Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Σχεδιασμός και υλοποίηση Διεπαφής προγραμματισμού εφαρμογών για ηλεκτρονικό κατάστημα.** **Design and Development of an API for an e-shop**
Ονοματεπώνυμο Φοιτητή	**Γεώργιος Κώστογλου**
Πατρώνυμο	**Σταύρος**
Αριθμός Μητρώου	**ΜΠΠΛ/18039**
Επιβλέπων	**Κωνσταντίνος Πατσάκης, Αναπληρωτής Καθηγητής**

Ημερομηνία Παράδοσης **Σεπτέμβριος 2021**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Πατσάκης Κωνσταντίνος Αναπληρωτής Καθηγητής	Αλέπης Ευθύμιος Αναπληρωτής Καθηγητής	Σακκόπουλος Ευάγγελος Επίκουρος Καθηγητής

Πίνακας περιεχομένων

1. Εισαγωγή

1.1. Περίληψη

Στην παρούσα διπλωματική εργασία , θα παρουσιαστεί η υλοποίηση μια Διεπαφής προγραμματισμού εφαρμογών (API), η οποία είναι απαραίτητη για την λειτουργία της διαδικτυακής εφαρμογής (webapp). Η διαδικτυακή εφαρμογή αφορά την λειτουργία ενός ηλεκτρονικού καταστήματος πώλησης διαφόρων ειδών. Το υλοποιημένο API δίνει κάποιες από τις βασικές δυνατότητες για ένα ηλεκτρονικό κατάστημα, όπως δημιουργία νέου λογαριασμού πελάτη , προσθήκη προϊόντων στο καλάθι του κλπ. Επίσης θα γίνει παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν για την υλοποίηση του API, όπως της γλώσσας Java και των τεχνολογιών της, συγκεκριμένα της βιβλιοθήκης Java spring boot (jpa , security).

Abstract

The dissertation introduces the implementation of an Application Programming Interface (API), which is necessary for the operation of the web application (webapp), will be presented. The web application concerns the operation of an online store selling various items. The implemented API gives some of the basic features for an online store, such as creating a new customer account, adding products to its cart, etc. It will also present the technologies used to implement the API, such as the Java language and its technologies, specifically the Java spring boot library (jpa, security).

1.2. Δομή της εργασίας

Η εργασία αυτή αποτελείται από 7 κεφάλαια. Στο πρώτο κεφάλαιο γίνεται μια σύντομη παρουσίαση της εργασίας. Το δεύτερο κεφάλαιο είναι το εισαγωγικό που περιγράφει τους στόχους της εργασίας αυτής και περιλαμβάνει επίσης μια σύντομη περιγραφή των τεχνολογιών που χρησιμοποιήθηκαν. Το τρίτο κεφάλαιο αναφέρει και αναλύει την μέθοδο και όλες τις τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής. Στο τέταρτο κεφάλαιο παρατίθενται τα σημαντικότερα κομμάτια του κώδικα και η βάση δεδομένων της εφαρμογής. Στο πέμπτο κεφάλαιο παρουσιάζονται οι λειτουργίες της εφαρμογής και η εκτέλεση τους μέσω ενός εργαλείου που μας παρέχει την δυνατότητα να πραγματοποιήσουμε http requests στο API. Στο έκτο κεφάλαιο γίνεται αναφορά των συμπερασμάτων αλλά και μερικές ιδέες για μελλοντικές επεκτάσεις για την εφαρμογή.

2. Εισαγωγή

2.1. Σκοπός διεξαγωγής της εργασίας

Σκοπός της παρούσας εργασίας είναι η δημιουργίας ενός server-side web API για μια διαδικτυακή εφαρμογή, συγκεκριμένα θα χρησιμοποιηθεί για την λειτουργία ενός ηλεκτρονικού καταστήματος πώλησης διαφόρων προϊόντων. Με το API θα δίνεται η δυνατότητα στο web app (front end) του ηλεκτρονικού καταστήματος να επικοινωνεί με τα διαθέσιμα endpoints του server με την χρήση HTTP requests και να επιστρέφονται HTTP response με δεδομένα τύπου json.

2.2. Τεχνολογίες

Στην παρούσα διπλωματική εργασία χρησιμοποιούμε την γλώσσα Java και τις τεχνολογίες της, συγκεκριμένα την βιβλιοθήκη Java spring boot (jpa , security) . Με σκοπό να μπορέσουμε να αναπτύξουμε μια Διεπαφή προγραμματισμού εφαρμογών (API) για την λειτουργία μιας διαδικτυακής εφαρμογής (webapp) για ένα ηλεκτρονικό κατάστημα. Η αρχιτεκτονική σχεδιασμού της εφαρμογής είναι το MVC (Model-view-controller) , εκτός από τις υλοποιημένες λειτουργίες που έχει η εφαρμογή , υπάρχει η δυνατότητα να προστεθούν και νέες . Το υπάρχον API δίνει τις βασικές δυνατότητες για ένα ηλεκτρονικό κατάστημα, όπως δημιουργία νέου λογαριασμού πελάτη , προσθήκη προϊόντων στο καλάθι του κλπ.

Το server-side Web API που σχεδιάστηκε στα πλαίσια της παρούσας διπλωματικής εργασίας είναι ουσιαστικά μια Διεπαφή προγραμματισμού εφαρμογών για τον web server. Μέσω αυτής δίνετε η δυνατότητα στον web server να δέχεται HTTP requests από τον χρήστη, συνήθως ένα web browser για ένα συγκεκριμένο resource και ο web server επιστρέφει το περιεχόμενο αυτού του ή κάποιο μήνυμα σφάλματος. Ένα resource που αποστέλλετε από τον web server μπορεί να είναι ένα προϋπάρχον διαθέσιμο αρχείο ή μπορεί να δημιουργείται την στιγμή που γίνεται το αίτημα που επικοινωνεί με τον server. Η επικοινωνία μεταξύ web browser και web server πραγματοποιείται από τα διαθέσιμα δημόσια εκτεθειμένα endpoints που αποτελούν την διεπαφή που σχεδιάστηκε, τα οποία ορίζουν την τοποθεσία των resources, συνήθως η πρόσβαση τους πραγματοποιείται μέσω ενός URL στο οποίο τα HTTP requests γίνονται post.

3. Μεθοδολογία υλοποίησης

3.1. Μέθοδος ανάλυσης και ανάπτυξης της εφαρμογής

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε η γλώσσα java και το framework spring boot. Το περιβάλλον ανάπτυξης που χρησιμοποιήθηκε είναι το intelliJ IDEA και για την αποθήκευση των δεδομένων της εφαρμογής χρησιμοποιήθηκε μια βάση προσωρινής αποθήκευσης δεδομένων η H2 Database. Στη συνέχεια θα αναλυθούν περισσότερο οι τεχνολογίες.

3.2. Γλώσσα προγραμματισμού JAVA

Η Java είναι μια δημοφιλής γλώσσα προγραμματισμού , αναπτύχθηκε το 1995.Ανήκει στην εταιρεία Oracle, και περισσότερες από 3 δισεκατομμύρια συσκευές χρησιμοποιούν Java. Μερικές από τις χρήσεις της είναι : Mobile applications (κυρίως Android apps),Desktop applications ,Web applications , Web servers and application servers, Games, Database connection και πολλά άλλα.

Η κύρια διαφορά μεταξύ Java και των άλλων γλωσσών προγραμματισμού είναι η μέθοδος με την οποία εκτελείται ο κώδικας Java. Σε αντίθεση με τις άλλες γλώσσες όπως το C ++, η Java μεταγλωττίζεται σε bytecode που μπορεί να εκτελεστεί σε οποιαδήποτε συσκευή με την Java Virtual Machine (JVM). Η C ++, από την άλλη πλευρά, μεταγλωττίζεται απευθείας στον κώδικα του μηχανήματος και, ως εκ τούτου, μπορεί να εκτελεστεί μόνο στην ίδια πλατφόρμα στην οποία συντάχθηκε. Άλλα πλεονεκτήματα της java είναι :

- Λειτουργεί σε διαφορετικές πλατφόρμες (Windows, Mac, Linux, Raspberry Pi κ.λπ.)
- Είναι μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού στον κόσμο,
- Είναι εύκολο να μάθει και απλό στη χρήση, Είναι ανοιχτού κώδικα και δωρεάν,
- Είναι ασφαλές, γρήγορο και ισχυρό
- Έχει τεράστια υποστήριξη από την κοινότητα (δεκάδες εκατομμύρια προγραμματιστές)
- Η Java είναι μια αντικειμενοστρεφής γλώσσα που δίνει μια σαφή δομή στα προγράμματα και επιτρέπει την επαναχρησιμοποίηση του κώδικα, μειώνοντας το κόστος ανάπτυξης
- Καθώς η Java είναι κοντά στα C ++ και C #, διευκολύνει τους προγραμματιστές να μεταβούν σε Java ή το αντίστροφο

3.3. Η αρχιτεκτονική MVC

Η αρχιτεκτονική Model-View-Controller (MVC) χωρίζει μια εφαρμογή σε τρία κύρια λογικά στοιχεία: το μοντέλο (model), την προβολή (view) και τον ελεγκτή (controller). Το κάθε στοιχείο είναι κατασκευασμένο για να χειρίζεται συγκεκριμένες πτυχές μιας εφαρμογής. Το MVC είναι ένα από τα πιο συχνά χρησιμοποιούμενα βιομηχανικά πρότυπα ανάπτυξης ιστοσελίδων.

