ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών

Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού και Τεχνητής Νοημοσύνης

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ανάπτυξη Εφαρμογής Ηλεκτρονικού Εμπορίου Με Spring Framework Και Java Virtual Threads** **E-Commerce Application Development With Spring Framework And Java Virtual Threads**
Ονοματεπώνυμο Φοιτητή	**ΙΩΑΝΝΗΣ ΚΟΥΤΣΙΟΥΜΑΡΗΣ**
Πατρώνυμο	**ΜΑΡΙΝΟΣ**
Αριθμός Μητρώου	**ΜΠΣΠ2318**
Επιβλέπων	**Ευθύμιος Αλέπης, Καθηγητής**

Ημερομηνία Παράδοσης **Μάρτιος 2026**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης	Μαρία Βίρβου	Διονύσιος Σωτηρόπουλος
Καθηγητής	Καθηγήτρια	Αναπληρωτής Καθηγητής

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή έχει ως κυρίαρχο σκοπό την ανάπτυξη ενός πλήρως λειτουργικού backend μιας εφαρμογής ηλεκτρονικού εμπορίου σε Java. Η τελική εφαρμογή έχει όλες τις παραδοσιακές λειτουργίες που αντιστοιχούν σε μια εφαρμογή ηλεκτρονικού εμπορίου, όπως παραδείγματος χάρη, ανάγνωση, προσθήκη, τροποποίηση και διαγραφή προϊόντων από τους υπάλληλους και διαχειριστές, καθώς και δυνατότητες προσθήκης σε λίστα «αγαπημένων», παραγγελίας και κριτικής ενός προϊόντος από χρήστες. Κατά την υλοποίηση αυτής δόθηκε ιδιαίτερη αξία στη χρήση των Java Virtual Threads και του Spring Framework. Τα Java Virtual Threads είναι μια καινούρια τεχνολογία της Java, η οποία εμφανίστηκε πρώτη φορά στη Java 19 (Preview) και στη συνέχεια παρουσιάστηκε ολοκληρωμένη στη Java 21 (Standard). Προσφέρει ποικίλα πλεονεκτήματα τόσο για τους προγραμματιστές όσο και για το τελικό πρόγραμμα, με πρωταρχικό την αξιόλογη ελάττωση της πολυπλοκότητας και των σφαλμάτων που προκύπτουν κατά την παραλληλοποίηση λειτουργιών από τους προγραμματιστές. Παράλληλα, το τελικό πρόγραμμα είναι πιο αποδοτικό, αναγνώσιμο και εύκολα επεκτάσιμο. Το Spring Framework είναι μια τεράστια συλλογή προγραμματιστικών εργαλείων τα οποία προσφέρουν στους προγραμματιστές μια πληθώρα δυνατοτήτων που δεν θα είχαν διαφορετικά, με έναν οργανωμένο και - κατά το δυνατό - απλό τρόπο.

Abstract

This master's thesis has as its primary purpose the development of a fully functional backend for an e-commerce application in Java. The completed application includes all traditional functionalities associated with an e-commerce application, such as reading, adding, updating and deleting products by employees and administrators, as well as adding products to a «favorites» list, ordering and reviewing products by users. During the development of this application, the use of Java Virtual Threads and the Spring Framework was considered highly valuable and desirable. Java Virtual Threads is a recent Java technology, which was first introduced in Java 19 (Preview) and was later presented as finalized in Java 21 (Standard). It offers numerous advantages for both developers and the resulting program, most notable of which is a significant reduction in complexity and errors that occur during the implementation of concurrent operations by the developers. Furthermore, the completed program is more efficient, readable and scalable. The Spring Framework is an extensive collection of programming tools that provide developers with several capabilities they would not otherwise have, in an organized and – when possible – straightforward manner.

Περιεχόμενα

Εικόνες

Εισαγωγή

Το διαδίκτυο αποτελεί για τον σύγχρονο άνθρωπο ένα απαραίτητο και επιθυμητό μέσο για να εκπληρώσει τις καθημερινές του ανάγκες. Οι ανάγκες αυτές αφορούν τομείς όπως τον εργασιακό, στον οποίο είναι σύνηθες να υποχρεούνται οι εργαζόμενοι να χρησιμοποιήσουν το διαδίκτυο για να εκπληρώσουν τα καθήκοντα τους, καθώς και τους τομείς της ψυχαγωγίας και της αγοράς προϊόντων. Συνέπεια του τελευταίου, είναι ότι οι ιστοσελίδες και οι εφαρμογές ηλεκτρονικού εμπορίου έχουν αυξανόμενη απήχηση και είναι συχνά σε προτίμηση, αντί της αγοράς προϊόντων από κάποιο φυσικό κατάστημα.

Στο πλαίσιο αυτό στηρίζετε το backend μιας εφαρμογής ηλεκτρονικού εμπορίου (e-commerce) που αποτελεί την παρούσα διπλωματική εργασία. Επίκεντρο της εργασίας αποτέλεσε η χρήση σύγχρονων εργαλείων, τεχνολογιών και τεχνικών ώστε ο παραγόμενος κώδικας να είναι αποτελεσματικός, ευπαρουσίαστος και να ακολουθεί βέλτιστες πρακτικές. Ύψιστης σημασίας θεωρήθηκε η αξιοποίηση του Spring Framework και των Java Virtual Threads, των οποίων η σωστή εφαρμογή επιφέρει τρομερή αύξηση της οργάνωσης και της απόδοσης του κώδικα. Όλες οι τεχνολογίες και μεθοδολογίες που εφαρμόστηκαν αναφέρονται αναλυτικά σε επόμενα κεφάλαια.

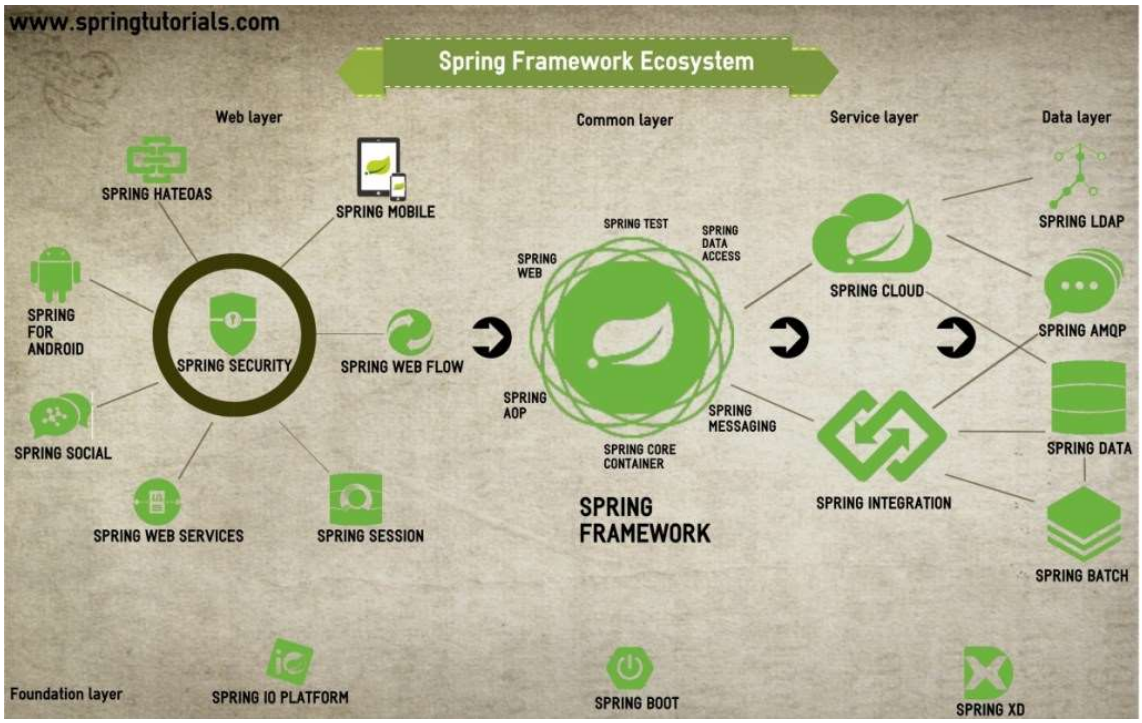
Η λειτουργικότητα της εφαρμογής προσφέρει ξεχωριστές δυνατότητες για τους χρήστες, οι οποίες εξαρτούνται από το αν ένας χρήστης είναι Υπάλληλος, Διαχειριστής ή Απλός Χρήστης. Οι απλοί χρήστες μπορούν να εκτελέσουν μόνο περιορισμένες από τις λειτουργίες που παρέχει η εφαρμογή, εκείνες που αφορούν πρωταρχικά την εγγραφή και σύνδεση τους στο σύστημα, και στη συνέχεια την ανάγνωση – αλλά όχι τροποποίηση – μη ευαίσθητων δεδομένων, ανάγνωση και τροποποίηση δικών τους στοιχείων (όπως το καλάθι αγορών τους, τις λίστες αγαπημένων προϊόντων και τις παραγγελίες τους) καθώς και αγορά και κριτική προϊόντων. Οι υπάλληλοι και σε ανώτατο βαθμό οι διαχειριστές, έχουν και συμπληρωματικές δυνατότητες που αφορούν τη διαχείριση του συστήματος, με χαρακτηριστικά παραδείγματα τη διαγραφή χρηστών και την προσθήκη ή τροποποίηση προϊόντων και εκπτώσεων.

Αρχιτεκτονική συστήματος

Η εκπόνηση της παρούσας διπλωματικής εργασίας δεν θα ήταν εφικτή χωρίς τη συνεργασία πολλών διαφορετικών τεχνολογιών που αφορούν την συγγραφή του κώδικα αλλά και εξωτερικά συστήματα. Ορισμένες από αυτές απευθύνονται στην οργάνωση, επεκτασιμότητα και τον έλεγχο του κώδικα από προγραμματιστική άποψη, ενώ άλλες απευθύνονται στην ομαλή λειτουργία του, την προσθήκη λειτουργικότητας, ή την επικοινωνία του κώδικα με ξεχωριστά συστήματα όπως μια βάση δεδομένων και έναν web server. Η επιλογή των τεχνολογιών έγινε λαμβάνοντας υπόψιν τις απαιτήσεις της εφαρμογής, σε συνδυασμό με τις βέλτιστες πρακτικές και την προτίμηση που υπάρχει για κάθε μια στην αγορά. Όλες οι τεχνολογίες που περιγράφονται παρακάτω, εφάπτονται ομαλά πάνω στη γλώσσα Java.

Spring Framework

Το Spring Framework είναι ένα μεγάλο και αναλυτικό framework ανοιχτού κώδικα με εργαλεία που απευθύνονται στην παραγωγή μοντέρνων εφαρμογών βασισμένων στη Java. Είναι εξαιρετικά δημοφιλές καθώς θεωρείται ευέλικτο και επιτρέπει επεκτάσιμο κώδικα μειωμένης πολυπλοκότητας και επαναληπτικότητας, το οποίο επιτυγχάνει με την αυτοματοποίηση πολλαπλών απαραίτητων, συνηθισμένων ή επαναλαμβανόμενων διαδικασιών. Το οικοσύστημα του Spring αποτελείται από πολλά μεμονωμένα συστατικά που με καθένα από αυτά υλοποιούνται διαφορετικές λειτουργίες. Κατά τη διάρκεια του υποκεφαλαίου αυτού αναλύονται λεπτομερώς τα συστατικά (modules) του Spring που χρησιμοποιήθηκαν για την εκπόνηση της εφαρμογής ηλεκτρονικού εμπορίου.



Εικόνα 1: Το οικοσύστημα του Spring Framework[1]

Spring Initializr

Το Spring Initializr δεν είναι απαραίτητο συστατικό για την ανάπτυξη μιας Java εφαρμογής και δεν είναι μέρος του κώδικα με αποτέλεσμα να μπορούσε να παραλειφθεί. Ωστόσο μια συνοπτική αναφορά του θεωρείται αξιοσημείωτη καθώς είναι ένα δημοφιλές και εύχρηστο εργαλείο. Είναι, λοιπόν, ένα ηλεκτρονικό εργαλείο το οποίο διευκολύνει τους προγραμματιστές (developers) να παράγουν ταχύτατα και δίχως δυσκολίες μια Spring Boot εφαρμογή, στην οποία μπορούν να επιλέξουν

απευθείας τις εξαρτήσεις (dependencies) που επιθυμούν και ρυθμίσεις που αφορούν, μεταξύ άλλων, ονόματα και περιγραφές, εκδώσεις (versions) ή εργαλεία χτισίματος (build tools). Έπειτα από την επιλογή των παραπάνω, οι προγραμματιστές κατέχουν πλέον ένα έτοιμο πρότυπο (template) , το οποίο δύνανται να εξελίξουν μέχρις ότου να παραχθεί το επιθυμητό πρόγραμμα.

Spring Boot

Ένα εξαιρετικά σημαντικό module του Spring είναι το Spring Boot. Ως κύριο στόχο έχει να μειώσει τον κόπο παραγωγής επαγγελματικών Spring εφαρμογών και υπηρεσιών (services) , το οποίο επιτυγχάνει με τις εξής τεχνικές:

- **Spring Boot Starters**: Περιλαμβάνουν δημοφιλείς βιβλιοθήκες και frameworks, ενώ η ένταξη τους σε μια εφαρμογή γίνεται με κριτήριο τις ανάγκες του συστήματος. Χαρακτηριστικά παραδείγματα είναι το Spring Boot Web Starter που διευκολύνει την υλοποίηση web υπηρεσιών (web services) και Application Programming Interfaces (API), ή το Security Starter που παρέχει δυνατότητες που αφορούν την αυθεντικοποίηση (authentication) και εξουσιοδότηση (authorization) χρηστών. Υπάρχουν συνολικά περίπου πενήντα ξεχωριστά Starters όπου το καθένα διαθέτει μοναδικές ιδιότητες.[2]
- **Αυτόματη ρύθμιση και αρχικοποίηση των εφαρμογών**: Το Spring Boot προ-εγκαθιστά ορισμένα απαραίτητα dependencies και αρχικοποιεί δεδομένα για αυτά, καθώς και για άλλες εξωτερικές βιβλιοθήκες που ενδεχομένως να συμπεριλάβει ένας προγραμματιστής. Οι επιλεγμένες ρυθμίσεις είναι δυνατό να τροποποιηθούν στη συνέχεια ανάλογα με τη κρίση του κάθε προγραμματιστή. Ανεξαρτήτως, οι προγραμματιστές επωφελούνται από μειωμένη πολυπλοκότητα και εξοικονόμηση χρόνου.
- **Standalone Spring εφαρμογές**: Δηλαδή, εφαρμογές των οποίων η λειτουργία δεν απαιτεί εξωτερικά συστήματα και βασίζεται αποκλειστικά στο περιβάλλον που προσφέρει το Spring Boot. Βασικότερο στοιχείο για την επίτευξη standalone Spring εφαρμογών η ενσωμάτωση κάποιου web server σε πολλά Spring Boot Starters, όπως το Web Starter που προσφέρει τον Tomcat web server. Παρά ταύτα, το αν η εκάστοτε εφαρμογή θα είναι standalone ή όχι καθορίζεται εν τέλει από τις ανάγκες της και τις επιλογές που έγιναν κατά την υλοποίηση της.

Το Spring Framework είναι εφικτό να χρησιμοποιηθεί δίχως το Spring Boot και, μάλιστα, η έλλειψη του συνεπάγεται αυξημένο έλεγχο πάνω στις ρυθμίσεις και τα συστήματα της εφαρμογής. Αυτό όμως συχνά δεν είναι επιθυμητό, διότι τα τελικά κέρδη που προσφέρει το Spring Boot στην ταχύτητα και απλοποίηση παραγωγής μιας εφαρμογής τείνουν να υπερτερούν σε σχέση με τον μειωμένο έλεγχο.[3]

Εικόνα 2: Σημαντικά οφέλη του Spring Boot[4]

Στη συνέχεια, περιγράφονται τα Spring Boot Starter dependencies που εντάχθηκαν στην εφαρμογή της παρούσας διπλωματικής εργασίας και τα χαρακτηριστικά που διαθέτουν.

Spring Boot Web Starter

Ολα τα Spring Boot Starters είναι dependencies που περιλαμβάνουν πολλά μικρότερα dependencies που τυπικά συνδυάζονται, ώστε να επιτευχθούν κάποιες συγκεκριμένες λειτουργικότητες. Το Web Starter περιλαμβάνει τα παρακάτω dependencies που απευθύνονται στην δημιουργία web εφαρμογών.

1) **Spring Web MVC**: Σε αυτό στηρίζονται web εφαρμογές που αφορούν Model-View-Controller (MVC) αρχιτεκτονική. Εν συντομία, τα models αφορούν την δομή των δεδομένων και την εγκυρότητα τους, τα views απευθύνονται στην διεπαφή του χρήστη και τον τρόπο που παρουσιάζονται σε αυτόν τα δεδομένα, ενώ τα controllers δέχονται εισερχόμενα HTTP αιτήματα (requests), σύμφωνα με τα οποία καλούν τις κατάλληλες λειτουργίες και τελικά επιστρέφουν στον χρήστη κάποια αποτελέσματα.
2) **Ενσωματωμένο Servlet Container**: Το προεπιλεγμένο servlet container του Web starter είναι το Tomcat, επιτρέπεται όμως η αντικατάσταση του από άλλα containers όπως το Jetty και το Undertow. Καθοριστικές λειτουργίες του ενσωματωμένου Tomcat, το οποίο χρησιμοποιήθηκε στο σύστημα ηλεκτρονικού εμπορίου, είναι η διαχείριση κάποιας πόρτας (port) με σκοπό να δέχεται HTTP requests, καθώς επίσης η διαχείριση των ίδιων των requests και νημάτων (threads).
3) **Jackson**: Οι βιβλιοθήκες Jackson απευθύνονται στην σειριοποίηση (serialization) αντικειμένων που διαχειρίζεται το πρόγραμμα σε ένα JSON και την αποσειριοποίηση (deserialization) JSON δεδομένων σε προγραμματιστικά αντικείμενα. Σε κώδικα Java, είναι

καλή πρακτική τα αντικείμενα αυτα να είναι Plain Old Java Objects (POJO) δηλαδή να είναι αντικείμενα κλάσεων οι οποίες δεν εκπληρώνουν κάποια άλλη εργασία, πέρα από το να είναι πρότυπα αντικειμένων.

4) **Logging Framework**: Εγκαθιστά το Logback framework, αποτελώντας ένα άμεσο ξεκίνημα στην καταγραφή συμβάντων.[5]

Η εφαρμογή ηλεκτρονικού εμπορίου που αποτελεί την παρούσα εργασία είναι, βεβαίως, μια web εφαρμογή και εξαρτάται από όλες τις προαναφερόμενες βιβλιοθήκες για την ορθή λειτουργία της.

