν το μοναδικό IDE με υποστήριξη για προηγμένη πλοήγηση κώδικα και refactoring. Είναι ένα εμπορικό προϊόν, όπου διατίθεται δωρεάν δοκιμή 30 ημερών (με όλες τις δυνατότητες) για όλες τις πλατφόρμες. Πιο πρόσφατα, έγινε διαθέσιμη μια έκδοση ανοιχτού κώδικα. Η τρέχουσα σταθερή έκδοση είναι **2021.3**. Προσφέρει υποστήριξη για τη σχεδίαση διαγραμμάτων κατηγορίας UML, οπτική μοντελοποίηση στο Hibernate, Spring, ανάλυση εξαρτήσεων και Maven. Εφαρμογές σε πολλές γλώσσες όπως Java, JavaScript, HTML, Python, Ruby, PHP και πολλές άλλες μπορούν να αναπτυχθούν χρησιμοποιώντας το IntelliJ. Το IntelliJ υποστηρίζει μια μεγάλη γκάμα πλαισίων και τεχνολογιών όπως JSP, JSF, EJB, Ajax, GWT, Struts, Spring, Hibernate και OSGi. Επιπλέον, διάφοροι διακομιστές εφαρμογών όπως GlassFish, JBoss, Tomcat και WebSphere υποστηρίζονται από την IntelliJ. Η IntelliJ καθιστά δυνατή την εύκολη ενσωμάτωση με CVS, Subversion, Ant, Maven και JUnit.



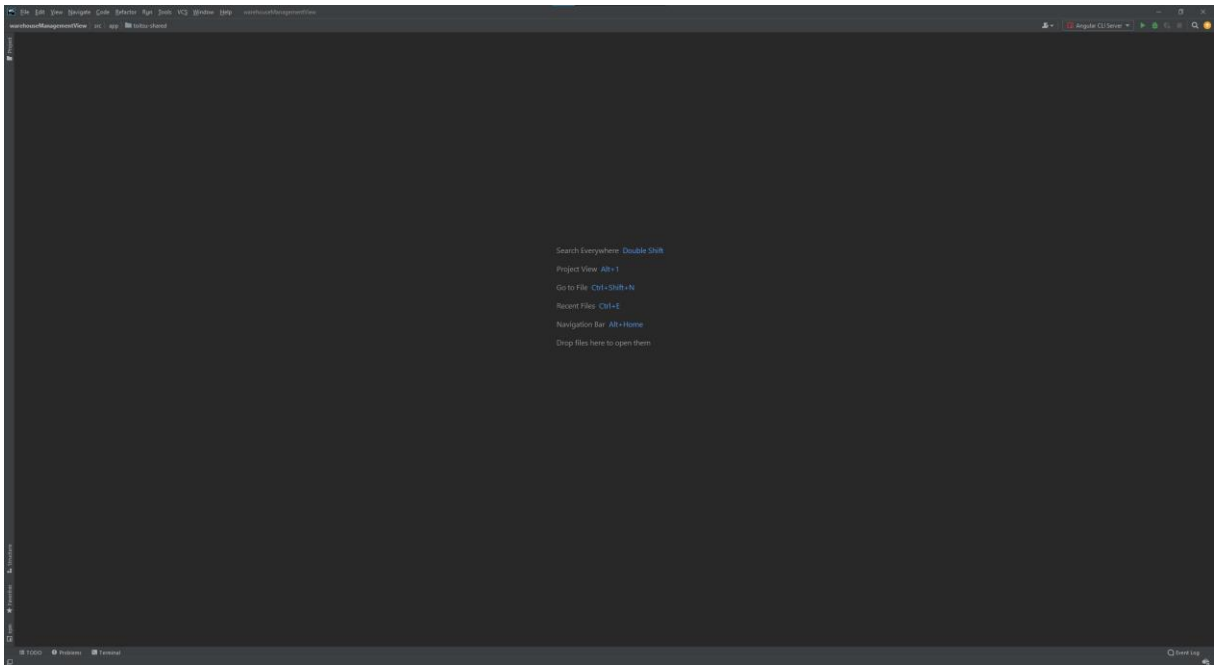
Εικόνα 5.1 Περιβάλλον Intellij IDEA

5.1.1.1 Διαφορές με άλλα IDE

Η υποστήριξη για το Maven είναι καλύτερη στο IntelliJ. Το IntelliJ IDEA διαθέτει ενσωματωμένο πρόγραμμα δημιουργίας GUI για Swing. Στην πραγματικότητα, η κοινότητα Java θεωρεί το GUI Builder της IntelliJ ως τον καλύτερο σχεδιαστή GUI αυτή τη στιγμή. Όσον αφορά την υποστήριξη XML, το IntelliJ προσφέρει την καλύτερη επιλογή. Διαθέτει ενσωματωμένο πρόγραμμα επεξεργασίας XML με εξελιγμένες δυνατότητες όπως συμπλήρωση κώδικα.

5.1.2 WebStorm

Το WebStorm(Εικόνα 4.2) είναι ένα IDE για τη σύνταξη κώδικα σε JavaScript και τις συναφείς τεχνολογίες, όπως TypeScript, React, Vue, Angular, Node.js, HTML και style sheets. Η τρέχουσα σταθερή έκδοση είναι **2021.3.1.** Ακριβώς όπως το IntelliJ IDEA και τα άλλα IDE της JetBrains, το WebStorm κάνει την εμπειρία της ανάπτυξής ενός προγράμματος πιο ευχάριστη, αυτοματοποιώντας τις εργασίες ρουτίνας και βοηθώντας το εύκολο χειρισμό πολύπλοκων εργασιών.



Εικόνα 5.2 Περιβάλλον Webstorm

5.2 Διαχείριση Βάσης Δεδομένων

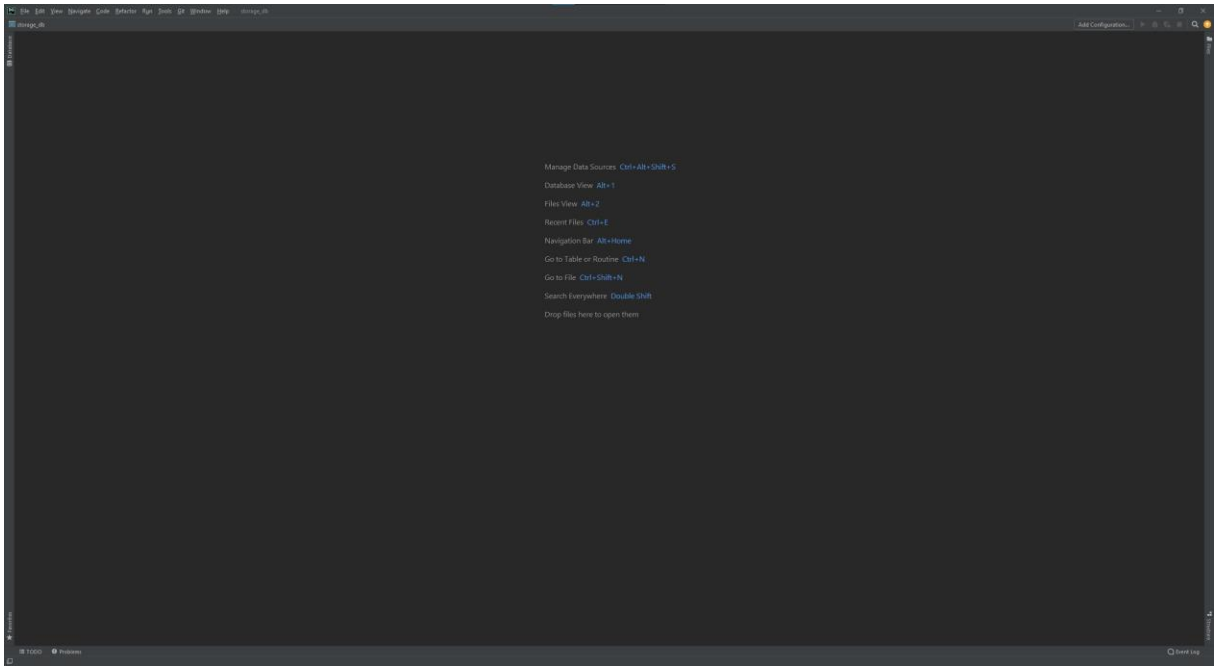
Με τον όρο Σύστημα Διαχείρισης Βάσης Δεδομένων, γνωστό ως Database Management system (DBMS), εννοείται είτε κάποιο λογισμικό μέσω του οποίου γίνεται η δημιουργία, η διαχείριση, η συντήρηση και η χρήση μιας ηλεκτρονικής βάσης δεδομένων, ανάλογα με τον τύπο βάσης δεδομένων που επιλέγεται ή ένα σύνολο αλληλοσυσχετιζόμενων προγραμμάτων που τρέχουν και διαχειρίζονται τα δεδομένα μιας τέτοιας βάσης. Το λογισμικό χρησιμοποιεί στερεότυπες (standard) μεθόδους καταλογοποίησης, ανάκτησης, και εκτέλεσης ερωτημάτων σχετικών με τα δεδομένα. Το σύστημα διαχείρισης οργανώνει τα εισερχόμενα δεδομένα με τρόπους χρησιμοποιήσιμους από εξωτερικούς χρήστες.

