ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΠΤΥΞΗ ΔΙΑΔΙΚΤΥΑΚΗΣ ΕΦΑΡΜΟΓΗΣ ΣΕ JAVA ΜΕ ΤΗΝ ΠΛΑΤΦΟΡΜΑ SPRING ΚΑΙ ΑΝΑΛΥΣΗ ΤΗΣ ΓΙΑ ΕΚΠΑΙΔΕΥΤΙΚΟΥΣ ΣΚΟΠΟΥΣ

Διπλωματική Εργασία

του

Καραγιάννη Κωνσταντίνου

Θεσσαλονίκη, 6/2019

ΑΝΑΠΤΥΞΗ ΔΙΑΔΙΚΤΥΑΚΗΣ ΕΦΑΡΜΟΓΗΣ ΣΕ JAVA ΜΕ ΤΗΝ ΠΛΑΤΦΟΡΜΑ SPRING ΚΑΙ ΑΝΑΛΥΣΗ ΤΗΣ ΓΙΑ ΕΚΠΑΙΔΕΥΤΙΚΟΥΣ ΣΚΟΠΟΥΣ

Καραγιάννης Κωνσταντίνος
A.M. : mai18022

Πτυχίο Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας, 2016

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων καθηγητής
Ξυνόγαλος Στέλιος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25/6/2019

Ξυνόγαλος Στέλιος	Χατζηγεωργίου Αλέξανδρος	Ευαγγελίδης Γεώργιος
.....	.....	.....

Καραγιάννης Κωνσταντίνος

.....

Περίληψη

Στον πολύ ανταγωνιστικό τομέα της ανάπτυξης διαδικτυακών εφαρμογών, είναι πολύ σημαντικό η ανάπτυξη τους να γίνεται με τη μεγαλύτερη δυνατή ακρίβεια και αποδοτικότητα. Η διαδικασία ανάπτυξης των Java EE εφαρμογών είναι συχνά πολύπλοκη. Ο σκοπός του Spring Framework είναι να διευκολύνει την διαδικασία της ανάπτυξης των εφαρμογών. Δημιουργήθηκε ως μια εναλλακτική επιλογή, αντί για τις «βαριές» επιχειρηματικές, Java τεχνολογίες, όπως είναι τα Enterprise Java Beans (EJB).

Σκοπός της μελέτης είναι η ανάπτυξη εκπαιδευτικού υλικού, για την εκμάθηση του Spring Framework. Αναπτύχθηκε μια διαδικτυακή εφαρμογή με τη χρήση του Spring Framework και ένας ιστότοπος, ο οποίος περιλαμβάνει τη διαδικασία ανάπτυξης της εφαρμογής. Η εφαρμογή είναι ένα τυπικό παράδειγμα ενός ηλεκτρονικού καταστήματος. Το εκπαιδευτικό υλικό του ιστότοπου μπορεί να αξιοποιηθεί από κάθε ενδιαφερόμενο για την εκμάθηση της ανάπτυξης διαδικτυακών εφαρμογών σε Java με το Spring Framework.

Λέξεις Κλειδιά: Spring Framework, Διαδικτυακή εφαρμογή, Εκπαιδευτικό υλικό, Java EE, Spring Boot

Abstract

In the highly competitive environment of web development, it is crucial that web applications are developed accurately, economically and efficiently. The development process of Java EE applications is often complicated. The Spring framework was created to address the complexity of enterprise application development and simplify it. Spring was created as an alternative to heavier enterprise Java technologies, such as Enterprise Java Beans (EJB).

The purpose of this thesis is the creation of educational material for the learning of Spring Framework. A web application was developed using the Spring Framework. The application is a typical e-shop. Furthermore, a website was created, which contains the development process of the above web application. The educational material of the website can be used from anyone interested in learning how to develop Java web applications, using the Spring Framework.

Keywords: Spring Framework, Web application, Educational material, Java EE, Spring Boot

Ευχαριστίες

Θα ήθελα να εκφράσω τις ευχαριστίες μου, στον επιβλέποντα καθηγητή μου κ. Στέλιο Ξυνόγαλο, για την άριστη συνεργασία και τη βοήθεια του καθ' όλη τη διάρκεια της μελέτης.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Συμβολισμοί

JPA	Java Persistence API
POJO	Plain Old Java Object
ORM	Object to Relational Mapping
CRUD	Create, Read, Update, Delete
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
XML	eXtensible Markup Language
JSON	JavaScript Object Notation
DOM	Document Object Model
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
API	Application Programming Interface

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Στον πολύ ανταγωνιστικό τομέα της ανάπτυξης διαδικτυακών εφαρμογών, είναι πολύ σημαντικό η ανάπτυξη τους να γίνεται με τη μεγαλύτερη δυνατή ακρίβεια και αποδοτικότητα. Η διαδικασία ανάπτυξης των επιχειρηματικών Java EE εφαρμογών είναι συχνά πολύπλοκη. Ο σκοπός του Spring Framework είναι να διευκολύνει τη διαδικασία της ανάπτυξης των εφαρμογών. Δημιουργήθηκε ως μια εναλλακτική επιλογή, αντί για τις «βαριές» επιχειρηματικές, Java τεχνολογίες, όπως είναι τα Enterprise Java Beans (EJB).

1.2 Σκοπός – Στόχοι

Σκοπός της μελέτης είναι η ανάπτυξη εκπαιδευτικού υλικού, για την εκμάθηση του Spring Framework. Αναπτύχθηκε μια διαδικτυακή εφαρμογή, με τη χρήση του Spring Framework και ένας ιστότοπος, ο οποίος περιλαμβάνει τη διαδικασία ανάπτυξης της εφαρμογής. Το εκπαιδευτικό υλικό του ιστότοπου μπορεί να αξιοποιηθεί από κάθε ενδιαφερόμενο για την εκμάθηση της ανάπτυξης διαδικτυακών εφαρμογών σε Java με το Spring Framework.

1.3 Συνεισφορά

Στην παρούσα μελέτη:

1. Δημιουργήθηκε μια διαδικτυακή εφαρμογή, με την οποία αναδεικνύεται η χρήση του Spring Framework, για τη διευκόλυνση της διαδικασίας ανάπτυξης διακτυακών εφαρμογών. Η εφαρμογή που αναπτύχθηκε είναι ένα τυπικό ηλεκτρονικό κατάστημα.
2. Η αξιοποίηση της εφαρμογής ως μελέτη περίπτωσης σε συνδυασμό με τα αναλυτικά βήματα υλοποίησης όλων των πτυχών της θεωρούμε ότι θα αποτελέσει ένα χρήσιμο εργαλείο για κάθε ενδιαφερόμενο.

1.4 Διάρθρωση της μελέτης

Κεφάλαιο 2: Στο δεύτερο κεφάλαιο παρουσιάζονται τα σημαντικότερα Java frameworks και αναλύεται το Spring Framework.

Κεφάλαιο 3: Το τρίτο κεφάλαιο περιγράφει τις τεχνολογίες, οι οποίες χρησιμοποιήθηκαν στην εφαρμογή.

Κεφάλαιο 4: Στο τέταρτο κεφάλαιο περιγράφεται ο εξυπηρετητής και η βάση δεδομένων της εφαρμογής.

Κεφάλαιο 5: Στο πέμπτο κεφάλαιο παρουσιάζονται οι απαιτήσεις και η διαδικασία ανάπτυξης της εφαρμογής.

Κεφάλαιο 6: Στο έκτο κεφάλαιο παρουσιάζεται η εφαρμογή και οι λειτουργίες της.

Κεφάλαιο 7: Στο έβδομο κεφάλαιο περιγράφεται το περιεχόμενο του εκπαιδευτικού ιστότοπου.

Κεφάλαιο 8: Στο όγδοο κεφάλαιο παρουσιάζονται τα συμπεράσματα, τα οποία προέκυψαν και γίνονται προτάσεις για μελλοντική επέκταση του θέματος.

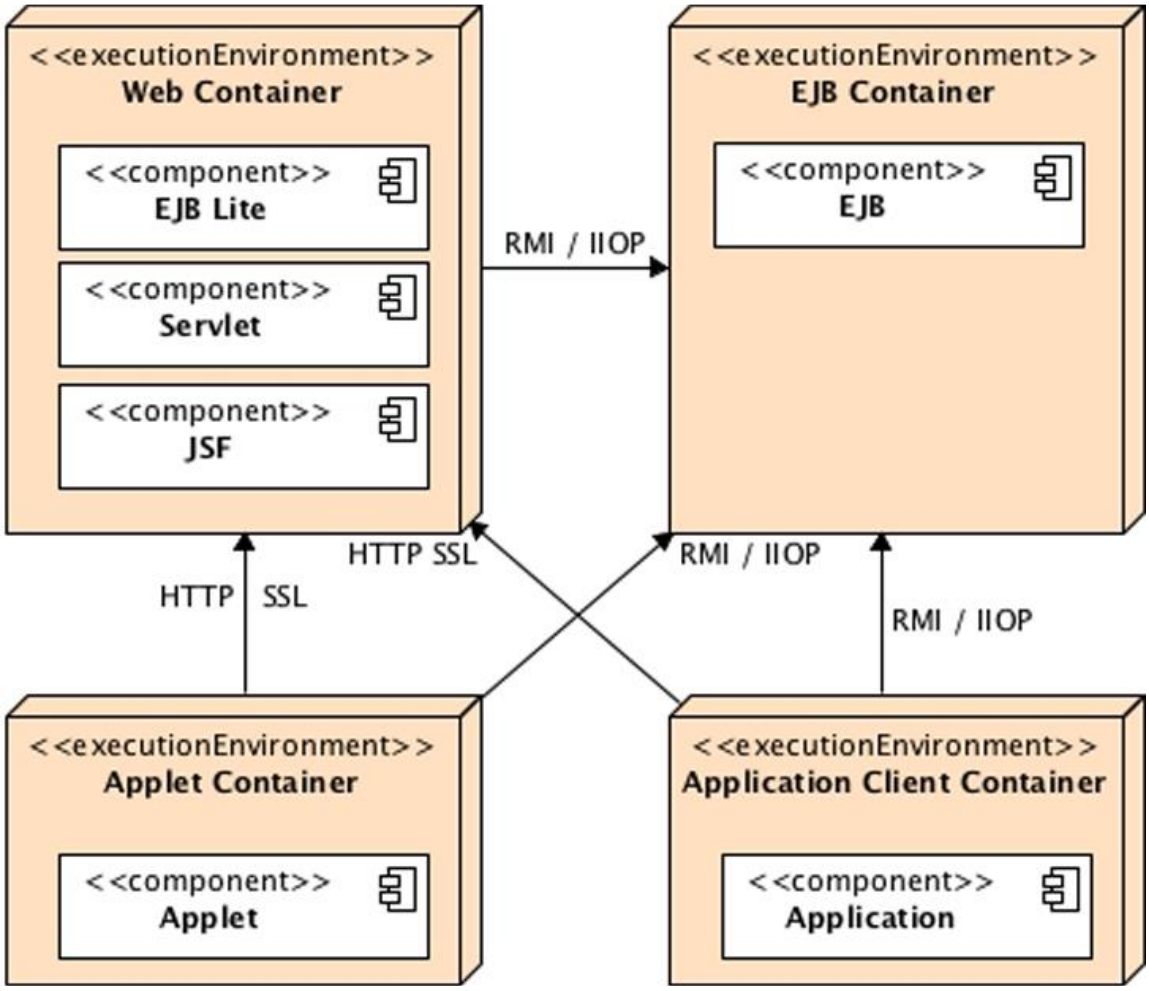
Κεφάλαιο 9: Στο τελευταίο κεφάλαιο παρατίθεται η βιβλιογραφία, η οποία χρησιμοποιήθηκε για την εκπόνηση της διπλωματικής εργασίας.