Spring Boot Validation Starter

Σε αντίθεση με το Web Starter, το Validation Starter είναι πιο περιορισμένο στην λειτουργικότητα που προσφέρει, καθώς εξετάζει αποκλειστικά την επικύρωση (validation) των δεδομένων του συστήματος. Το validation των δεδομένων - που συνεπάγεται και την αξιοπιστία τους - γίνεται με το Jakarta Validation framework που εισάγεται αυτόματα, και τα λεγόμενα «annotations» με τα οποία αυτό αξιοποιείται. Η αξία του Validation Starter γίνεται ευδιάκριτη στο παράδειγμα εγγραφής ενός email, για τα οποία προϋπόθεση είναι να ακολουθούν μια συγκεκριμένη μορφή. Η μορφή αυτή εξασφαλίζεται ταχύτατα με τη χρήση του «@Email» annotation στο αντίστοιχο πεδίο.[6]

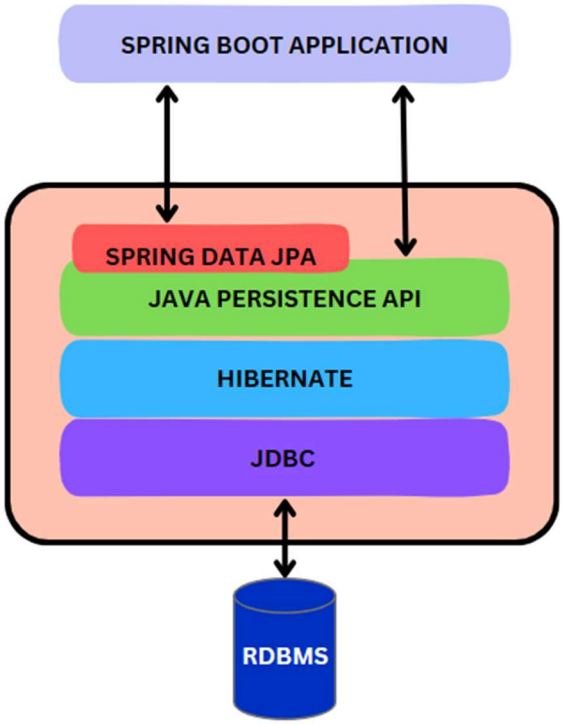
Annotation	Purpose	Example
@NotNull	Ensures the value is not null	@NotNull(message = "Name is required")
@Size	Validates the length of a string or collection	@Size(min = 3, max = 20)
@Min, @Max	Ensures a number is within a range	@Min(18) for age validation
@Email	Checks for a valid email format	@Email(message = "Invalid email")
@Pattern	Validates using a regex pattern	@Pattern(regexp = "\\d{10}")

Εικόνα 3: Ενδεικτικά Jakarta Validation annotations[7]

Spring Boot Data JPA Starter

Περιλαμβάνει το Spring Data JPA και dependencies που εξασφαλίζουν την αποτελεσματική λειτουργία του. Το Spring JPA (Java Persistence API) ουσιαστικά απλοποιεί την χρήση των Object-Relational Mapping (ORM) εργαλείων με αρχικοποιημένα δεδομένα και ρυθμίσεις, καθώς και annotations που επικοινωνούν με αυτά χωρίς την ανάγκη μεγάλης ποσότητας κώδικα.

Τα ORMs είναι εργαλεία ή τεχνικές που γεφυρώνουν τα προγραμματιστικά αντικείμενα που περιέχουν οι Object Oriented (OO) γλώσσες προγραμματισμού, όπως είναι η Java, με μια σχεσιακή βάση δεδομένων, αλλά και το αντίστροφο. Το προ εγκατεστημένο ORM που χρησιμοποιείται από το Data JPA Starter είναι το Hibernate, υπάρχει όμως η επιλογή αντικατάστασης του με κάποιο άλλο ORM. Επιπρόσθετα, το Data JPA Starter περιλαμβάνει ένα dependency για το Java Database Connectivity (JDBC) που επιτυγχάνει την σύνδεση με την βάση δεδομένων με το ίδιο το σύστημα και την εκτέλεση SQL queries.[8][9]



Εικόνα 4: Τα περιεχόμενα του Spring Data JPA και η επικοινωνία τους με το σύστημα[10]

Spring Boot Test Starter

Το τελευταίο Starter που εισάχθηκε στην εφαρμογή ηλεκτρονικού εμπορίου είναι το Test Starter. Συνενώνει πολλές βιβλιοθήκες που αφορούν την δημιουργία test του κώδικα και της αλληλεπίδρασης του με εξωτερικά συστήματα, συμπεριλαμβανομένου μεταξύ άλλων, τις φημισμένες JUnit, Hamcrest και Mockito. Τα tests του κώδικα και του καθολικού συστήματος παράλληλα με την υλοποίηση του, είναι καθοριστικός παράγοντας στην ταχύτατη εξιχνίαση προγραμματιστικών και λογικών σφαλμάτων, για μικρά και ιδιαίτερα για μεγάλα συστήματα. Αποτρέπουν επίσης σφάλματα κατά τη μελλοντική συντήρηση ή διόρθωση του συστήματος.[11]

Στο πλαίσιο της παρούσας διπλωματικής, το Test Starter εισάχθηκε με ορατότητα των βιβλιοθηκών του μόνο στα tests που παράχθηκαν, όπως άλλωστε συνηθίζεται, και χρησιμοποιήθηκαν ιδιαίτερα οι βιβλιοθήκες JUnit και Mockito για την παραγωγή λεγόμενων Unit tests. Κάθε Unit test επαληθεύει την ορθή λειτουργία ενός μικρού τμήματος κώδικα, αγνοώντας όμως παράγοντες όπως την επικοινωνία του κώδικα με εξωτερικά συστήματα.

Spring Security

Ένα άλλο αξιόλογο module του Spring είναι το Spring Security. Είναι προφανές ότι η ασφάλεια ενός συστήματος και η προστασία του από κακόβουλα πρόσωπα είναι ύψιστης αξίας. Μάλιστα, όσο αυξάνεται το μέγεθος του συστήματος και των δεδομένων που διαχειρίζεται, τόσο αυξάνεται και ανάγκη της ασφάλειας των δεδομένων αυτών. Το Spring Security είναι καθοριστικό εργαλείο για την ασφάλεια των Spring εφαρμογών, διότι προσφέρει έναν αποτελεσματικό τρόπο να ικανοποιηθούν οι ανάγκες αυθεντικοποίησης (authentication) και εξουσιοδότησης (authorization) μιας εφαρμογής, όπως επίσης η προφύλαξη της από πολλές κοινές επιθέσεις ασφαλείας.

Κατά την διαδικασία συγγραφής του κώδικα της διπλωματικής, χρησιμοποιήθηκαν τρία dependencies που ανήκουν στο Spring Security. Security Core, Security Web και Security Config.[12]

- Το Security Core περιλαμβάνει συστατικά που είναι απαραίτητα για το συνολικό Spring Security και επομένως, είναι υποχρεωτική προσθήκη για την αξιοποίηση του Spring Security και των υπόλοιπων υπό συστατικών του.
- Το Security Web αναφέρεται στην ασφάλεια μιας εφαρμογής κατά την επικοινωνία της με το δίκτυο. Παραδείγματος χάρη, χρησιμοποιείται για να ορίσει «φίλτρα» , όπου ένα φίλτρο μπορεί να είναι η επιτρεπόμενη χρήση κάποιον από τα URL της εφαρμογής μόνο από αυθεντικοποιημένους χρήστες.
- Το Security Config επιτρέπει την ενεργοποίηση και μετατροπή διαφόρων ρυθμίσεων ασφαλείας.[13]

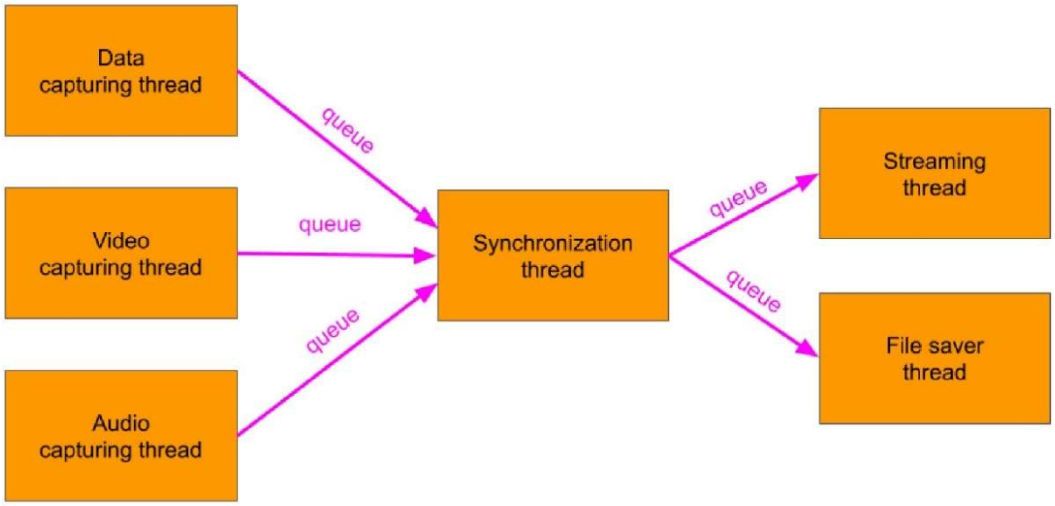
Virtual Threads

Εκτός του Spring Framework, δόθηκε υψηλή αξία και στην αξιοποίηση των Java Virtual Threads. Προτού αναλυθούν εις βάθος τα Virtual Threads, οφείλει να γίνει αναφορά στις έννοιες των νημάτων (threads) και του multi-threading.

Η έννοια του νήματος απευθύνεται σε ένα πολύ μικρό και οικονομικό - ως προς τη χρήση υπολογιστικών πόρων – σύνολο εντολών που μπορεί να εκτελέσει ένας επεξεργαστής. Το Java Virtual Machine (JVM) από το οποίο εξαρτάται η Java, χρησιμοποιεί αυτόματα τουλάχιστον ένα νήμα για να εκτελέσει τον κώδικα. Στη συνέχεια, το σύστημα και οι προγραμματιστές έχουν την δυνατότητα να παράξουν επιπλέον νήματα ώστε να επιτύχουν την παράλληλη εκτέλεση τμημάτων κώδικα. Κάθε νήμα μπορεί να διαθέτει αποκλειστικά δεδομένα όπως μεταβλητές (variables), αλλά χρησιμοποιεί κοινή μνήμη και πόρους. Αυτό συνεπάγεται πως τα νήματα επιτρέπεται να συνεργάζονται μεταξύ τους, αλλά και ότι είναι πιθανό να βρεθούν σε διαμάχη αν δεν εφαρμοστούν με προσοχή.

Η χρήση πολλαπλών νημάτων είναι μια τεχνική η οποία ονομάζεται multi-threading και αποσκοπεί σε αυξημένη απόδοση για το τελικό σύστημα. Η αυξημένη απόδοση είναι ιδιαίτερα εμφανής σε συστήματα πολλών επεξεργαστών, όπου με την ορθή εφαρμογή του multithreading θα μπορούσε να εξισορροπηθεί η κατανάλωση των πόρων του συστήματος, σε αντίθεση με το να επιβαρύνεται - ανά κάθε χρονική στιγμή – αποκλειστικά ένας επεξεργαστής με την εκτέλεση των λειτουργιών, καθώς και να ολοκληρωθεί το πρόγραμμα γρηγορότερα. [14][15]

Στην εικόνα 5 γίνεται ευδιάκριτο πως πολλαπλά νήματα θα μπορούσαν να λειτουργούν παράλληλα για αυξημένη αποτελεσματικότητα του συστήματος. Κάθε νήμα αναλαμβάνει μια λειτουργία ανεξάρτητη από τις υπόλοιπες και την εκτελεί. Όταν όλα τα νήματα έχουν ολοκληρώσει τη λειτουργία τους, κάποιο νήμα συλλέγει τα αποτελέσματα τους (το synchronization thread) και συνεχίζει την εκτέλεση του υπόλοιπου συστήματος, εισάγοντας τα προηγούμενα αποτελέσματα ως δεδομένα στις επερχόμενες λειτουργίες.



Εικόνα 5: Παράδειγμα χρήσης multi-threading[16]

Μέχρι προσφάτως, η Java υποστήριζε ένα μοναδικό είδος νημάτων, τα λεγόμενα platform ή classic threads που διαθέτουν τα παρακάτω βασικά χαρακτηριστικά:

- Καθένα από αυτά τα νήματα που δημιουργεί το JVM και «τρέχει» κάποια χρονική στιγμή, αντιστοιχείται απευθείας σε ένα νήμα του λειτουργικού συστήματος (Operating System Thread).
- Σε περίπτωση που σε κάποια χρονική στιγμή «τρέχουν» περισσότερα platform threads σε σχέση με τον συνολικό αριθμό των OS threads που διαθέτει το σύστημα, τότε οι πόροι του συστήματος δεν αρκούν για να υποστηρίξουν την παράλληλη εκτέλεση όλων των platform threads, με αποτέλεσμα να υπάρχει ανταγωνισμός ανάμεσα τους.
- Όταν ένα platform thread κατά την εκτέλεση του περιμένει κάποια άλλη λειτουργία ή σύστημα να ολοκληρωθεί και να του επιστρέψει απαραίτητα δεδομένα, δεν αναστέλλει την λειτουργία του. Παρότι δεν καταναλώνει πόρους του επεξεργαστή, εξακολουθεί να χρησιμοποιεί ένα OS thread, το οποίο κατά συνέπεια δεν μπορεί να αξιοποιηθεί αποδοτικά για κάποιο άλλο platform thread. Το OS thread απελευθερώνεται μόνο όταν το αρχικό platform thread ολοκληρώσει την εκτέλεση του.
- Η δημιουργία των platform threads είναι μια απαιτητική διαδικασία για το σύστημα. Επομένως, η πρακτική που ακολουθείται τυπικά είναι η δημιουργία περιορισμένου αριθμού platform threads, τα οποία κάθε φορά που τερματίζουν την εκτέλεση τους δεν καταστρέφονται, αλλά είναι πάλι διαθέσιμα για να ξεκινήσουν κάποια άλλη λειτουργία.[17]

Η κυκλοφορία της έκδοσης Java 19 στις 20 Σεπτεμβρίου του 2022, παρουσίασε για πρώτη φορά (σε πρώιμο επίπεδο) ένα καινούριο είδος νημάτων του JVM, τα Java Virtual Threads. Ολοκληρωθήκαν με τη Java 21 ένα χρόνο μετά, στις 19 Σεπτεμβρίου του 2023, και πρωταρχικός τους σκοπός ήταν να διευκολύνουν την συγγραφή και την συντήρηση εφαρμογών παράλληλης εκτέλεσης κώδικα. Τα κύρια χαρακτηριστικά τους παρατίθενται ως εξής:

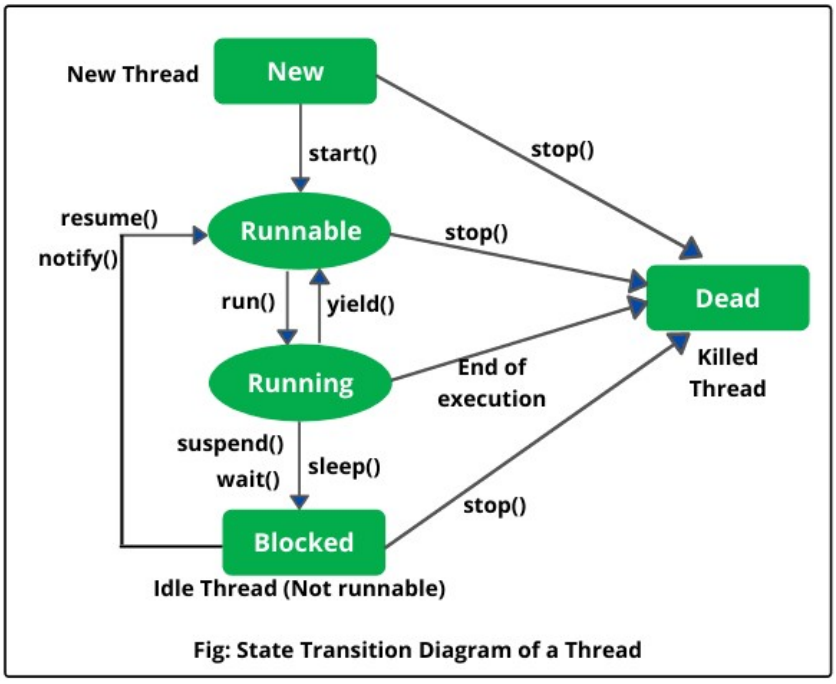
- Τα virtual threads αντί να αντιστοιχίζονται σε OS Threads, αντιστοιχίζονται με τον ίδιο τρόπο σε platform threads.
- Όταν ένα virtual thread κατά την εκτέλεση του περιμένει κάποια άλλη λειτουργία να του επιστρέψει δεδομένα, αναστέλλει τη λειτουργία του προσωρινά και σταματάει να καταναλώνει πόρους και να απασχολεί το platform thread που το διαχειριζόταν, μέχρις ότου να λάβει τα δεδομένα αυτά. Τότε, «ξυπνάει» και χρησιμοποιεί ξανά κάποιο platform thread (είτε το προηγούμενο είτε κάποιο διαφορετικό) για να συνεχίσει την εκτέλεση του. Παρά ταύτα, οφείλει να γίνει αναφορά στο γεγονός ότι υπάρχουν κάποιες – πολύ περιορισμένες –

περιστάσεις όπου τα virtual threads όταν βρίσκονται σε αναμονή κάνουν το λεγόμενο «pinning» και δεν απελευθερώνουν τα platform/OS threads. Μια τέτοια περίσταση ήταν η εκτέλεση τμημάτων κώδικα που ανήκουν μέσα στη λέξη-κλειδί «synchronized» , κάτι που άλλαξε στην πορεία με την εισαγωγή της έκδοσης 24 της Java.[18]

- Όπως και τα platform threads, στο ενδεχόμενο όπου «τρέχουν» μια χρονική στιγμή περισσότερα virtual threads συγκριτικά με τα OS threads που διαθέτει το σύστημα, τότε δεν θα είναι εφικτή η ταυτόχρονη λειτουργία τους και θα υπάρχει ανταγωνισμός ανάμεσα τους. Ωστόσο, τα virtual threads είναι σχεδιασμένα να εκτελούν μικρά τμήματα κώδικα, μέσα στα οποία θα αναγκάζονται να περιμένουν δεδομένα. Όπως αναφέρθηκε προηγουμένως, υπό αυτές τις συνθήκες τα virtual threads είναι ικανά να αναστέλλουν τη λειτουργία τους, με αποτέλεσμα να τείνουν να φτάνουν σε πολύ υψηλότερο πληθυσμό δίχως πρόβλημα ανταγωνισμού, αφού δεν χρησιμοποιούν platform/OS threads καθ' όλη τη διάρκεια ζωής τους.
- Η δημιουργία των virtual threads είναι υπολογιστικα πολύ πιο οικονομική διαδικασία σε σύγκριση με τα platform threads. Απόρροια αυτού, είναι πως δεν συνιστάται η επαναχρησιμοποίηση τους. Προτιμότερη είναι η καταστροφή και επαναδημιουργία τους.
- Η χρήση τους σε λειτουργίες που απαιτούν αυξημένους υπολογιστικούς πόρους είναι λανθάνουσα. Τα virtual threads ούτε έχουν τη δυνατότητα να απελευθερώσουν τα platform threads εάν δεν περιμένουν ανεξάρτητα δεδομένα, ούτε είναι ταχύτερα από εκείνα τα threads. Επομένως, η πιθανώς αυξημένη ποσότητα virtual threads που θα «έτρεχαν» ταυτόχρονα και αδιάκοπα στο σύστημα, μπορεί ταχύτατα να καθυστερήσει σημαντικά την εκτέλεση όλων των λειτουργιών.[17][19]

Έχοντας εξερευνήσει τα χαρακτηριστικά των virtual threads, είναι ευδιάκριτο πως σε μια εφαρμογή ηλεκτρονικού εμπορίου, όπου επιβάλλεται η ταυτόχρονη εκτέλεση πολλών λειτουργιών (όπως παράλληλων αγορών προϊόντων από διαφορετικούς χρήστες) και όπου υπάρχει συχνή επικοινωνία με εξωτερικά συστήματα όπως μια βάση δεδομένων, η υλοποίηση με virtual threads παρέχει ποικίλα οφέλη στην αποδοτικότητα του συστήματος.