Ιδωμένο από μία άλλη οπτική γωνία, το σύστημα διαχείρισης βάσης δεδομένων είναι ένας διαχειριστής αρχείων (file manager) που διαχειρίζεται δεδομένα σε βάσεις δεδομένων παρά αρχεία σε συστήματα αρχείων, τα οποία είναι μία άλλη μορφή βάσης δεδομένων.

Πέραν της καταλογοποίησης το πλήρες σύστημα διαχείρισης βάσης δεδομένων ευρετηριάζει ή θα έπρεπε να ευρετηριάζει τα δεδομένα και να βελτιστοποιεί τους πίνακες δεδομένων του. Το σημαντικότερο όλων είναι ότι πρέπει να φροντίζει για την ακεραιότητα των εισαγόμενων στοιχείων και την απόδοσή τους με πολλούς διαφορετικούς τύπους, ανάλογα με ιδιαίτερες ανάγκες του χρήστη. Αντίθετα προς τα συστήματα διαχείρισης των δεδομένων που επεξεργάζονται και αλλάζουν τα δεδομένα σύμφωνα με τα προσδοκώμενα αποτελέσματα από έναν ιδιαίτερο αλγόριθμο, αποδίδοντας λογικό περιεχόμενο, το σύστημα διαχείρισης βάσης δεδομένων χρησιμοποιεί εκτελεί τους ελάχιστους δυνατούς μαθηματικούς υπολογισμούς καθώς ο κύριος στόχος του η οργάνωση, η διαχείριση και η απόδοση δεδομένων σε περίπτωση ζήτησης.

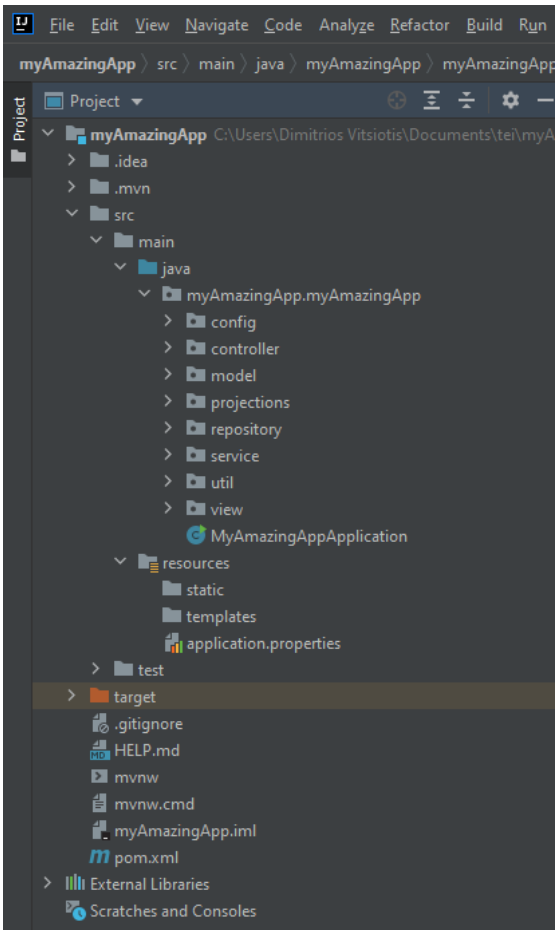
5.2.1 DataGrip

Το DataGrip(Εικόνα 4.3) είναι ένα περιβάλλον διαχείρισης βάσεων δεδομένων για προγραμματιστές[3]. Είναι σχεδιασμένο για την αναζήτηση, τη δημιουργία και τη διαχείριση βάσεων δεδομένων. Οι βάσεις δεδομένων μπορούν να λειτουργούν τοπικά, σε διακομιστή ή στο cloud Το DataGrip είναι συμβατό με MySQL, PostgreSQL, Microsoft SQL Server, Microsoft Azure, Oracle, Amazon Redshift, Sybase, DB2, SQLite, HyperSQL, Apache Derby και H2. Αναπτύχθηκε και αυτό όπως το Intellij IDEA και το WebStorm από την JetBrains και όπως σε αυτά έτσι και εδώ προσφέρεται η αυτόματη συμπλήρωση κώδικα, εντοπίζει πιθανά σφάλματα στον κώδικα μας και προτείνει τις καλύτερες επιλογές για να τα διορθώσουμε.



Εικόνα 5.3 Περιβάλλον DataGrip

5.3 Βάσικες έννοιες



Εικόνα 4.4 Project Structure

5.3.1 Rest Controller

Ένα controller περιέχει την πληροφορία που θα χρειαστεί να έχει μια http αίτηση(request) για να επιστρέψει την πληροφορία για την οποία σχεδιάστηκε. Τα κύρια στοιχεία ενός controller για να γίνει ένα reuest σε αυτό είναι το **U1RI αίτησης** στο οποίο θα γίνετε η κλήση, τα Πεδία κεφαλίδας μηνύματος αίτησης HTTP, Προαιρετικά πεδία **σώματος μηνύματος αίτησης[2].** Η απόκριση (response) ενός controller συνήθως επιστρέφει Πεδία **κεφαλίδας μηνύματος απόκρισης(response message header)** HTTP και κάποια Προαιρετικά πεδία **σώματος μηνύματος απόκρισης(response message body).**

```
@Controller
public class MyController {
```

5.3.1.1 URI αίτησης (request URI)

Το URI αίτησης, που αποτελείται από τα εξής: {URI-scheme} :// {URI-host} / {resource-path} ? {query-string} . Παρόλο που το URI αίτησης περιλαμβάνεται στην κεφαλίδα του μηνύματος αίτησης, αναφέρεται ξεχωριστά εδώ επειδή οι περισσότερες γλώσσες ή πλαίσια απαιτούν να διαβιβάζεται ξεχωριστά από το μήνυμα αίτησης.

- Σχήμα URI: Υποδεικνύει το πρωτόκολλο που χρησιμοποιείται για τη μετάδοση της αίτησης. Για παράδειγμα, http ή https.
- Κεντρικός υπολογιστής URI: Καθορίζει το όνομα τομέα ή τη διεύθυνση IP του διακομιστή όπου φιλοξενείται το τελικό σημείο της υπηρεσίας REST, όπως myserver.contoso.com.
- Διαδρομή πόρου: Καθορίζει τον πόρο ή τη συλλογή πόρων και μπορεί να περιλαμβάνει πολλά τμήματα που χρησιμοποιούνται από την υπηρεσία κατά τον προσδιορισμό της επιλογής αυτών των πόρων.
- Συμβολοσειρά ερωτήματος (προαιρετικό): Παρέχει πρόσθετες απλές παραμέτρους, όπως η έκδοση του API ή τα κριτήρια επιλογής πόρου.

5.3.1.2 Πεδία κεφαλίδας μηνύματος αίτησης(request message header fields) HTTP

Μια απαιτούμενη μέθοδος HTTP (γνωστή και ως λειτουργία ή ενέργεια), που υποδεικνύει στην υπηρεσία τον τύπο της λειτουργίας που ζητάτε. Οι Υπηρεσίες αναφοράς των API REST υποστηρίζουν τις μεθόδους DELETE, GET, HEAD, PUT, POST και PATCH.

Προαιρετικά πρόσθετα πεδία κεφαλίδας, όπως απαιτείται από το καθορισμένο URI και τη μέθοδο HTTP.

5.3.1.3 Προαιρετικά πεδία σώματος μηνύματος αίτησης(Optional request message body fields) HTTP

Για παράδειγμα, οι λειτουργίες POST περιέχουν αντικείμενα με κωδικοποίηση MIME που μεταβιβάζονται ως σύνθετες παράμετροι. Για λειτουργίες POST ή PUT, ο τύπος της κωδικοποίησης MIME για το σώμα πρέπει επίσης να καθοριστεί στην κεφαλίδα αίτησης Content-type. Ορισμένες υπηρεσίες απαιτούν τη χρήση ενός συγκεκριμένου τύπου MIME, όπως application/json.