2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

2.1 Java Enterprise Edition (Java EE)

Η Java είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού, γενικού σκοπού, η οποία δημιουργήθηκε από τον James Gosling της εταιρίας Sun Microsystems το 1995 [1]. Είναι μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού και χρησιμοποιείται ιδιαίτερα για διαδικτυακές εφαρμογές, αρχιτεκτονικής πελάτη - εξυπηρετητή. Ένας σημαντικός σχεδιαστικός στόχος της Java ήταν η φορητότητα, που σημαίνει πως τα προγράμματα, τα οποία είναι γραμμένα σε Java, πρέπει να τρέχουν σε οποιοδήποτε συνδυασμό υλικού υπολογιστή και λειτουργικού συστήματος, όπως Windows, Linux, Unix. Αυτό επιτυγχάνεται, μεταγλωττίζοντας τον κώδικα της Java, σε μια ενδιάμεση αναπαράσταση, η οποία λέγεται Java bytecode, αντί γλώσσας μηχανής συγκεκριμένης αρχιτεκτονικής. Οι Java bytecode εντολές είναι ανάλογες με τις εντολές γλώσσας μηχανής, αλλά εκτελούνται στην εικονική μηχανή της Java (Java Virtual Machine).

Η Java Platform Enterprise Edition (Java EE) είναι μια έκδοση της Java, η οποία επεκτείνει την έκδοση Java Platform Standard Edition (Java SE), για την ανάπτυξη κατανεμημένων, εμπορικών εφαρμογών [2]. Πιο συγκεκριμένα, είναι ένα σύνολο από προδιαγραφές, οι οποίες εφαρμόζονται από διαφορετικά containers. Τα containers είναι Java EE περιβάλλοντα εκτέλεσης (Εικόνα 2-1), τα οποία έχουν συγκεκριμένο ρόλο, υποστηρίζουν ένα σύνολο από APIs και παρέχουν υπηρεσίες στα επιμέρους συστατικά (components) όπως ασφάλεια, πρόσβαση στη βάση δεδομένων και διαχείριση συναλλαγών. Ανάλογα με το είδος της εφαρμογής, επιλέγεται το κατάλληλο container, το οποίο θα τη φιλοξενήσει.



Εικόνα 2-1 Java EE Containers [3]

2.2 Java Web Frameworks

Στον πολύ ανταγωνιστικό τομέα της ανάπτυξης διαδικτυακών εφαρμογών, είναι πολύ σημαντικό η ανάπτυξη τους να γίνεται με τη μεγαλύτερη δυνατή ακρίβεια και αποδοτικότητα. Αυτό μπορεί να επιτευχθεί ευκολότερα, χρησιμοποιώντας πλαίσια εργασίας λογισμικού (software framework). Ένα πλαίσιο εργασίας λογισμικού είναι ένα καθολικό, επαναχρησιμοποιούμενο περιβάλλον λογισμικού, το οποίο παρέχει συγκεκριμένες λειτουργικότητες, ως μέρος μιας μεγαλύτερης πλατφόρμας λογισμικού, για να διευκολύνει την ανάπτυξη εφαρμογών λογισμικού [4].

2.2.1 Spring Framework

Το Spring είναι το πιο δημοφιλές framework ανάπτυξης Java EE εφαρμογών. Είναι ένα βασισμένο σε αιτήματα (request-based), ελαφρύ, τμηματικό, framework ανοιχτού λογισμικού, το οποίο δημιουργήθηκε από τον Rod Johnson το 2003 [5].

Ο σκοπός της δημιουργίας του ήταν να διευκολύνει τη διαδικασία της ανάπτυξης των εφαρμογών. Δημιουργήθηκε ως μια εναλλακτική επιλογή, αντί για τις «βαριές» επιχειρηματικές Java τεχνολογίες, όπως είναι τα Enterprise Java Beans (EJB). Το Spring προσφέρει ένα ελαφρύτερο προγραμματιστικό μοντέλο σε σχέση με τα EJBs [6]. Επιτρέπει την ανάπτυξη επιχειρηματικών εφαρμογών, χρησιμοποιώντας μόνο POJOs. Έτσι, για την ανάπτυξη μιας εφαρμογής, μπορεί να χρησιμοποιηθεί ένα web container όπως ο Tomcat, αντί για έναν εξυπηρετητή εφαρμογών (application server).

2.2.2 JavaServer Faces

Το JavaServer Faces (JSF) είναι ένα βασισμένο σε μέρη (component-based), MVC framework, μέρος της Java EE, το οποίο απλοποιεί τη δημιουργία διεπαφών χρήστη, χρησιμοποιώντας επαναχρησιμοποιούμενα μέρη της διεπαφής χρήστη (UI components) σε μια σελίδα [7].

Βασισμένο στο οδηγούμενο από μέρη (component-driven), σχεδιαστικό μοντέλο διεπαφών χρήστη (UI design-model), το JavaServer Faces χρησιμοποιεί XML αρχεία, τα οποία λέγονται view templates ή Facelets views. Το FacesServlet επεξεργάζεται τα αιτήματα, φορτώνει το κατάλληλο view template, δημιουργεί ένα δέντρο από μέρη (components), διαχειρίζεται τα γεγονότα και στέλνει την απάντηση στον πελάτη (συνήθως HTML). Η κατάσταση των μερών της διεπαφής χρήστη και των άλλων αντικειμένων αποθηκεύεται στο τέλος κάθε αιτήματος, σε μια διαδικασία που λέγεται stateSaving [8].

2.2.3 Google Web Toolkit

Το Google Web Toolkit είναι ένα framework ανοιχτού λογισμικού για τη δημιουργία RICH Internet Applications (RIA), χρησιμοποιώντας τη Java. Όταν μεταγλωττιστεί ο κώδικας, παράγεται κώδικας Javascript, κατάλληλος για κάθε πρόγραμμα περιήγησης [9]. Το GWT είναι ένα γενικό σύνολο εργαλείων για τη δημιουργία JavaScript λειτουργικότητας στην πλευρά του πελάτη, υψηλής αποδόσεως

και αφήνει σημαντικές αποφάσεις αρχιτεκτονικής αποκλειστικά στον προγραμματιστή [10].

Οι προγραμματιστές μπορούν να χρησιμοποιήσουν ολοκληρωμένα περιβάλλοντα ανάπτυξης (integrated development environment), όπως το Eclipse, για τη δημιουργία και την αποσφαλμάτωση της εφαρμογής του πελάτη (client), σαν να ήταν μια εφαρμογή Java. Επίσης, παρέχει μια βιβλιοθήκη από γραφικά στοιχεία (widgets), για τις περισσότερες εργασίες μιας εφαρμογής [11].

2.2.4 Apache Struts 2

Το Apache Struts 2 είναι ένα βασισμένο σε αιτήματα (request-based), MVC framework ανοιχτού λογισμικού, για την ανάπτυξη Java EE εφαρμογών. Αναπτύχθηκε αρχικά από τον Craig McClanahan και μετά δόθηκε στο Apache Foundation τον Μάιο του 2000, όπου έγινε γνωστό ως Jakarta Struts. Το Struts 2 χρησιμοποιεί και επεκτείνει το Java Servlet API, για να προωθήσει τη χρήση της αρχιτεκτονικής MVC [12].

Κάθε Struts 2 εφαρμογή αποτελείται από τρία κομμάτια [13]:

1. **Action class**: Είναι μια POJO κλάση.
2. **Controller**: Τα HTTP φίλτρα χρησιμοποιούνται ως controllers. Πραγματοποιούν βασικές λειτουργίες, όπως παρεμβολή και επαλήθευση αιτημάτων.
3. **View**: Χρησιμοποιείται για την εμφάνιση των δεδομένων. Το Struts 2 υποστηρίζει templating τεχνολογίες όπως JSP, Freemarker και Velocity.

Επίσης, το Struts 2 παρέχει υποστήριξη για AJAX και μπορεί να ενσωματώσει τεχνολογίες όπως το Hibernate και το Spring στην εφαρμογή.