Ολοκληρώνοντας το κεφάλαιο των νημάτων, στην εικόνα 6 παρουσιάζεται ο κύκλος ζωής ενός νήματος. Αρχικά, όταν δημιουργείται είναι σε κατάσταση «New» και εξελίσσεται γρήγορα στην κατάσταση «Runnable» μόλις είναι έτοιμο να εκτελέσει κάποιο τμήμα κώδικα. Στη συνέχεια, όταν κάποιος επεξεργαστής εκτελέσει το νήμα, αυτό τροποποιείται σε κατάσταση «Running». Εάν ο επεξεργαστής σταματήσει να το διαχειρίζεται τότε το νήμα επιστρέφει πάλι σε κατάσταση «Runnable» , ενώ αν το νήμα αναστείλει τη λειτουργία του και βρίσκεται σε αναμονή, τότε είναι σε κατάσταση «Blocked». «Ξυπνώντας» από την κατάσταση «Blocked» , το νήμα μόλις είναι έτοιμο να συνεχίσει φτάνει ξανά σε κατάσταση «Runnable». Τελικά, με την ολοκλήρωση της εκτέλεσης του νήματος, ή με την οριστική διακοπή του οποτεδήποτε κατά τη διάρκεια του κύκλου ζωής του, το νήμα σταματάει και καταστρέφεται φτάνοντας την κατάσταση «Dead».



Εικόνα 6: Ο κύκλος ζωής μιας διεργασίας[20]

Build Tools

Η σύγχρονη ανάπτυξη Java εφαρμογών έχει καθιερωθεί να αξιοποιεί κάποιο «build tool» για τον τελικό έλεγχο και τη μεταγλώττιση του πηγαίου κώδικα σε κάποιο εκτελέσιμο αρχείο. Η εκτέλεση των tests του κώδικα, η μετατροπή του σε γλώσσα μηχανής (compilation) και η οργάνωση του σε πακέτα, είναι κοπιαστικές διαδικασίες όταν γίνονται χειροκίνητα. Αντιθέτως, build tools όπως τα δημοφιλή Gradle και Maven ανταποκρίνονται στην ανάγκη αυτοματοποίησης των διαδικασιών αυτών με πολλαπλούς τρόπους. Τα πλεονεκτήματα των build tools παρατίθενται αναλυτικά ως εξής:

1. Ένα βασικό θετικό χαρακτηριστικό τους είναι πως εξοικονομούν μεγάλη ποσότητα χρόνου και κόπου για τους προγραμματιστές, καθώς με τη χρήση τους οι διαδικασίες εκτέλεσης των τεστ, του compilation και άλλων, γίνονται ταχύτατα με μόνο λίγα πλήκτρα του υπολογιστή.
2. Απλοποιούν την διαδικασία εισαγωγής dependencies για την εφαρμογή. Με ελάχιστο κώδικα τα εργαλεία αυτά βρίσκουν από το διαδίκτυο τα επιθυμητά dependencies και τα εισάγουν στην εφαρμογή. Ταυτόχρονα, εξασφαλίζεται ότι κάθε προγραμματιστής που διαχειρίζεται την εφαρμογή θα έχει εγκατεστημένα τα συγκεκριμένα dependencies στις συγκεκριμένες εκδόσεις τους.
3. Ελαχιστοποιούν σε ανώτατο βαθμό τα ανθρώπινα σφάλματα. Αρχικά, ελαττώνουν την κακή επικοινωνία και τα σφάλματα που ενδεχομένως να πρόκυπταν στα πλαίσια μιας ομάδας εξαιτίας των προαναφερόμενων διαφοροποιήσεων στα dependencies της εφαρμογής. Επιπρόσθετα, ελαχιστοποιούν τα ανθρώπινα σφάλματα που πιθανώς να παράγονταν από το χειροκίνητο compilation και την παράληψη τελικών tests από απερισκεψία.
4. Επιπρόσθετα, τα build tools εξασφαλίζουν παρόμοια αποτελέσματα κατά την εκτέλεση της εφαρμογής ανεξαρτήτως λειτουργικού συστήματος.
5. Τελευταία, είναι απαραίτητα για αυτοματοποιήσεις των ελέγχων και του deployment της εφαρμογής κάθε φορά που δημοσιεύεται καινούριος κώδικας. Παραδείγματος χάρη, για γρήγορα και ορθά updates ή μεταγενέστερες διορθώσεις.[21]

Η αξία των build tools παρουσιάζεται και συνοπτικά στην εικόνα 7.

Εικόνα 7: Πλεονεκτήματα των build tools[21]

Το Gradle και το Maven είναι και τα δυο ποιοτικά εργαλεία που μπορούν να ικανοποιήσουν τις ανάγκες της εφαρμογής ηλεκτρονικού εμπορίου. Ανάμεσα τους, επιλέχθηκε το Gradle λόγω αυξημένης προσωπικής εμπειρίας, καθώς και προτίμησης στην γλώσσα Groovy που αυτό αξιοποιεί, σε αντίθεση με την XML του Maven.

JSON Web Tokens

Τα JSON Web Tokens (JWT) είναι μια αποδοτική και ασφαλής τεχνική για τη μεταφορά δεδομένων ως ένα JSON αντικείμενο. Είναι ιδιαίτερα διαδεδομένα για χρήση σε web εφαρμογές και APIs, ώστε να επιτυγχάνεται η αυθεντικοποίηση και η εξουσιοδότηση των χρηστών. Υπάρχουν δυο διαφορετικοί μέθοδοι για τη κατασκευή ενός ασφαλούς JWT, που αφορούν τον τρόπο με τον οποίο τα tokens υπογράφονται.

Η πρώτη μέθοδος αφορά ένα «μυστικό κλειδί» που διαθέτει ο δημιουργός του token. Με τη συνεισφορά κάποιου κρυπτογραφικού αλγόριθμου και του μυστικού κλειδιού, το token «υπογράφεται» ψηφιακά. Στη συνέχεια, για να επαληθευθεί ότι το token δημιουργήθηκε από την αναμενόμενη οντότητα και ότι τα δεδομένα του δεν έχουν τροποποιηθεί αφότου έχει γίνει η υπογραφή του, ελέγχεται χρησιμοποιώντας πάλι το ίδιο μυστικό κλειδί. Η επαλήθευση γίνεται από το σύστημα που κατασκεύασε το JWT, καθώς το μυστικό κλειδί πρέπει να παραμένει προστατευμένο από εξωτερικά συστήματα και πρόσωπα, αφού σε αντίθετη περίπτωση θα ήταν αδύνατη η ταυτοποίηση του δημιουργού.

Η δεύτερη μέθοδος αντικαθιστά το μυστικό κλειδί με έναν συνδυασμό κλειδιών, ένα ιδιωτικό (private) και ένα δημόσιο (public). Το private κλειδί είναι κρυφό και ανήκει στον κατασκευαστή του JWT, ο οποίος με αυτό υπογράφει ψηφιακά το token. Ωστόσο, το private κλειδί δεν επαναχρησιμοποιείται για την επαλήθευση του. Η επαλήθευση του token απαιτεί ένα συγκεκριμένο public κλειδί, το οποίο μπορεί να διανεμηθεί δίχως κίνδυνο ασφαλείας. Δηλαδή, υπάρχει διαφοροποίηση στο κλειδί που παράγει το JWT και στο κλειδί που το επαληθεύει, επιτρέποντας έτσι την ασφαλή εξακρίβωση του δημιουργού του και των δεδομένων του από πολλαπλά εξωτερικά συστήματα ή χρήστες.

Κάθε JWT αποτελείται από τρία απαραίτητα στοιχεία, το Header, το Payload και το Signature, τα οποία διαχωρίζονται με το σύμβολο της τελείας.

- **Header**: Παρέχει εισαγωγικές πληροφορίες για το ίδιο το Token, όπως τον τύπο του και τον αλγόριθμο που εφαρμόστηκε για την υπογραφή του.
- **Payload**: Η κατασκευή ενός JWT γίνεται με σκοπό την ασφαλή μεταφορά δεδομένων. Τα δεδομένα αυτά βρίσκονται στο payload. Ενδεικτικά, τέτοια δεδομένα ενδεχομένως να είναι

το όνομα κάποιου χρήστη, αν αυτός είναι διαχειριστής ή οχι, η ημερομηνία και ώρα παραγωγής και λήξης του token.

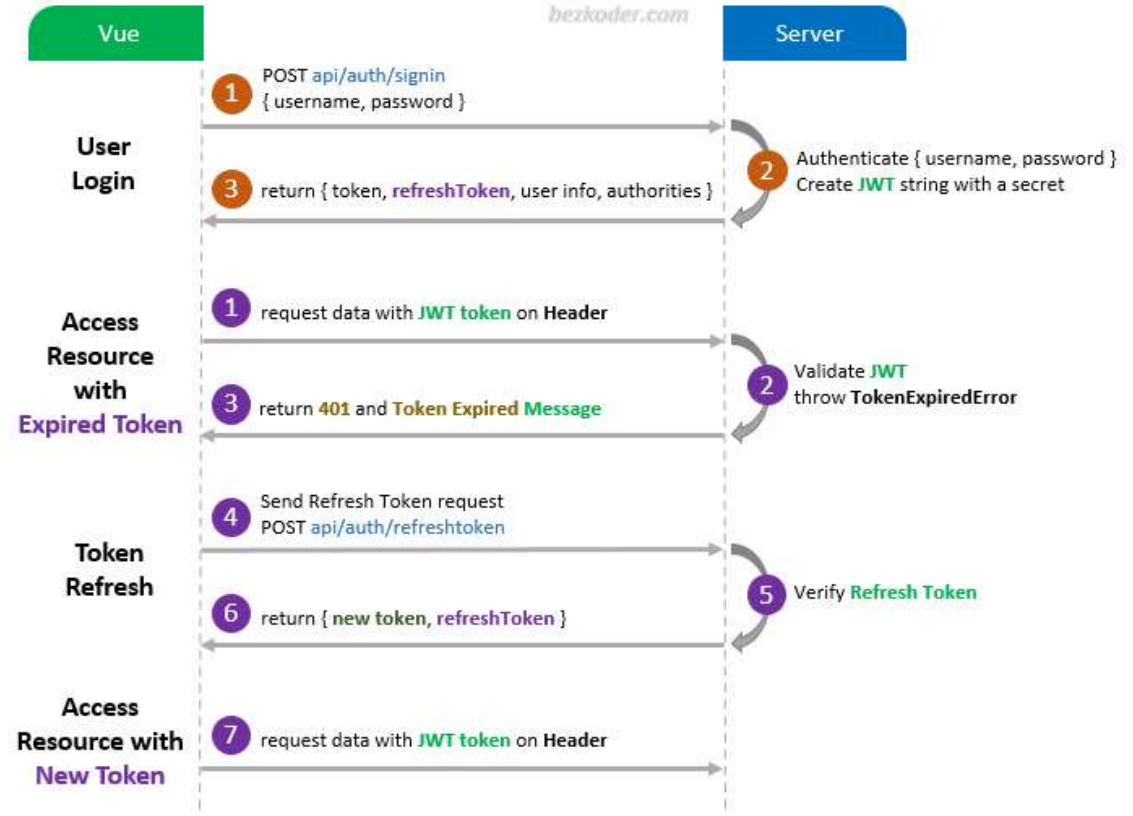
- **Signature**: Το signature είναι η ψηφιακή υπογραφή του token. Παράγεται από κάποιον κρυπτογραφικό αλγόριθμο συγχωνεύοντας το header, το payload και το μυστικό/ιδιωτικό κλειδί.

Παρότι τα JWT παρέχουν ασφάλεια σε τομείς τροποποίησης και αξιοπιστίας των πληροφοριών τους, το header και το payload τους είναι δυνατό να αποκωδικοποιηθούν σε μορφή JSON από οποιονδήποτε χρήστη. Κατά συνέπεια, δεν συνιστάται η αποθήκευση ευαίσθητων δεδομένων σε αυτά. Σε περίπτωση που η απόκρυψη των δεδομένων θεωρείται απαραίτητη, είναι εφικτό να επιτευχθεί αυτό συμπεριλαμβάνοντας και κρυπτογράφηση του token. Τότε, μετά την υπογραφή του με το private κλειδί του, ο αποστολέας θα το κρυπτογραφήσει με το public κλειδί του παραλήπτη. Όταν ο παραλήπτης λάβει το JWT θα το αποκρυπτογραφήσει με το private κλειδί του και θα το επαληθεύσει με το public κλειδί του αποστολέα.[22]

Κατά την υλοποίηση της εφαρμογής ηλεκτρονικού εμπορίου, όταν κάνει login κάποιος χρήστης λαμβάνει αμέσως δυο JWTs, ένα «access token» σε μορφή κειμένου καθώς και ένα «refresh token» ως cookie. Το access token έχει μικρή διάρκεια ζωής - ώστε να προστατεύεται από κακόβουλα πρόσωπα - και χρησιμοποιείται από τον χρήστη σε μεταγενέστερα requests που κάνει στο σύστημα για να αποδειχθεί η ταυτότητα του και να εξεταστεί το επίπεδο πρόσβασης του. Να γίνει, δηλαδή, πρώτα το authentication και στη συνέχεια το authorization του. Αφότου έχει λήξει το access token ενός χρήστη, παύει να γίνεται αποδεκτό για αυθεντικοποίηση και πρέπει να παραχθεί καινούριο, το οποίο είναι εφικτό με δυο τρόπους. Εάν απουσιάζει κάποιο έγκυρο ή μη ληγμένο refresh token, τότε ο χρήστης οφείλει να κάνει πάλι login. Ωστόσο, αν διαθέτει ένα μη ληγμένο refresh token cookie τότε παρέχοντας αυτό το token στο σύστημα, ο χρήστης δύναται να αποκτήσει ένα καινούριο access token και ένα νέο refresh token cookie, ενώ το προηγούμενο διαγράφεται από τη βάση δεδομένων, με αποτέλεσμα να μην θεωρείται πλέον έγκυρο.

Η τεχνική όπου το refresh token διαγράφεται ή γίνεται ανενεργό μετά την παραγωγή κάποιου καινούριου, ονομάζεται «refresh token rotation» και αποσκοπεί πάλι στην προστασία των tokens από κακόβουλα πρόσωπα, δεδομένου φυσικά ότι τηρούνται τα κατάλληλα μέτρα ασφαλείας κατά την αποθήκευση, προβολή και χρήση τους. Η αποθήκευση του ως ένα cookie που είναι αόρατο από την γλώσσα Javascript και μπορεί να αποσταλεί και να αποθηκευτεί μόνο σε σελίδες που ακολουθούν το HTTPS πρωτόκολλο, αποτελούν ένα τέτοιο μέτρο ασφαλείας. Επιπλέον, να σημειωθεί πως η βέλτιστη χρήση των refresh token προϋποθέτει ότι θα γίνεται αυτόματος εντοπισμός του ληγμένου access token από το σύστημα και αυτόματη ανανέωση του, αν υπάρχει ένα έγκυρο refresh token. Ωστόσο, για λόγους απλότητας η παρούσα εφαρμογή δεν υποστηρίζει αυτή την αυτοματοποίηση. Σε περίπτωση που ο χρήστης κάνει αποσύνδεση από τον λογαριασμό του, διαγράφονται αμέσως τα refresh tokens του από τη βάση.

Στην εικόνα 8 παριστάνεται ένα παράδειγμα χρήσης των JWT access και refresh tokens. Ξεκινώντας, κάποιος χρήστης συνδέεται στον λογαριασμό του, αυθεντικοποιείται και λαμβάνει το access και refresh token του. Στη συνέχεια κάποιος χρήστης επιχειρεί να λάβει δεδομένα από το σύστημα κατέχοντας ένα ληγμένο access token. Τότε το σύστημα δεν του δίνει πρόσβαση αλλά τον ενημερώνει για το σφάλμα. Στα βήματα 4,5 και 6 αποστέλνεται στο σύστημα το έγκυρο refresh token του χρήστη με αποτέλεσμα εκείνος να λαμβάνει πάλι καινούρια και έγκυρα access και refresh tokens. Στο βήμα 7 ο χρήστης πλέον ζητάει ξανά δεδομένα στο σύστημα με το νέο, έγκυρο access token του.