5.3.1.4 Πεδία κεφαλίδας μηνύματος απόκρισης HTTP(HTTP response message header fields)

Έναν κωδικό κατάστασης HTTP, που κυμαίνεται από κωδικούς επιτυχίας 2xx έως κωδικούς σφάλματος 4xx ή 5xx. Εναλλακτικά, μπορεί να επιστρέφεται ένας κωδικός κατάστασης που καθορίζεται από την υπηρεσία, όπως υποδεικνύεται στην τεκμηρίωση του API.

Προαιρετικά πρόσθετα πεδία κεφαλίδας, όπως απαιτείται για την υποστήριξη της απόκρισης της αίτησης, όπως μια κεφαλίδα απόκρισης Content-type.

5.3.1.5 Προαιρετικά πεδία σώματος μηνύματος απόκρισης(HTTP response message header fields)

Τα αντικείμενα απόκρισης με κωδικοποίηση MIME επιστρέφονται στο σώμα της απόκρισης HTTP, όπως μια απόκριση από μια μέθοδο GET που επιστρέφει δεδομένα. Συνήθως, αυτά τα αντικείμενα επιστρέφονται σε δομημένη μορφή όπως JSON ή XML, όπως υποδεικνύεται από την κεφαλίδα απόκρισης Content-type.

5.3.1.6 Τεκμηρίωση(documentation) API

Ένα σύγχρονο REST API απαιτεί σύγχρονη τεκμηρίωση API. Η REST API έχει δημιουργηθεί με βάση την προδιαγραφή OpenAPI (a.k.a. τις προδιαγραφές swagger) και η τεκμηρίωση είναι διαθέσιμη στο SwaggerHub. Εκτός από την τεκμηρίωση του API, το SwaggerHub σάς βοηθά να δημιουργήσετε μια βιβλιοθήκη για προγράμματα-πελάτη στη γλώσσα της επιλογής σας, όπως JavaScript, TypeScript, C#, Java, Python, Ruby και πολλές άλλες.

Με τη χρήση της βιβλιοθήκης Swagger η τεκμηρίωση γίνετε απευθείας στη δήλωση του κάθε controller στην οποία αναφέρεται μπορεί να αναφέρεται η απόκριση του και τα πεδία σώματος που μπορεί να έχει.

5.3.2 Services

Τα Service Components είναι το αρχείο κλάσεων που περιέχει τον χαρακτηρισμό @Service. Αυτά τα αρχεία κλάσεων χρησιμοποιούνται για τη συγγραφή της επιχειρησιακής λογικής σε ένα διαφορετικό επίπεδο, χωριστά από το αρχείο κλάσεων @RestController.[4]

Η λογική για τη δημιουργία ενός αρχείου κλάσης service component παρουσιάζεται εδώ

```
public interface ProductService {
}
```

Η κλάση που θα υλοποιήσει τις μεθόδους του παραπάνω Interface έχει το χαρακτήρισμό @Service.

```
@Service
public class CountryService implements ICountryService {
```

Εφόσον έχουμε υλοποιήσει τους Rest Controllers και γνωρίζουμε τι θέλουμε μας επιστρέφει συνέχεια έχουμε να υλοποίήσουμε αυτήν την επιχειρησιακή λογική στα Services. Εδώ μπορούμε να κάνουμε ελέγχους των δεδομένων που θέλουμε να ανακτήσουμε από τη βάση και να τα εμφανίσουμε στο χρήστη, να κάνουμε μετατροπή αυτών και να διαγράψουμε. Μπορούμε να υλοποιήσουμε τη λογική του CRUD όπως αναφέρθηκε σε άλλη ενότητα δηλαδή τη δημιουργίας, τροποποίησης και διαγραφής μια εγγραφής.

Τα services μπορούν εκμεταλευτούν τις υλοποιήσεις άλλων services χρησιμοποιώντας το annotation @Autowired.

```
@Autowired
   ProductService productService;
```

Με αυτό τον τρόπο αποφεύγουμε να γράφουμε και να συντηρούμε διπλό κώδικα.

5.3.3 Repository

Το @Rerpository είναι ένα annotation στη Spring που δηλώνει ότι η κλάση είναι ένα Repository[5]. Το Repository είναι ένας μηχανισμός που ενθυλακώνει τις λειτουργίες της αναζήτης, ανάκτησης και αποθήκευσης ενός αντικειμένου. Στην ουσία είναι κλάση που επικοινωνεί με τη βάση μας και αποτελεί των ενδιάμεσο μεταξύ αυτής και του service. Από τη βάση ανακτούμε τις εγγραφές που αναφέρονται στην κλάση με το annotation @Entity. Υπάρχουν πολλοί τρόποι να επικοινωννήσεις με το repository είτε είναι με τη χρήση της JPA, είτε με τη δημιουργία Qclasses, ακόμα κάνοντας χρήση sql ερωτημάτων JPA QUERY, native queries κά.

```
@Repository
public interface CountryRepository extends CrudRepository<Country, Long> {

}
```

5.3.3.1 Entity

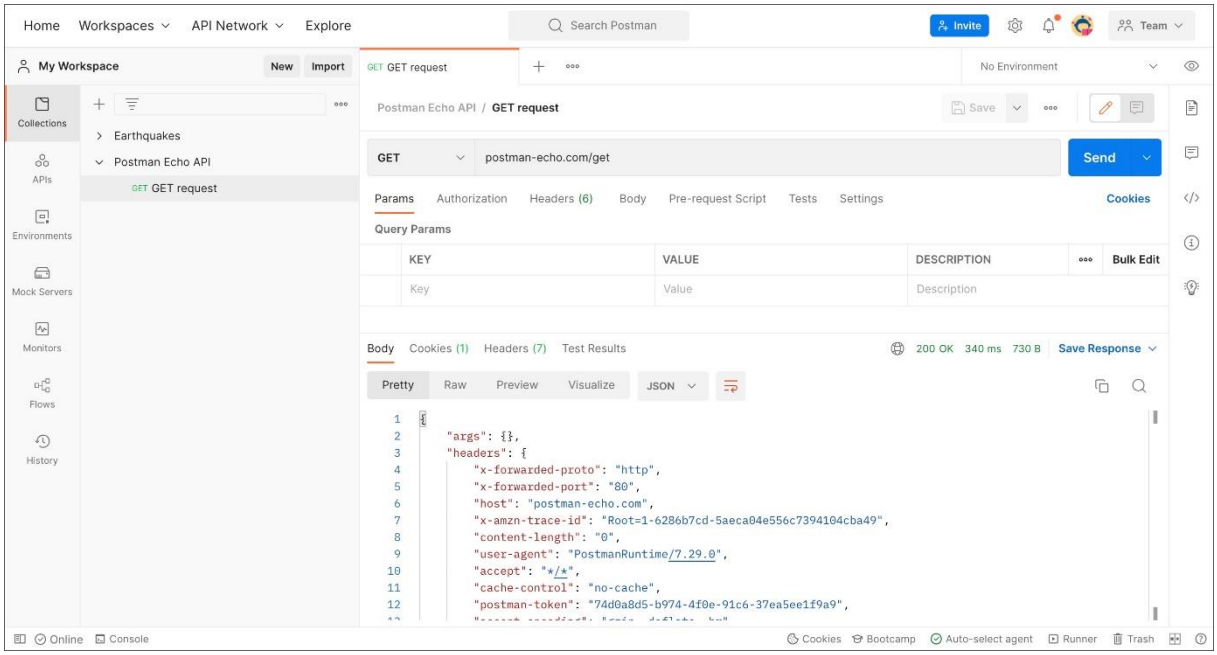
Ο χαρακτηρισμός @Entity καθορίζει ότι η κλάση είναι μια οντότητα και αντιστοιχίζεται σε έναν πίνακα της βάσης δεδομένων. Η σημείωση @Table καθορίζει το όνομα του πίνακα της βάσης δεδομένων που θα χρησιμοποιηθεί για την αντιστοίχιση. Για κάθε οντότητα πρέπει να ορίζονται τουλάχιστον δύο annotations: @Entity και @Id. Ο χαρακτηρισμός @Id προσδιορίζει το πρωτεύον κλειδί μιας οντότητας και ο χαρακτηρισμός @GeneratedValue δίνει τη στρατηγική δημιουργίας των τιμών των πρωτευόντων κλειδιών.

```
@Entity
@Table(name = "countries")
public class Country {
```

5.4 Postman - Έργαλείο για των έλεγχο των κλήσεων στο back end

Ένα εργαλείο για τον έλεγχο των μηνυμάτων αίτησης/απόκρισης HTTP είναι το Postman. Το Postman είναι μια δωρεάν εφαρμογή web μεσολάβησης για τον εντοπισμό σφαλμάτων που μπορούν να εμποδίσουν τις αιτήσεις REST, το οποίο διευκολύνει τη διάγνωση των μηνυμάτων αίτησης/απόκρισης HTTP.