2.2.5 Συγκριτική ανάλυση

Θα πραγματοποιηθεί συγκριτική ανάλυση των frameworks ως προς τα εξής:

- **Δημοτικότητα**: Η σειρά δημοτικότητας των παραπάνω frameworks είναι η εξής [14]:
    1. *Spring*: με βαθμολογία 89
    2. *JavaServer Faces*: με βαθμολογία 81
    3. *Google Web Toolkit*: με βαθμολογια 77
    4. *Apache Struts 2*: με βαθμολογία 64

Η βαθμολογία υπολογίζεται από τον μέσο όρο της βαθμολογίας GitHub, η οποία βασίζεται στον αριθμό των αστεριών του αντίστοιχου repository του framework στο GitHub και της βαθμολογίας Stack Overflow, η οποία βασίζεται στον αριθμό των ερωτήσεων στο Stack Overflow με ετικέτα το όνομα του αντίστοιχου framework.

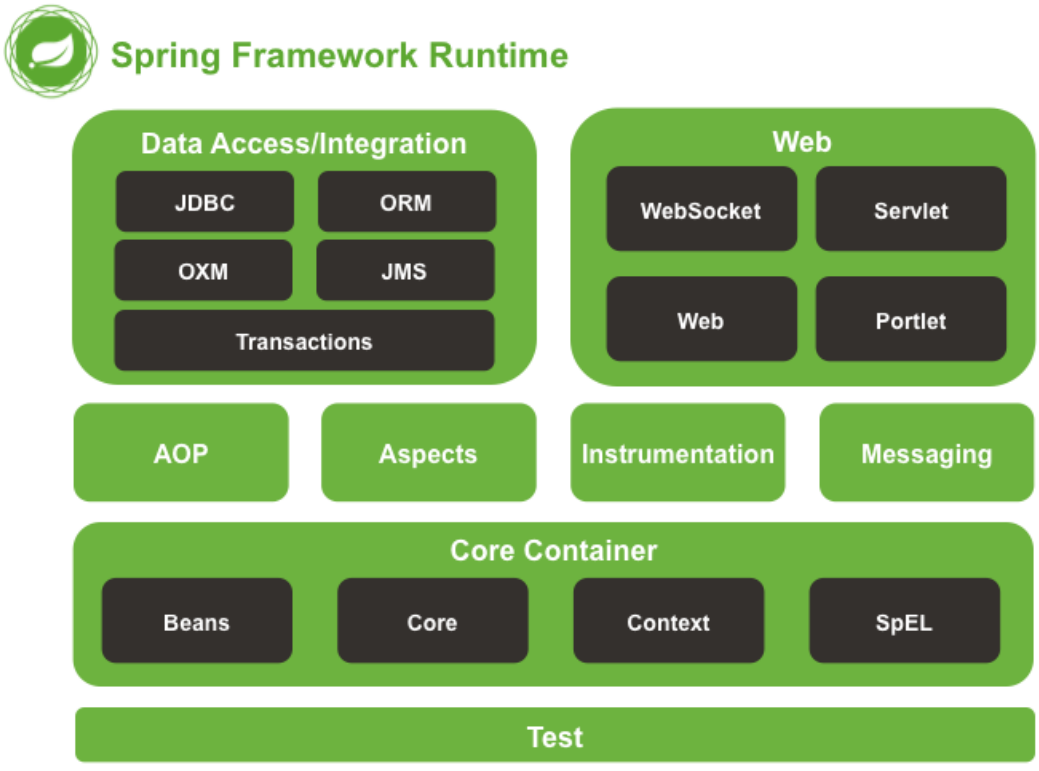
- **Κατηγορία**: Το Spring MVC και το Apache Struts 2 είναι βασισμένα σε αιτήματα (request-based) frameworks, τα οποία χρησιμοποιούν controllers για την διαχείριση των αιτημάτων, ενώ το JavaServer Faces είναι βασισμένο σε συστατικά (component-based), το οποίο αφαιρεί τα εσωτερικά της διαχείρισης των αιτημάτων και ενθυλακώνει την λογική σε επαναχρησιμοποιούμενα συστατικά [15]. Το Google Web Toolkit χρησιμοποιείται για τη δημιουργία πλούσιων διαδικτυακών εφαρμογών (Rich Internet Applications), οι οποίες τρέχουν στον περιηγητή του χρήστη και έχουν χαρακτηριστικά διεπαφής χρήστη, που συναντούνται σε εφαρμογές επιτραπέζιων υπολογιστών [16].
- **Πολυπλοκότητα**: Το Spring και το JavaServer Faces είναι αρκετά πολύπλοκα. Για να χρησιμοποιήσει κάποιος το Spring MVC, χρειάζεται να γνωρίζει το τμήμα (module) Spring Core, το οποίο πραγματοποιεί την έγχυση εξαρτήσεων (dependency injection). Επίσης, απαιτείται αρκετός κώδικας XML ή Java για ρυθμίσεις. Η πολυπλοκότητα του JavaServer Faces προέρχεται από τις προδιαγραφές Java EE και την ανάγκη χρήσης ενός Java EE εξυπηρετητή εφαρμογών (application server) [17]. Το Apache Struts 2 και το Google Web Toolkit είναι λιγότερο πολύπλοκα. Η γνώση της αρχιτεκτονικής MVC είναι αναγκαία για τη χρήση του Apache Struts 2, ενώ για το Google Web Toolkit, ο προγραμματιστής χρειάζεται να γνωρίζει μόνο Java. Μπορεί να σέρνει τα συστατικά (components) στη οθόνη σε λειτουργία σχεδίασης (Design mode) και το Google Web Toolkit θα παράγει αυτόματα τον αντίστοιχο κώδικα.
- **Documentation**: Το Spring, το JavaServer Faces που υποστηρίζεται από την Oracle και το Google Web Toolkit που υποστηρίζεται από την Google, περιέχουν εκτεταμένο υποστηρικτικό υλικό στις επίσημες ιστοσελίδες τους, όπως εγχειρίδια και αναλυτικούς οδηγούς για αρχάριους και προχωρημένους προγραμματιστές. Αντίστοιχα, το Apache Struts 2

περιέχει και αυτό υποστηρικτικό υλικο στην ιστοσελίδα του, το οποίο όμως είναι πιο περιορισμένο και όχι τόσο οργανωμένο, όσο των παραπάνω frameworks.

2.3 Spring Framework

2.3.1 Spring Modules

Το Spring Framework αποτελείται από χαρακτηριστικά (features), τα οποία είναι οργανωμένα σε 20 τμήματα (modules). Αυτά τα τμήματα είναι ομαδοποιημένα στο Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging και Test, όπως φαίνεται στην Εικόνα 2-2.

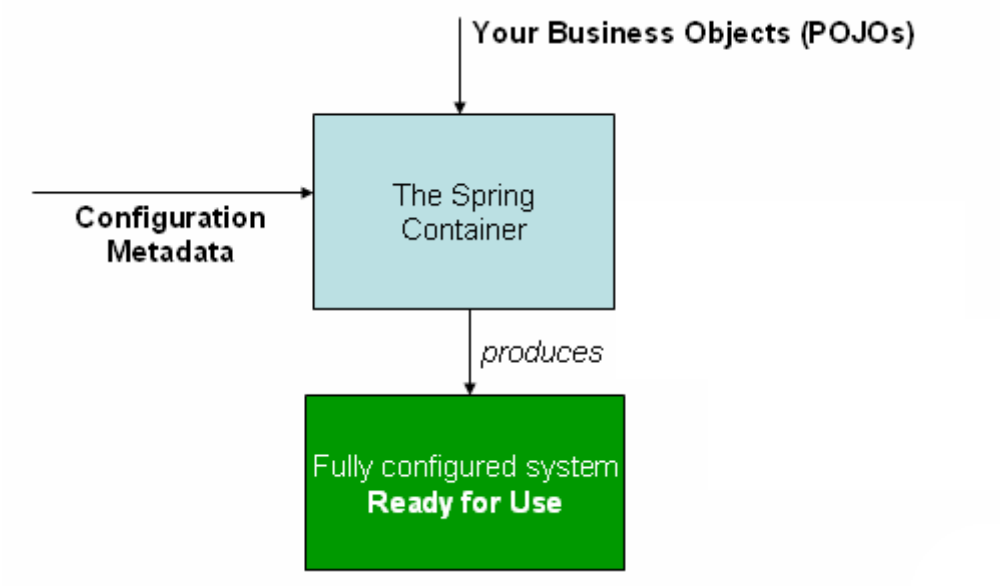


Εικόνα 2-2 Spring Framework Modules [18]

2.3.2 Spring IoC Container

Η αντιστροφή των εξαρτήσεων (Inversion of Control) είναι γνωστή και ως dependency injection (DI). Είναι μια διαδικασία, στην οποία τα αντικείμενα ορίζουν τις εξαρτήσεις τους (τα άλλα αντικείμενα με τα οποία δουλεύουν), μόνο μέσω παραμέτρων κατασκευαστή, παραμέτρων σε μια μέθοδο «εργοστάσιο» (factory method) ή μέσω ιδιοτήτων (properties), τα οποία ορίζονται στο στιγμιότυπο αντικειμένου, αφού αυτό κατασκευαστεί ή επιστραφεί από μέθοδο «εργοστάσιο». Το container ύστερα εγχέει (inject) τις εξαρτήσεις, όταν δημιουργεί το αντικείμενο κόκκο (bean). Η διαδικασία είναι ουσιαστικά η αντίστροφη, από το να ελέγχει το αντικείμενο - bean την αρχικοποίηση ή την τοποθεσία των εξαρτήσεων του, κατασκευάζοντας απευθείας τις κλάσεις.

Η διεπαφή org.springframework.context.ApplicationContext αντιπροσωπεύει το Spring IoC container και είναι υπεύθυνη για την αρχικοποίηση, ρύθμιση και συναρμολόγηση των αντικειμένων (Εικόνα 2-3). Το container παίρνει τις οδηγίες, ως προς ποια αντικείμενα να αρχικοποιήσει, να ρυθμίσει και να συναρμολογήσει, διαβάζοντας τα μεταδεδομένα ρυθμίσεων (configuration metadata). Τα μεταδεδομένα ρυθμίσεων αναπαρίστανται με XML, Java annotations ή με κώδικα Java.