Εικόνα 8: Παράδειγμα χρήσης access και refresh tokens[23]

PostgreSQL

Η βάση δεδομένων που επιλέχθηκε για την υλοποίηση της εφαρμογής ηλεκτρονικού εμπορίου είναι η PostgreSQL. Είναι μια αξιόπιστη, ανοιχτού κώδικα, σχεσιακή βάση δεδομένων η οποία είναι επεκτάσιμη (scalable) και αποδοτική σε συστήματα που υποστηρίζουν πολλαπλές, ταυτόχρονες συναλλαγές (transactions). Το γεγονός πως είναι σχεσιακή βάση αποτελεί ανεκτίμητο προσόν για την παρούσα εφαρμογή, αφού οι σχεσιακές βάσεις έχουν τη δυνατότητα να αποτυπώνουν και να διαχειρίζονται καλύτερα δεδομένα που σχετίζονται άμεσα και αλληλεπιδρούν μεταξύ τους. Χαρακτηριστικό παράδειγμα του συσχετισμού των δεδομένων είναι η ύπαρξη χρηστών, οι οποίοι πραγματοποιούν και κατέχουν παραγγελίες που αφορούν προϊόντα. Κατά συνέπεια, για μια εφαρμογή ηλεκτρονικού εμπορίου η χρήση κάποιας μη σχεσιακής βάσης δεδομένων, όπως της MongoDB, θα ήταν εφικτή όμως δεν θα μπορούσε να ανταπεξέλθει με παρόμοια προγραμματιστική ευκολία και επιτυχία στις ανάγκες της εφαρμογής.[24]

Ανάπτυξη Και Παρουσίαση Συστήματος

Αφότου πλέον έχει γίνει αναλυτική επεξήγηση όλων των τεχνολογιών από τις οποίες αποτελείται η εφαρμογή ηλεκτρονικού εμπορίου, στο κεφάλαιο αυτό εξετάζεται πως οι τεχνολογίες υλοποιήθηκαν στην πράξη και γίνεται παρουσίαση του συνολικού συστήματος

Πρακτική υλοποίηση

Καταρχάς, είναι απαραίτητο να δοθεί στο Spring πρόσβαση στη PostgreSQL βάση δεδομένων.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/eCommerceAPIdb
spring.datasource.username=koutsioj
#for security reasons, the password is retrieved from an environment variable on the pc
spring.datasource.password=${ECOMMERCE_API_POSTGRESQL_PASSWORD}
spring.datasource.driver-class-name=org.postgresql.Driver
```

Εικόνα 9: Ρυθμίσεις πρόσβασης στη βάση δεδομένων

Με αυτές τις εντολές στο «application.properties» του Spring ορίζονται το url, username και password μιας PostgreSQL βάσης δεδομένων που έχει δημιουργηθεί σε προγενέστερο χρόνο ώστε να επιτευχθεί η σύνδεση.

Τα αρχεία «application.properties» και «build.gradle» περιλαμβάνουν γενικές ρυθμίσεις και θα παρουσιάζονται τμήματα τους όταν αυτό κρίνεται σημαντικό κατά την επερχόμενη ανάλυση της λειτουργίας του συστήματος.

Η λειτουργικότητα αποτελείται από πέντε κυρίαρχα πακέτα που περιλαμβάνουν υπό-πακέτα και κλάσεις:

1. Application
2. Configurations
3. Domain
4. Service
5. Web

Application Πακέτο

Το πρώτο πακέτο περιλαμβάνει μόνο την κλάση με την «main» μέθοδο από την οποία ξεκινάει η εφαρμογή. Εκτελώντας την, ενεργοποιείται το Spring Boot χάρη στο @SpringBootApplication annotation, που αξιοποιώντας τα επόμενα annotations εντοπίζει στα αρχεία του συστήματος και χρησιμοποιεί τις κλάσεις που έχουν τα @Component , @Entity και @Repository αντίστοιχα.

```
@SpringBootApplication  koutsioj *
@ComponentScan(basePackages = "unipi.koutsioj.eCommerceAPI")
@EntityScan("unipi.koutsioj.eCommerceAPI.domain")
@EnableJpaRepositories("unipi.koutsioj.eCommerceAPI.domain")
public class ECommerceApiApplication {
    static void main(String[] args) { SpringApplication.run(ECommerceApiApplication.class, args); }
}
```

Εικόνα 10: Annotations της main κλάσης

Το @Component δηλώνει ότι μια κλάση την εντοπίζει και την διαχειρίζεται το Spring. Το ίδιο κάνουν και τα υπόλοιπα δυο annotations, έχοντας όμως τα πρόσθετα χαρακτηριστικά ότι το @Entity δηλώνει ότι μια κλάση απευθύνεται σε POJO αντικείμενα που μπορούν να εισαχθούν σε μια βάση δεδομένων, ενώ το @Repository δηλώνει ότι η κλάση / διεπαφή (interface) αφορά την επικοινωνία αντικειμένων με τη βάση.

Configurations Πακέτο

Το Configurations πακέτο εμπεριέχει υπό-πακέτα και κλάσεις που δεν εκτελούν νέες λειτουργίες, όμως ορίζουν ρυθμίσεις ή αποτελούν προϋποθέσεις για να προχωρήσει ομαλά το σύστημα.

Το Auditing υπό-πακέτο του επιτρέπει σε αντικείμενα που εισάγονται ή τροποποιούνται στη βάση δεδομένων να λαμβάνουν αυτόματα το όνομα του χρήστη που τα ενημέρωσε τελευταίος και την ημερομηνία που αυτό συνέβη, με τη συνεισφορά των αντίστοιχων annotations «@LastModifiedBy» , «@LastModifiedDate» και του «@EntityListeners(AuditingEntityListener.class)» στις κλάσεις των αντικειμένων. Το τελευταίο ενεργοποιεί το απαραίτητο entity listener για να λειτουργήσουν ορθά τα πρώτα δυο.

Το Config υπό-πακέτο περιέχει γενικές ρυθμίσεις και πρότυπα για δημιουργία κλειδαριών για ομαλή παραλληλοποίηση λειτουργιών. Οι κλειδαριές θα εξηγηθούν παραπάνω στη συνέχεια.

Το Authentication υπο-πακέτο έχει περισσότερο ενδιαφέρον καθώς αποτελείται από πολλές κλάσεις που απευθύνονται κυρίως στην αυθεντικοποίηση, αλλά και στην εξουσιοδότηση των χρηστών. Η κλάση «AccessValidator» αναλαμβάνει το τμήμα της εξουσιοδότησης και, όταν χρειάζεται, απαγορεύει απλούς χρήστες να λάβουν πρόσβαση σε δυνατότητες που προορίζονται για υπάλληλους και διαχειριστές. Η κλάση AuthConfiguration έχει την ιδιαιτερότητα ότι «τρέχει» αυτόματα μόλις τρέξει η εφαρμογή, το οποίο οφείλεται στα τρία annotations στην εικόνα.

```
@Configuration  no usages  koutsioj
//disables some default security setting and the app uses settings from this class instead
@EnableWebSecurity
//enables security for methods, allowing them to use annotations like @PreAuthorize
@EnableMethodSecurity(prePostEnabled = true, securedEnabled = true, jsr250Enabled = false)
public class AuthConfiguration {
```

Εικόνα 11: Configuration Annotations

Καθήκον της είναι η εκτέλεση του securityFilterChain bean το οποίο ορίζει σε ποια requests του χρήστη πρέπει να γίνεται αυθεντικοποίηση του, προτού δοθεί σε αυτόν πρόσβαση, καθώς επίσης ρυθμίσεις που απευθύνονται στην ορθή λειτουργία των JWT.

```
@Bean  no usages  koutsioj
SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http.csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf.disable()) //we use JWT tokens in the authorization header, we don't use or need csrf tokens
            .authorizeHttpRequests(( AuthorizationManagerRequestMat... authorize) -> {
                authorize.requestMatchers( ...patterns: "/auth/logout/**").authenticated(); //note: this needs to be BEFORE ("/auth/**").permitAll(); or else it gets ignored
                authorize.requestMatchers( ...patterns: "/auth/**").permitAll(); //no auth needed for these requests
                //authorize.requestMatchers(HttpMethod.OPTIONS, "/**").permitAll();
                authorize.dispatcherTypeMatchers(DispatcherType.ERROR).permitAll(); //allows error messages, otherwise they would be be displayed as "401 unauthorized" with no body
                authorize.anyRequest().authenticated(); //any request that doesn't match the requestMatchers needs authentication
            });

    http.httpBasic(Customizer.withDefaults()) //http basic authentication, instead of 'formLogin' for example
            .sessionManagement( SessionManagementConfigurer<HttpSecurity> session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS)) //don't create session (we use JWT instead)
            .addFilterBefore(authenticationFilter, UsernamePasswordAuthenticationFilter.class); //add the custom filter BEFORE the standard auth filter in the chain, so that this one is use

    return http.build(); //build and return the custom authentication chain filter (it's used internally)
}
```

Εικόνα 12: Μεθοδος securityFilterChain

Μια ακόμα κλάση που ξεκινάει αυτόματα είναι η AdminInit του πακέτου Initializers. Ωστόσο, η αιτία που ξεκινάει αυτόματα δεν οφείλεται σε κάποιο annotation, οφείλεται στο γεγονός ότι κάνει implement την ApplicationRunner. Η AdminInit χρησιμοποιεί το annotation @Autowired, που αποτελεί ουσιαστικά τον τρόπο με τον οποίο το Spring συνδέει διαφορετικές κλάσεις αντί να δηλώνονται ρητά αντικείμενα τους, καθώς και το @Value annotation με το οποίο ορίζει τιμές μεταβλητών από το αρχείο «application.properties» όπου δηλώνονται. Κατά την αυτόματη εκτέλεση της μεθόδου «run», εξετάζεται εάν υπάρχει ήδη στη βάση κάποιος χρήστης με ρόλο διαχειριστή

(admin) και σε περίπτωση που δεν υπάρχει, παράγεται και αποθηκεύεται ένας καινούριος με προκαθορισμένα δεδομένα.

```
@Component  1 usage  koutsioj *
public class AdminInit implements ApplicationRunner {
    @Autowired  2 usages
    private UserRepository userRepository;
    @Autowired  1 usage
    private PasswordEncoder passwordEncoder;

    @Value("${admin.username}")  1 usage
    private String username;
    @Value("${admin.email}")  1 usage
    private String email;
    @Value("${admin.password}")  1 usage
    private String password;

    private static final Logger log = LoggerFactory.getLogger(AdminInit.class);  1 usage

    @Transactional  koutsioj *
    @Override
    public void run(ApplicationArguments args) {
        if (userRepository.existsByUserRolesContaining(User.UserRole.ADMIN)) {
            return;
        }
        User admin = new User();
        Set<User.UserRole> adminRole = new HashSet<>();
        adminRole.add(User.UserRole.ADMIN);
        adminRole.add(User.UserRole.EMPLOYEE);
        admin.setUsername(username);
        admin.setEmail(email);
        admin.setPassword(passwordEncoder.encode(password));
        admin.setUserRoles(adminRole);

        userRepository.save(admin);
        log.debug("Admin initialized.");
    }
}
```

Εικόνα 13: Κλάση παραγωγής του πρώτου διαχειριστή

Οι μεταβλητές που αφορούν τον admin στο «application.properties» λαμβάνονται από τον υπολογιστή που εκτελεί το πρόγραμμα μέσω των Environment Variables των Windows, για λόγους προστασίας των δεδομενων:

```
#--------------ADMIN INITIALIZATION-------#
admin.username=${ECOMMERCE_ADMIN_USERNAME}
admin.password=${ECOMMERCE_ADMIN_PASSWORD}
admin.email=${ECOMMERCE_ADMIN_EMAIL}
```

Εικόνα 14: Στοιχεία του πρώτου διαχειριστή

Επιστρέφοντας ξανά στο authentication πακέτο, οι κλάσεις CustomUserDetails και CustomUserDetailsService αποσκοπούν στο να λαμβάνουν έναν χρήστη από τη βάση δεδομένων από το username ή το email του και να επιστρέφουν σημαντικά δεδομένα για αυτόν. Ολοκληρώνοντας με το authentication, η κλάση JWTAuthProvider απευθύνεται στα JWT tokens – παραδείγματος χάρη, στην παραγωγή και την επικύρωση τους – ενώ η JWTAuthFilter λαμβάνει από κάθε request των χρηστών την «Authorization» κεφαλίδα (header) που περιέχει το JWT token του χρήστη - το οποίο απαιτείται από τα περισσότερα requests για να εκτελεστούν ορθά - και στη συνέχεια, αν αυτό υπάρχει και είναι έγκυρο τότε ο χρήστης αυθεντικοποιείται.

Domain Πακέτο

Το Domain επίπεδο απευθύνεται στα βασικά τμήματα της εφαρμογής, τα οποία είναι ασυσχέτιστα από εξωτερικά συστήματα και διευκρινίζουν την λογική οργάνωση και τους κανόνες που διατηρεί η εφαρμογή, ανεξαρτήτως από το πως αυτά πραγματοποιούνται με μεθόδους. Δηλαδή, περιλαμβάνει οπωσδήποτε δεδομένα όπως entities (π.χ. μια κλάση «Product» που αποτελεί το υπόδειγμα για τη δημιουργία όλων των αντικειμένων) και Data Transfer Objects (DTOs) τα οποία συνδέουν τους χρήστες με τα αντικείμενα προσφέροντας τους ορατότητα και πρόσβαση μόνο σε συγκεκριμένες πληροφορίες που χρειάζονται για τα αντικείμενα. Επίσης, στο domain επίπεδο αποθηκεύονται ακόμα τα repositories που καθιστούν την επαφή της βάσης δεδομένων με τα αντικείμενα.

Service Πακέτο

Ως Service επίπεδο ονομάζεται το επίπεδο της εφαρμογής που σχετίζεται με τις μεθόδους που χρησιμοποιούν έμπρακτα την λογική και τους κανόνες του domain επιπέδου. Επομένως, το service πακέτο εμπεριέχει οπωσδήποτε τα λεγόμενα services, τα οποία υλοποιούν τις λειτουργικότητες του συστήματος. Εκεί, λοιπόν, εκτελούνται όλοι οι έλεγχοι, η μεταφορά δεδομένων, η τροποποίηση της βάσης και ότι άλλο απαιτείται ώστε να επιτευχθούν ποικίλες δυνατότητες, κάποιες εκ των οποίων είναι η εγγραφή και σύνδεση χρηστών, η ανάγνωση, δημιουργία, τροποποίηση και διαγραφή προϊόντων και εκπτώσεων, η δημιουργία κριτικής και παραγγελιών, η ανανέωση των tokens και πολλές ακόμα. Αξιόλογοι, μάλιστα, είναι και οι τρόποι με τους οποίους τα services εκτελούν multithreading με τα Java Virtual Threads που αναλύθηκαν προηγουμένως.

Η μέθοδος «getAllCartProducts» στον κώδικα που ακολουθεί λαμβάνει τα προϊόντα που κάθε χρήστης έχει αποθηκεύσει στο καλάθι αγορών του. Τα virtual threads χρησιμοποιούνται ξεκινώντας από τη γραμμή 48 του κώδικα ώστε να υπολογιστεί γρήγορα η τιμή του κάθε προϊόντος, η οποία υπολογίζεται δυναμικά αφότου πρώτα ελεγχθεί εάν το κάθε προϊόν βρίσκεται σε ενεργή έκπτωση. Στη γραμμή 48 λοιπόν, δημιουργείται ένας «executor» ο οποίος ρυθμίζεται έτσι ώστε κάθε ενέργεια που εκτελεί, να την εκτελεί σε ένα νέο virtual thread. Για καθένα προϊόν που λήφθηκε, ο executor παράγει ένα καινούριο virtual thread που υπολογίζει την τελική τιμή του προϊόντος και μετά καταστρέφεται αυτόματα. Ο υπολογισμός αυτός είναι γρήγορος και δεν περιλαμβάνει «blocking» κώδικα, δηλαδή κώδικα που αναμένει αποτελέσματα από κάποιο άλλο σύστημα. Κατά συνέπεια, σε ένα πραγματικό σενάριο χρήσης του συστήματος, τα virtual threads δεν είναι επικερδή για την συγκεκριμένη περίσταση, παρουσιάστηκαν απλώς ως παράδειγμα χρήσης.

Στον κώδικα που ακολουθεί φαίνεται επίσης η μεταβλητή «pageable». Αν θεωρηθεί ότι ενδεχομένως να υπάρχουν πάρα πολλοί χρήστες και πάρα πολλά προϊόντα στην εφαρμογή, τότε το

να λάβει η εφαρμογή όλα τα προϊόντα στα καλάθια τους μπορεί να την καθυστερήσει σημαντικά. Για την επίλυση του προβλήματος αυτού χρησιμοποιείται το pageable, που επιτρέπει στο σύστημα να εμφανίζει τα προϊόντα σε διαφορετικές σελίδες, ώστε να μην απαιτείται από αυτό να τα λαμβάνει όλα μαζί και, επομένως, εξοικονομεί πόρους και ταχύτητα.

```
45      public Page<CartProduct> getAllCartProducts(Pageable pageable) {  2 usages  koutsioj
46          Page<CartProduct> cartProducts = cartProductRepository.findAll(pageable);
47
48          try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {
49              //calculate new price for any product currently discounted
50              for (CartProduct cartProduct : cartProducts.getContent()) {
51                  executor.execute(() -> { //run the price calculations for each cart product in a virtual thread
52                      Product product = cartProduct.getProductReference();
53                      Discount discount = cartProduct.getProductReference().getDiscount();
54
55                      Optional<BigDecimal> finalPrice = productService.calcDiscountedProductPrice(product, discount);
56                      finalPrice.ifPresent(product::setPrice);
57                      cartProduct.setPrice(product.getPrice()
58                              .multiply(BigDecimal.valueOf(cartProduct.getQuantity())));
59                  });
60              }
61          }
62          return cartProducts;
63      }
```

Εικόνα 15: παραγωγή virtual threads με executor

Ένας διαφορετικός τρόπος που υλοποιήθηκαν τα virtual threads, και μάλιστα με πραγματικό κέρδος, είναι στη μέθοδο «registerUser» η οποία αφορά την εγγραφή ενός χρήστη στο σύστημα. Στην γραμμή 84 του κώδικα παρακάτω, δημιουργείται μια μεταβλητή «taskScope» η οποία όποτε χρησιμοποιεί την μέθοδο «fork()» εκτελεί τον κώδικα που βρίσκεται μέσα σε αυτή σε ένα καινούριο virtual thread. Η μέθοδος «join()» στη γραμμή 99 υποχρεώνει το πρόγραμμα να περιμένει την – επιτυχής ή εσφαλμένη - ολοκλήρωση των virtual threads που δημιουργήθηκαν προτού προχωρήσει. Επιπλέον, η μεταβλητή αυτή ρυθμίζεται έτσι ώστε αν οποιοδήποτε virtual thread που παραχθεί καταλήξει σε σφάλμα, να αναστέλλεται αυτόματα και η λειτουργία των υπόλοιπων virtual threads.

Εν τέλει, η μεταβλητή αυτή χρησιμοποιείται ώστε να γίνει σε ένα virtual thread αναζήτηση του username με το οποίο προσπάθησε να κάνει εγγραφή ο χρήστης στη βάση δεδομένων, ενώ σε ένα δεύτερο thread να γίνει αναζήτηση του email που εισήγαγε. Αν βρεθεί χρήστης που να έχει κάνει ήδη εγγραφή προηγουμένως με το επιλεγμένο username ή email τότε αμέσως παύει το άλλο thread - αν δεν έχει ολοκληρωθεί - και εμφανίζεται στον χρήστη το αντίστοιχο ενημερωτικό μήνυμα σφάλματος χωρίς να γίνει επιτυχημένη εγγραφή του. Σε περίπτωση που δεν βρεθεί προηγούμενος χρήστης με κάποιο από τα προαναφερόμενα στοιχεία, τότε συνεχίζεται ομαλά το πρόγραμμα και γίνεται επιτυχής εγγραφή του χρήστη στη βάση δεδομένων. Καθώς η εκτέλεση των virtual threads περιλαμβάνει αναζήτηση στη βάση δεδομένων (δηλαδή blocking κώδικα) και είναι εφικτό να υπάρχουν πάρα πολλοί εγγεγραμμένοι χρήστες στην εφαρμογή, αυτή τη φορά η χρήση των Java virtual threads επιφέρει αληθινό κέρδος ταχύτητας.