Το Postman(εικόνα 4.5) είναι μια πλατφόρμα API για τη δημιουργία και τη χρήση APIs. Μπορούμε αφού έχουμε πρώτα 'χτίσει' το back end της εφαρμογή μας να κάνουμε HTTP κλήσεις στα URL που έχουμε ορίσει στους controllers τις κάθε οντότητας. Με αυτό τον τρόπο μπορούμε να κάνουμε το testing της εφαρμογής μας και να βλέπουμε ότι αυτό που έχουμε υλοποήσει μέχρι στιγμής δουλεύει σωστά. Για να το καταλάβουμε αυτό, είναι χρήσιμο να γνωρίζουμε τους κωδικούς των σφαλμάτων που μας επιστρέφει το postman μετά από κάθε κλήση.



Εικόνα 4.5 Περιβάλλον Postman

5.4.1 Δημιουργία μιας Http κλήσης

Αρχικά έχοντας ανοιχτή την πλατφόρμα του postman επιλέγουμε από το new και στη συνέχεια το HTTP Request. Επιλέγουμε τη μέθοδο του HTTP request που είναι αυτή που έχουμε ορίσει στον controller που θέλουμε να κάνουμε το testing. Οι μέθοδοι που είναι διαθέσιμοι στην πλατφόρμα μας είναι αρκετοί, οστόσο αυτοί που χρησιμοποιήθηκαν ηταν η GET, POST, PUT, DELETE. Η πρώτη μέθοδος χρησιμοπιήθηκε για κλήσεις που είχαμε να πάρουμε δεδομένα και να τα εμφανίσουμε στο χρήστη, ενώ οι υπόλοπιες τρεις για να καλύψουν το CRUD της φαρμογής, δηλάδη το Create , Update και Delete.

Δίπλα από τη μέθοδο που θα επιλέξουμε από τη λίστα γράφουμε το url του http request. Ένα http request είναι της μορφής **http://localhost:8080/products/all** όπου από το πρώτο κομμάτι **http://** καταλαβαίνουμε ότι πρόκυται για http κλήση, το **localhost:8080** είναι η πόρτα της εφαρμογής μας στην οποία θα γίνει η κλήση, το **products** είναι το όνομα της οντότητας της εφαμογής και το **all** ο controller, ο οποίος θα μας επιστρέφει το αποτέλεσμα.

Επιπλέον μπορούμε να προσθέσουμε και παραμέτρους που μπορεί να έχει ένα request όπως για παράδειγμα το id αν θέλουμε να πάρουμε συγκεκριμένη εγγραφή. Σε αυτή την περίπτωση στο tab params, συμπληρώνουμε το KEY με τη λέξη κλειδί που ορίσαμε στο controller για να επιστρέψει την εγγραφή που χρειαζόμαστε βάσει της τιμής VALUE που θα του δώσουμε. Το DESCRIPTION αποτελεί προαιρετικό πεδίο και είναι μια περιγραφή της παραμέτρου της κλήσης, χωρίς βέβαια να την επηρεάζει.

Στο tab Authorization προσθέτουμε το τύπο του authorization που χρησιμοποιούμε για να κάνουμε κλήση στους controllers. Αν έχουμε ορίσει στον κώδικα ότι πρέπει να είναι εξουσιοδοτημένος ο χρήστης που θα κάνει την κλήση και δεν έχει οριστεί το authorization και τα υπόλοιπα στοιχεία που χρειάζεται ο τύπος που θα επιλέξει, τότε η κλήση επιστρέφει με σφάλα 401 από το back end μας.

Το tab Body το συμπληρώνουμε συνήθως όταν το request που κάνουμε είναι τύπου POST. Αυτό μας επιτρέπει να στείλουμε πίσω στον κώδικα μας τα στοιχεία μια ολόκληρης εγγραφής που έχουμε ορίσει στον κώδικα και ο αντίστοιχος controller περιμένει ως παράμετρο. Το POST το χρησιμοποιούμε κυρίως για τη δημιουργία μιας εγγραφής, οπότε το body περιέχει όλα τα πεδία που έχουμε ορίσει στον κώδικά μας ότι περιμένουμε κατά το request. Επίσης μπορεί να χρησιμοποιηθεί και για την ενημέρωση μια εγγραφής. Σημαντικό στις POST κλήσεις είναι το body που στέλνουμε, στην περίπτωση μας της μορφής JSON να είναι όπως το έχουμε ορίσει στην αντίστοιχη DTO κλάση που κάνουμε την κλήση. Αν δεν είναι η ονομασία και η σειρά των πεδίων στο body του postman με αυτό του DTO της κλάσης, το response θα επιστρέψει με τον κωδικό 400.

Στη συνέχεια πατώντας το Send στέλνουμε το http request και αν πήγαν όλα καλά θα λάβουμε την απάντηση μας από το back end της εφαρμογής, αλλιώς ο postman θα μας εμφανίσει τον κωδικό του σφάλματος, στο status του response, που συνέβει και απέτυχε το request.

5.4.2 Postman – Response/Απάντηση της Http κλήσης

Παραπάνω αναφέρθηκαν κάποιοι κωδικοί κατάστασης που επιστρέφει ο postman όταν κάνουμε μια κλήση στο back end της εφαρμογής μας. Του κωδικούς αυτούς είναι αναγκαίο να τους γνωρίζει ο προγραμματιστής που χτίζει την εφαρμογή αφού μπορεί να καταλάβει ποιο είναι το πρόβλημα και που στο κομμάτι του κώδικα που κάνει το test.

Παρακάτω θα δούμε κάποιους από τους πιο συχνούς κωδικούς που μπορεί να συναντήσει ο προγραμματιστής στην απάντηση της κλήσης που θα κάνει.

- **200**: Αυτός ο κωδικός χρησιμοποιείται για μια επιτυχημένη κλήση.
- **400**: Αυτός ο κωδικός χρησιμοποιείται για το Bad Request. Εάν έχουμε κάποιο τυπογραφικό ή μας λείπουν κάποιες απαιτούμενες παράμετροι, τότε το αίτημα δεν θα γίνει κατανοητό από το back end.
- **401**: Αυτός ο κωδικός χρησιμοποιείται για μη εξουσιοδοτημένη πρόσβαση. Εάν ο έλεγχος ταυτότητας του κλήσης που ορίσαμε απέτυχε ή ο χρήστης δεν έχει δικαιώματα για τις ζητούμενες λειτουργίες.
- **404**: Εμφανίζεται όταν έχουμε κάνει λάθος του URL της κλήσης, με αποτέλεσμα να μη μπορεί να τη βρει.
- **405**: Αυτό θα εμφανιστεί αν η μέθοδος της κλήσης δεν επιτρέπεται ή αν η ζητούμενη μέθοδος δεν υποστηρίζεται. Συμβαίνει όταν έχουμε ορίσει μια κλήση POST και στην κλήση στο postman βάλουμε κάποια άλλη.
- **500**: Ο κωδικός αυτό εμφανίζεται όταν έχουμε σφάλμα στο back end μας.

Οι παραπάνω κωδικοί ανάλογα με την υλοποίηση που έχει γίνει στο back end μπορεί να επιστρέψει κάποια απάντηση στο response που θα είναι πιο κατανόητο από το προγραμματιστή. Αυτό είναι χρήσιμο στην περίπτωση που ο κωδικό κατάστασης είναι σφάλμα, κάνοντας έτσι την εύρεση του σφάλματος πιο γρήγορη.

Βιβλιογραφικές αναφορές

[1] https://ubunlog.com/el/intellij-idea-ide-java/

[2] https://docs.microsoft.com/el-gr/power-bi/report-server/rest-api

[3] https://ubunlog.com/el/datagrip-ide-bases-de-datos/

[4] https://www.tutorialspoint.com/spring_boot/spring_boot_service_components.htm

[5] https://zetcode.com/springboot/repository/

Επίλογος

Στο κεφάλαιο αυτό έγινε μια μικρή αναφορά των IDE που χρησιμοποιήθηκαν για την ανάπτυξη του κώδικα, τον έλεγχο αυτού αλλά και τη διαχείριση της βάσης. Αναφέρθηκαν κάποιες βασικές έννοιες που είναι απαραίτητες για την ανάπτυξη μιας REST εφαρομογής και πώς κάνουμε τον έλεγχο κάνοντας HTTP κλήσεις στο back end (server) πριν ακόμα υλοποιήσουμε το UI αυτής.