Ακολουθούν τα 3 μοντέλα που αποτελούν την αρχιτεκτονική Model View Controller.

Model

Το στοιχείο Μοντέλο είναι υπεύθυνο για την λογική που σχετίζεται με τα δεδομένα που χρησιμοποιεί ο χρήστης. Το μοντέλο μπορεί να χρησιμοποιείται είτε για την μεταφορά των δεδομένων μεταξύ των

στοιχείων προβολής και ελεγκτή ή οποιαδήποτε άλλα δεδομένα που σχετίζονται με τη λογική του ηλεκτρονικού καταστήματος. Θα μπορούσε για παράδειγμα, ένα αντικείμενο καλάθι να ανακτήσει τις πληροφορίες καλάθι από τη βάση δεδομένων, θα το χειριστεί και θα τα ενημερώσει ξανά στη βάση δεδομένων ή θα τα χρησιμοποιήσει για την απόδοση δεδομένων.

View

Η χρήση του στοιχείου View έχει να κάνει με τις λειτουργίες διεπαφής χρήστη (UI) της εφαρμογής.

Controller

Ο Controller χρησιμοποιείται για την επικοινωνία μεταξύ του Model και View . Επεξεργάζεται όλη την πληροφορία με βάση την λογική της εφαρμογής (business logic), τα εισερχόμενα αιτήματα, την διαχείριση των δεδομένων σύμφωνα με το Model. Για παράδειγμα ο Cart controller θα διαχειριστεί όλα τα αιτήματα ζήτησης και καταχώρησης πληροφορίας και θα ενημερώσει την βάση χρησιμοποιώντας το Cart model.

3.4. Spring framework

Τα frameworks είναι έτοιμα «κομμάτια» κώδικα που χρησιμοποιούνται από προγραμματιστές για να δημιουργήσουν εφαρμογές. Επίσης μπορούν να επαναχρησιμοποιούνται σαν πρότυπα και οι προγραμματιστές συμπληρώνουν κώδικα όπου είναι απαραίτητο.

Το Spring framework είναι το πιο δημοφιλές framework για την enterprise Java. Εκατομμύρια προγραμματιστές σε όλο τον κόσμο χρησιμοποιούν το Spring Framework για να δημιουργήσουν κώδικα υψηλής απόδοσης, εύκολα ελεγχόμενο και επαναχρησιμοποιήσιμο. Το spring framework είναι «ανοιχτού κώδικα» Με τις κύριες λειτουργίες Spring Framework μπορούν να αναπτυχθούν οποιαδήποτε είδος εφαρμογής Java, αλλά υπάρχουν επεκτάσεις για την κατασκευή εφαρμογών ιστού που μπορούν να χρησιμοποιηθούν συνδυαστικά με την Java EE. Το Spring framework διευκολύνει τη χρήση της J2EE και προωθεί καλές πρακτικές προγραμματισμού, επιτρέποντας ένα μοντέλο προγραμματισμού που βασίζεται σε POJO (plain old Java object).

3.5. Hibernate

Η Hibernate είναι ένα framework της γλώσσας java, η οποία απλοποιεί την ανάπτυξη εφαρμογών που επικοινωνούν με βάση δεδομένων. Είναι ανοικτού κώδικα, ORM (Object Relational Mapping) εργαλείο. Η Hibernate όχι μόνο πραγματοποιεί το mapping από java classes σε πίνακες βάσης δεδομένων (και από java τύπους δεδομένων σε SQL τύπους δεδομένων), επίσης υποστηρίζει data query CRUD (create, read, update, delete) λειτουργίες. Ένα database query είναι ένα αίτημα στη βάση που έχει να κάνει με την επεξεργασία ή ανάκτηση δεδομένων.

3.6. H2 Database

Η βάση δεδομένων H2 είναι μια σχεσιακού τύπου βάση ανοιχτού κώδικα java , μπορεί να ρυθμιστεί είτε σαν in-memory είτε persistent , συνήθως η H2 χρησιμοποιείται για development και testing και όχι για production.

3.7. Insomnia

Το Insomnia είναι ένα rest client εργαλείο ελέγχου. Ο σκοπός του είναι να καταγράψει την απάντηση ενός rest api στέλνοντας HTTP requests ώστε να ελέγξουμε εάν το rest api λειτουργεί όπως θα θέλαμε ή όχι.

3.8. Tomcat

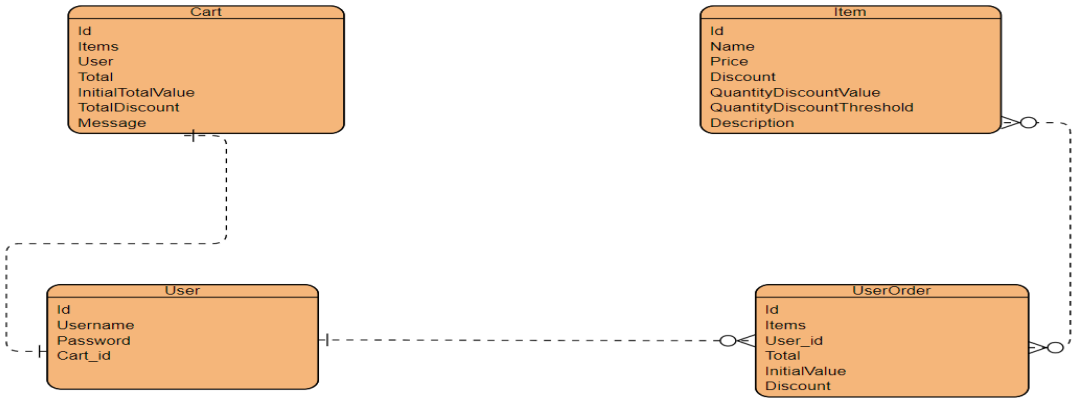
Ο Apache Tomcat είναι ένα δωρεάν και open-source implementation των Java Servlet, Java Server Pages (JSP), Java Expression Language και WebSocket technologies.Ο Tomcat παρέχει ένα HTTP web server περιβάλλον στο οποίο τρέχει Java κώδικας. Η ανάπτυξη και συντήρηση του Tomcat γίνεται από μια ανοικτή κοινότητα προγραμματιστών υπό την αιγίδα της Apache Software Foundation. Η χρήση του Tomcat server θεωρείτε μια από τις καλύτερες επιλογές , λόγο των διάφορων δυνατοτήτων του όπως η επεκτασιμότητα, αλλά καθώς είναι και μια αποδεδειγμένα καλά δοκιμασμένη και σταθερή core engine.

4. Υλοποίηση εργασίας

Σε αυτό το κεφάλαιο θα γίνει λεπτομερής περιγραφή του κώδικα υλοποίησης της εφαρμογής. Στην πρώτη υποενότητα γίνεται ανάλυση της βάσης δεδομένων της εφαρμογής και στη συνέχεια του κώδικα της εφαρμογής. Ο οποίος βασίζετε στο μοντέλο σχεδιασμού MVC και το java framework spring.

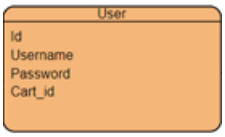
4.1. Βάση δεδομένων

Όπως ήδη αναφέρθηκε , το σύστημα διαχείρισης βάσεων δεδομένων H2 επιλέχθηκε για την υλοποίηση της εφαρμογής. Στη συνέχεια ακολουθεί η ανάλυση του μοντέλου της βάσης.



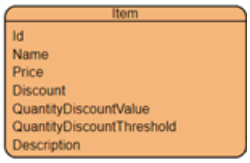
Διάγραμμα Οντοτήτων – Συσχετίσεων

4.1.1. User



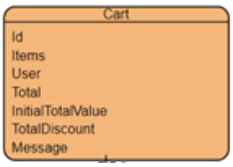
Η οντότητα user είναι αυτή στην οποία αποθηκεύουμε τα στοιχεία των χρηστών του ηλεκτρονικού καταστήματος. Σε αυτό τον πίνακα αποθηκεύετε το username, password και το καλάθι του κάθε πελάτη. Επίσης με την συσχέτιση με την οντότητα cart έχουμε την δυνατότητα να δούμε τα αντικείμενα του κάθε πελάτη, το σύνολο της αξίας τους κ.α. .

4.1.2. Item



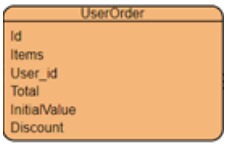
Η οντότητα Item είναι αυτή στην οποία αποθηκεύουμε τα στοιχεία των προϊόντων του ηλεκτρονικού καταστήματος.

4.1.3. Cart



Η οντότητα Cart είναι αυτή στην οποία αποθηκεύουμε τα στοιχεία του καλαθιού κάθε πελάτη του ηλεκτρονικού καταστήματος.