```java
@Transactional  4 usages  koutsioj *
public String registerUser(@RequestBody UserRegistrationDTO request) throws BadRequestException {
    log.info("User registration starting");
    //execute the 2 callables in 2 different virtual threads and finish early if one of them throws an exception
    try (StructuredTaskScope taskScope = StructuredTaskScope.open()) { //default "open()" behaviour is scope failure if a subtask fails

        taskScope.fork(() -> {
            if (userRepository.findByUsername(request.username()).isPresent()) {
                throw new BadRequestException("Username is already taken");
            }
            return null;
        });
        taskScope.fork(() -> {
            if (userRepository.findByEmail(request.email()).isPresent()) {
                throw new BadRequestException("Email is already taken");
            }
            return null;
        });

        taskScope.join();
    }
    catch (StructuredTaskScope.FailedException e) { //if an exception was thrown in a subtask, throw it for the controller to catch
        if (e.getCause() instanceof BadRequestException) {
            throw new BadRequestException(e.getCause().getMessage());
        }
        else {
            throw new RuntimeException(e.getCause().getMessage());
        }
    } catch (InterruptedException _) {
        //ignore
    }
```

Εικόνα 16: Παραγωγή virtual threads με StructuredTaskScope

Η εφαρμογή μιας μεταβλητής τύπου «StructuredTaskScope» , όπως ήταν η μεταβλητή taskScope, είναι ιδιαίτερα εύχρηστη. Πάρα ταύτα, οφείλεται να αναφερθεί πως η δυνατότητα αυτή παρέχεται από την έκδοση Java 25 Preview. Είναι δεδομένο ότι οι «preview» εκδώσεις προσφέρουν δυνατότητες που πιθανώς να αλλάξουν στον τρόπο υλοποίησης και λειτουργίας - ή ενδεχομένως ακόμα και να αναιρεθούν – σε επερχόμενες εκδώσεις της Java και δεν συνίστανται για περιβάλλοντα παραγωγής. Συνίστανται μόνο για δοκιμές, πειράματα και ελέγχους. Μάλιστα, η ενεργοποίηση των δυνατοτήτων τους απαιτεί μετατροπή ρυθμίσεων, το οποίο έγινε με της εξής εντολές στο αρχείο «build.gradle» :

```
// enable preview features
tasks.withType(JavaCompile).configureEach { JavaCompile it ->
    options.compilerArgs += "--enable-preview"
}
tasks.withType(Test).configureEach { Test it ->
    jvmArgs += "--enable-preview"
}
tasks.withType(JavaExec).configureEach { JavaExec it ->
    jvmArgs += "--enable-preview"
}
```

Εικόνα 17: Ενεργοποίηση preview δυνατοτήτων

Web Πακέτο

Ολοκληρώνοντας, το Web επίπεδο και αντίστοιχα το web πακέτο αφορούν την διαχείριση των HTTP requests που γίνονται από κάποιο άλλο σύστημα ή χρήστη. Αρχικά εξετάζουν την εγκυρότητα αυτών

και των πληροφοριών που ενδεχομένως μεταφέρουν, και ακολούθως αν όλα είναι ορθά, καλούν το κατάλληλο service για να πραγματοποιηθεί η επιθυμητή ενέργεια. Σε περίπτωση που εντοπιστεί κάποιο σφάλμα κατά τη διαδικασία αυτή, υπάρχουν «Handlers» που λαμβάνουν πολλά πιθανά σφάλματα και επιστρέφουν στον χρήστη το κατάλληλο μήνυμα κάθε φορά, χωρίς να διακόπτουν πλήρως τη λειτουργία της εφαρμογής.

Επιπρόσθετα, στο παράδειγμα που ακολουθεί παρουσιάζεται και ένας τρόπος ελέγχου εξουσιοδότησης του χρήστη/συστήματος, που εκτελείται πριν από την λειτουργικότητα διαγραφής του. Συγκεκριμένα, εξετάζεται εάν ο εγγεγραμμένος χρήστης που έκανε το request έχει δικαιώματα «ADMIN» ή «EMPLOYEE» και αν ο χρήστης αυτός έχει το ίδιο id με τον χρήστη που προσπαθεί να διαγράψει, δηλαδή, αν προσπαθεί να διαγράψει τον εαυτό του. Αν οποιαδήποτε συνθήκη από τις παραπάνω είναι αληθής, τότε το request προχωράει προς εκτέλεση. Διαφορετικά, σταματάει απευθείας και γίνεται ενημέρωση του χρήστη.

```
//deletes one user
@DeleteMapping("/{id}/delete")  no usages  koutsioj
@PreAuthorize("hasAnyAuthority('ADMIN','EMPLOYEE') or #id == principal.id")
public ResponseEntity<String> deleteUser(@PathVariable Long id) {
    log.info("Received request for /users/{id}/delete");
    userService.deleteUser(id);
    log.info("/users/{id}/delete request completed successfully");
    return ResponseEntity.ok( body: "User with id: "+id+" deleted successfully");
}
```

Εικόνα 18: @PreAuthorize annotation

Στο Web πακέτο ενεργοποιούνται επιπλέον τα virtual threads χάρη στην εντολή :

```
spring.threads.virtual.enabled=true
```

Εικόνα 19: Ενεργοποίηση virtual threads για το Spring

Η οποία βρίσκεται στο αρχείο ρυθμίσεων «application.properties» και καλεί το Spring να εφάπτει αυτόματα κάθε request που λαμβάνει σε ένα ξεχωριστό virtual thread.

Η παράλληλη εκτέλεση πολλών requests συνεπάγεται την πιθανότητα να αλληλεπιδρούσαν δίχως τη θέληση τους οι διάφοροι χρήστες μεταξύ τους, προκαλώντας ψευδής συνθήκες και λανθασμένες καταστάσεις. Ένα ενδεικτικό παράδειγμα είναι η ταυτόχρονη παραγγελία του ίδιου προϊόντος από δυο διαφορετικούς χρήστες ενώ το προϊόν διαθέτει μόνο ένα τελευταίο απόθεμα. Τότε, προτού προλάβει να ολοκληρωθεί η παραγγελία του πρώτου χρήστη και επομένως να τροποποιηθεί το απόθεμα του προϊόντος, είναι δυνατό να ξεκινήσει η παραγγελία του δευτέρου, με τελικό αποτέλεσμα να γίνουν, λανθασμένα, δεκτές και οι δυο παραγγελίες. Για την αποφυγή αυτού και άλλων παρόμοιων προβλημάτων, αξιοποιήθηκε μια τεχνική «κλειδώματος» των προϊόντων, όπου είναι απαραίτητη, η οποία αποτρέπει την παράλληλη χρήση ενός προϊόντος από πάνω από έναν χρήστη.

Στο παράδειγμα που ακολουθεί, όταν ξεκινάει μια παραγγελία, παράγεται μια μεταβλητή τύπου «ReentrantLock». Χρησιμοποιώντας το ID κάθε προϊόντος, αυτή εξασφαλίζει ότι τα προϊόντα που επιχειρεί να παραγγείλει ένας χρήστης θα «κλειδωθούν» με αποτέλεσμα να μην μπορούν να χρησιμοποιηθούν από άλλους χρήστες που προσπαθούν και αυτοί να τα κλειδώσουν, έως ότου να «ξεκλειδωθούν» από τον αρχικό χρήστη. Το σύστημα πραγματοποιεί τους ελέγχους και τις ενέργειες που απαιτεί με τα προϊόντα αυτά – όπως την τροποποίηση των αποθεμάτων τους – και μόλις ολοκληρωθούν όλες οι ενέργειες - επιτυχώς ή ανεπιτυχώς - τα προϊόντα ξεκλειδώνονται.

```
try {
    //try to lock all the needed products based on their ids
    sortedCartProducts.forEach( CartProduct cartProduct -> {
        ReentrantLock productLock = productLockManager.getLock(cartProduct.getProductReference().getId());
        log.debug("Trying to lock product with id:{}",cartProduct.getId());
        try {
            if (productLock.tryLock( timeout: 5, TimeUnit.SECONDS)) { //try for 5 seconds
                productLocks.add(productLock);
                log.debug("Locked product with id:{}",cartProduct.getId());
            } else { //if time expired throw exception
                throw new IllegalStateException("Could not get lock for product with id : "+cartProduct.getProductReference().getId());
            }
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    });

    //check that order quantity does not exceed existing quantity for any product
    sortedCartProducts.forEach( CartProduct cartProduct -> {
        Product product = cartProduct.getProductReference();
        if (cartProduct.getQuantity() > product.getAvailableInventory()) {
            throw new IllegalStateException("Requested quantity of product with id: " + cartProduct.getProductReference().getId() + " exceeds available quantity");
        } else {
            product.setAvailableInventory(product.getAvailableInventory() - cartProduct.getQuantity());
            productRepository.save(product);
            log.info("Inventory updated for product with id:{}",cartProduct.getId());
        }
    });
```

```
} finally { //Last thing before the transaction ends must be to release the locks
    log.debug("Unlocking products");
    productLocks.reversed().forEach(ReentrantLock::unlock);
}
```

Εικόνα 20: Παράδειγμα χρήσης ReentrantLock

Test Πακέτο

Το τελευταίο βασικό πακέτο, είναι το πακέτο των «test» το οποίο εμπεριέχει όλα JUnit tests του κώδικα που ενεργοποιούνται αυτόματα κατά το build της εφαρμογής με το Gradle. Οι δυο εικόνες που ακολουθούν αποτελούν ενδεικτικά tests, ενώ σε ολόκληρο το πακέτο περιλαμβάνονται παραπάνω από τριάντα unit tests.

Η πρώτη εικόνα παριστάνει ένα πολύ απλό test που τρέχει αυτόματα δυο φορές, μια για κάθε τιμή του «enum EmailAndUsername» και ελέγχει ότι αν κάποιος χρήστης επιχειρήσει να φτιάξει λογαριασμό με username ή email που είναι ήδη καταχωρημένο από άλλο χρήστη, τότε η εγγραφή του αποτυγχάνει, όπως είναι επιθυμητό.

```java
@Mock  3 usages
private RefreshTokenRepository refreshTokenRepository;
@InjectMocks //injects the @Mock fields in the corresponding fields in the authService  9 usages
private AuthService authService;

private enum EmailAndUsername {EMAIL, USERNAME}  2 usages  new *

@BeforeEach  new *
void setUp() {
    SecurityContextHolder.setContext(securityContext); //necessary for tests where authService methods use the security cont
}

@ParameterizedTest  new *
@EnumSource(EmailAndUsername.class) //runs the test once for each enum value
void registerUser_throws_whenEmailOrUsernameAlreadyExists(EmailAndUsername emailAndUsername) {
    UserRegistrationDTO userDto = new UserRegistrationDTO( username: "userX",  password: "passwordX",  email: "emailX@ABC.com");

    switch (emailAndUsername) {
        case EMAIL -> when(userRepository.findByEmail(userDto.email()))
                .thenReturn(Optional.of(new User()));
        case USERNAME -> when(userRepository.findByUsername(userDto.username()))
                .thenReturn(Optional.of(new User()));
    }
    assertThrows(BadRequestException.class,() -> authService.registerUser(userDto));
}
```

Εικόνα 21: Test ελέγχου μοναδικότητας των email και username των χρηστών

Στη δεύτερη εικόνα φαίνεται ένα πιο σύνθετο test, το οποίο ελέγχει ότι όταν ένας χρήστης επιχειρεί να προσθέσει ένα προϊόν σε μια λίστα «αγαπημένων» προϊόντων, τότε το σύστημα επαληθεύει ότι υπό κατάλληλες συνθήκες δεν προκύπτει σφάλμα και ο χρήστης έχει την πρόσβαση να εκτελέσει την ενέργεια αυτή για τη λίστα του συγκεκριμένου χρήστη που έχει επιλέξει με βάση το id του. Παράλληλα ελέγχει ότι αν όλα ολοκληρώθηκαν με επιτυχία, τότε το προϊόν πραγματικά προστέθηκε στην λίστα του επιλεγμένου χρήστη.

```java
@Test  new *
void addProductToFavorites_validatesUserAccessAndAddsToFavoriteLists_whenValid() throws IllegalAccessException, BadRequestException {
    UUID userId = UUID.fromString( name: "1");
    UUID productId = UUID.fromString( name: "2");

    Map<String, Object> userAndProductMap = new HashMap<>();

    userAndProductMap.put("user",new User());
    User user = (User) userAndProductMap.get("user");
    user.setFavoritedProducts(new HashSet<>());

    userAndProductMap.put("product",new Product());
    Product product = (Product) userAndProductMap.get("product");
    product.setUsersWhoFavoritedProduct(new HashSet<>());

    //userAndProductMap with its added values is considered the return value of findUserAndProduct when executed
    doReturn(userAndProductMap) //stubs findUserAndProduct without running it
            .when(userService).findUserAndProduct(any(), any());

    //merge is called twice in the service. This gets both the User and the Product object, whichever is given each time
    doAnswer( InvocationOnMock invocation -> invocation.getArgument( i: 0))
            .when(entityManager).merge(any());

    assertDoesNotThrow(() -> userService.addProductToFavorites(userId,productId));

    //verify the user access was checked
    verify(accessValidator, times( wantedNumberOfInvocations: 1)).validateUserAccess(userId);

    //check that the product/user were added in the proper collections
    assertTrue(user.getFavoritedProducts().contains(product));
    assertTrue(product.getUsersWhoFavoritedProduct().contains(user));
}
```

Εικόνα 22: Test ορθής προσθήκης προϊόντος στη λίστα αγαπημένων

Logback Αρχείο

Αξιοσημείωτο είναι επίσης το αρχείο «logback-spring.xml» που τροποποιεί τον τρόπο με τον οποίο δημιουργούνται, αποθηκεύονται και διαγράφονται τα Logback logs του συστήματος. Το πρώτο «appender» που παρουσιάζεται παρακάτω στη γραμμή 4, απευθύνεται στα συμβάντα που εμφανίζονται στην κονσόλα του προγράμματος, ενώ ο appender στη γραμμή 11 απευθύνεται στα συμβάντα που αποθηκεύονται σε ένα εξωτερικό αρχείο. Τα logs του αρχείου διαγράφονται αυτόματα μετά από 7 ημέρες ή αν έχουν ξεπεράσει συνολικά τα 500MB χώρου στο δίσκο. Το τελευταίο τμήμα του κώδικα περιορίζει τους τύπους των log που θα εμφανίζονται. Όπως φαίνεται από το «pattern» των logs, κάθε log συμπεριλαμβάνει μεταξύ άλλων τα id του αιτήματος που έγινε στο σύστημα, καθώς και το id του χρήστη που έκανε το αίτημα, αν αυτά είναι διαθέσιμα. Οι πληροφορίες αυτές παρέχονται αυτόματα από το σύστημα μετά την επαλήθευση του access token στην κλάση «JWTAuthFilter».

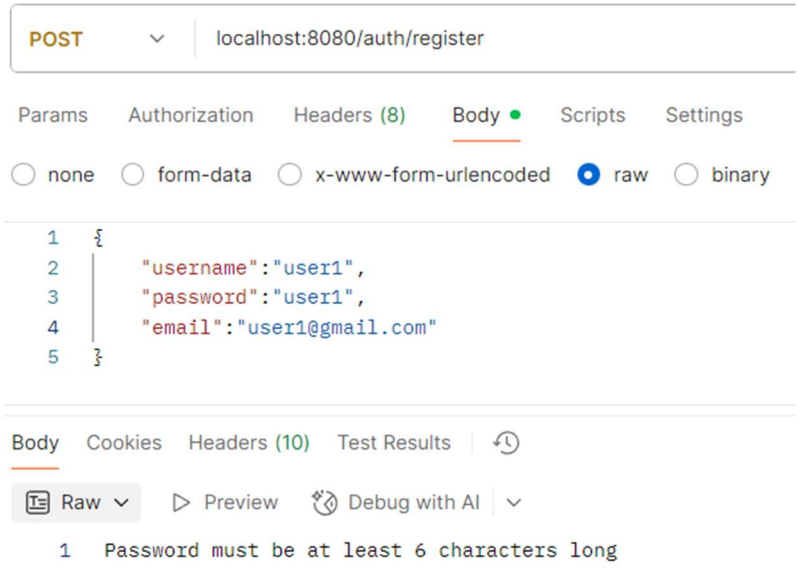
```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!-- logging config -->
<configuration>
    <appender name="console" class="ch.qos.logback.core.ConsoleAppender"> <!-- console appender -->
        <encoder>
            <!-- show date, thread, level etc -->
            <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} -- %msg -- requestId='%X{requestId}' requestUser_Id='%X{requestUserId}'%n</pattern>
        </encoder>
    </appender>

    <appender name="file" class="ch.qos.logback.core.rolling.RollingFileAppender"> <!-- rolling file appender -->
        <file>logs/app.log</file> <!-- current log file -->
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} -- %msg -- requestId='%X{requestId}' requestUser_Id='%X{requestUserId}'%n</pattern>
        </encoder>

        <!-- Automatic log deletion by date -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>logs/app-%d{yyyy-MM-dd}.log</fileNamePattern>
            <!-- uses the last time digit in fileNamePattern to determine the expiry date. Currently it's 7 days (7 * dd) -->
            <maxHistory>7</maxHistory>
            <!-- total size limit across all logs) -->
            <totalSizeCap>500MB</totalSizeCap>
        </rollingPolicy>
    </appender>

    <logger name="org.hibernate" level="ERROR"/>    <!-- automatically only showing error level logs from hibernate -->