Κεφάλαιο 6ο: Λειτουργίες και δυνατότητες της εφαρμογής

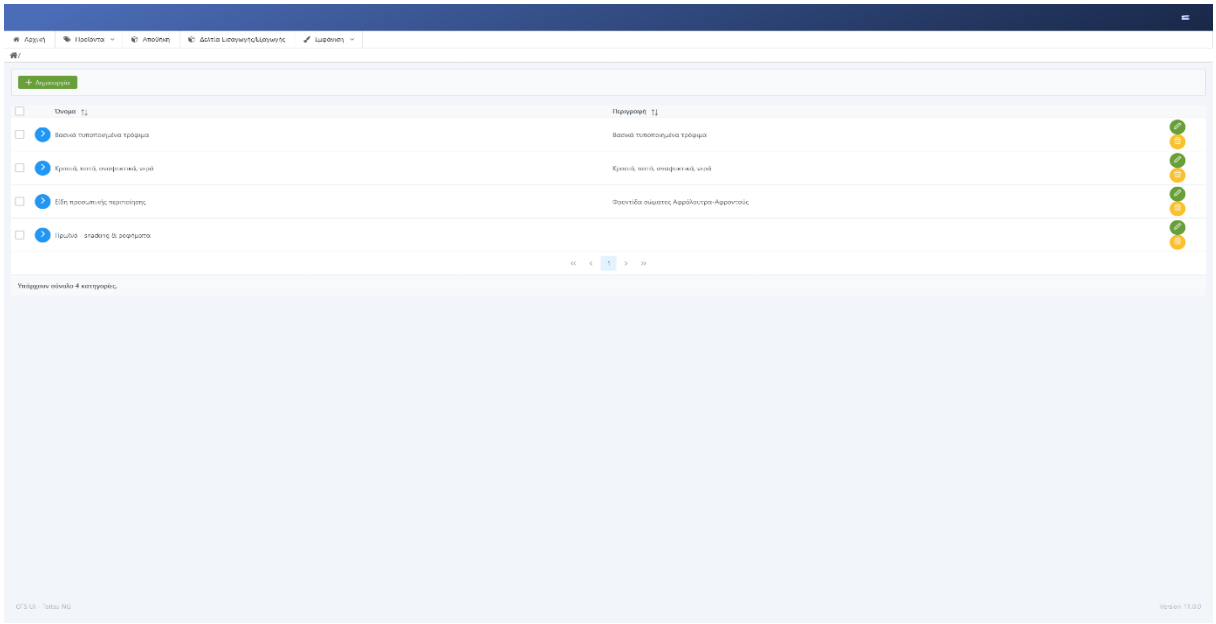
Σε αυτό το κεφάλαιο θα αναφερθούν οι δυνατότες που προσφέρουν άλλες εφαρμογές με πολυετή παρουσία στο χώρο του εμπορίου και τι μπορεί η παρούσα εφαρμογή να κάνει σε σχέση με αυτές. Επίσης αναλύεται οι λειτουργίες που μπορεεί να κάνει ο χρήστης μέσα από αυτή την εφαρμογή και πως αυτές θα τον βοηθήσουν να διαχειρίζεται την καθημερινή ρουτίνα της εργασίας του.

6.1 Αποθήκη

6.1.1 Ευρετήριο

Στο ευρετήριο(Εικόνα 6.1) εμφανίζεται μια λίστα με όλες τις αποθήκες που είναι καταχωρημένες στην εφαρμογή. Οι εγγραφές ανακτούνται όλες με το άνοιγμα του παραθύρου και υπάρχει ένα πεδίο για τη γρήγορη αναζήτηση κάποιας από αυτές.

Η εμφάνιση των αποθηκών γίνεται σε δενδροειδή λίστα ώστε να μπορέσουμε να εμφανίσουμε το σύνολο των τμημάτων από τα οποία αποτελείται η κάθε μία.

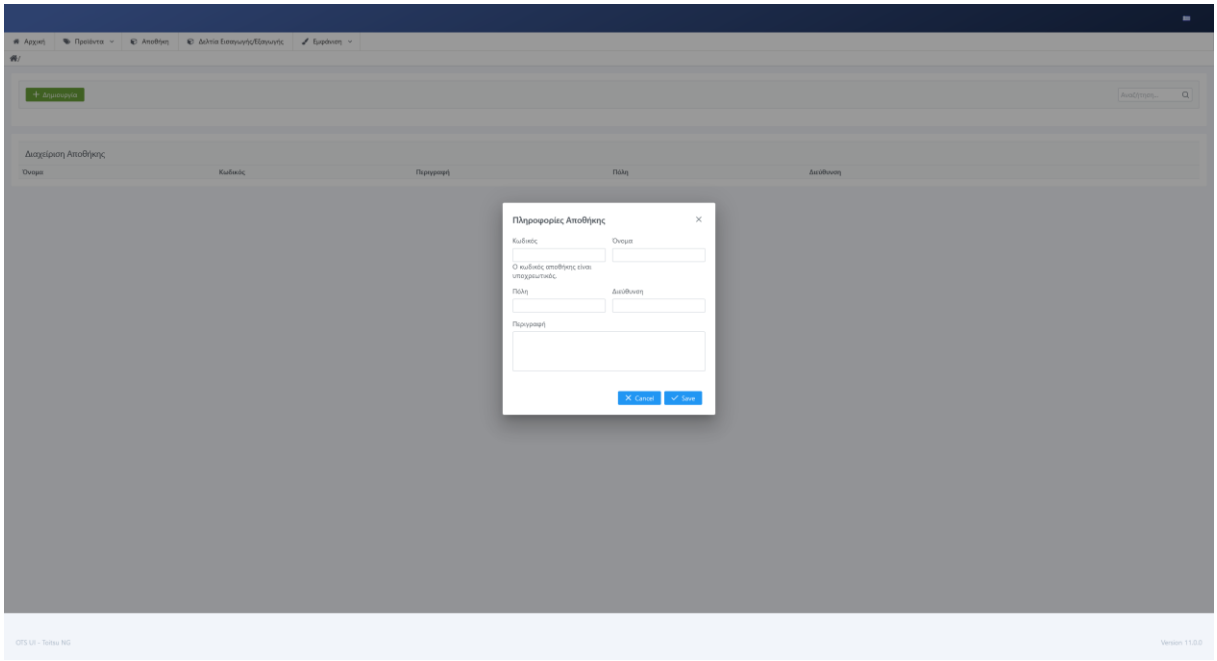


Εικόνα 6.1 Ευρεήριο αποθηκών

6.1.2 Λειτουργίες CRUD

Ο χρήστης της εφαρμογής θα μπορεί να καταχωρήσει στην εφαρμογή την αποθήκη(Εικόνα 5.2) που θέλει και να προσθέσει σε αυτή κάποια στοιχεία. Τα στοιχεία που πρέπει να καταχωρήσει είναι **Όνομα, Κωδικός, Περιγραφή, Πόλη, Διεύθυνση.** Ο κωδικός της αποθήκης είναι μοναδικό πεδίο που σημαίνει ότι δεν μπορεί να χρησιμοποίηθει περισσότερο της μιας φοράς. Μετά την καταχώρηση της αποθήκης ο χρήστης μπορεί εύκολα να επεξεργαστεί τα στοιχεία της αν κρίνει αναγκαίο, ακόμα και την αφαιρέσει. Κάθε ενέργεια διαγραφής που κάνει είναι μη ανατρέψιμη για αυτό το λόγο έχει μπει μήνυμα επιβεβαίωσης της.

Η εφαρμογή υποστηρίζει την παρακολούθηση αποθέματος σε ξεχωριστές αποθήκες που μπορεί να έχουν μεγάλη χιλιομετρική απόσταση μεταξύ τους.