4.1.4. UserOrder



Η οντότητα UserOrder είναι αυτή στην οποία αποθηκεύουμε τα στοιχεία των παραγγελιών των χρηστών του ηλεκτρονικού καταστήματος. Η οντότητα UserOrder συσχετίζεται με την User και προκύπτει η αντιστοίχιση κάθε παραγγελίας με έναν χρήστη. Επίσης η οντότητα UserOrder συσχετίζεται με την οντότητα Item.

4.2. Pom.xml

Το Ακρωνύμιο POM είναι για τις λέξεις Project Object Model. Είναι βασικό κομμάτι για την λειτουργία του Maven. Είναι ένα XML αρχείο το οποίο βρίσκεται στο βασικό φάκελο του project με την ονομασία pom.xml. Το συγκεκριμένο αρχείο περιέχει πληροφορίες για το project και για την παραμετροποίηση του, που χρησιμοποιεί το maven για το build του . Το pom.xml περιέχει επίσης τα goals και τα plugins. Καθώς εκτελεί ένα task ή goal , το Maven αναζητά στο POM τις απαραίτητες πληροφορίες παραμετροποίησης και στη συνέχεια τις εκτελεί. Μερικές από τις πληροφορίες που παρέχονται από το αρχείο pom είναι : project dependencies , plugins, goals, build profiles, project version.

Παραθέτετε κώδικας του ανωτέρου αρχείου:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.5.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>nith-app</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>nith-app</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>11</java.version>
        <maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>
    </properties>
```

```xml
<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <dependency>
        <groupId>com.auth0</groupId>
        <artifactId>java-jwt</artifactId>
        <version>3.10.2</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
    </dependency>

    <dependency>
        <groupId>com.h2database</groupId>
```

```
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.30</version>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>


        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.6.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>RELEASE</version>
            <scope>test</scope>
        </dependency>

    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <release>11</release>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

4.3. Υλοποίηση μοντέλου (model) στην JAVA με χρήση JPA

4.3.1 User

```java
package com.example.demo.model.persistence;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonProperty;


@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonProperty
    private long id;

    @Column(nullable = false, unique = true)
    @JsonProperty
    private String username;

    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    @Column(nullable = false)
    private String password;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "cart_id", referencedColumnName = "id")
    @JsonIgnore
    private Cart cart;

    public Cart getCart() { return cart; }

    public void setCart(Cart cart) { this.cart = cart; }

    public long getId() { return id; }

    public void setId(long id) { this.id = id; }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public void setPassword(String password) { this.password = password; }

    public String getPassword() { return password; }
}

```

4.3.2 Item

```java
package com.example.demo.model.persistence;

import ...

@Entity
@Table(name = "item")
public class Item {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonProperty
    private Long id;

    @Column(nullable = false)
    @JsonProperty
    private String name;

    @Column(nullable = false)
    @JsonProperty
    private BigDecimal price;

    @Column(nullable = false)
    @JsonProperty
    private BigDecimal discount;

    @Column(nullable = false)
    @JsonProperty
    private BigDecimal quantityDiscountValue;


    @Column(nullable = false)
    @JsonProperty
    private BigDecimal quantityDiscountThreshold;


    @Column(nullable = false)
    @JsonProperty
    private String description;

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Item other = (Item) obj;
        if (id == null) {
            if (other.id != null)
```

```
            if (other.id != null)
                return false;
        } else if (!id.equals(other.id))
            return false;
        return true;
    }

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public BigDecimal getPrice() { return price; }

    public void setPrice(BigDecimal price) { this.price = price; }

    public String getDescription() { return description; }

    public void setDescription(String description) { this.description = description; }


    public BigDecimal getDiscount() { return discount; }

    public void setDiscount(BigDecimal discount) { this.discount = discount; }



    public BigDecimal getQuantityDiscountValue() { return quantityDiscountValue; }

    public void setQuantityDiscountValue(BigDecimal quantityDiscountValue) {
        this.quantityDiscountValue = quantityDiscountValue;
    }

    public void setQuantityDiscountThreshold(BigDecimal quantityDiscountThreshold) {
        this.quantityDiscountThreshold = quantityDiscountThreshold;
    }

    public BigDecimal getQuantityDiscountThreshold() { return this.quantityDiscountThreshold; }


}
```

4.3.3 Cart

```java
package com.example.demo.model.persistence;

import ...

@Entity
@Table(name = "cart")
public class Cart {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonProperty
    @Column
    private Long id;

    @ManyToMany
    @JsonProperty
    @Column
    private List<Item> items;

    @OneToOne(mappedBy = "cart")
    @JsonProperty
    private User user;

    @Column
    @JsonProperty
    private BigDecimal total;


    @JsonProperty
    @Column
    private BigDecimal initialTotalValue;

    @JsonProperty
    @Column
    private BigDecimal totalDiscount;

    @JsonProperty
    @Column
    private String message;


    public BigDecimal getTotal() { return total; }

    public void setTotal(BigDecimal total) { this.total = total; }

    public User getUser() { return user; }

    public void setUser(User user) { this.user = user; }

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public List<Item> getItems() { return items; }

    public void setItems(List<Item> items) { this.items = items; }
```

```
    public void addItem(Item item) {
        if(items == null) {
            items = new ArrayList<>();
        }
        items.add(item);
        if(total == null) {
            total = new BigDecimal( val: 0);
        }
        initialTotalValue = initialTotalValue.add(item.getPrice());
//      total = total.add(item.getPrice());
    }

    public void removeItem(Item item) {
        if(items == null) {
            items = new ArrayList<>();
        }
        items.remove(item);
        if(total == null) {
            total = new BigDecimal( val: 0);
        }
        initialTotalValue = initialTotalValue.subtract(item.getPrice());
//      total = total.subtract(item.getPrice());
    }

    public BigDecimal getInitialTotalValue() { return initialTotalValue; }

    public void setInitialTotalValue(BigDecimal initialTotalValue) { this.initialTotalValue = initialTotalValue; }

    public BigDecimal getTotalDiscount() { return totalDiscount; }

    public void setTotalDiscount(BigDecimal totalDiscount) { this.totalDiscount = totalDiscount; }

    public String getMessage() { return message; }

    public void setMessage(String message) { this.message = message; }
}
```

4.3.4 UserOrder

```java
package com.example.demo.model.persistence;

import java.math.BigDecimal;
import java.util.List;
import java.util.stream.Collectors;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;

@Entity
@Table(name = "user_order")
public class UserOrder {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonProperty
    @Column
    private Long id;

    @ManyToMany(cascade = CascadeType.ALL)
    @JsonProperty
    @Column
    private List<Item> items;

    @ManyToOne
    @JoinColumn(name="user_id", nullable = false, referencedColumnName = "id")
    @JsonProperty
    private User user;

    @JsonProperty
    @Column
    private BigDecimal total;

    @JsonProperty
    @Column
    private BigDecimal initialValue;

    @JsonProperty
    @Column
    private BigDecimal discount;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }
```

```java
    public List<Item> getItems() { return items; }

    public void setItems(List<Item> items) { this.items = items; }

    public User getUser() { return user; }

    public void setUser(User user) { this.user = user; }

    public static UserOrder createFromCart(Cart cart) {
        UserOrder order = new UserOrder();
        order.setItems(cart.getItems().stream().collect(Collectors.toList()));
        order.setTotal(cart.getTotal());
        order.setUser(cart.getUser());
        return order;
    }

    public BigDecimal getTotal() { return total; }

    public void setTotal(BigDecimal total) { this.total = total; }

    public BigDecimal getInitialValue() { return initialValue; }

    public void setInitialValue(BigDecimal initialValue) { this.initialValue = initialValue; }

    public BigDecimal getDiscount() { return discount; }

    public void setDiscount(BigDecimal discount) { this.discount = discount; }
}
```

4.4. Υλοποίηση υπηρεσιών (services)

4.4.1 CheckoutService

```
package com.example.demo.service;

import com.example.demo.model.persistence.Cart;

public interface CheckoutService {
    Cart applyCheckoutDetails(Cart cart);
}
```