Εικόνα 2-3 Spring Inversion Of Control Container [19]

Στο Spring, τα αντικείμενα που αποτελούν τη ραχοκοκαλιά της εφαρμογής και διαχειρίζονται από το Spring IoC Container λέγονται κόκκοι (beans). Ο κόκκος είναι ένα

αντικείμενο, το οποίο αρχικοποιείται, δημιουργείται και διαχειρίζεται από το Spring IoC container. Αλλιώς, είναι ένα από τα πολλά αντικείμενα τις εφαρμογής.

3 Τεχνολογίες εφαρμογής

Στο κεφάλαιο αυτό θα αναλυθούν οι τεχνολογίες, οι οποίες χρησιμοποιήθηκαν για τη δημιουργία της εφαρμογής. Η εφαρμογή φιλοξενήθηκε στην πλατφόρμα Heroku και η βάση δεδομένων που χρησιμοποιήθηκε ήταν η PostgreSQL. Για την υλοποίηση του εξυπηρετητή, χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java (Java EE), το framework Spring Boot και η βιβλιοθήκη Spring Data JPA. Την υλοποίηση του JPA παρείχε το Hibernate.

Για την υλοποίηση του πελάτη, χρησιμοποιήθηκαν οι τεχνολογίες HTML, CSS, Javascript, Typescript και το framework Angular.

3.1 Spring Boot

Το Spring Boot είναι ένα ανοιχτού κώδικα Java framework και αποτελεί τη λύση προσέγγισης «κανόνας πάνω από ρυθμίσεις» (convention-over-configuration) του Spring, για τη δημιουργία αυτόνομων εφαρμογών, επιπέδου παραγωγής, τις οποίες απλά τρέχουμε. Περιέχει ρυθμίσεις και βιβλιοθήκες του Spring, οι οποίες είναι οι πιο κατάλληλες, σύμφωνα με την άποψη του Spring για την έναρξη της ανάπτυξης της εφαρμογής με τη λιγότερη διαδικασία ρυθμίσεων.

Οι στόχοι του Spring Boot είναι οι εξής [20]:

- Η αποφυγή των πολύπλοκων ρυθμίσεων XML (configuration) του Spring
- Η ανάπτυξη Spring εφαρμογών επιπέδου παραγωγής, με τον ευκολότερο τρόπο
- Η μείωση του χρόνου ανάπτυξης της εφαρμογής
- Η παροχή ενός ευκολότερου τρόπου έναρξης της διαδικασίας ανάπτυξης της εφαρμογής

3.2 JPA

Το Java Persistence API (JPA) είναι προδιαγραφές μέρος της Java EE, για την αποθήκευση αντικείμενων της Java στη σχεσιακή βάση δεδομένων. Θεωρείται η πιο συνηθισμένη προσέγγιση για Object to Relational Mapping (ORM) στη βιομηχανία της Java [21].

Είναι ένα σύνολο από διεπαφές, οι οποίες απαιτούν μια υλοποίηση. Δημοφιλείς υλοποιήσεις του JPA είναι το Hibernate, το EclipseLink και το Apache OpenJPA. Η υλοποίηση του JPA λέγεται πάροχος διατήρησης (persistence provider).

Το JPA επιτρέπει στους προγραμματιστές να δουλεύουν απευθείας με Plain Old Java Objects (POJO), αντί με εντολές SQL, για να πραγματοποιήσουν τις διάφορες λειτουργίες CRUD στη σχεσιακή βάση δεδομένων.

Η αντιστοίχιση των κλάσεων της Java με τους πίνακες της βάσης δεδομένων πραγματοποιείται με Java annotations ή XML.

3.3 Spring Data JPA

Το Spring Data JPA είναι μέρος των βιβλιοθηκών Spring Data και δίνει τη δυνατότητα δημιουργίας JPA repositories. Είναι ένα framework, το οποίο παρέχει ένα επιπλέον επίπεδο αφαίρεσης πάνω από τον πάροχο JPA [22].

Ο στόχος της αφαίρεσης Spring Data repository είναι να μειώσει σημαντικά τον κώδικα που απαιτείται για να δημιουργήσουμε το επίπεδο πρόσβασης στη βάση δεδομένων (data access) για τις διάφορες αποθήκες δεδομένων (persistence stores).

Το βασικό repository, που παρέχει το Spring Data είναι το CrudRepository. Αυτό παρέχει τις βασικές λειτουργίες CRUD. Για παράδειγμα, δημιουργούμε το ακόλουθο repository για τον πίνακα της βάσης categories (id, name). Επεκτείνοντας το JpaRepository, το Spring Data JPA μας παρέχει τις υλοποιήσεις των μεθόδων για την εισαγωγή, ανάκτηση, διαγραφή και ενημέρωση εγγραφών στον πίνακα.

```
@Repository
public interface CategoryRepository extends JpaRepository<Category, Integer>{}
```

3.4 PostgreSQL

Η PostgreSQL είναι μία ανοιχτού λογισμικού, σχεσιακή βάση δεδομένων, η οποία χρησιμοποιεί και επεκτείνει τη γλώσσα SQL. Μπορεί να χειριστεί φόρτο εργασίας από μικρές εφαρμογές σε ένα μηχάνημα, μέχρι μεγάλες, οι οποίες εξυπηρετούν πολλούς χρήστες ταυτόχρονα.

Έχει ισχυρή φήμη για την αποδεδειγμένη αρχιτεκτονική της, την αξιοπιστία, την ακεραιότητα των δεδομένων και την επεκτασιμότητα της. Τρέχει σε όλα τα μεγάλα

λειτουργικά συστήματα, είναι συμμορφωμένη με το ACID από το 2001 και παρέχει ισχυρά πρόσθετα, όπως το PostGIS geospatial database extender [23].

3.5 HTML

Η HTML (Hypertext Markup Language) είναι markup γλώσσα, για τη δημιουργία ιστοσελίδων και διαδικτυακών εφαρμογών. Περιγράφει τη δομή της ιστοσελίδας σημασιολογικά. Τα προγράμματα περιήγησης (web browsers) λαμβάνουν το HTML αρχείο από τον εξυπηρετητή και το εμφανίζουν στον χρήστη [24].