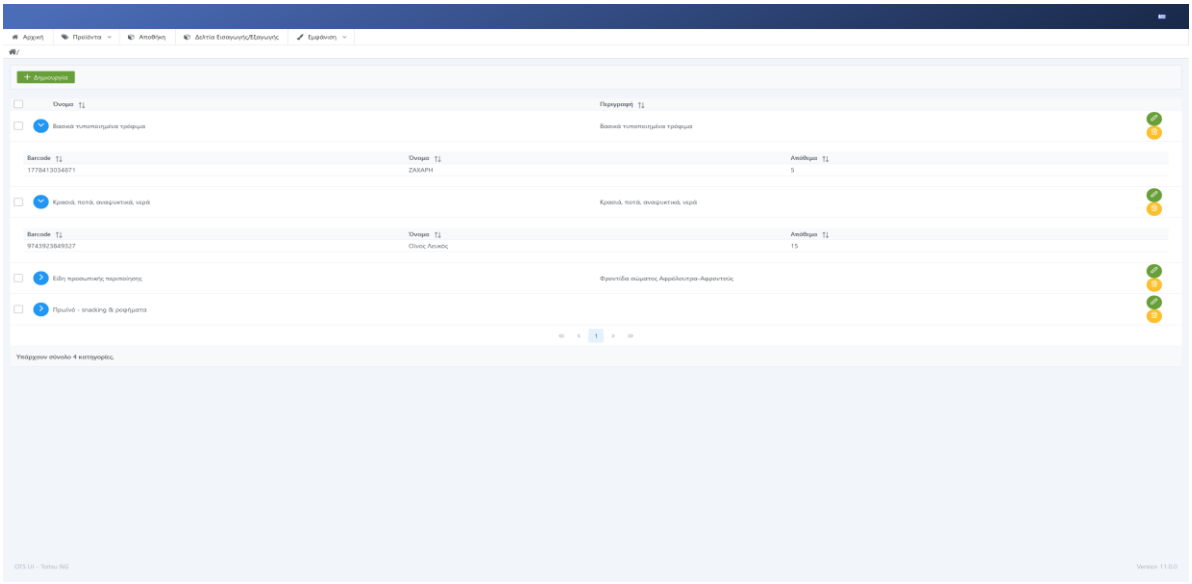


Εικόνα 6.2 Δημιουργία αποθήκης

6.2 Τμήμα αποθήκης

6.2.1 Λειτουργίες CRUD

Στη συνέχεια της δημιουργίας και καταχώρησης της αποθήκης στη βάση δεδομένων ο χρήστης πρέπει να δημιουργήσει τα τμήματα(Εικόνα 6.3) από τα οποία αυτή αποτελείται. Το τμήμα συνδέεται με την αποθήκη που θα επιλέξει ο χρήστης. Κάθε τμήμα όπως και η αποθήκη έχει κάποια στοιχεία που πρέπει να συμπληρωθούν. Τα στοιχεία αυτά είναι **κωδικός τμήματος, όνομα, περιγραφή και ο αριθμός των ραφιών.** Ο κωδικός του τμήματος όπως και της αποθήκης είναι μοναδικός. Μετά την καταχώρηση τμήματος όπως και στην αποθήκη ο χρήστης έχει τη δυνατότητα της επεξεργασίας και διαγραφή του αντίστοιχα.



Εικόνα 6.3 Ευρετήριο Τμήματος αποθήκης

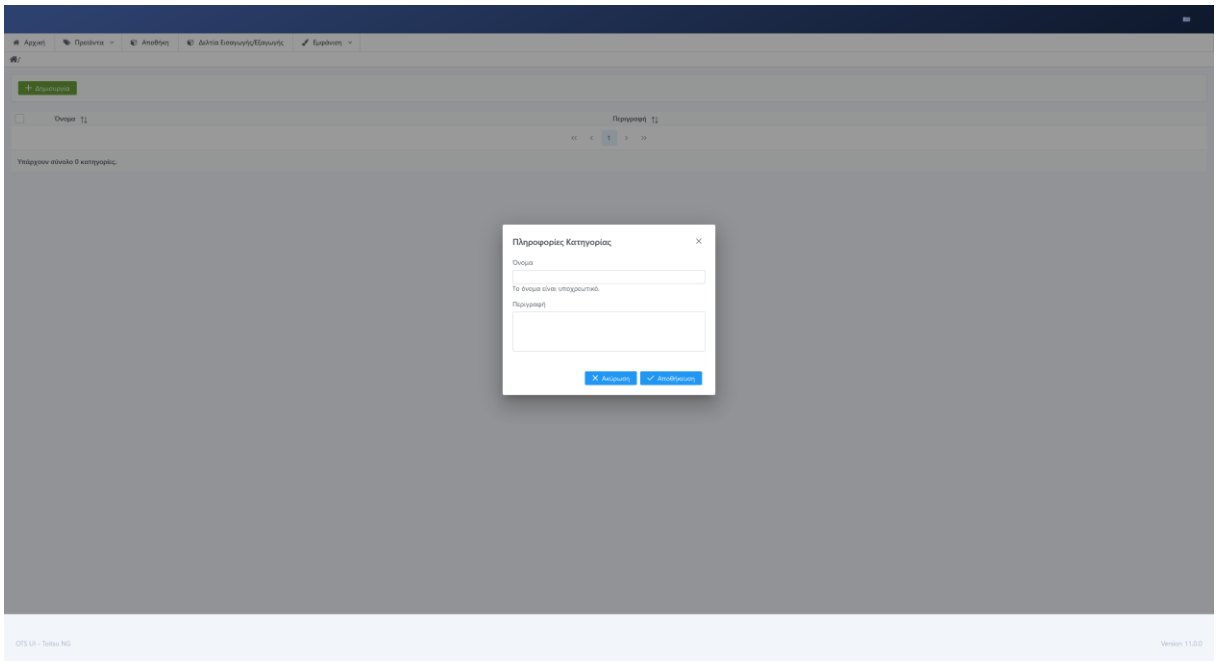
5.3 Κατηγορία Προϊόντος

6.3.1 Ευρετήριο

Στο ευρετήριο εμφανίζεται μια λίστα με όλες τις κατηγορίες που είναι καταχωρημένες στην εφαρμογή. Οι εγγραφές ανακτούνται όλες με το άνοιγμα του παραθύρου και υπάρχει ένα πεδίο για τη γρήγορη αναζήτηση κάποιας από αυτές. Η εμφάνιση των κατηγοριών γίνεται σε λίστα που εμφανίζει το όνομα και την περιγραφή τους.

6.3.2 Λειτουργίες CRUD

Ο χρήστης μπορεί να δημιουργήσει και να καταχωρήσει κατηγορίες προϊόντων(Εικόνα 6.4) που θα χρησιμοποιήσει για να κατηγοριοποίησει τα προϊόντα που θα εισάγει στη συνέχεια. Τα πεδία που πρέπει να συμπληρωθούν είναι πολύ απλά και δύο στο πλήθος τους. Αυτά είναι ένα **όνομα και μια περιγραφή.** Ο χρήστης έχει τη δυνατότητα της επεξεργασίας και διαγραφής μιας κατηγορίας.