4.4.2. CheckoutServiceImpl

```
package com.example.demo.service;


import com.example.demo.model.persistence.Cart;
import com.example.demo.model.persistence.Item;
import com.example.demo.model.persistence.repositories.CartRepository;
import com.example.demo.model.persistence.repositories.ItemRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;


@Service
public class CheckoutServiceImpl implements CheckoutService{

    @Autowired CartRepository cartRepository;
    @Autowired ItemRepository itemRepository;

    private static final BigDecimal DISCOUNT = new BigDecimal(10);
    private static final BigDecimal ONE_HUNDRED                    = new
BigDecimal(100);
    private static final BigDecimal PRICE_DISCOUNT_THRESHOLD         =
BigDecimal.valueOf(60);


    //Implement discount logic
    @Override public Cart applyCheckoutDetails(Cart cart) {
```

```
        return passCartThroughDiscountsFilter(cart);
    }


    private Cart passCartThroughDiscountsFilter(Cart cart) {
        /*  STEP 1: SET UP                                  */
        var initialTotalValue = cart.getInitialTotalValue();
        //Finish setting up the initial cart
        finishInitialCartSetup(cart);
        //Quantity threshold discount
        quantityThresholdFilter(cart);
        //Total price threshold (e.g. £60) discount
        totalPriceThresholdFilter(cart);
        //FUTURE--->Bundle of different products threshold dicount
        //**********************************
        return cart;
    }

    private void quantityThresholdFilter(Cart cart) {

        BigDecimal totalDiscount = BigDecimal.ZERO;
        BigDecimal cnt = new BigDecimal(0);
        List<Long> idsChekcked = new ArrayList<>();
        var items = cart.getItems();
        final var size = items.size();
        for (int i = 0; i < size; i++) {
            if (idsChekcked.contains(items.get(i).getId()) ||

items.get(i).getQuantityDiscountThreshold().compareTo(BigDecimal.ZERO) ==
0) {
                continue;
            }
            cnt = cnt.add(BigDecimal.valueOf(1));

            for (int j = i + 1; j < size; j++) {
                if (items.get(j).getId() == items.get(i).getId()) {
                    cnt = cnt.add(BigDecimal.valueOf(1));
                }
            }
            final var quantityDiscountThreshold =
items.get(i).getQuantityDiscountThreshold();
            if (cnt.compareTo(quantityDiscountThreshold) >= 0) {
                cart.setMessage("Quantity based discount is applied");
                for (int k = 0; k < size; k++) {
                    if (items.get(k).getId() == items.get(i).getId()) {
                        final var item = items.get(k);

item.setDiscount(item.getPrice().subtract(item.getQuantityDiscountValue()
));
                        final BigDecimal discount = item.getDiscount();
```

```
                        cart.getItems().get(k).setDiscount(discount);
                        totalDiscount = totalDiscount.add(discount);
                    }
                }
                idsChekcked.add(items.get(i).getId());
            }

        }
        cart.setTotal(cart.getTotal().subtract(totalDiscount));
        cart.setTotalDiscount(totalDiscount);
    }

    private void finishInitialCartSetup(Cart cart) {
        cart.setTotal(cart.getInitialTotalValue());
        cart.setTotalDiscount(BigDecimal.ZERO);
        cart.setMessage("No discount applied");
        for (int i = 0; i < cart.getItems().size(); i++) {
            cart.getItems().get(i).setDiscount(BigDecimal.ZERO);
        }
    }


    private void totalPriceThresholdFilter(Cart cart) {
        BigDecimal totalAfterQuantityReductions = cart.getTotal();
        if
(totalAfterQuantityReductions.compareTo(PRICE_DISCOUNT_THRESHOLD) > 0) {

cart.setTotal(getDiscountedPrice(totalAfterQuantityReductions,
DISCOUNT));

cart.setTotalDiscount(cart.getInitialTotalValue().subtract(cart.getTotal(
)));
            cart.setMessage(cart.getMessage() + " and " + DISCOUNT + "%
discount is applied because the order is above " +
PRICE_DISCOUNT_THRESHOLD);
        }
    }

    private BigDecimal getDiscountedPrice(BigDecimal initialValue,
BigDecimal discountPercentage) {
        return
initialValue.subtract(initialValue.multiply(discountPercentage.divide(ONE
_HUNDRED)));
    }

}
```

4.4.3. UpdateCartContentService

```
package com.example.demo.service;

import com.example.demo.model.persistence.Cart;
import com.example.demo.model.persistence.Item;
import com.example.demo.model.persistence.User;

public interface UpdateCartContentsService {
    Cart updateCart(User user, Item item, int quantity, boolean
addToCart);
}
```

4.4.4. UpdateCartContentServiceImp

```
package com.example.demo.service;

import com.example.demo.model.persistence.Cart;
import com.example.demo.model.persistence.Item;
import com.example.demo.model.persistence.User;
import com.example.demo.model.persistence.repositories.CartRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.List;


@Service
public class UpdateCartContentsServiceImpl implements
UpdateCartContentsService {

    @Autowired private CheckoutService checkoutService;
    @Autowired private CartRepository  cartRepository;

    @Override public Cart updateCart(User user, Item item, int quantity,
boolean addToCart) {
        var cart = user.getCart();
        if (addToCart) {
            cart = checkoutService.applyCheckoutDetails(addToCart(cart,
item, quantity));
        } else {
            cart =
checkoutService.applyCheckoutDetails(removeFromCart(cart, item,
quantity));
```

```
        }
        cartRepository.save(cart);
        return cart;
    }

    private Cart removeFromCart(Cart cart, Item item, int quantity) {
        for (int i = 0; i < quantity; i++) {
            if (cart.getItems().contains(item)) {
                cart.removeItem(item);
            }
        }
        if (cart.getItems().isEmpty()) {
            return constructEmptyCart(cart);
        }
        return cart;
    }

    private Cart addToCart(Cart cart, Item item, int quantity) {
        for (int i = 0; i < quantity; i++) {
            cart.addItem(item);
        }
        return cart;
    }

    private Cart constructEmptyCart(Cart cart) {
        cart.setTotal(BigDecimal.ZERO);
        cart.setMessage("The cart is empty");
        return cart;
    }
}
```

4.4.5. UserService

```
package com.example.demo.service;

import com.example.demo.model.persistence.User;

public interface UserService {
    User createUser(String username, String password);
}
```

4.4.6. UserServiceImpl

```
package com.example.demo.service;

import com.example.demo.controllers.UserController;
import com.example.demo.model.persistence.Cart;
import com.example.demo.model.persistence.User;
import com.example.demo.model.persistence.repositories.CartRepository;
import com.example.demo.model.persistence.repositories.UserRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;

@Service
public class UserServiceImpl implements UserService{

    private static final Logger                log =
LoggerFactory.getLogger(UserController.class);
    @Autowired private   BCryptPasswordEncoder bCryptPasswordEncoder;
    @Autowired private CartRepository cartRepository;
    @Autowired private UserRepository userRepository;

    @Override public User createUser(String username, String password) {
        User user = new User();
        user.setUsername(username);
        Cart cart = new Cart();
        cart.setTotal(BigDecimal.ZERO);
        cart.setInitialTotalValue(BigDecimal.ZERO);
        cart.setTotalDiscount(BigDecimal.ZERO);
        cartRepository.save(cart);
        user.setCart(cart);
        user.setPassword(bCryptPasswordEncoder.encode(password));
        userRepository.save(user);
        log.info("New user {} created", user.getUsername());
        return user;
    }
}
```

4.5. Υλοποίηση ελεγκτών (controllers)

4.5.1. CartController

```
package com.example.demo.controllers;


import java.util.Optional;

import com.example.demo.model.persistence.repositories.CartRepository;
import com.example.demo.service.CheckoutService;
import com.example.demo.service.UpdateCartContentsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.persistence.Cart;
import com.example.demo.model.persistence.Item;
import com.example.demo.model.persistence.User;
import com.example.demo.model.persistence.repositories.ItemRepository;
import com.example.demo.model.persistence.repositories.UserRepository;
import com.example.demo.model.requests.ModifyCartRequest;

@RestController
@RequestMapping("/api/cart")
public class CartController {

   //DI with autowiring

   @Autowired UpdateCartContentsService updateCartContentsService;

   //DI with Constructor
   private UserRepository userRepository;
   private CartRepository cartRepository;
   private ItemRepository itemRepository;

   @Autowired
   public CartController(
         UserRepository userRepository,
         CartRepository cartRepository,
         ItemRepository itemRepository
```

```
    ) {
        this.userRepository = userRepository;
        this.cartRepository = cartRepository;
        this.itemRepository = itemRepository;
    }

    @GetMapping("/{username}")
    public ResponseEntity<Cart> getCart(@PathVariable String username) {
        User user = userRepository.findByUsername(username);
        if(user == null) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(cartRepository.findByUser(user));
    }

    @PostMapping("/addToCart")
    public ResponseEntity<Cart> addTocart(@RequestBody ModifyCartRequest
request) {

        User user = userRepository.findByUsername(request.getUsername());
        //CHECKS
        if(user == null) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }
        Optional<Item> item = itemRepository.findById(request.getItemId());
        if(!item.isPresent()) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }

        return ResponseEntity.ok(updateCartContentsService.updateCart(user,
item.get(), request.getQuantity(), true));
    }




    @PostMapping("/removeFromCart")
    public ResponseEntity<Cart> removeFromcart(@RequestBody
ModifyCartRequest request) {

        User user = userRepository.findByUsername(request.getUsername());
        //CHECKS
        if(user == null) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }
        Optional<Item> item = itemRepository.findById(request.getItemId());
        if(!item.isPresent()) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }
```

```
      return ResponseEntity
            .ok(updateCartContentsService.updateCart(user, item.get(),
request.getQuantity(), false));
   }

}
```