Δομικά στοιχεία των HTML σελίδων είναι οι ετικέτες HTML (tags). Χρησιμοποιούμε ετικέτες για να προσθέσουμε περιεχόμενο στη σελίδα. Οι περισσότερες ετικέτες αποτελούνται από ζευγάρια `<p> </p>`, εκτός από μερικές όπως η `<img>`. Το κλασσικό παράδειγμα Hello world σε HTML, γράφεται ως εξής (Κώδικας 3-1):

```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Κώδικας 3-1 Πρόγραμμα Hello World σε HTML

3.6 CSS

Η CSS (Cascading Style Sheets) είναι μια γλώσσα μορφοποίησης φύλλου (style sheet), η οποία χρησιμοποιείται για την περιγραφή της αναπαράστασης ενός αρχείου, που είναι γραμμένο σε HTML. Είναι σχεδιασμένη, έτσι ώστε να επιτρέπει τον διαχωρισμό του περιεχομένου και της αναπαράστασης, συμπεριλαμβανομένων της διάταξης, των χρωμάτων και γραμματοσειρών [25].

Τα πλεονεκτήματα της χρήσης CSS είναι τα εξής [26]:

- **Εξοικονόμηση χρόνου**: Μπορούμε, να γράψουμε το CSS αρχείο μια φορά και να το χρησιμοποιήσουμε σε πολλά αρχεία HTML.
- **Η σελίδα φορτώνεται γρηγορότερα**: Δεν χρειάζεται, να προσθέτουμε χαρακτηριστικά μορφοποίησης (style attributes) σε κάθε HTML ετικέτα.

Γράφουμε μια φορά τον κανόνα για την ετικέτα και εφαρμόζεται σε κάθε εμφάνιση της στο αρχείο.

- **Ευκολία συντήρησης**: Όταν θέλουμε, να κάνουμε μια καθολική αλλαγή, αλλάζουμε τον κανόνα και όχι κάθε εμφάνιση της ετικέτας.
- **Συμβατότητα με διαφορετικές συσκευές**: Μπορούμε, να προσαρμόσουμε και να βελτιστοποιήσουμε την εμφάνιση του περιεχομένου, ανάλογα με τον τύπο της συσκευής.

3.7 Javascript

Στην εφαρμογή χρησιμοποιήθηκε έμμεσα η Javascript, αφού η TypeScript μεταγλωττίζεται σε Javascript. Η Javascript είναι μια υψηλού επιπέδου, διερμηνευμένη γλώσσα προγραμματισμού, η οποία ακολουθεί τις προδιαγραφές ECMAScript [27].

Χρησιμοποιείται κυρίως για τη δημιουργία διαδραστικών και δυναμικών ιστοσελίδων, οπότε αποτελεί αναπόσπαστο κομμάτι των διαδικτυακών εφαρμογών. Η μεγάλη πλειοψηφία των ιστοσελίδων χρησιμοποιεί Javascript. Αν και αρχικά ήταν γνωστή ως γλώσσα σεναρίου των ιστοσελίδων, τώρα χρησιμοποιείται και στη πλευρά του εξυπηρετητή, όπως π.χ. στο Node.js [28].

3.8 TypeScript

Η TypeScript είναι μια αυστηρά τυποποιημένη (strongly typed), αντικειμενοστρεφής γλώσσα. Σχεδιάστηκε από τον Anders Hejlsberg (σχεδιαστής της C#) της Microsoft. Είναι ένα τυποποιημένο, υπερσύνολο της Javascript και μεταγλωττίζεται σε Javascript [29]. Το framework Angular έχει γραφτεί αποκλειστικά σε Typescript.

Η TypeScript προήλθε από το γεγονός ότι η χρήση της Javascript, στην ανάπτυξη εφαρμογών μεγάλης έκτασης, έχει ως αποτέλεσμα τη δημιουργία πολύπλοκου κώδικα [30]. Σε αντίθεση με την Javascript, ο μεταγλωττιστής της TypeScript μεταγλωττίζει τον κώδικα και παράγει σφάλματα μεταγλώττισης, εάν βρει συντακτικά λάθη. Επίσης, η TypeScript είναι αυστηρά τυποποιημένη και υποστηρίζει έννοιες αντικειμενοστρεφούς προγραμματισμού, όπως κλάσεις, διεπαφές και κληρονομικότητα.

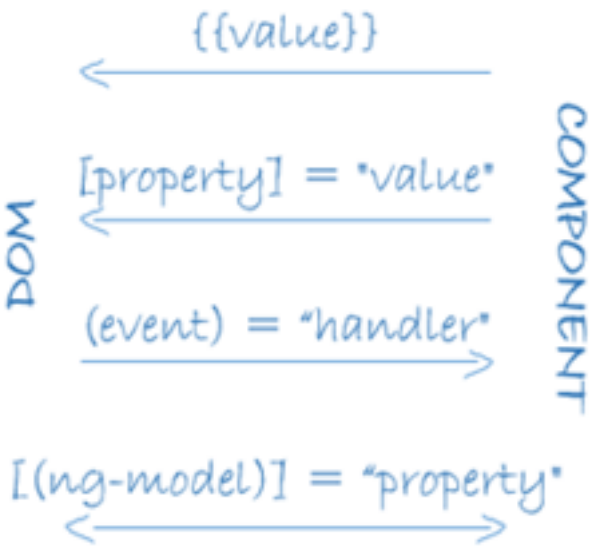
3.9 Angular

Η Angular είναι ένα ανοιχτού λογισμικού, front-end framework της Google, για τη δημιουργία εφαρμογών πελάτη, χρησιμοποιώντας HTML και TypeScript. Είναι γραμμένη σε TypeScript. Παρέχει τις βασικές και προαιρετικές λειτουργίες ως ένα σύνολο από βιβλιοθήκες TypeScript, τις οποίες μπορούμε να εισάγουμε στις εφαρμογές [31].

Μια Angular εφαρμογή αποτελείται από NgModules. Ένα module είναι ένα συνεκτικό τμήμα κώδικα, το οποίο παρέχει ένα πλαίσιο μεταγλώττισης (compilation context) για τα components και πραγματοποιεί μια συγκεκριμένη λειτουργία. Κάθε εφαρμογή έχει ένα module «ρίζα» (root module), που λέγεται AppModule, το οποίο παρέχει τον μηχανισμό εκκίνησης της εφαρμογής.

Σημαντικό δομικό στοιχείο της Angular είναι τα components. Component είναι μια κλάση με το decorator `@Component` και ελέγχει ένα κομμάτι της οθόνης, το οποίο λέγεται view. Η κλάση περιέχει τα δεδομένα, τη λογική της εφαρμογής και είναι συσχετισμένη με ένα HTML template. Κάθε Angular εφαρμογή έχει τουλάχιστον ένα component, το οποίο λέγεται component «ρίζα» (root component). Αυτό μπορεί, να περιέχει και άλλα components, τα οποία λέγονται components «παιδιά» (child components).

Τέλος, η Angular παρέχει οδηγίες (directives) για την αλλαγή της δομής του DOM, προσθέτοντας, αφαιρώντας ή τροποποιώντας στοιχεία (elements) και σύμβολα (binding markup) για την σύνδεση των δεδομένων της εφαρμογής με το DOM. Το παρακάτω διάγραμμα δείχνει τις τέσσερις μορφές σύνδεσης δεδομένων (Εικόνα 3-1). Κάθε μορφή έχει μια κατεύθυνση: προς το DOM, από το DOM ή και τα δύο.



Εικόνα 3-1 Angular – Data Binding [32]

4 Δημιουργία εξυπηρετητή και βάσης δεδομένων

Στο κεφάλαιο αυτό περιγράφονται ο εξυπηρετητής, ο οποίος θα φιλοξενεί την εφαρμογή, η λογική αποθήκευσης των εικόνων στο Amazon S3 και οι πίνακες της βάσης δεδομένων.

4.1 Heroku

Για την φιλοξενία της εφαρμογής επιλέχθηκε το Heroku. Το Heroku είναι μια πλατφόρμα νέφους ως υπηρεσία (cloud platform as a service), η οποία υποστηρίζει πολλές γλώσσες προγραμματισμού [33]. Κάνει τη διαδικασία της εγκατάστασης (deployment), της ρύθμισης και της διαχείρισης της εφαρμογής, όσο πιο απλές γίνονται, επιτρέποντας στους προγραμματιστές να ασχοληθούν αποκλειστικά με την ανάπτυξη του κώδικα.

Η πλατφόρμα Heroku χρησιμοποιεί το μοντέλο του container, για να τρέξει και να κλιμακώσει όλες τις εφαρμογές του Heroku. Τα containers, τα οποία χρησιμοποιούνται στο Heroku λέγονται dynos. Τα dynos είναι απομονωμένα, εικονικά Linux containers, τα οποία είναι σχεδιασμένα να εκτελούν κώδικα, σύμφωνα με εντολές του χρήστη. Η εφαρμογή μπορεί να κλιμακωθεί χρησιμοποιώντας έναν συγκεκριμένο αριθμό από dynos, ανάλογα με τις απαιτήσεις των πόρων [34].

Η διαδικασία της εγκατάστασης της εφαρμογής, πραγματοποιείται με το δημοφιλές σύστημα ελέγχου εκδόσεων (version control system) Git. Επίσης, το Heroku παρέχει πάνω από 150 πρόσθετα, όπως η PostgreSQL, για την επέκταση της εφαρμογής.