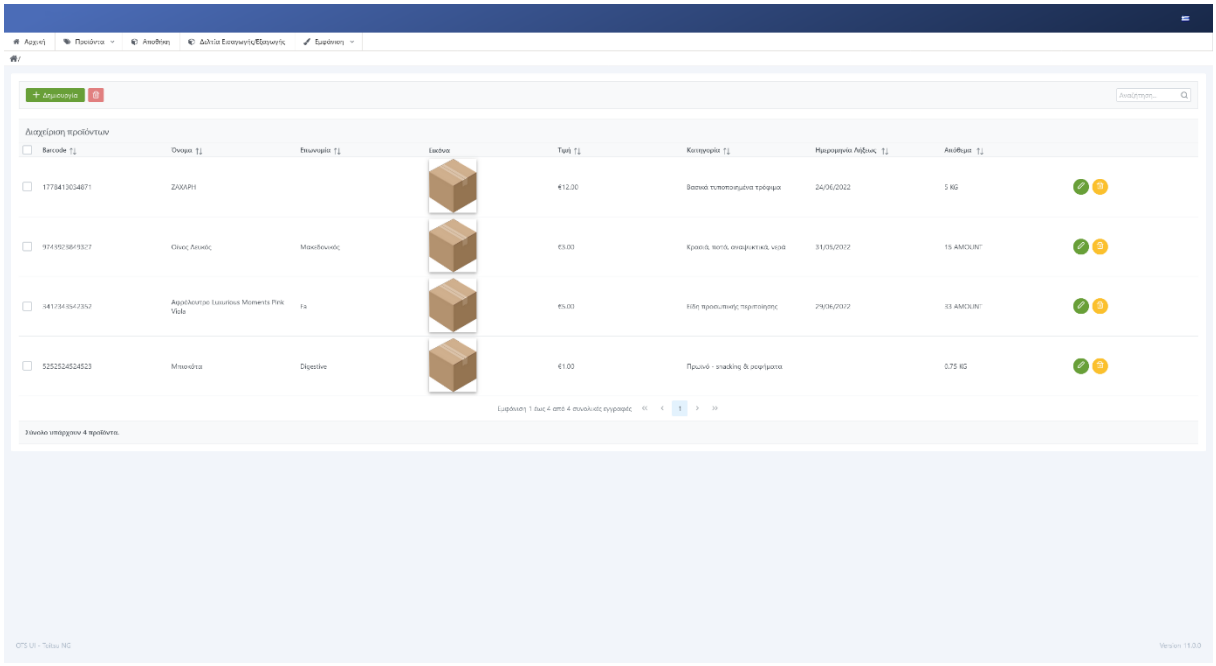


Εικόνα 6.4 Δημιουργία κατηγορίας προϊόντος

6.4 Προϊόν

6.4.1 Ευρετήριο

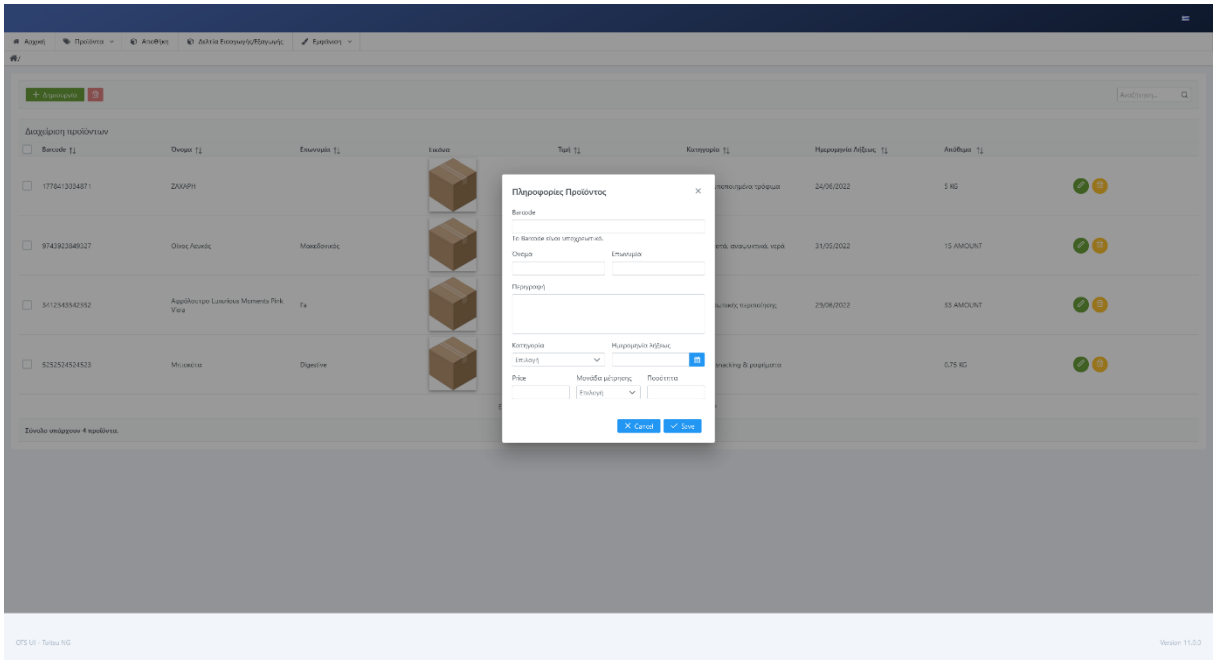
Στο ευρετήριο εμφανίζεται μια λίστα με όλες τα προϊόντα(Εικόνα 6.5) που είναι καταχωρημένα στην εφαρμογή. Οι εγγραφές ανακτούνται όλες με το άνοιγμα του παραθύρου και υπάρχει ένα πεδίο για τη γρήγορη αναζήτηση κάποιας από αυτές. Η εμφάνιση των προϊόντων γίνεται σε λίστα που εμφανίζει κάποια βασικά πεδία και την εικόνα του κάθε προϊόντος.



Εικόνα 6.5 Ευρετήριο Προϊόντων

6.4.2 Λειτουργίες CRUD

Ο χρήστης μπορεί να δημιουργήσει και να καταχωρήσει προϊόντα(Εικόνα 6.6) που θα χρησιμοποιήσει για να κατηγοριοποίησει τα προϊόντα που θα εισάγει στη συνέχεια. Τα πεδία που πρέπει να συμπληρωθούν είναι **barcode, όνομα, επωνυμία, κατηγορία, ημερομηνία λήξης, τιμή, είδος μέτρησης, ποσότητα και περιγραφή προϊόντος.** Ο χρήστης έχει τη δυνατότητα της επεξεργασίας και διαγραφής ενός προϊόντος.



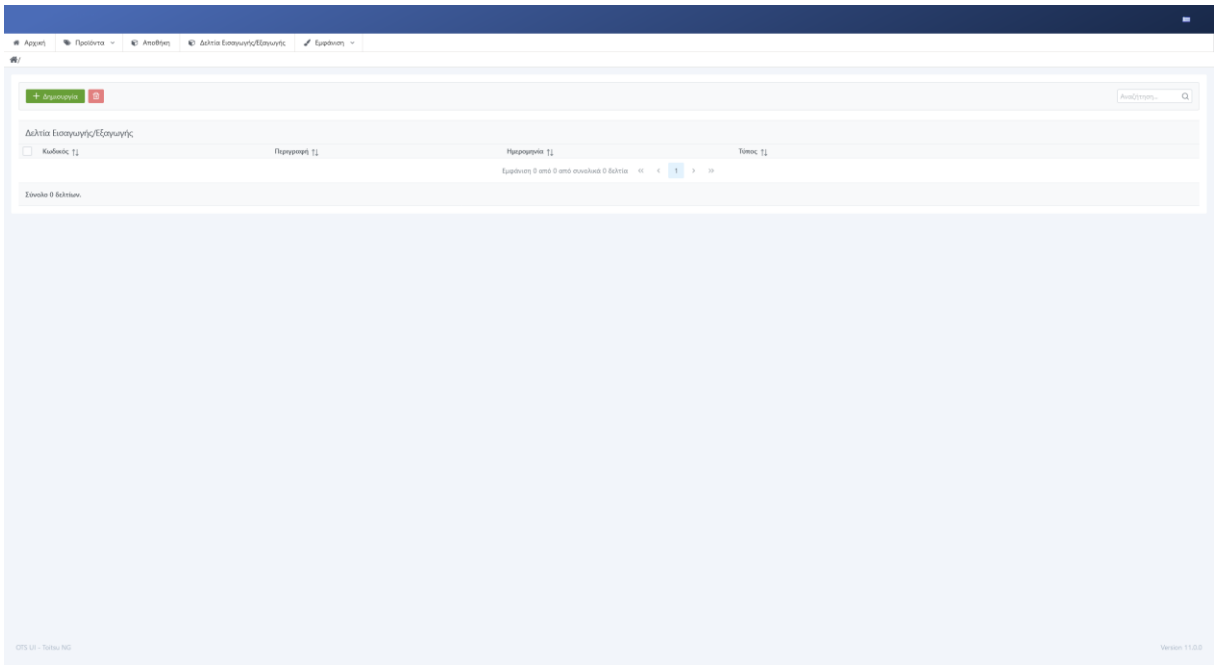
Εικόνα 6.6. Δημιουργία/Εμφάνιση στοιχείων προϊόντος

5.5 Δελτία Εισαγωγή/ Εξαγωγής Προϊόντων

Ένα δελτίο κίνησης βοηθάει το χρήστη στην καταγραφή της εισαγωγή και εξαγωγής των προϊόντων σε μια βιβλιοθήκη. Αυτό βοηθάει στη διαχείριση του αποθέματος αφού εύκολα μπορούμε να προβλέψουμε κα να αποφύγουμε την έλλειψη κάποιου προϊόντος.

5.5.1 Ευρετήριο

Στο ευρετήριο εμφανίζεται μια λίστα με όλα τα δελτία(Εικόνα 5.7) που έχουν καταχωρηθεί στην εφαρμογή. Οι εγγραφές ανακτούνται όλες με το άνοιγμα του παραθύρου και υπάρχει ένα πεδίο για τη γρήγορη αναζήτηση κάποιας από αυτές. Η εμφάνιση των δελτίων γίνεται σε λίστα που εμφανίζει κάποια βασικά πεδία τους.