    <!-- Root logger -->
    <root level="INFO"> <!-- Only showing info, warning and error logs -->
        <appender-ref ref="console"/>
        <appender-ref ref="file"/>
    </root>
</configuration>
```

Εικόνα 23: Logging ρυθμισεις

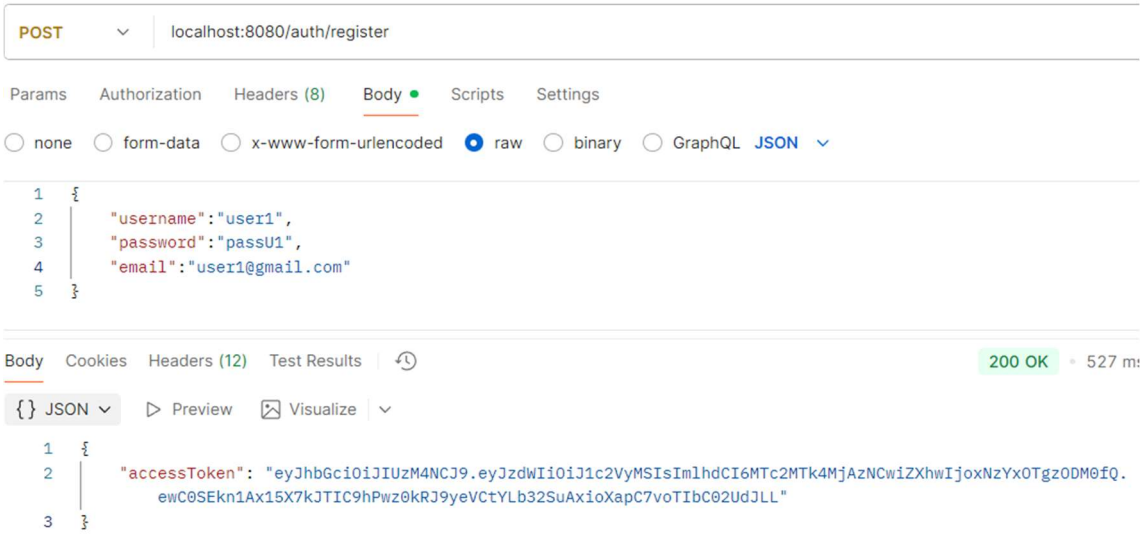
Παρουσίαση συστήματος

Σε αυτό το τμήμα παρουσιάζονται ενδεικτικά παραδείγματα ολοκληρωμένων ροών εργασιών που εκτελούνται από το σύστημα και κάποιον χρήστη. Η παρουσίαση των HTTP requests και των αποτελεσμάτων τους γίνεται με τη συνεισφορά του εργαλείου «Postman».

Πρωταρχικά, όπως αναφέρθηκε προηγουμένως, γίνεται η σύνδεση με τη βάση δεδομένων και αρχικοποιείται ένας χρήστης-διαχειριστής. Οποιοσδήποτε άλλος χρήστης για να αξιοποιήσει την εφαρμογή πρέπει να κάνει δημιουργία λογαριασμού. Πρέπει, λοιπόν, να κάνει ένα POST request στο «... /auth/register» url και να καταχωρήσει ένα username, email και password. Σε περίπτωση που το password ή το email δεν είναι έγκυρα, όπως φαίνεται παρακάτω, γίνεται ενημέρωση του χρήστη και δεν καταχωρείται στη βάση. Παρόμοιοι έλεγχοι εγκυρότητας βρίσκονται σε πολλά σημεία ολόκληρης της εφαρμογής με υποχρεωτικά πεδία, περιορισμούς ποσοτήτων και άλλα, όμως για λόγους συνοπτικότητας αυτή θα είναι η τελευταία αναφορά που θα γίνει προς αυτούς.



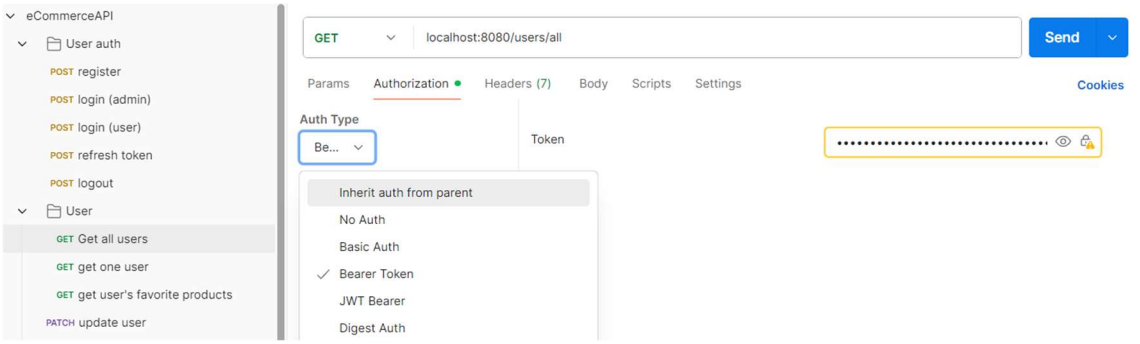
Εικόνα 24: Register με μη έγκυρο τύπο password



Εικόνα 25: Register με έγκυρα στοιχεία

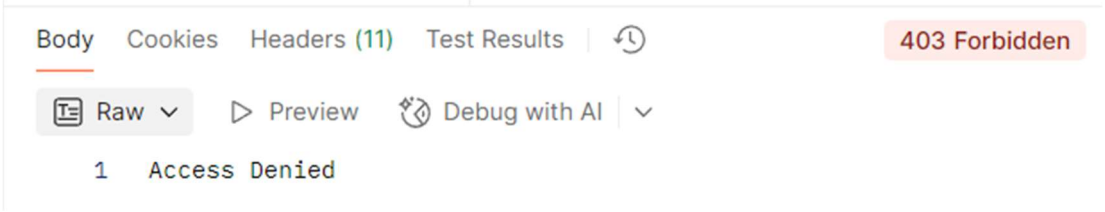
Το password του χρήστη προτού αποθηκευτεί γίνεται hash με χρήση του αλγορίθμου Bcrypt, ο οποίος είναι ένας αξιόπιστος, σκόπιμα αργός και κατά συνέπεια αποτελεσματικός αλγόριθμος για password hashing, αφού καθυστερεί τους πιθανούς επιτεθέντες.

Αφότου ο χρήστης κάνει λογαριασμό και λάβει το access token του (και το rotation token cookie) έχει με αυτό πρόσβαση σε διάφορες λειτουργίες. Παραδείγματος χάρη, με αυθεντικοποίηση με το access token του ως «Bearer token» μπορεί να επιχειρήσει να δει όλους τους χρήστες του συστήματος



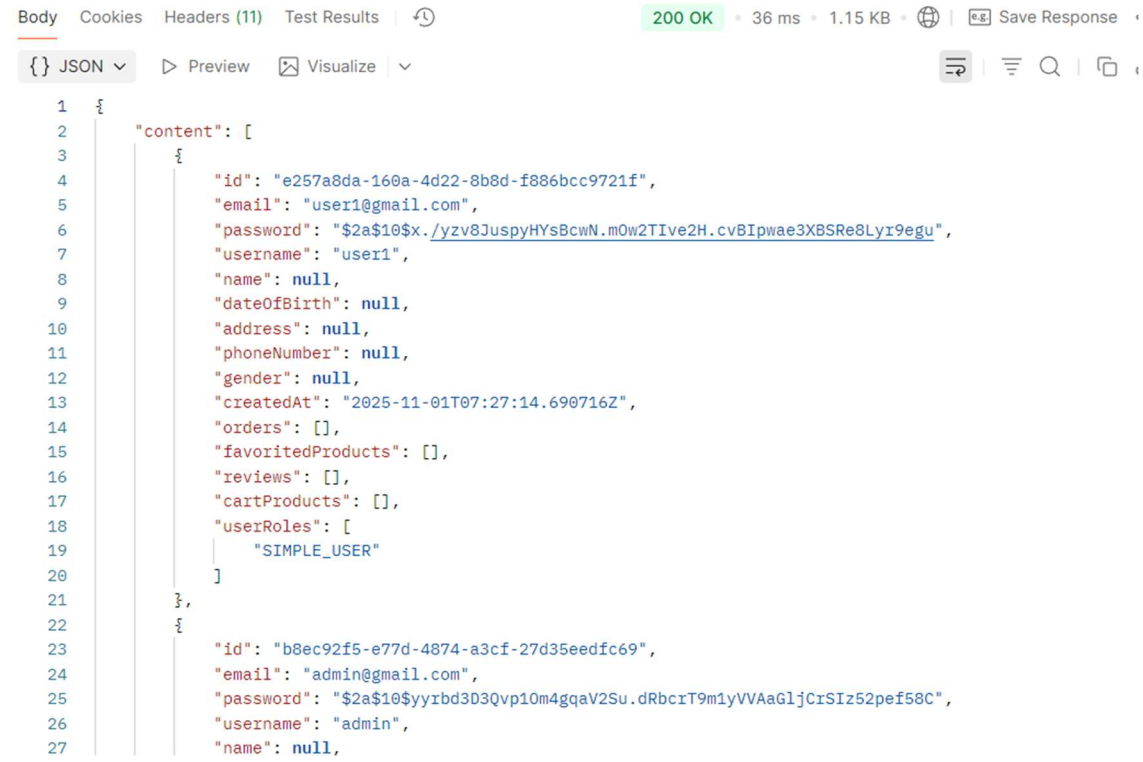
Εικόνα 26: Χρήση Json Web Token

Η λειτουργία αυτή όμως, είναι διαθέσιμη μόνο σε χρήστες-διαχειριστές, με αποτέλεσμα να λαμβάνει ένα «access denied» μήνυμα



Εικόνα 27: Απαγόρευση πρόσβασης του χρήστη

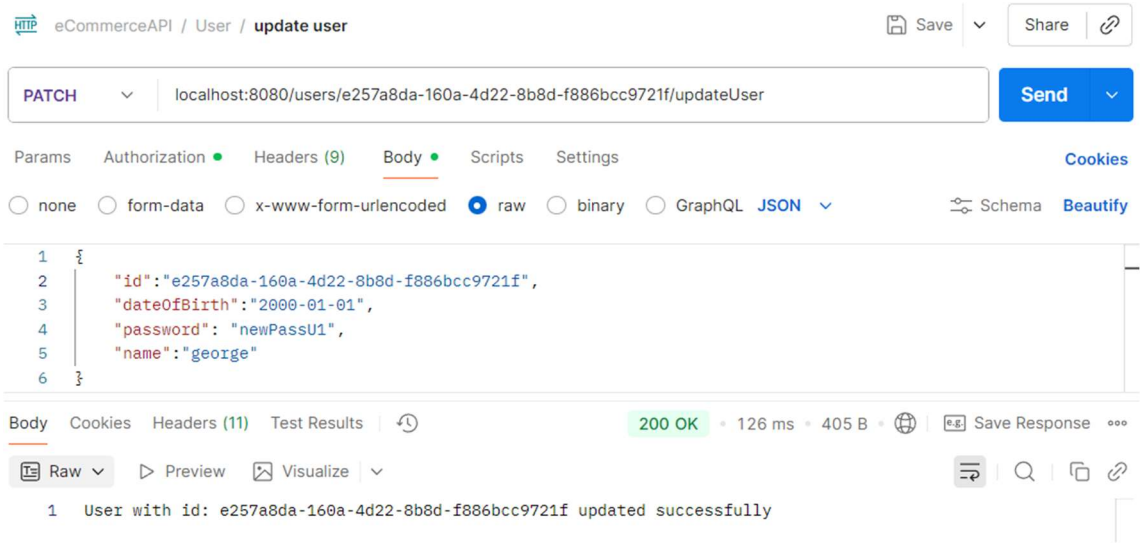
Η αποστολή του ίδιου request με ένα access token του διαχειριστή παράγει αποτελέσματα της παρακάτω μορφής



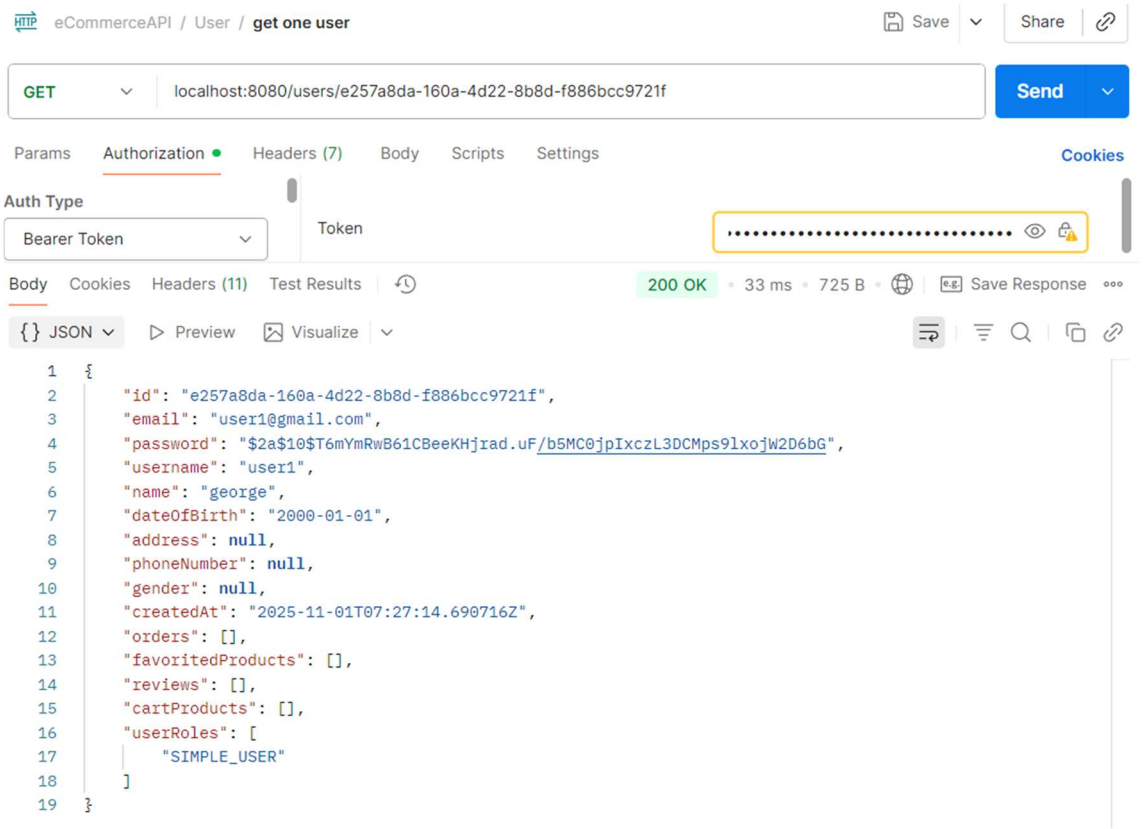
Εικόνα 28: Αποτέλεσμα επιτυχημένης ανάγνωσης όλων των χρηστών

Παριστάνονται, λοιπόν, οι χρήστες με όλα τους τα δεδομένα σε μορφή αρχείου JSON. Βεβαίως, τα passwords είναι κρυπτογραφημένα.

Ο μόνος χρήστης στον οποίο έχει πρόσβαση ένας απλός χρήστης είναι αυτός με το ίδιο του το id, δηλαδή ο εαυτός του. Στη συνέχεια, ο χρήστης φαίνεται να κάνει τροποποίηση κάποιων ενδεικτικών στοιχείων του και να βλέπει το ανανεωμένο προφίλ του.

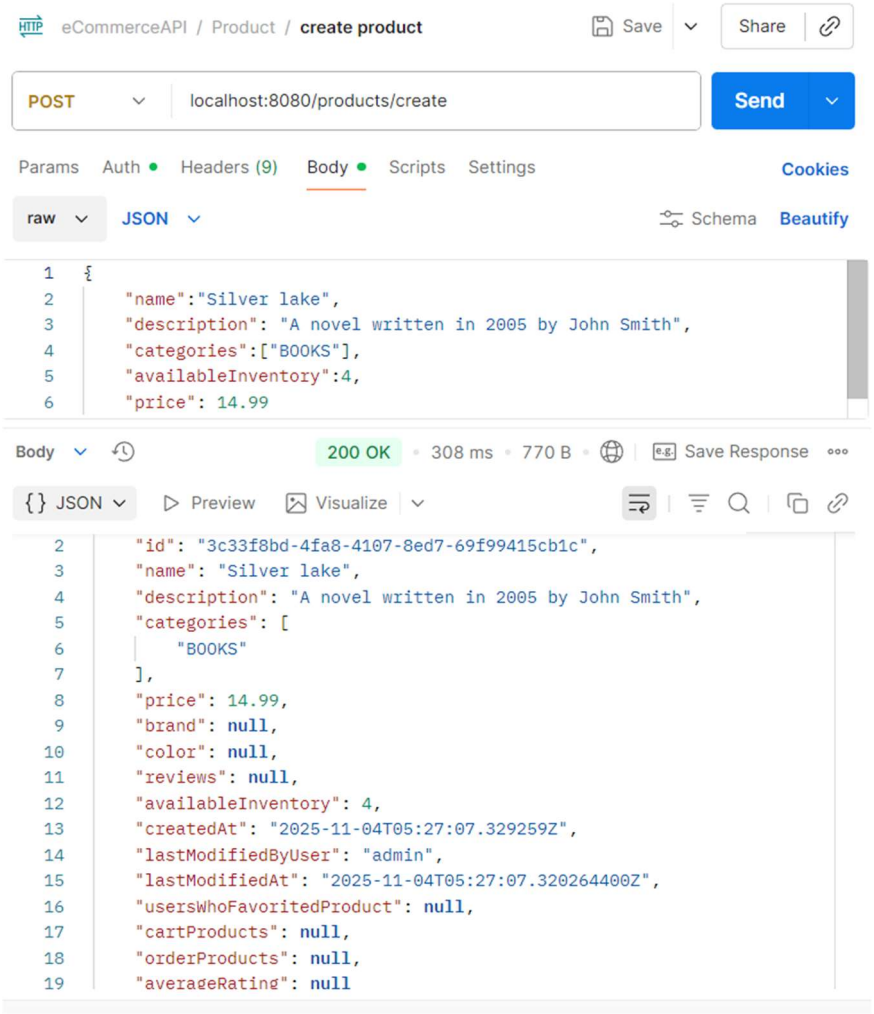


Εικόνα 29: Τροποποίηση χρήστη από τον εαυτό του



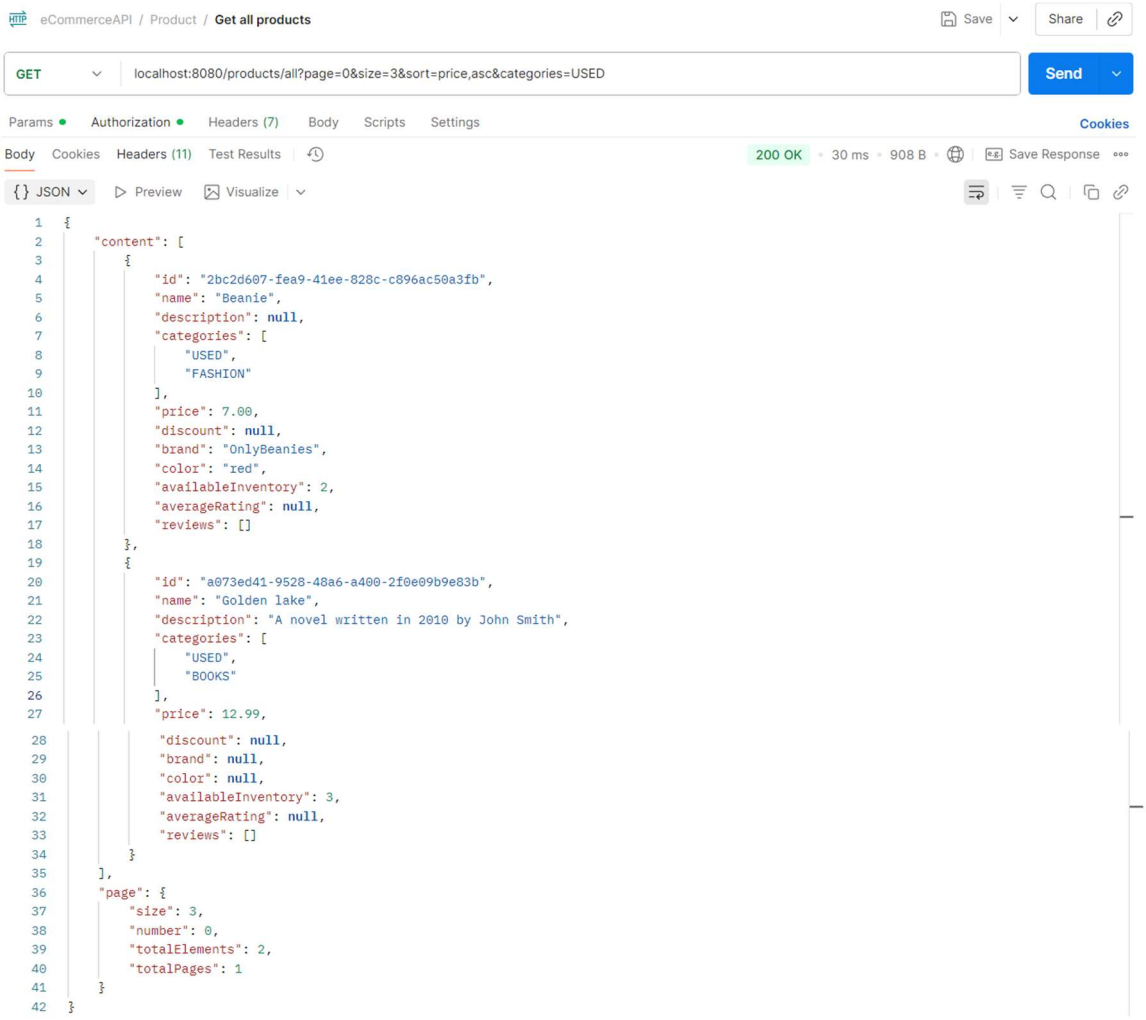
Εικόνα 30: Αποτέλεσμα επιτυχημένης τροποποίησης χρήστη

Αφότου ο χρήστης τροποποιήσει τα στοιχεία του, είναι πιθανό να επιθυμεί να προσθέσει προϊόντα στη λίστα «αγαπημένων» του ή στο καλάθι του. Προτού γίνει παρουσίαση των διαδικασιών αυτών, είναι απαραίτητο να προστεθούν προϊόντα από κάποιον υπάλληλο ή διαχειριστή.



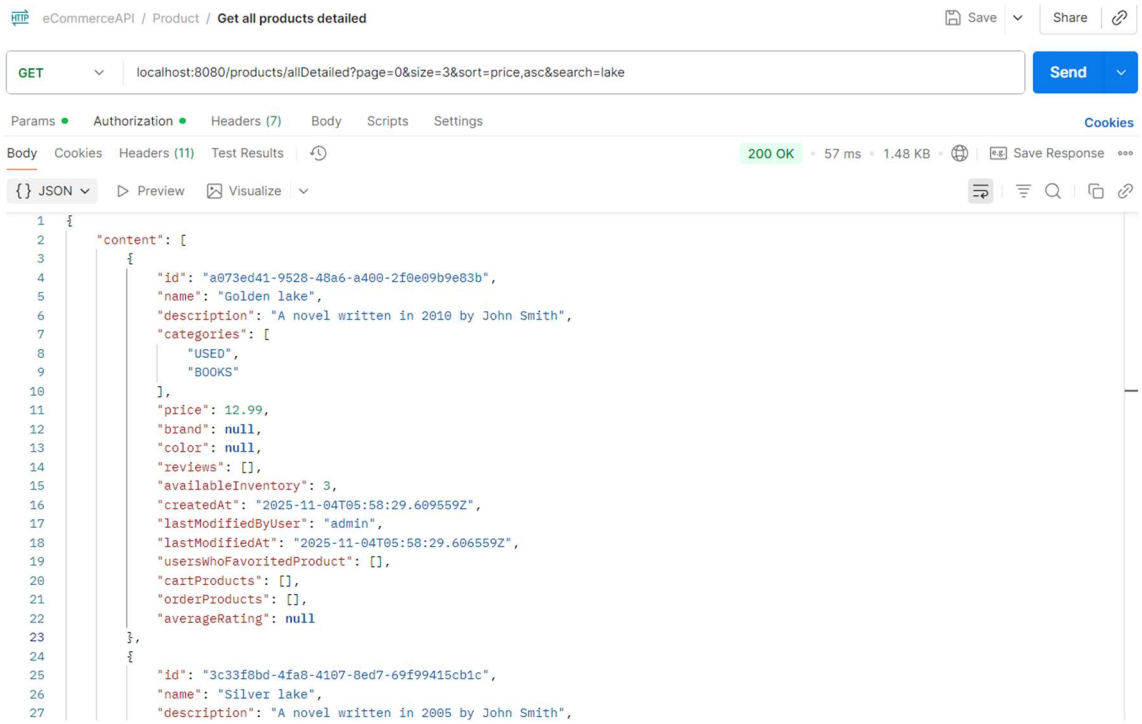
Εικόνα 31: Δημιουργία προϊόντος από διαχειριστή

Φυσικά, προσφέρεται σε όλους τους χρήστες η λειτουργία να δουν ένα ή παραπάνω από τα προϊόντα που διατίθενται από την εφαρμογή και να ενεργοποιήσουν φίλτρα για να περιορίσουν τα αποτελέσματα τους, αν το επιθυμούν.



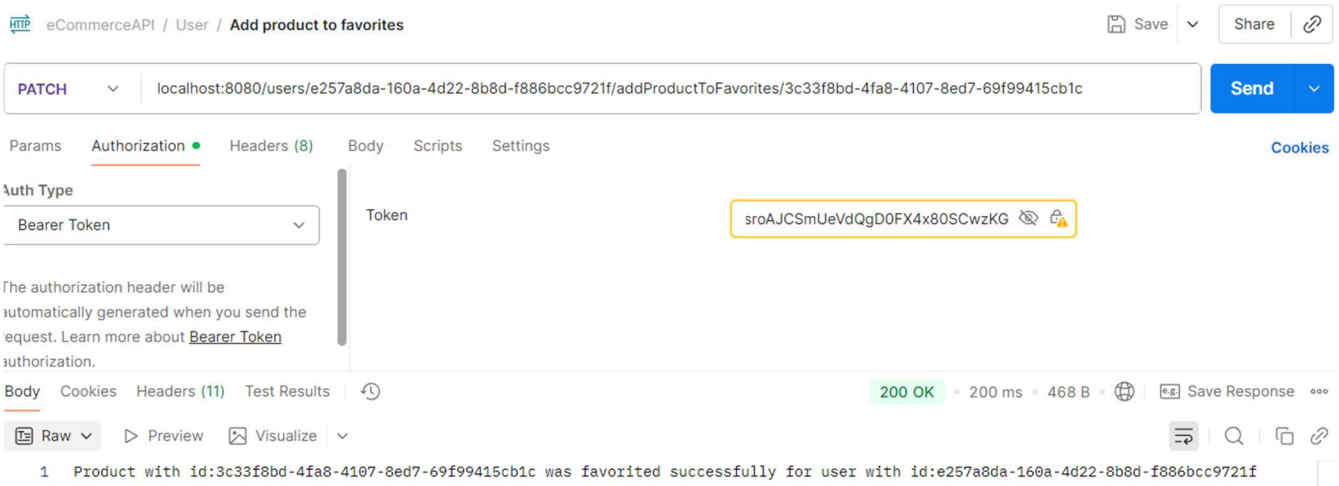
Εικόνα 32 : Αποτέλεσμα αναζήτησης προιόντων με φιλτρα από απλό χρηστη

Όπως φαίνεται από το παραπάνω παράδειγμα, το προϊόν με όνομα «Silver lake» που παράχθηκε προηγουμένως, δεν βρίσκεται στα αποτελέσματα της αναζήτησης καθώς δεν ανήκει στην κατηγορία «USED» που ζητάται από το φίλτρο. Επιπρόσθετα, οι διαχειριστές και οι υπάλληλοι διαθέτουν άλλη μια λειτουργία προβολής προϊόντων, η οποία προσφέρει συμπληρωματικές πληροφορίες που είναι καλύτερο να μην είναι ορατές σε άλλους χρήστες.



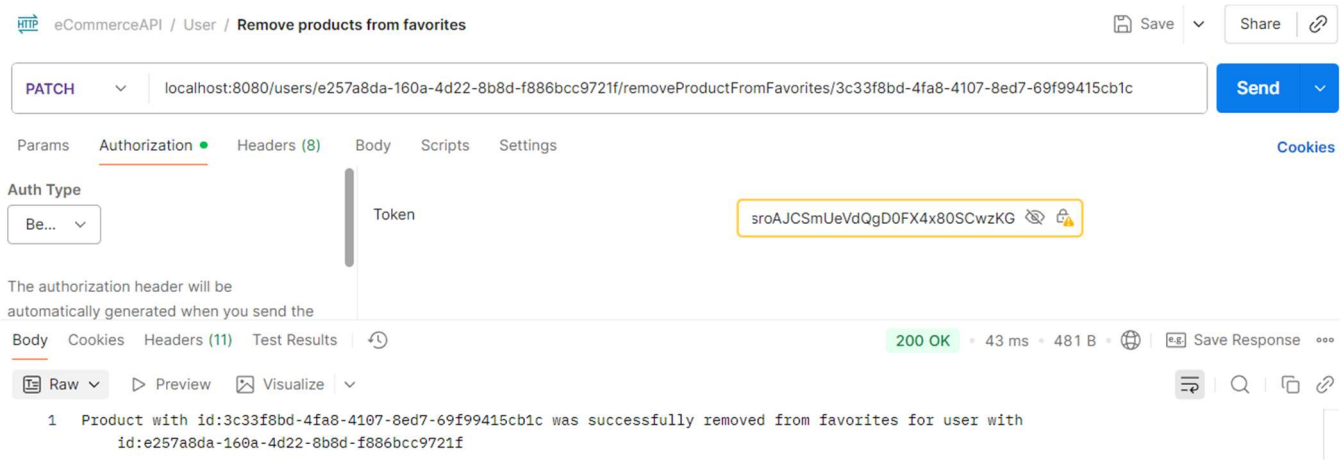
Εικόνα 33: Αποτέλεσμα αναζήτησης προϊόντων με φίλτρα από διαχειριστή

Τώρα που έχουν δημιουργηθεί κάποια προϊόντα, ο χρήστης μπορεί να προσθέσει οποιοδήποτε από αυτά στα αγαπημένα του.



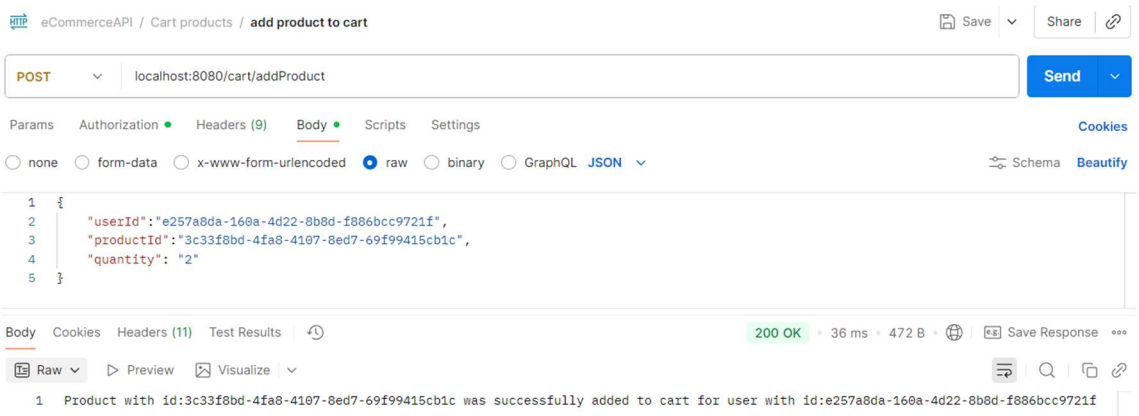
Εικόνα 34: Προσθήκη προϊόντος σε λίστα αγαπημένων

Ενώ με παρόμοιο τρόπο έχει τη δυνατότητα και να το αφαιρέσει από τη λίστα.



Εικόνα 35: Αφαίρεση προϊόντος από λίστα αγαπημένων

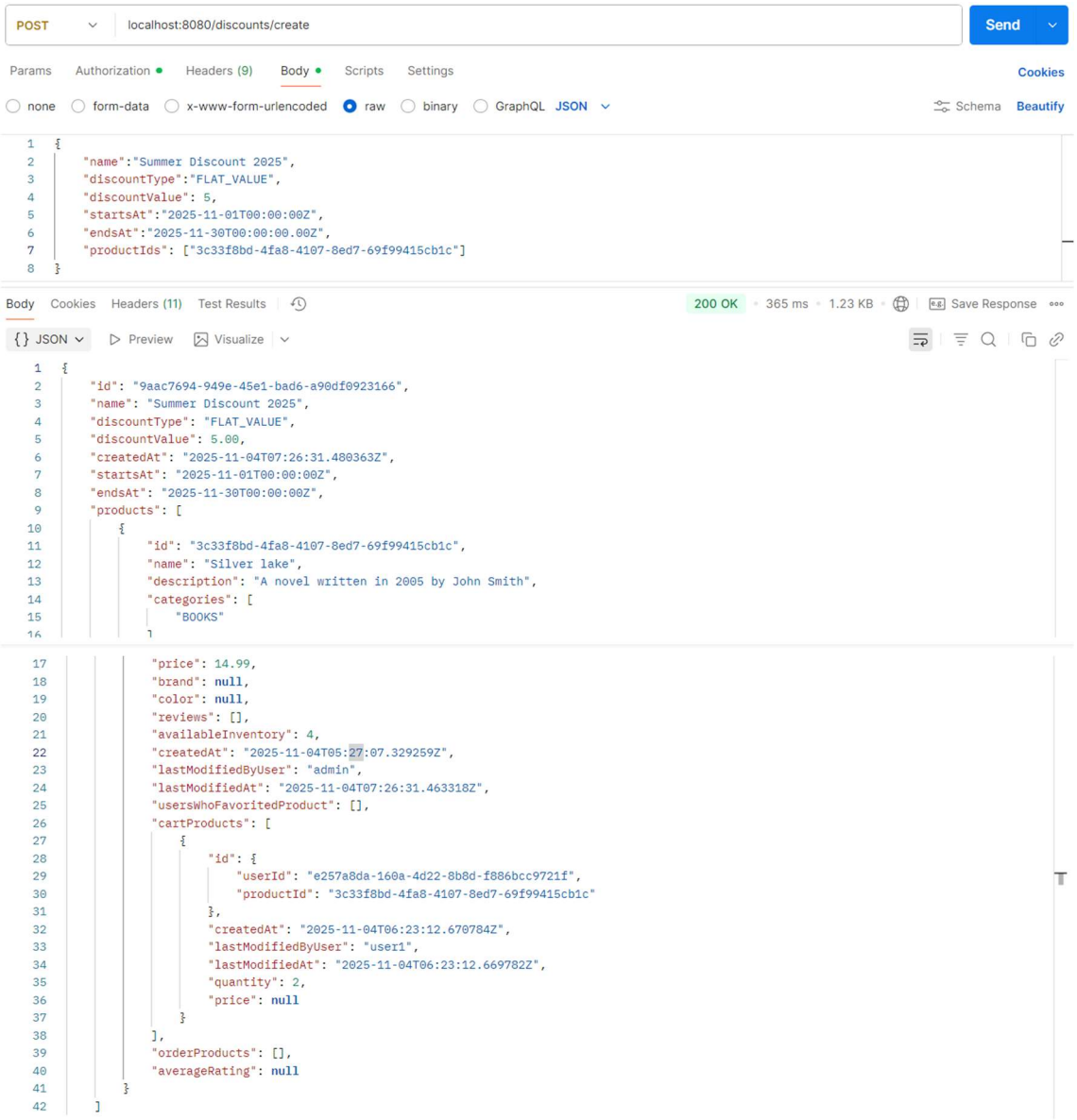