4.2 Amazon S3

Κάθε dyno έχει το δικό του εφήμερο σύστημα αρχείων [35]. Οποιεσδήποτε αλλαγές στο σύστημα αρχείων διαρκούν στο dyno μέχρι να τερματιστεί ή να επανεκκινηθεί. Κάθε dyno εκκινεί με ένα καθαρό αντίγραφο του σύστηματος αρχείων, από την πιο πρόσφατη εγκατάσταση. Οπότε, για την αποθήκευση τον φωτογραφιών της εφαρμογής, θα χρησιμοποιήσουμε την υπηρεσία Amazon S3. Το Amazon S3 είναι μια υπηρεσία αποθήκευσης του Amazon Web Services (AWS), στην οποία μπορούμε να αποθηκεύσουμε αντικείμενα μέσω μιας διεπαφής υπηρεσίας Ιστού (web service interface) [36].

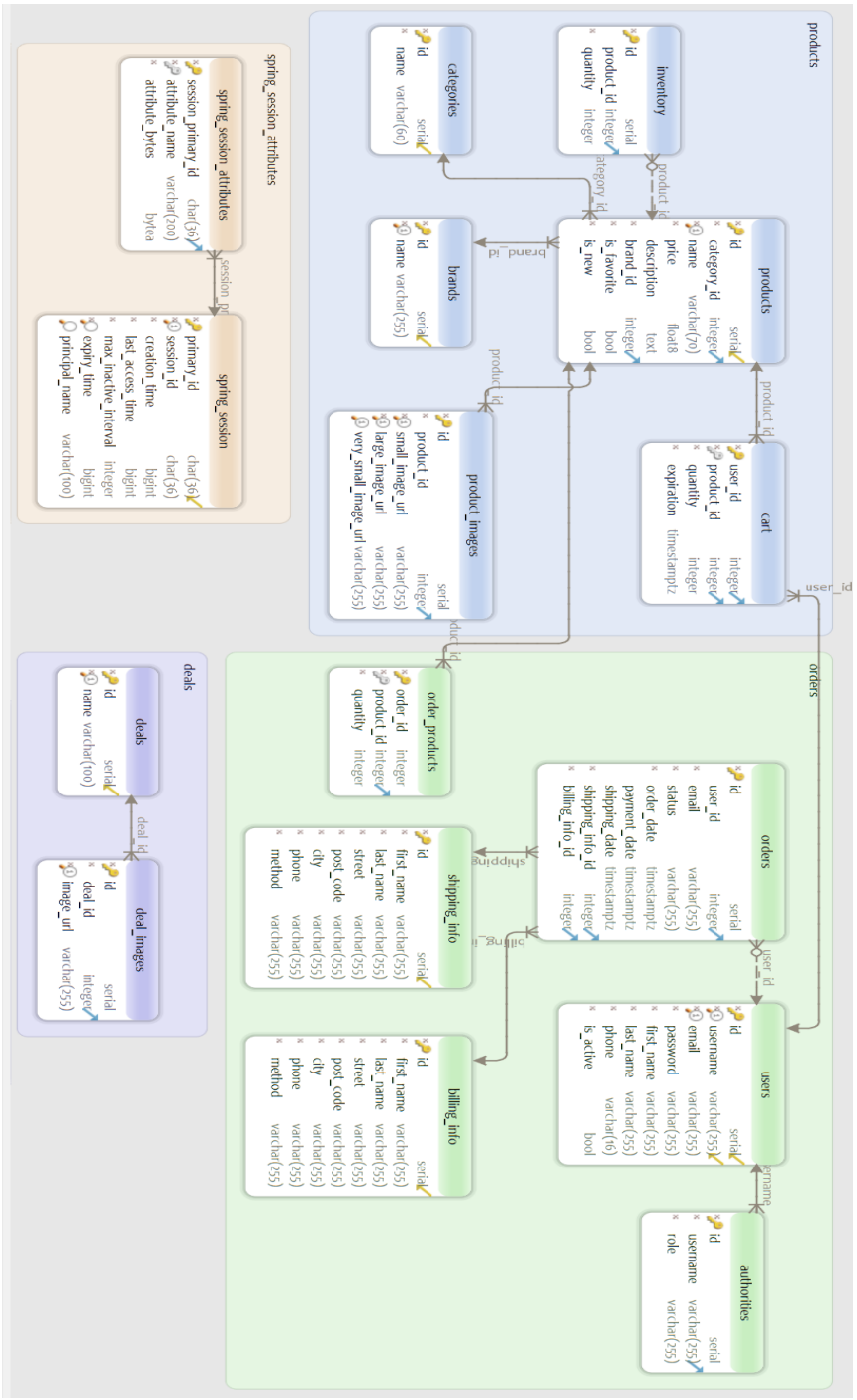
Βασικές μονάδες αποθήκευσης του Amazon S3 είναι τα αντικείμενα, τα οποία οργανώνονται σε κάδους (buckets). Κάθε αντικείμενο έχει ένα μοναδικό κλειδί (key). Μπορούμε να διαχειριστούμε τους κάδους μέσω της κονσόλας του Amazon S3 ή προγραμματιστικά μέσω του AWS SDK και του Amazon S3 REST API. Τα ονόματα του κάδου και του κλειδιού είναι επιλεγμένα έτσι ώστε να μπορούμε να ανακτήσουμε τα αντικείμενα μέσω HTTP URL: *http://s3.amazonaws.com/bucket/key*.

Στην εφαρμογή υπάρχουν έξι κατηγορίες προϊόντων (ιχθυέλαια, αρώματα, μέταλλα, σαμπουάν, υπερτροφές, βιταμίνες) και κάθε προϊόν έχει την ίδια εικόνα σε τρία μεγέθη (πολύ μικρό, μικρό, μεγάλο). Οι εικόνες των προϊόντων αποθηκεύονται στον κάδο springeshop-bucket και το URL του κάδου είναι το *https://s3.eu-central-1.amazonaws.com/springeshop-bucket/*. Για παράδειγμα, η μικρή εικόνα του προϊόντος apivita-shampoo-dry-dandruff-250ml, το οποίο ανήκει στην κατηγορία σαμπουάν, βρίσκεται στο URL *https://s3.eu-central-1.amazonaws.com/springeshop-bucket/products/small/shampoos/apivita-shampoo-dry-dandruff-250ml-small.jpg* .

4.3 Δημιουργία βάσης δεδομένων PostgreSQL

4.3.1 Σχεσιακό σχήμα βάσης δεδομένων

Αφού προστέθηκε το πρόσθετο (add-on) PostgreSQL στην εφαρμογή του Heroku, δημιουργήθηκε η βάση δεδομένων της εφαρμογής, η οποία περιέχει 16 πίνακες. Στην Εικόνα 4-1 της επόμενης σελίδας, δίνεται το σχεσιακό σχήμα της βάσης δεδομένων.



Εικόνα 4-1 Σχεσιακό σχήμα βάσης δεδομένων

4.3.2 Πίνακες βάσης δεδομένων

Η βάση δεδομένων περιλαμβάνει τους εξής πίνακες:

- Πίνακας categories

  Ο πίνακας περιέχει τις κατηγορίες των προϊόντων (Πίνακας 4-1).

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
name (character varying (60))	Όνομα κατηγορίας

Πίνακας 4-1 Πίνακας categories

- Πίνακας brands

  Ο πίνακας περιέχει τις εταιρίες των προϊόντων (Πίνακας 4-2)..

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
name (character varying (60))	Όνομα εταιρίας

Πίνακας 4-2 Πίνακας brands

- Πίνακας products

  Ο πίνακας περιέχει τα προϊόντα του καταστήματος (Πίνακας 4-3). Μια κατηγορία μπορεί να περιέχει πολλά προϊόντα, ενώ ένα προϊόν ανήκει μόνο σε μια κατηγορία. Δηλαδή, μια γραμμή του πίνακα categories συσχετίζεται με πολλές γραμμές του πίνακα products και μια γραμμή του πίνακα products συσχετίζεται με μία μόνο του categories.

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
category_id (integer)	Ξένο κλειδί - Δείχνει το id της κατηγορίας, που ανήκει
name (character varying (70))	Όνομα κατηγορίας
price (double precision)	Τιμή του προϊόντος
description (text)	Περιγραφή του προϊόντος
brand_id (integer)	Ξένο κλειδί - Δείχνει το id της εταιρίας, που ανήκει
is_favorite (boolean)	Δηλώνει εάν είναι αγαπημένο
is_new (boolean)	Δηλώνει εάν είναι καινούριο

Πίνακας 4-3 Πίνακας products

- Πίνακας product_images

  Ο πίνακας περιέχει τις εικόνες των προϊόντων (Πίνακας 4-4). Μια εικόνα αντιστοιχεί σε ένα προϊόν και το αντίστροφο.

Όνομα / Τύπος πεδίου	**Λειτουργία πεδίου**
id (serial)	Κύριο κλειδί
product_id (integer)	Ξένο κλειδί - Δείχνει το id του προϊόντος
small_image_url (varchar (255))	URL της εικόνας μικρού μεγέθους
large_image_url (varchar (255))	URL της εικόνας μεγάλου μεγέθους
very_small_image_url (varchar (255))	URL της εικόνας πολύ μικρού μεγέθους

Πίνακας 4-4 Πίνακας product_images

- Πίνακας inventory

  Ο πίνακας περιέχει το απόθεμα των προϊόντων (Πίνακας 4-5). Μια γραμμή του πίνακα inventory αντιστοιχεί σε μια γραμμή του πίνακα products και το αντίστροφο.

Όνομα / Τύπος πεδίου	**Λειτουργία πεδίου**
id (serial)	Κύριο κλειδί
product_id (integer)	Ξένο κλειδί - Δείχνει το id του προϊόντος
quantity (integer)	Απόθεμα του προϊόντος

Πίνακας 4-5 Πίνακας inventory

- Πίνακας deals

  Ο πίνακας περιέχει τις προσφορές του καταστήματος (Πίνακας 4-6).

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
name (character varying (100))	Όνομα προσφοράς

Πίνακας 4-6 Πίνακας deals

- Πίνακας deal_images

  Ο πίνακας περιέχει τις εικόνες των προσφορών (Πίνακας 4-7). Μια εικόνα αντιστοιχεί σε μια προσφορά και το αντίστροφο.

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
deal_id (integer)	Ξένο κλειδί - Δείχνει το id της προσφοράς
image_url (integer)	URL της εικόνας της προσφοράς

Πίνακας 4-7 Πίνακας deal_images

- Πίνακας users

  Ο πίνακας περιέχει τους εγγεγραμμένους χρήστες του καταστήματος (Πίνακας 4-8).

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
username (varchar (255))	Όνομα λογαριασμού του χρήστη
email (varchar (255))	Email του χρήστη
password (varchar (255))	Κωδικός του χρήστη
first_name (varchar (255))	Όνομα του χρήστη
last_name (varchar (255))	Επίθετο του χρήστη
phone (varchar (255))	Τηλέφωνο του χρήστη
is_active (boolean)	Δηλώνει, εάν είναι ενεργοποιημένος ο λογαριασμός του

Πίνακας 4-8 Πίνακας users

- Πίνακας authorities

  Ο πίνακας περιέχει τον ρόλο του χρήστη (Πίνακας 4-9).

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
username (varchar (255))	Ξένο κλειδί - Δείχνει το όνομα του χρήστη
role (varchar (255))	Ρόλος του χρήστη

Πίνακας 4-9 Πίνακας authorities

- Πίνακας cart