4.5.2. ItemController

```
package com.example.demo.controllers;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.model.persistence.Item;
import com.example.demo.model.persistence.repositories.ItemRepository;

@RestController
@RequestMapping("/api/item")
public class ItemController {

   private ItemRepository itemRepository;

   @Autowired
   public ItemController(ItemRepository itemRepository) {
      this.itemRepository = itemRepository;
   }

   @GetMapping
   public ResponseEntity<List<Item>> getItems() {
      return ResponseEntity.ok(itemRepository.findAll());
   }

   @GetMapping("/{id}")
   public ResponseEntity<Item> getItemById(@PathVariable Long id) {
      return ResponseEntity.of(itemRepository.findById(id));
   }

   @GetMapping("/name/{name}")
```

```
    public ResponseEntity<List<Item>> getItemsByName(@PathVariable String
name) {
        List<Item> items = itemRepository.findByName(name);
        return items == null || items.isEmpty() ?
ResponseEntity.notFound().build()
                : ResponseEntity.ok(items);

    }

}
```

4.5.3. OrdeController

```
package com.example.demo.controllers;

import java.util.List;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.model.persistence.User;
import com.example.demo.model.persistence.UserOrder;
import com.example.demo.model.persistence.repositories.OrderRepository;
import com.example.demo.model.persistence.repositories.UserRepository;

@RestController
@RequestMapping("/api/checkout")
public class OrderController {

    @Autowired private UserRepository  userRepository;
    @Autowired private OrderRepository orderRepository;



    @PostMapping("/submit/{username}")
    public ResponseEntity<UserOrder> submit(@PathVariable String username)
{
```

```
        User user = userRepository.findByUsername(username);
        if(user == null) {
            return ResponseEntity.notFound().build();
        }
        UserOrder order = UserOrder.createFromCart(user.getCart());
        orderRepository.save(order);
        return ResponseEntity.ok(order);
    }

    @GetMapping("/history/{username}")
    public ResponseEntity<List<UserOrder>> getOrdersForUser(@PathVariable
String username) {
        User user = userRepository.findByUsername(username);
        if(user == null) {
            return ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(orderRepository.findByUser(user));
    }
}
```

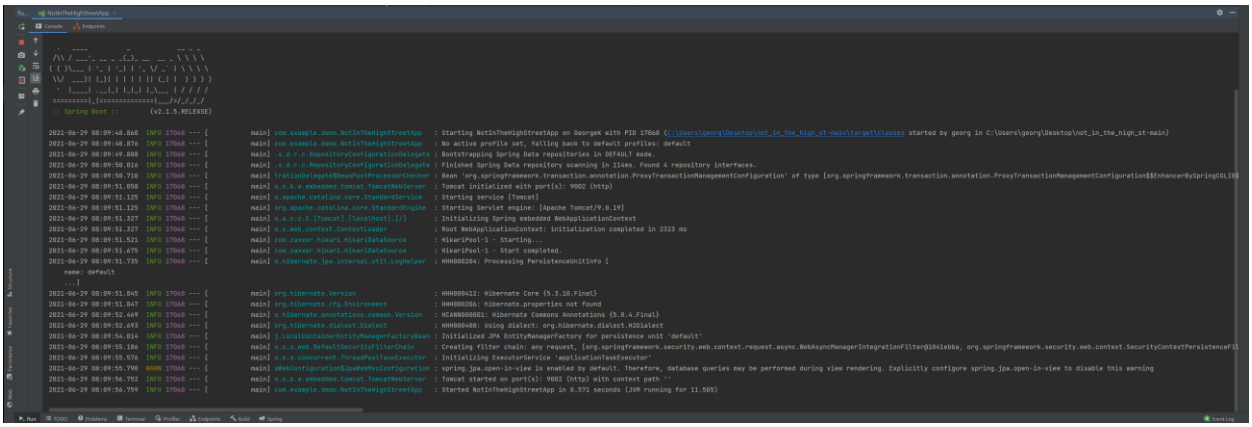
4.5.4. UserController

```
package com.example.demo.controllers;

import com.example.demo.service.UserService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.example.demo.model.persistence.User;
import com.example.demo.model.persistence.repositories.UserRepository;
import com.example.demo.model.requests.CreateUserRequest;


@RestController
@RequestMapping("/api/user")
public class UserController {
```

```
   private static final Logger log =
LoggerFactory.getLogger(UserController.class);

   @Autowired UserService userService;
   @Autowired UserRepository userRepository;


   @GetMapping("/id/{id}")
   public ResponseEntity<User> findById(@PathVariable Long id) {
      log.debug("UserController.findById called with id {}", id);
      return ResponseEntity.of(userRepository.findById(id));
   }

   @GetMapping("/{username}")
   public ResponseEntity<User> findByUserName(@PathVariable String
username) {
      log.debug("UserController.findByUserName called with username {}");
      User user = userRepository.findByUsername(username);
      return user == null ? ResponseEntity.notFound().build() :
ResponseEntity.ok(user);
   }

   @PostMapping("/create")
   public ResponseEntity<User> createUser(@RequestBody CreateUserRequest
createUserRequest) {
      log.debug("UserController.createUser called with username {}",
createUserRequest.getUsername());
      if (
            createUserRequest.getPassword().length() <= 6 ||

!createUserRequest.getPassword().equals(createUserRequest.getConfirmPassw
ord())
      ) {
         log.error("Cannot create user {} because the password is
invalid", createUserRequest.getUsername());
         return ResponseEntity.badRequest().build();
      }
      User user = userService.createUser(createUserRequest.getUsername(),
createUserRequest.getPassword());
      log.info("New user {} created", createUserRequest.getUsername());
      return ResponseEntity.ok(user);
   }

}
```

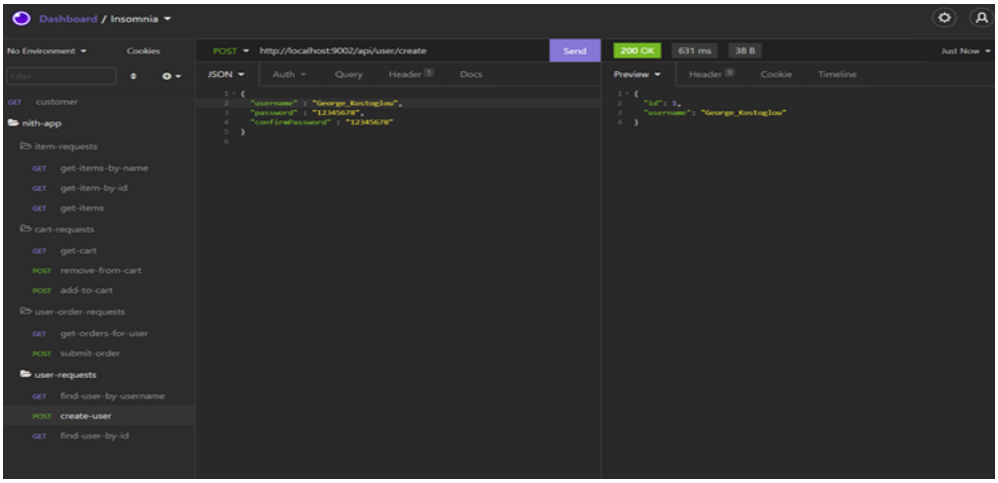
5. Λειτουργίες API/ Εκτέλεση εφαρμογής

Με την εκκίνηση της εφαρμογής ο ενσωματωμένος server tomcat που μας παρέχει το spring framework ξεκινάει.

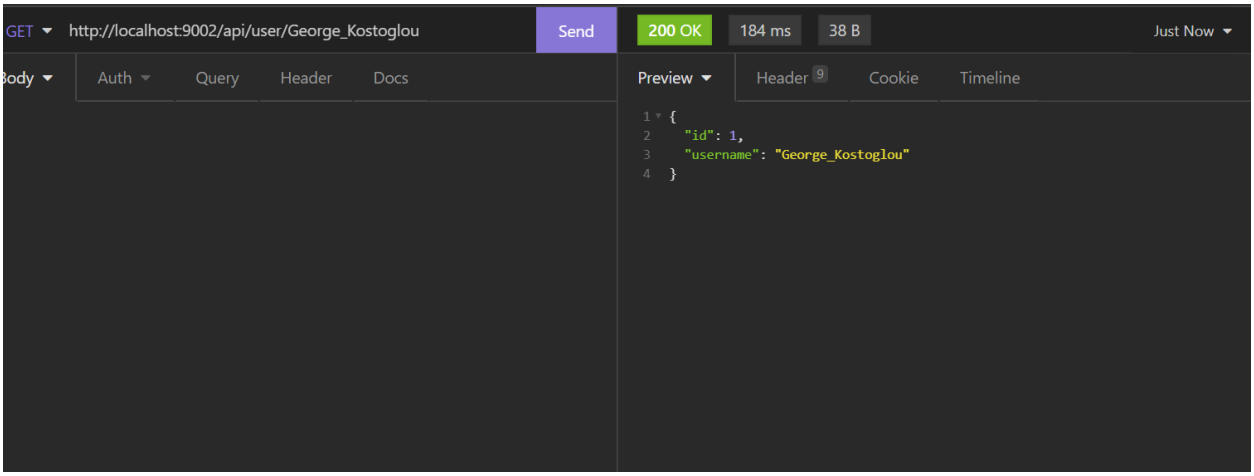


Στην συνέχεια με την βοήθεια του εργαλείου insomnia θα εκτελέσουμε τα αντίστοιχα http requests για την δοκιμή του API που δημιουργήσαμε.