Οποιαδήποτε στιγμή ο χρήστης μπορεί να δει τη λίστα αγαπημένων του με ένα από get request. Επιπλέον, δύναται να προσθέσει προϊόντα στο καλάθι του για αγορά με το εξής post request.



Εικόνα 36: Προσθήκη προϊόντος σε καλάθι αγορών

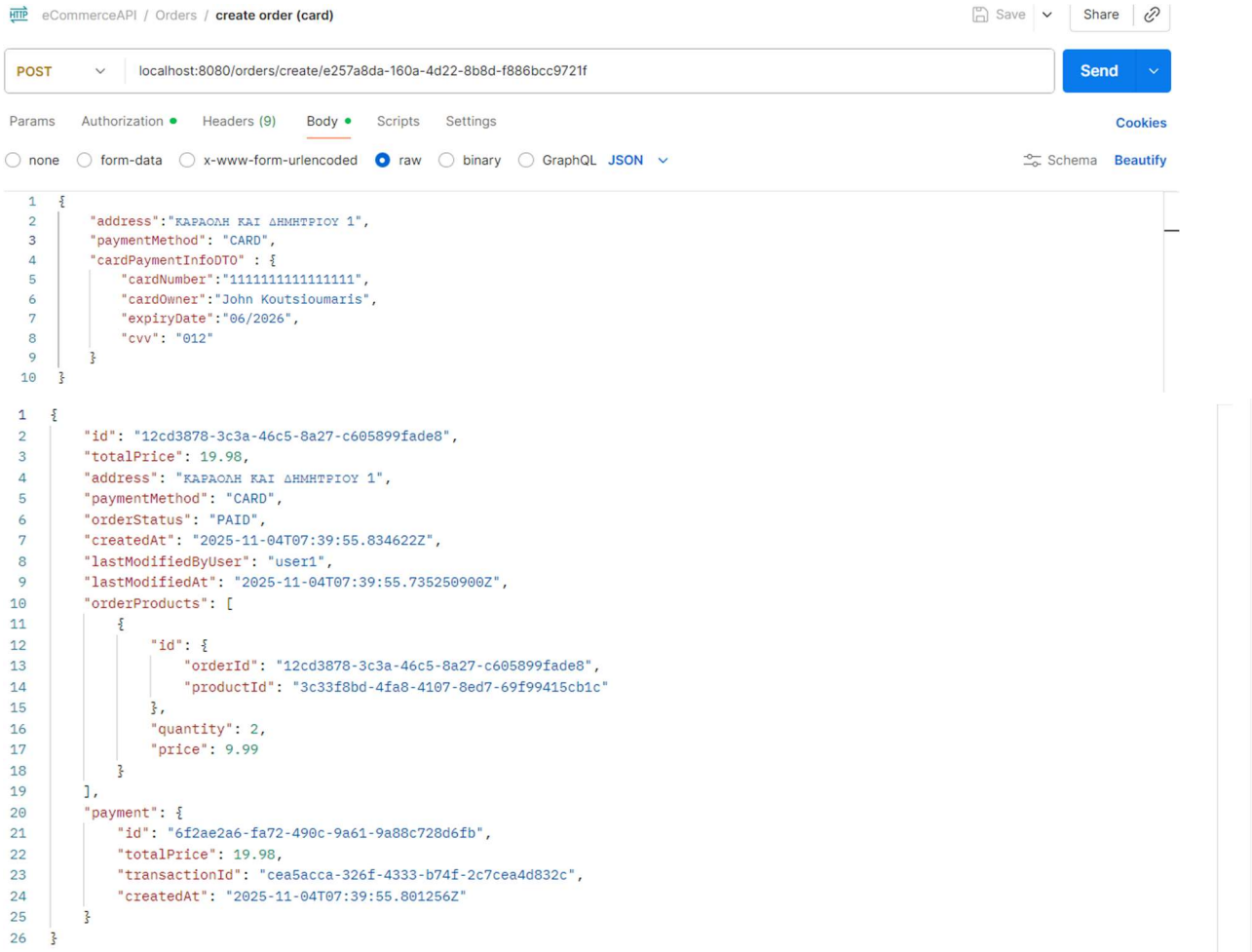
Βεβαίως, υπάρχουν ακόμα και δυνατότητες προβολής, τροποποίησης και διαγραφής/αφαίρεσης των προϊόντων στο καλάθι του χρήστη. Η τροποποίηση απευθύνεται στην επιθυμητή ποσότητα του προϊόντος.

Μια ακόμα αξιοσημείωτη λειτουργικότητα, είναι η προσθήκη εκπτώσεων για προϊόντα. Οι εκπτώσεις επιτρέπεται να είναι είτε «FLAT_VALUE» δηλαδή μια σταθερή τιμή έκπτωσης που θα εφαρμόζεται σε όλα τα προϊόντα της έκπτωσης, ή εναλλακτικά, τύπου «PERCENTAGE» που απευθύνεται σε ένα ποσοστό έκπτωσης πάνω στην τιμή των προϊόντων. Προϊόντα μπορούν να προστεθούν σε μια έκπτωση κατά τη δημιουργία της αλλά και μεταγενέστερα, ή να αφαιρεθούν.



Εικόνα 37: Δημιουργία έκπτωσης

Αφότου ο χρήστης έχει προϊόντα στο καλάθι του, του επιτρέπεται να τα κάνει παραγγελία. Επιτρέπονται παραγγελίες με πληρωμές τύπου «CARD» και «CASH» όπου η διαφορά τους είναι πως η τύπου CARD πληρωμή προσομοιάζει μια στοιχειώδης πληρωμή κάρτας και εσκεμμένα έχει 20% ποσοστό αποτυχίας.



Εικόνα 38: Παραγγελία με κάρτα

Αν εξεταστούν προσεκτικά τα ID των προϊόντων στην εικόνα δημιουργίας προϊόντος, έκπτωσης και παραγγελίας, παρατηρείται ότι η παραγγελία αφορά δυο αντίγραφα του βιβλίου «Silver lake» όπου η αρχική του τιμή ήταν 14.99 αλλά είναι σε έκπτωση των 5 ευρώ, με αποτέλεσμα ο χρήστης να πληρώνει 9.99 για το καθένα, και το συνολικό του κόστος να είναι 19.98 . Η έκπτωση υπολογίζεται αυτόματα κατά την προβολή όπως και την παραγγελία των προϊόντων, δεδομένου ότι είναι ενεργή κατά τη συγκεκριμένη χρονική περίοδο.

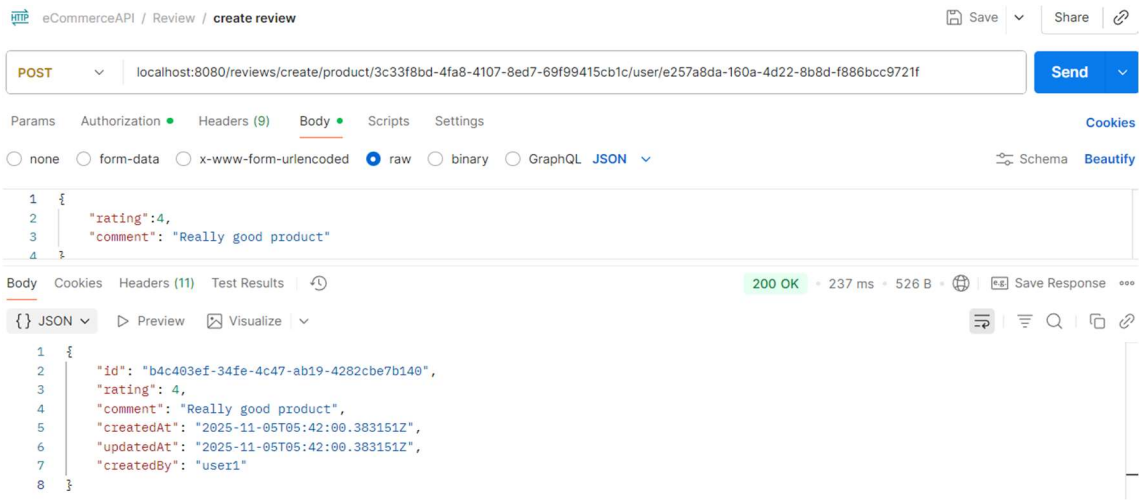
Καθώς η πληρωμή γίνεται με κάρτα το «orderStatus» είναι αυτόματα «PAID» , ενώ σε περίπτωση αντικαταβολής (CASH) η πληρωμή θα ήταν σε αναμονή και το orderStatus θα ήταν «PENDING». Το orderStatus είναι εφικτό να αλλάξει μετά την ολοκλήρωση της παραγγελίας από χρήστες με ρόλους υπάλληλου ή διαχειριστή.

Ένα επιπρόσθετο, ιδιαίτερο στοιχείο των παραγγελιών είναι πως δεν παρέχεται από το σύστημα η δυνατότητα τροποποίησης ή διαγραφής τους, διότι αντιμετωπίζονται ως αποδεικτικά συναλλαγών τα οποία δεν συνιστάται να αλλοιώνονται ή να καταστρέφονται.

Όταν ολοκληρωθεί μια παραγγελία, τα προϊόντα που παραγγέλθηκαν αποθηκεύονται αυτόματα στη βάση δεδομένων ως «OrderProducts» , όπου το καθένα από αυτά αντιστοιχίζει ένα προϊόν σε μια παραγγελία.

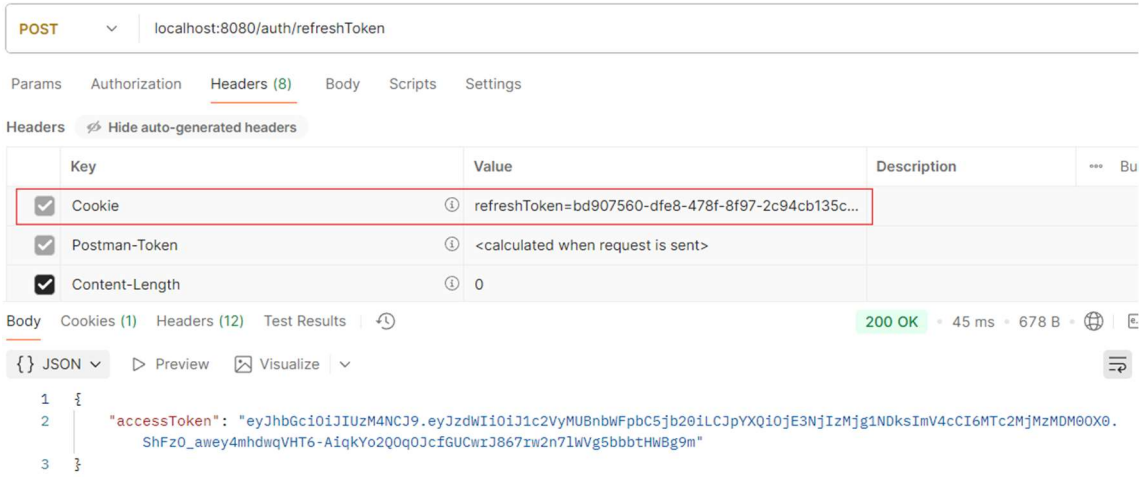
Η εφαρμογή παρέχει και τη δυνατότητα στους χρήστες να κάνουν κριτική σε προϊόντα, εφόσον τα έχουν παραγγείλει πρώτα τουλάχιστον μια φορά. Παρότι στο αποτέλεσμα του Postman στην εικόνα που ακολουθεί δεν φαίνεται η σύνδεση της κριτικής με τον χρήστη που την έκανε και το

προϊόν στο οποίο έγινε, οι πληροφορίες αυτές αποθηκεύονται εσωτερικά στη βάση δεδομένων. Με αντίστοιχο τρόπο γίνεται και τροποποίηση των κριτικών.



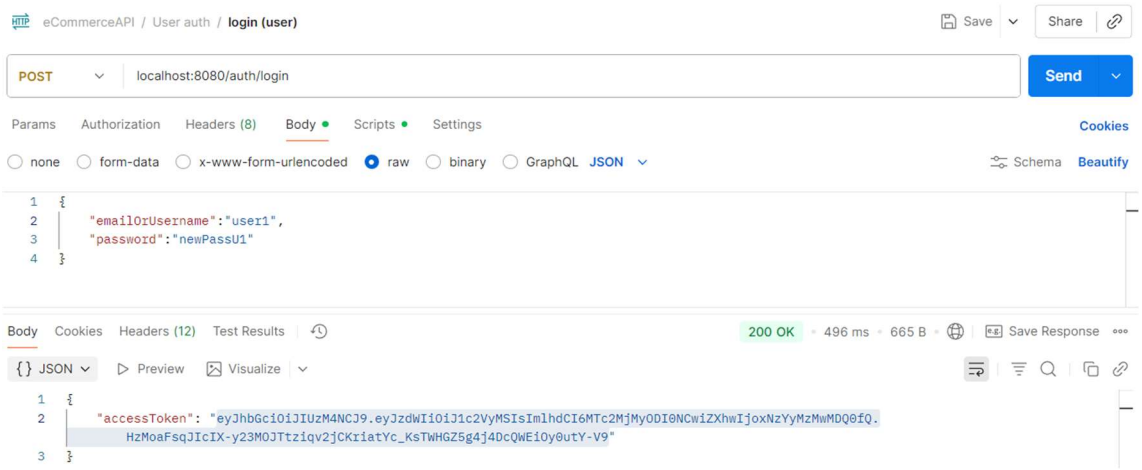
Εικόνα 39: Κριτική αγορασμένου προϊόντος

Κατά την εκτέλεση οποιουδήποτε request, εάν ο χρήστης παρατηρήσει ότι το access token του έληξε, τότε οφείλει είτε να αξιοποιήσει το refresh token cookie του για να αποκτήσει ένα καινούριο access token



Εικόνα 40: Refresh token cookie κεφαλίδα

Είτε, αν δεν είναι ούτε αυτό σε ισχύ, να κάνει σύνδεση με το username ή το email του και το password



Εικόνα 41: Επιτυχημένο login χρήστη

Όλες οι παραπάνω λειτουργικότητες που αναλύθηκαν κατά την παρουσίαση της εφαρμογής, είναι αντιπροσωπευτικές μιας πραγματικής ροής ενεργειών που θα έπραττε ένας απλός χρήστης με τελικό σκοπό να αγοράσει προϊόντα, σε συνδυασμό με έναν διαχειριστή που εισάγει στην εφαρμογή τα απαραίτητα αντικείμενα. Να σημειωθεί όμως, ότι περιγράφηκαν μόνο οι πιο αξιοσημείωτες λειτουργικότητες, ενώ δεν έγινε αναφορά σε άλλες, ιδιαίτερα απλές λειτουργίες που ενεργούν με παρόμοιους τρόπους, όπως ενδεικτικά η δυνατότητα διαγραφής χρηστών και προϊόντων, αποσύνδεσης χρήστη καθώς και προβολής ενός ή όλων των εκπτώσεων.

Συμπεράσματα

Κατά την ανάπτυξη της εφαρμογής ηλεκτρονικού εμπορίου διεξάχθηκε αναλυτική έρευνα για τις τεχνολογίες που επιλέχθηκαν και τις βέλτιστες τεχνικές με τις οποίες οι τεχνολογίες μπορούσαν να αξιοποιηθούν αποτελεσματικά. Το τελικό αποτέλεσμα ήταν η παραγωγή ενός αξιόπιστου backend της εφαρμογής που παρέχει στους χρήστες πολυπληθείς δυνατότητες, όπως αυτές που αφορούν δημιουργία, προβολή, τροποποίηση και διαγραφή χρηστών, προϊόντων, εκπτώσεων και κριτικών, προσθήκη και αφαίρεση προϊόντων σε λίστα αγαπημένων και σε καλάθι αγορών, καθώς επίσης παραγγελία και προσομοίωση πληρωμής προϊόντων.