Εικόνα 6.7 Ευρετήριο Δελτίων Εισαγωγής/Εξαγωγής

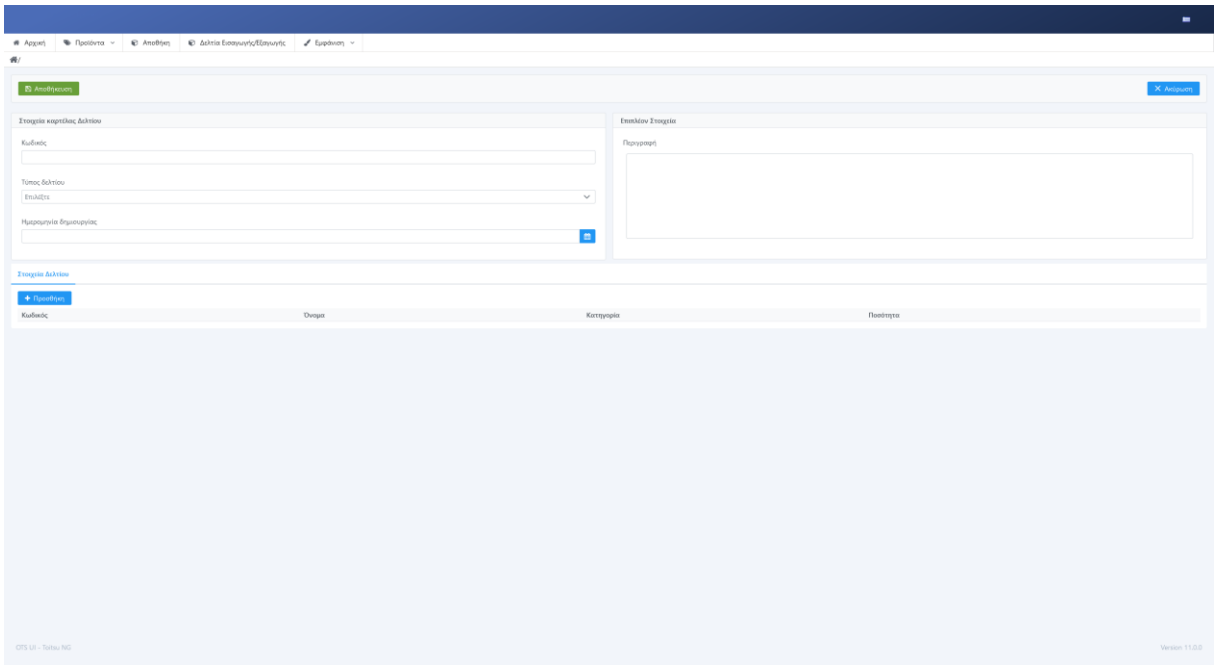
5.5.2 Λειτουργίες CRUD

Ο χρήστης μπορεί να δημιουργήσει και να καταχωρήσει ένα δελτίο κίνησης(Εικόνα 6.8) και να προσθέσει σε αυτό τα προϊόντα που θα περιλαμβάνει. Τα πεδία που πρέπει να συμπληρωθούν είναι για την κεφαλή του δελτίου ο **κωδικός δελτίου, ο τύπος του, ημερομηνία εισαγωγής/εξαγωγής, περιγραφή και ο χρήστης δημιουργίας**, ενώ για την κάθε γραμμή του δελτίου τα πεδία που μπορούν συμπληρωθούν είναι το **προϊόν** στο οποίο αναφέρεται, τη **ποσότητα** του καθώς και την **αποθηκή** και **τμήμα** στο οποίο αυτό εισάγεται ή εξάγεται αντίστοιχα.

Ο χρήστης έχει τη δυνατότητα της επεξεργασίας και διαγραφής κάποιου δελτίου κίνησης.

Επίσης με την καταγραφή κάθε δελτίου κίνησης στο σύστημα κρατάμε ένα ιστορικό παρακολούθησης που μπορεί αργότερα να ανατρέξει ο κάθε χρήστης και να πάρει πληροφορίες που θα του είναι χρήσιμες για κάποια αναφορά.

Με την εισαγωγή και εξαγωγή ενός προϊόντος ενημερώνεται αντίστοιχα και το απόθεμα για το συγκεκριμένο προϊόν για την αποθήκη που αντιστοιχεί. Με το τρόπο αυτό παρακολουθούμε σε πραγματικό χρόνο το απόθεμα και μπορούμε να προβλέψουμε τυχόν ελλείψεις που μπορεί να προκύωουν από την αύξηση της ζήτησης.



Εικόνα 6.8 Δημιουργία/Εμφάνιση δελτίου

Κεφάλαιο 7ο: Συμπεράσματα ή/και προτάσεις βελτίωσης

Το κεφάλαιο αυτό περιέχει κάποια γενικά συμπεράσματα που δημιουργήθηκαν κατά την υλοποίηση της εφαρμοφής και κάποιες προτάσεις βελτίωσεις που θα μπορούσαν να γίνουν για να αυξήσουν πιο πολύ την ασφάλεια του συστήματος.

7.1 Συμπεράσματα

Η υλοποίηση μιας cloud εφαρμογής έχεις πολλά πλεονεκτήματα σε σχέση με μια windows client based. Το σημαντικότερο πλενέκτημα της είναι, η εύκολη προσπέλαση της από οπουδήποτε βρισκόμαστε και ότι δεν χρειαζόμαστε συστήματα μεγάλων απαιτήσεων για να την τρέξουν, αφού παίζει σε κάθε browser.

Η υλοποίηση τέτοιου είδους εφαρμογών, σύμφωνα με τα παραπάνω συνεπάγεται ότι με λίγες μόνο αλλαγές μπορούν να τρέξουν σε οποιαδήποτε συσκευή που έχει σύνδεση στο διαδίκτυο.

Η αξιοποίηση του διαδικτύου μας δίνει τη δυνατότητα να μπορούμε να κάνουμε πολύ πιο εύκολα εγκατάστασεις τέτοιων εφαρμογών εύκολα και γρήγορα, ανεξαρτήτων το που βρίσκεται ο πελάτης. Επίσης μας καθιστά πιο εύκολη τη διαδίκαση της αναβάθμισης της εφαρμογής οποτεδήποτε χρειαστεί και σε πολύ σύντομο χρονικό διάστημα, σύμφωνα πάντα και με τους πόρους που διαθέτει ο κάθε πελάτης.

Το συμπέρασμα είναι ότι Cloud εφαρμογές γίνονται όλο και πιο αναγκαίες όσο προχωράμε μέσα στο χρόνο καθώς το διαδίκτυο και η τεχνολογία εξαπλώνεται ραγδαία σε κάθε γωνιά του πλανήτη. Όστοσο η ανάπτυξη τέτοιων εφαρμογών πρέπει να γίνεται με τεράστια εστίαση στο κομμάτι της ασφάλειας του χρήστη αφού το διαδίκτυο κρύβει πολλούς κινδύνους. Πρέπει να χτίζουμε τις εφαρμογές μας με τέτοιο τρόπο που θα αποτρέπουν κάποιο κακόβουλο χρήστη ή/και λογισμικό να προκαλέσει προβλήματα στο σύστημα μας.

Η ασφάλεια των Cloud συστημάτων είναι ένα μεγάλο ζήτημα που απαιτεί τεράστια σημασία κυρίως όταν πίσω από αυτό το σύστημα βίσκονται δεδομένα που δεν πρέπει να έχει ο καθένας στην κατοχή του.

7.2 Προτάσεις βελτίωσης

Η βελτίωση που θα έκανα στο σύστημα μου είναι να προσθέσω μια αυτοματοποιημένη ενημέρωσης του χρήστη για την κατάσταση του αποθέματος του κάθε προϊόντος η οποία θα γίνεται με αλγόριθμο που θα προβλέπει αυτή την κατάσταση. Ο αλγόριθμος βλέποντας την κατάσταση του αποθέματος σε πραγματικό χρόνο και τη ζήτηση που έχει το κάθε προϊόν ξεχωριστά ανά αποθήκη θα ειδοποιεί το χρήστη όταν καταλαβάινει ότι πρόκειται να εξαντληθεί. Το σύστημα επίσης θα ενημερώνει το χρήστη για την κοντινότερη αποθήκη από την οποία μπορεί να γίνει εφοδιασμός του αντίστοιχος προϊόντος.

Για το κομμάτι της ασφάλειας του συστήματος θα έφτιαχνα ως ξεχωριστό προτζεκτ ένα σύστημα αυθεντικοποίησης του χρήστη. Η εφαρμογή μας θα διασυνδέται με το σύνστημα αυτό, με σκοπό να αυθεντικοποιήσει το χρήστη και να του δώσει πρόσβαση σε αυτή ανάλογα με το ρόλο του. Ένα τέτοιο σύνστημα θα αποτρέψει κάποιο κακόβουλο χρήστη να έχει πρόσβαση στην εφαρμογή και να προκαλέσει σημαντικά προβλήματα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Internet Site

[1] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model
[2] https://euclid.ee.duth.gr/courses/old/2007-08/Databases/DraftSlides/LecDB02-ER-Large.pdf
[3] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
[4] https://kassapoglou.github.io/spring/spring-unit1-introduction-to-spring-framework.html
[5] https://www.baeldung.com/java-jar-war-packaging
[6] https://kassapoglou.github.io/maven/maven.html
[7] https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model
[8] https://euclid.ee.duth.gr/courses/old/2007-08/Databases/DraftSlides/LecDB02-ER-Large.pdf
[9] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
[10] https://ubunlog.com/el/intellij-idea-ide-java/
[11] https://docs.microsoft.com/el-gr/power-bi/report-server/rest-api
[12] https://ubunlog.com/el/datagrip-ide-bases-de-datos/
[13] https://www.tutorialspoint.com/spring_boot/spring_boot_service_components.htm
[14] https://zetcode.com/springboot/repository/