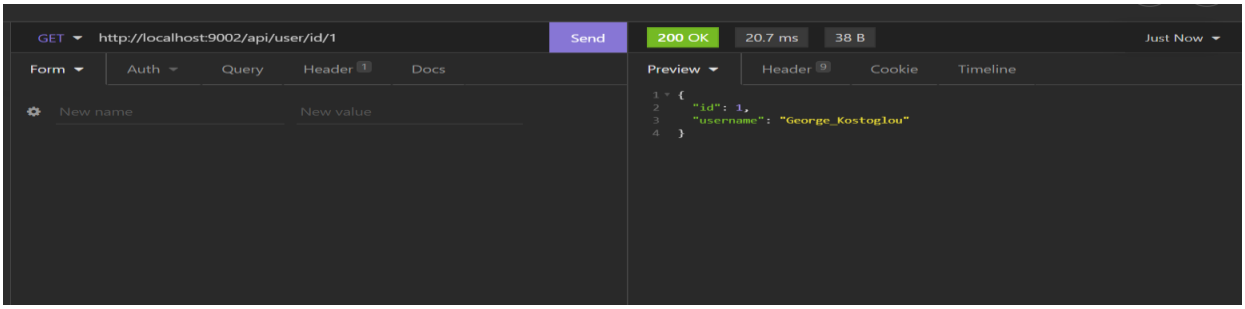
5.1.Δημιουργία λογαριασμού



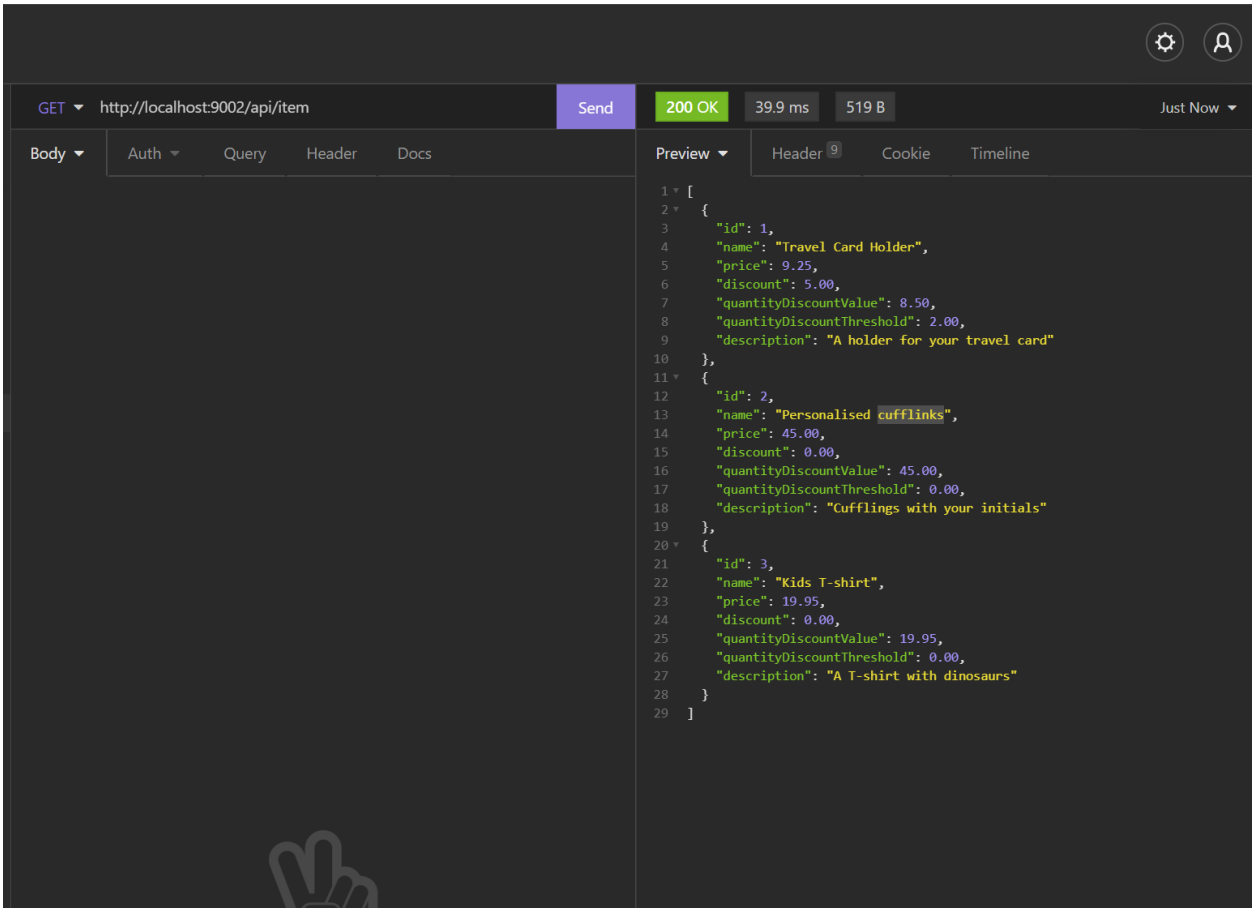
5.2. Αναζήτηση χρήστη με username



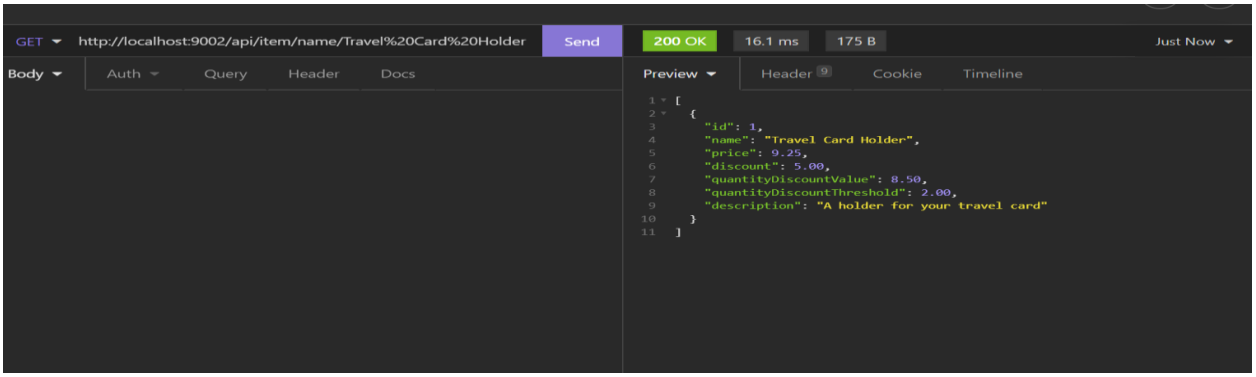
5.3. Αναζήτηση χρήστη με user id



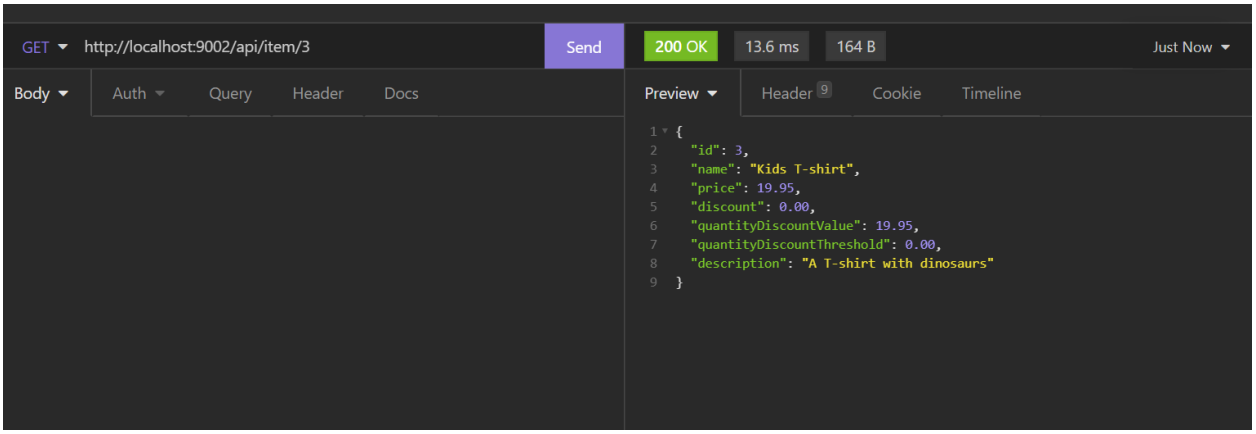
5.4 Αναζήτηση όλων των προϊόντων



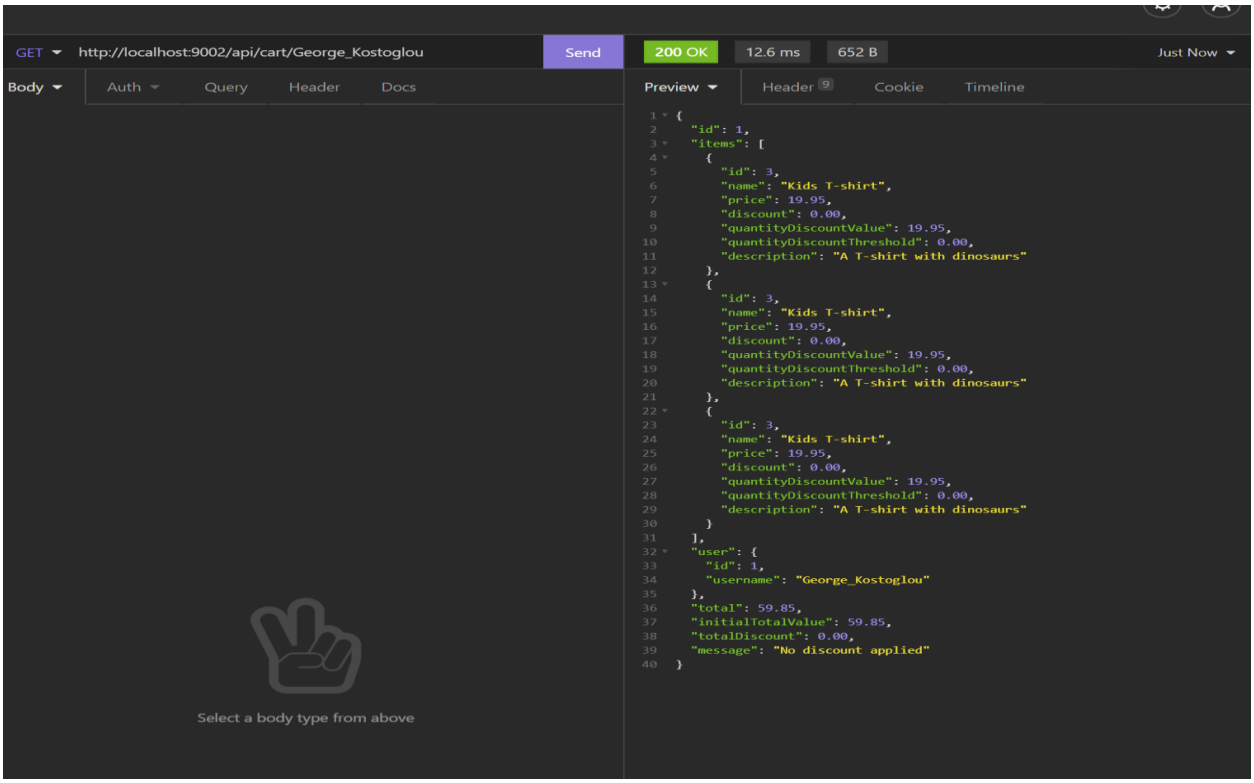
5.5. Αναζήτηση προϊόντος με όνομα



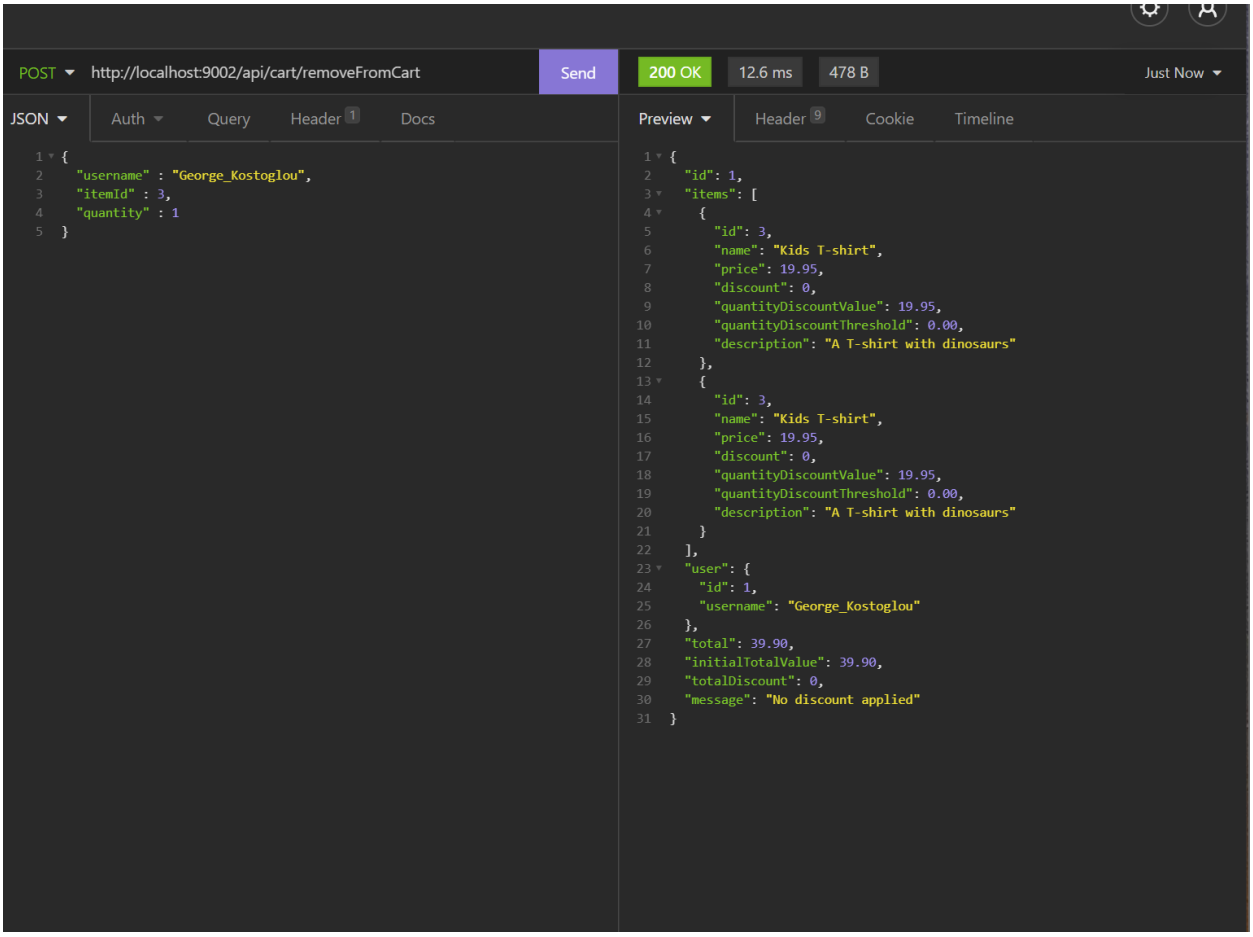
5.6 Αναζήτηση προϊόντος με id



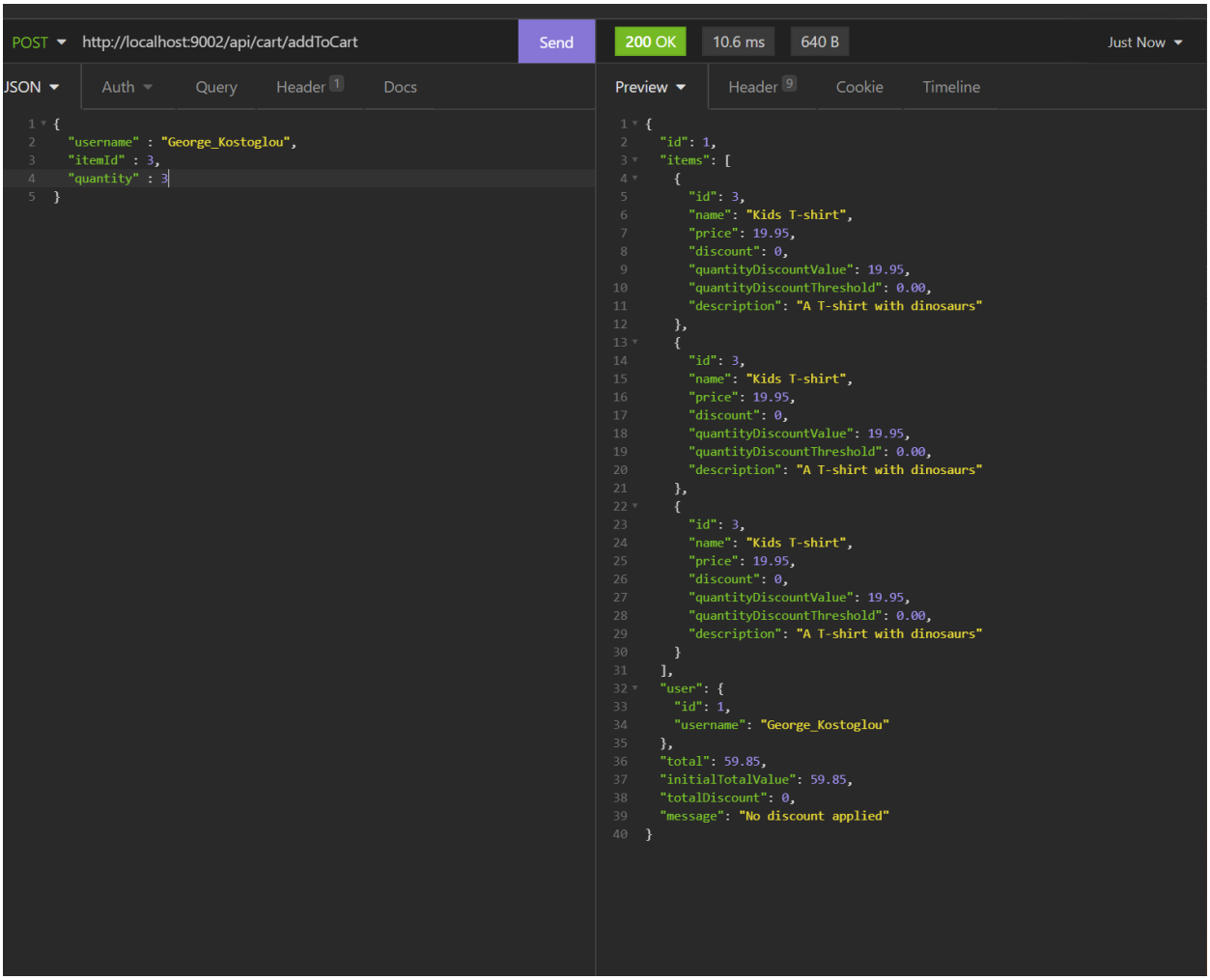
5.7. Αναζήτηση καλαθιού χρήστη



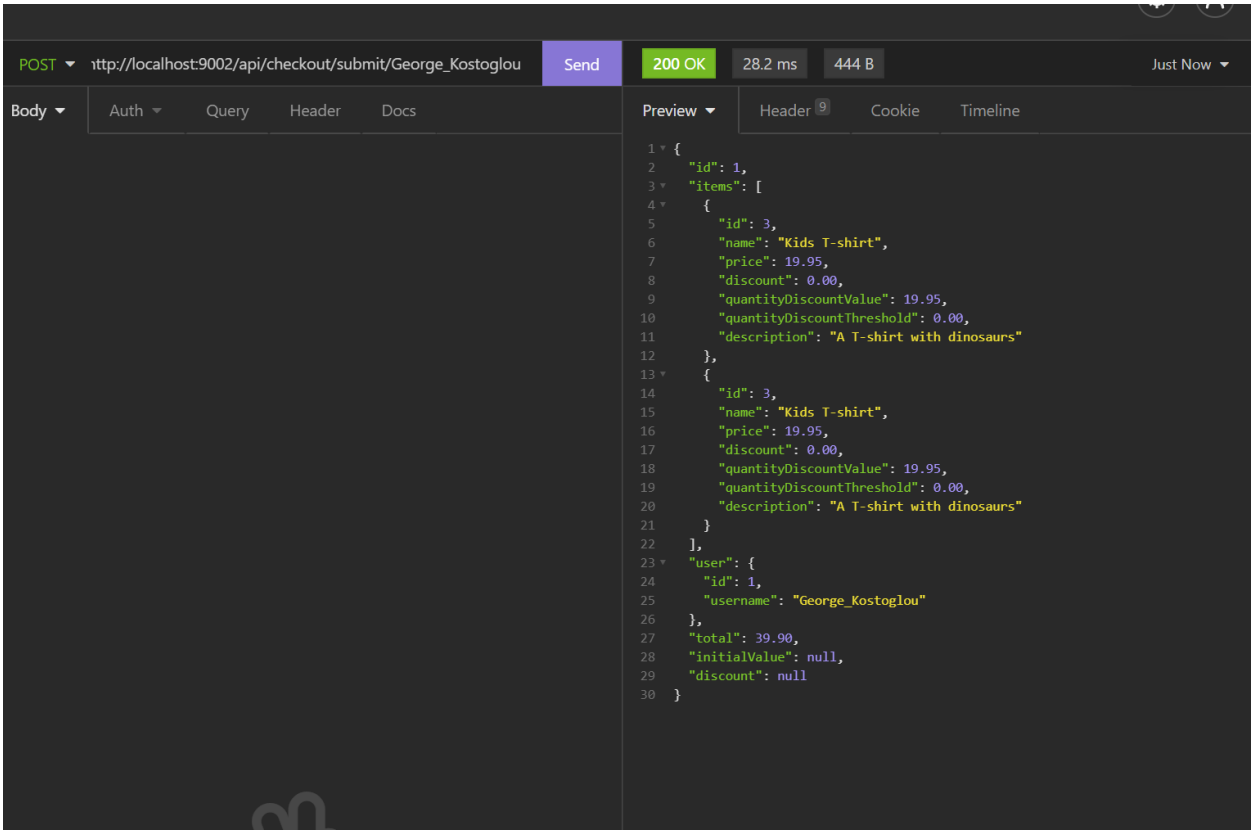
5.8 Αφαίρεση προϊόντος από καλάθι



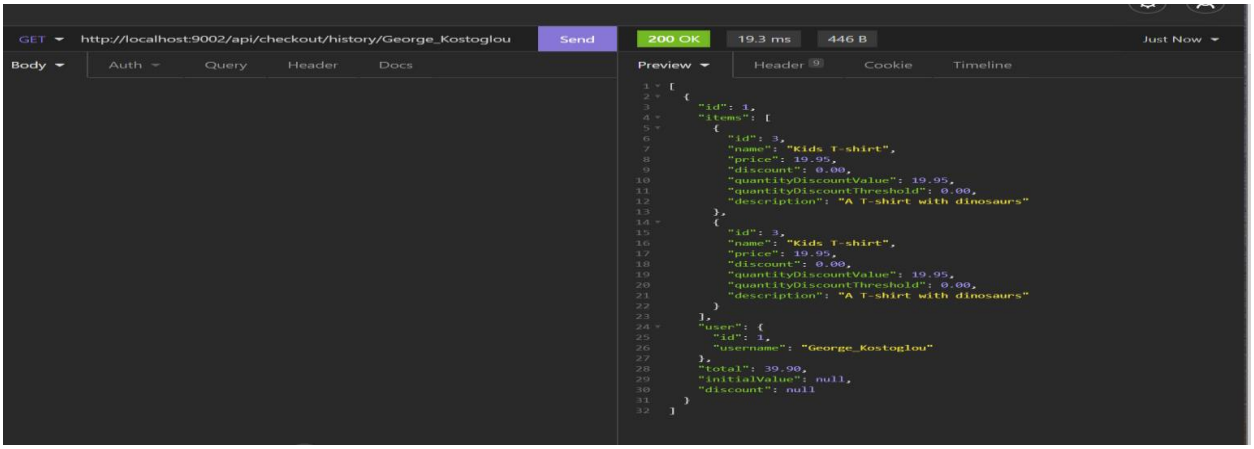
5.9. Προσθήκη προϊόντος στο καλάθι



5.10.Καταχώρηση παραγγελίας



5.11.Αναζήτηση παραγγελιών χρήστη



6. Συμπεράσματα & Επεκτάσεις