Η αυθεντικοποίηση και εξουσιοδότηση των χρηστών πραγματοποιήθηκε με τα JSON Web Tokens. Η επιλογή τους πραγματοποιήθηκε με δεδομένο ότι αν ληφθούν υπόψιν βέλτιστες πρακτικές, αποτελούν ένα ποιοτικό εργαλείο για τους χρήστες παρέχοντας τους ικανοποιητική ασφάλεια και ευκολία στη χρήση. Τέτοιες πρακτικές είναι η σύντομη διάρκεια ζωής των access token, καθώς και η τεχνική του refresh token rotation που στοχεύουν στην ελαχιστοποίηση της πιθανότητας επιτυχημένης χρήσης τους σε περίπτωση κλοπής τους από κακόβουλους χρήστες.

Επιπλέον, δόθηκε ιδιαίτερη βαρύτητα στην παράλληλη εκτέλεση ανεξάρτητων τμημάτων κώδικα με Java Virtual Threads, με σκοπό να γίνει μια αναλυτική παρουσίαση τους σε θεωρητικό υπόβαθρο, περιγράφοντας την ανάπτυξη της τεχνολογίας, τον τρόπο λειτουργίας της και τα πλεονεκτήματα της, καθώς και σε πρακτικό κομμάτι με τα παραδείγματα υλοποίησης τους από άποψη κώδικα.

Ανώτατου ενδιαφέροντος ήταν ακόμα η αξιοποίηση των δυνατοτήτων του Spring Framework και πολλών υπο-συστατικών που αυτό εμπεριέχει, τα οποία συνολικά συνεπάγονται αυξημένη οργάνωση, αυτοματοποιήσεις, αποτελεσματικές βιβλιοθήκες και annotations για την ταχεία ανάπτυξη συντηρήσιμων και αποδοτικών εφαρμογών.

Μελλοντικές Επεκτάσεις

Πιθανές μελλοντικές επεκτάσεις και βελτιώσεις που θα μπορούσαν να συμπεριληφθούν στο σύστημα, αποτελούν η παραγωγή ενός frontend τμήματος, που βεβαίως είναι προϋπόθεση για να κυκλοφορήσει μια εφαρμογή ηλεκτρονικού εμπορίου στην αγορά, καθώς και η υλοποίηση ενός πραγματικού, πλήρως λειτουργικού συστήματος πληρωμής.

Η εφαρμογή προτού εξελιχθεί σε εμπορικό προϊόν θα όφειλε επιπλέον να περιλαμβάνει περισσότερα unit tests, καθώς και να εισάγει διαφορετικών ειδών tests όπως integration tests που επαληθεύουν την ορθή αλληλεπίδραση των διαφορετικών συστατικών του συστήματος, ή performance tests τα οποία εξετάζουν την σταθερότητα και ταχύτητα της εφαρμογής.

Ολοκληρώνοντας, εξαιρετικές προσθήκες θα ήταν η ενημέρωση του χρήστη για την επιτυχία των αγορών του μέσω email, όπως επίσης η σύνδεση του με το λεγόμενο «Two-factor authentication (2FA)» , που θα απαιτούσε από τον χρήστη να συμπληρώσει κατά τη σύνδεση και έναν κωδικό πρόσβασης περιορισμένης χρονικής ισχύος που θα λάμβανε στο email ή στο τηλέφωνο του.

Βιβλιογραφία

1) https://springtutorials.com/spring-ecosystem/
2) https://medium.com/@khairmuhammadmemon/understanding-spring-boot-starters-71f09b3ae8f3
3) https://www.ibm.com/think/topics/java-spring-boot
4) https://java2blog.com/spring-boot-tutorial/
5) https://medium.com/@ByteCodeBlogger/what-makes-up-spring-boot-starter-web-in-a-spring-boot-application-dec6517052ad
6) https://www.javacodegeeks.com/2021/02/validation-in-spring-boot-applications.html
7) https://blog.devops.dev/getting-started-with-validation-in-spring-boot-using-validator-1a7ab6d2d1f5
8) https://www.geeksforgeeks.org/java/spring-boot-spring-data-jpa/
9) https://www.progress.com/datadirect-connectors/faqs/datadirect-jdbc-faqs/how-does-jdbc-work
10) https://medium.com/@aprayush20/spring-data-jpa-with-spring-boot-9e3b261e4cc7
11) https://www.springcloud.io/post/2023-05/spring-boot-unit-test/#gsc.tab=0
12) https://spring.io/projects/spring-security
13) https://docs.spring.io/spring-security/reference/modules.html
14) https://www.digitalocean.com/community/tutorials/multithreading-in-java
15) https://www.datacamp.com/doc/java/threads
16) https://topdeveloperacademy.com/articles/the-hidden-benefits-of-java-multithreading
17) https://senoritadeveloper.medium.com/javas-virtual-vs-platform-threads-and-what-s-new-in-jdk-24-22de93f51a74
18) https://openjdk.org/jeps/491
19) https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html#GUID-DC4306FC-D6C1-4BCC-AECE-48C32C1A8DAA
20) https://www.scientecheasy.com/2020/08/life-cycle-of-thread-in-java.html/
21) https://www.geeksforgeeks.org/java/what-is-a-build-tool/
22) https://www.jwt.io/introduction#how-json-web-tokens-work
23) https://www.bezkoder.com/vue-refresh-token/
24) https://www.ibm.com/think/topics/postgresql
25) **Eugenia A. Politou, Efthimios Alepis, Constantinos Patsakis: Profiling tax and financial behaviour with big data under the GDPR. Comput. Law Secur. Rev. 35(3): 306-329 (2019)**
26) **Fran Casino, Eugenia A. Politou, Efthimios Alepis, Constantinos Patsakis: Immutability and Decentralized Storage: An Analysis of Emerging Threats. IEEE Access 8: 4737-4744 (2020)**
27) **Nancy Alonistioti, Evangelia-Aikaterini Tsichrintzi, Konstantina Chrysafiadi, Efthimios Alepis: Requirements for Fuzzy Logic in Personalisation of Fire Emergency Alerts. IISA 2023: 1-8**
28) **Efthymios Alepis, Aristea Kontogianni: Smartphone Crowdsourcing and Data Sharing Towards Advancing User Experience and Mobile Services. Int. J. Interact. Mob. Technol. 14(3): 38-53 (2020)**