Η λειτουργία και ο σχεδιασμός ενός ηλεκτρονικού καταστήματος απαιτεί μεθοδικότητα. Εκμεταλλευμένοι τις σύγχρονες τεχνολογίες, το έργο του σχεδιασμού και υλοποίησής μπορεί να απλοποιηθεί σε πολλά επίπεδα συγκριτικά με τον σχεδιασμό αντίστοιχων εφαρμογών στο παρελθόν. Στην παρούσα εφαρμογή έγινε μια υλοποίηση ενός server-side web API που καλύπτει μερικές από τις βασικές λειτουργίες για ένα ηλεκτρονικό κατάστημα. Ως μελλοντική επέκταση της εφαρμογής θα μπορούσαν να προστεθούν κάποιες επιπλέον λειτουργίες όπως σύνδεση με εξωτερικό σύστημα για την ενημέρωση της εφαρμογής με το πραγματικό απόθεμα των προϊόντων για πώληση. Για την δημιουργία του front end της εφαρμογής θα μπορούσαν να χρησιμοποιηθούν τα framework Bootstrap και Angular. Το Bootstrap CSS framework παρέχει πρότυπα σχεδίασης για φόρμες , κουμπιά, πλοήγηση και άλλα δομικά στοιχεία διεπαφής. Το framework Angular είναι μια πλατφόρμα για την δημιουργία single-page client εφαρμογών χρησιμοποιώντας τις γλώσσες HTML και TypeScript. Τα βασικά δομικά στοιχεία του Angular είναι τα components, αυτά προσδιορίζουν τα views τα οποία είναι screen elements τα οποία μπορεί να επιλέγει το Angular να τα τροποποιεί σύμφωνα με την λογική και τα δεδομένα της εφαρμογής. Τα Components χρησιμοποιούν services τα οποία παρέχουν συγκεκριμένη λειτουργικότητα. Μέσω των services θα μπορεί η front end εφαρμογή να επικοινωνεί με το server-side web API. Επίσης θα μπορούσε να αντικατασταθεί η H2 in memory database, με μια persist on the disk βάση ώστε να αποθηκεύονται μόνιμα τα δεδομένα.

7. Βιβλιογραφία

https://www.w3schools.com/

https://www.tutorialspoint.com/mvc_framework/

https://www.redhat.com/en/topics/cloud-native-apps/what-is-a-Java-framework

https://www.javatpoint.com/hibernate-tutorial

https://www.tutorialspoint.com/hibernate/index.html

https://en.wikipedia.org