  Ο πίνακας περιέχει τα προϊόντα του καλαθιού του χρήστη (Πίνακας 4-10). Πρωτεύον κλειδί είναι ο συνδυασμός των ξένων κλειδιών (user_id, product_id). Μια γραμμή του πίνακα αντιπροσωπεύει ένα προϊόν του καλαθιού του χρήστη. Η συσχέτιση του πίνακα cart με τους πίνακες users και products είναι 1-1.

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
user_id (integer)	Ξένο κλειδί - Δείχνει το id του χρήστη
product_id (integer)	Ξένο κλειδί - Δείχνει το id του προϊόντος
quantity (integer)	Ποσότητα του προϊόντος
expiration (timestamp with time zone)	Ορίζει την ώρα και ημερομηνία που λήγει το καλάθι

Πίνακας 4-10 Πίνακας cart

- Πίνακας shipping_info

  Ο πίνακας περιέχει τις πληροφορίες αποστολής της παραγγελίας (Πίνακας 4-11).

Όνομα / Τύπος πεδίου	Λειτουργία πεδίου
id (serial)	Κύριο κλειδί
first_name (varchar (255))	Όνομα του χρήστη
last_name (varchar (255))	Επίθετο του χρήστη
street (varchar (255))	Διεύθυνση του χρήστη
post_code (varchar (255))	Ταχυδρομικός κώδικας του χρήστη

city (varchar (255))	Πόλη διαμονής του χρήστη
phone (varchar (255))	Τηλέφωνο του χρήστη
method (varchar (255))	Μέθοδος αποστολής

Πίνακας 4-11 Πίνακας shipping_info

- Πίνακας billing_info

  Ο πίνακας περιέχει τις πληροφορίες χρέωσης της παραγγελίας (Πίνακας 4-12).

Όνομα / Τύπος πεδίου	**Λειτουργία πεδίου**
id (serial)	Κύριο κλειδί
first_name (varchar (255))	Όνομα του χρήστη
last_name (varchar (255))	Επίθετο του χρήστη
street (varchar (255))	Διεύθυνση του χρήστη
post_code (varchar (255))	Ταχυδρομικός κώδικας του χρήστη
city (varchar (255))	Πόλη διαμονής του χρήστη
phone (varchar (255))	Τηλέφωνο του χρήστη
method (varchar (255))	Μέθοδος χρέωσης

Πίνακας 4-12 Πίνακας billing_info

- Πίνακας orders

  Ο πίνακας περιέχει τις παραγγελίες των χρηστών (Πίνακας 4-13). Η συσχέτιση του πίνακα orders με τους πίνακες users, billing_info, shipping_info είναι 1-1.

Όνομα / Τύπος πεδίου	**Λειτουργία πεδίου**
id (serial)	Κύριο κλειδί
user_id (integer)	Ξένο κλειδί - Δείχνει το id του χρήστη
email (character varying (255))	Email χρήστη
status (character varying (255))	Κατάσταση παραγγελίας
order_date (timestamp with time zone)	Ημερομηνία δημιουργίας παραγγελίας
payment_date (timestamp with time zone)	Ημερομηνία πληρωμής

shipping_date (timestamp with time zone)	Ημερομηνία αποστολής
shipping_info_id (integer)	Ξένο κλειδί - Δείχνει το id των πληροφοριών αποστολής
billing_id (integer)	Ξένο κλειδί - Δείχνει το id των πληροφοριών χρέωσης

Πίνακας 4-13 Πίνακας orders

- Πίνακας order_products

  Ο πίνακας περιέχει τα προϊόντα της παραγγελίας (Πίνακας 4-14). Πρωτεύον κλειδί είναι ο συνδυασμός των ξένων κλειδιών (order_id, product_id). Μια γραμμή του πίνακα αντιπροσωπεύει ένα προϊόν της παραγγελίας του χρήστη. Η συσχέτιση του πίνακα cart με τους πίνακες orders και products είναι 1-1.

Όνομα / Τύπος πεδίου	**Λειτουργία πεδίου**
order_id (integer)	Ξένο κλειδί - Δείχνει το id της παραγγελίας
product_id (integer)	Ξένο κλειδί - Δείχνει το id του προϊόντος
quantity (integer)	Ποσότητα του προϊόντος

Πίνακας 4-14 Πίνακας order_products

- Πίνακας spring_session

  Ο πίνακας περιέχει τις συνεδρίες των χρηστών (Πίνακας 4-15).

Όνομα / Τύπος πεδίου	**Λειτουργία πεδίου**
primary_id (character (36))	Κύριο κλειδί
session_id (character (36))	Id της συνεδρίας
creation_time (bigint)	Ώρα δημιουργίας συνεδρίας
last_access_time (bigint)	Ώρα τελευταίας σύνδεσης
max_inactive_interval (integer)	Μέγιστο χρονικό διάστημα, στο οποίο ο χρήστης είναι ανενεργός

expiry_time (bigint)	Ώρα λήξης συνεδρίας
principal_name (character varying (100))	Όνομα χρήστη

Πίνακας 4-15 Πίνακας spring_session

5 Ανάλυση απαιτήσεων και υλοποίηση εφαρμογής

Για την εκμάθηση του Spring framework, δημιουργήθηκε ένα ηλεκτρονικό κατάστημα, στο οποίο ο χρήστης μπορεί να δει τα προϊόντα του φαρμακείου, να αναζητήσει προϊόντα βάσει κάποιου όρου αναζήτησης και να πραγματοποιήσει εικονικές παραγγελίες.

5.1 Ανάλυση απαιτήσεων

Οι λειτουργικές απαιτήσεις της εφαρμογής ορίζονται ως εξής :

- Δημιουργία λογαριασμού στο κατάστημα
- Είσοδος στο κατάστημα
- Αποσύνδεση
- Εμφάνιση προϊόντων κατηγορίας
- Αναζήτηση προϊόντων
- Εμφάνιση πληροφοριών προϊόντος
- Εμφάνιση καλαθιού
- Πραγματοποίηση παραγγελίας

5.2 Περιπτώσεις χρήσης

Παρακάτω, παρουσιάζονται οι περιπτώσεις χρήσης της εφαρμογής υπό μορφή λεκτικής περιγραφής, δίνοντας την κανονική και εναλλακτική ροή κάθε χρήσης.

Δημιουργία λογαριασμού στο κατάστημα

Κανονική ροή:

1. Ο χρήστης βρίσκεται στην οθόνη εγγραφής, συμπληρώνει τα στοιχεία του και πατάει το κουμπί «Εγγραφή».
2. Μεταφέρεται στην οθόνη εισόδου, όπου εμφανίζεται μήνυμα επιτυχίας.

Εναλλακτική ροή:

2.1 Ο χρήστης είτε δεν έχει συμπληρώσει κάποιο πεδίο, είτε έχει εισάγει email, που χρησιμοποιείται ήδη από άλλο χρήστη.

2.2 Η εφαρμογή εμφανίζει στην οθόνη εγγραφής μήνυμα σφάλματος.

2.3 Ο χρήστης συνεχίζει από το βήμα 1 της κανονικής ροής.

Είσοδος στο κατάστημα

Κανονική ροή:

1. Ο χρήστης βρίσκεται στην οθόνη εισόδου, συμπληρώνει τα στοιχεία του και πατάει το κουμπί «Είσοδος».
2. Μεταφέρεται στην αρχική οθόνη, όπου εμφανίζεται το όνομα του.

Εναλλακτική ροή:

2.1 Ο χρήστης είτε δεν έχει συμπληρώσει κάποιο πεδίο, είτε έχει εισάγει email, που χρησιμοποιείται ήδη από άλλο χρήστη, είτε ο κωδικός είναι λάθος.

2.2 Η εφαρμογή εμφανίζει στην οθόνη εγγραφής μήνυμα σφάλματος.

2.3 Ο χρήστης συνεχίζει από το βήμα 1 της κανονικής ροής.

Αποσύνδεση

Κανονική ροή:

1. Ο χρήστης πατάει το κουμπί «Αποσύνδεση», στο πάνω μέρος οποιασδήποτε οθόνης.
2. Εάν δεν είναι ήδη στην αρχική οθόνη, μεταφέρεται σε αυτή.

Εμφάνιση προϊόντων κατηγορίας

Κανονική ροή:

1. Σε οποιαδήποτε οθόνη, ο χρήστης πατάει στην κατηγορία, την οποία θέλει.
2. Εμφανίζονται τα προϊόντα της κατηγορίας.

Αναζήτηση προϊόντων

Κανονική ροή:

1. Σε οποιαδήποτε οθόνη, ο χρήστης πληκτρολογεί τον όρο στο πεδίο αναζήτησης και πατάει στο κουμπί με το σύμβολο της αναζήτησης.

2. Μεταφέρεται στην οθόνη αναζήτησης, όπου εμφανίζονται τα σχετικά προϊόντα.

Εναλλακτική ροή:

2.1 Εμφανίζεται το μήνυμα «Δεν βρέθηκαν προϊόντα με αυτά τα κριτήρια».

Εμφάνιση πληροφοριών προϊόντος

Κανονική ροή:

1. Ο χρήστης εισέρχεται στην αρχική οθόνη και πατάει πάνω στην εικόνα του προϊόντος ή στο όνομα.
2. Μεταφέρεται στην οθόνη του προϊόντος, όπου εμφανίζονται οι πληροφορίες του.

Εμφάνιση καλαθιού

Κανονική ροή:

1. Σε οποιαδήποτε οθόνη, ο χρήστης πατάει στο εικονίδιο του καλαθιού στο πάνω μέρος της οθόνης.
2. Μεταφέρεται στην οθόνη του καλαθιού, όπου εμφανίζονται τα προϊόντα του καλαθιού.

Πραγματοποίηση παραγγελίας

Κανονική ροή:

1. Ο χρήστης βρίσκεται στην οθόνη καλαθιού και πατάει το κουμπί «Ολοκλήρωση αγοράς».
2. Μεταφέρεται στην οθόνη ολοκλήρωσης αγοράς, όπου συμπληρώνει τα στοιχεία της παραγγελίας.
3. Μεταφέρεται στην οθόνη παραγγελίας, όπου εμφανίζονται τα στοιχεία της παραγγελίας του.

Εναλλακτική ροή:

2.1 Ο χρήστης δεν έχει συμπληρώσει κάποιο πεδίο ή δεν έχει τη σωστή μορφή.

2.2 Η εφαρμογή εμφανίζει στην οθόνη εγγραφής μήνυμα σφάλματος.

2.3 Ο χρήστης συνεχίζει από το βήμα 2 της κανονικής ροής.

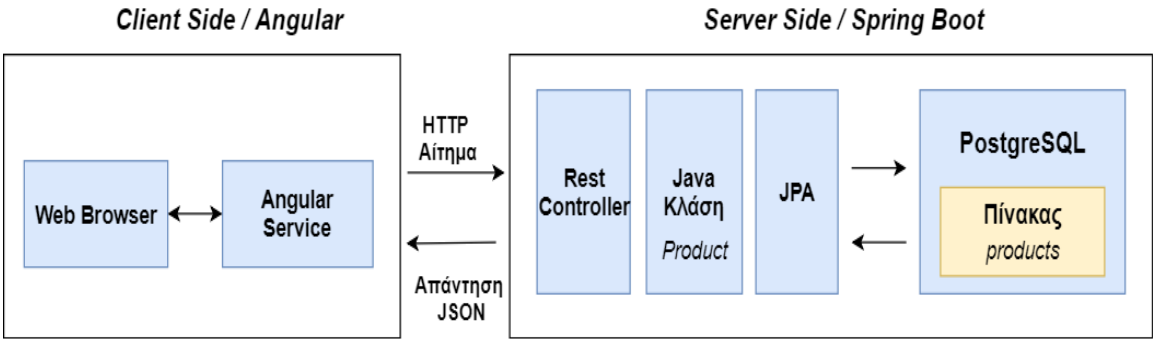
5.3 Αρχιτεκτονική εφαρμογής

Παρακάτω, αναλύεται η αρχιτεκτονική της εφαρμογής.

5.3.1 Αρχιτεκτονική πελάτη - εξυπηρετητή

Η αρχιτεκτονική πελάτη - εξυπηρετητή (client – server) χωρίζει την εφαρμογή σε δύο κομμάτια, τον πελάτη και τον εξυπηρετητή. Μια τέτοια εφαρμογή δημιουργείται σε ένα δίκτυο υπολογιστών, το οποίο συνδέει τον πελάτη με τον εξυπηρετητή. Ο εξυπηρετητής παρέχει την κεντρική λειτουργικότητα: πολλοί πελάτες μπορούν να συνδεθούν ταυτόχρονα με τον εξυπηρετητή και να του ζητήσουν να πραγματοποιήσει μια λειτουργία. Ο εξυπηρετητής παίρνει τα αιτήματα του πελάτη, πραγματοποιεί τη λειτουργία και επιστρέφει τα αποτελέσματα στον πελάτη [37].

Η Spring Boot εφαρμογή, η οποία τρέχει στον απομακρυσμένο server στο Heroku αποτελεί το κομμάτι του εξυπηρετητή (Εικόνα 5-1). Δημιουργούμε ένα RESTful API, το οποίο παρέχει τα δεδομένα της βάσης δεδομένων στον πελάτη. Η Angular εφαρμογή, η οποία τρέχει στον περιηγητή δεδομένων του χρήστη αποτελεί το κομμάτι του πελάτη. Ο κώδικας φορτώνεται στην αρχή και ύστερα ο πελάτης πραγματοποιεί HTTP αιτήματα στον εξυπηρετητή, καταναλώνοντας το API, για να πάρει τα δεδομένα και να ανανεώσει την σελίδα.

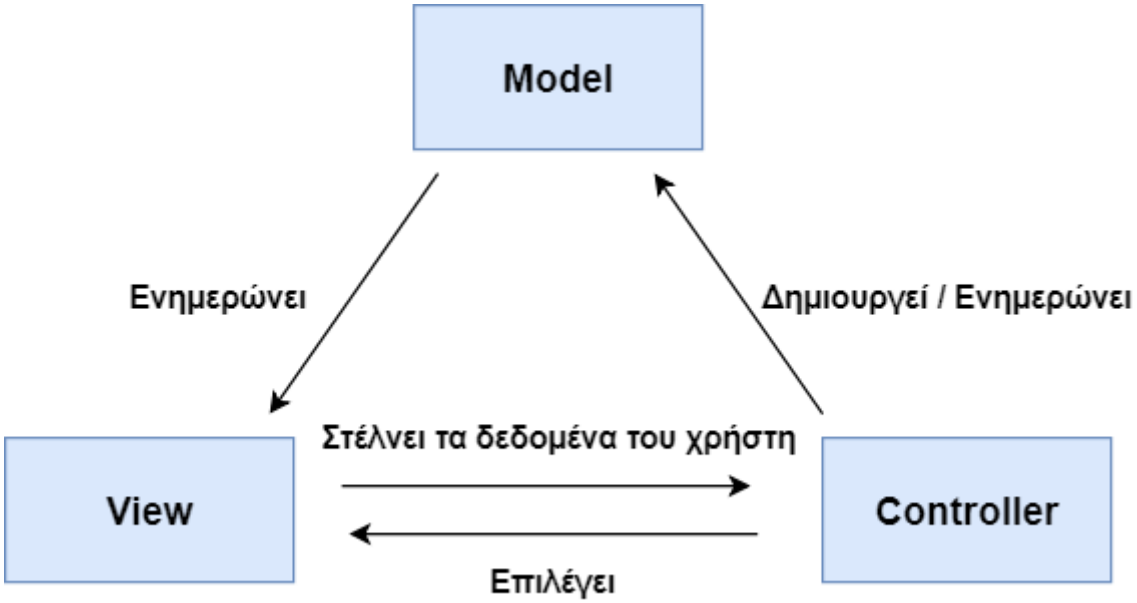


Εικόνα 5-1 Αρχιτεκτονική εφαρμογής [38]

5.3.2 Αρχιτεκτονική Spring Boot εφαρμογής

Στο κομμάτι του εξυπηρετητή, μπορούμε να διακρίνουμε τρία επίπεδα, όσον αφορά την αρχιτεκτονική [39].

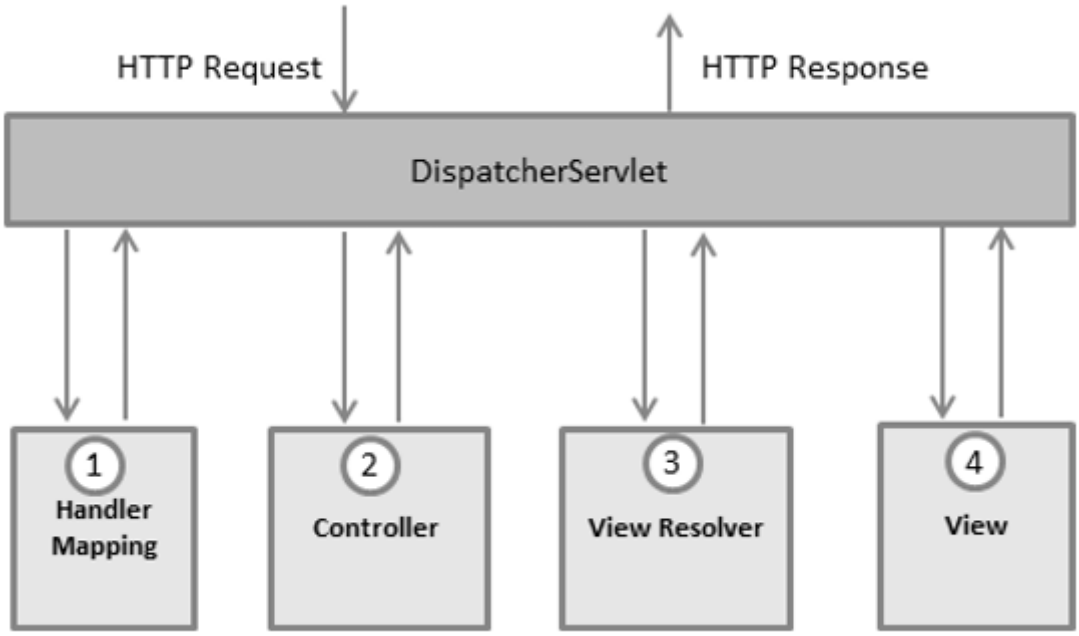
- **Επίπεδο δεδομένων (Data Tier)**: Περιλαμβάνει τους μηχανισμούς αποθήκευσης των δεδομένων. Σε αυτό το επίπεδο, βρίσκονται οι κλάσεις repositories, οι οποίες πραγματοποιούν τις διάφορες λειτουργίες στη βάση δεδομένων.
- **Επίπεδο επιχείρησης (Business Tier)**: Περιλαμβάνει τη λογική της εφαρμογής. Περιέχει τις κλάσεις που βρίσκονται στο πακέτο springeshop.service και καθορίζουν το πως θα συμπεριφερθεί η εφαρμογή.
- **Επίπεδο παρουσίασης (Presentation Tier)**: Σε αυτό το επίπεδο, εφαρμόζεται το μοντέλο αρχιτεκτονικής Model-View-Controller (MVC) (Εικόνα 5-2). Το MVC διαχωρίζει τα διάφορα κομμάτια της εφαρμογής (λογική εισόδων (input logic), επιχειρηματική λογική (business logic) και λογική διεπαφής χρήστη (UI logic)) και παρέχει χαμηλή σύζευξη μεταξύ αυτών των στοιχείων.



Εικόνα 5-2 Αρχιτεκτονική MVC

- Το Model ενθυλακώνει τα δεδομένα της εφαρμογής και γενικά αποτελείται από τα POJOs.
- Το View είναι υπεύθυνο για την εμφάνιση των δεδομένων του Model και γενικά το τελικό προϊόν που παράγει, είναι HTML, την οποία μπορεί, να μεταφράσει ο περιηγητής δεδομένων του πελάτη.
- Ο Controller είναι υπεύθυνος για την διαχείριση των αιτημάτων του χρήστη τη δημιουργία του κατάλληλου Model, το οποίο περνάει στο View, έτσι ώστε να το εμφανίσει.

Το Spring MVC, όπως και άλλα διαδικτυακά frameworks, είναι σχεδιασμένο γύρω από το πρότυπο σχεδίασης Front Controller, όπου ένα κεντρικό Servlet, το Dispatcher Servlet, παρέχει έναν μοιραζόμενο αλγόριθμο για την επεξεργασία των αιτημάτων, ενώ η πραγματική δουλειά ανατίθεται και πραγματοποιείται από ρυθμιζόμενα μέρη (components) [40].



Εικόνα 5-3 Dispatcher Sevlet [41]

Όταν γίνεται ένα HTTP αίτημα στο *DispatcherServlet*, πραγματοποιείται η εξής ακολουθία γεγονότων (Εικόνα 5-3):

1. Όταν λάβει ένα HTTP αίτημα, το *DispatcherServlet* συμβουλεύεται το *HandlerMapping*, για να καλέσει τον κατάλληλο *Controller*.
2. Ο *Controller* παίρνει το αίτημα και καλεί την κατάλληλη μέθοδο, που θα το διαχειριστεί, ανάλογα με την HTTP μέθοδο. Η μέθοδος του *Controller* θα δημιουργήσει τα δεδομένα του *Model*, σύμφωνα με την επιχειρηματική λογική και θα επιστρέψει το όνομα του *DispatcherServlet*View στο *DispatcherServlet*.
3. Το *DispatcherServlet* περνάει το λογικό όνομα του *View* στον *ViewResolver* και αυτός επιστρέφει την πραγματική υλοποίηση του *View*.
4. Το *DispatcherServlet* περνάει τα δεδομένα στο *View* και επιστρέφει την απάντηση στον περιηγητή δεδομένων του πελάτη.

5.4 RESTful API

Στην Spring Boot εφαρμογή δημιουργούμε ένα RESTful API, το οποίο καταναλώνει η Angular εφαρμογή, πραγματοποιώντας HTTP αιτήματα για να ανανεώσει τις σελίδες.

Το Representational State Transfer (REST) είναι ένα στυλ αρχιτεκτονικής λογισμικού, το οποίο ορίζει ένα σύνολο από περιορισμούς για τη δημιουργία υπηρεσιών Ιστού (web services). Παρουσιάστηκε και ορίστηκε από τον Roy Fielding το 2000. Οι υπηρεσίες Ιστού, οι οποίες συμμορφώνονται με αυτούς τους περιορισμούς, ονομάζονται RESTful υπηρεσίες Ιστού και παρέχουν διαλειτουργικότητα μεταξύ υπολογιστικών συστημάτων στο διαδίκτυο [42].

Οι RESTful υπηρεσίες Ιστού επιτρέπουν στα αιτούμενα συστήματα να έχουν πρόσβαση και να χειρίζονται κειμενικές αναπαραστάσεις των διαδικτυακών πόρων, χρησιμοποιώντας ένα αναγνωριστικό (uniform) και ένα προκαθορισμένο σύνολο από λειτουργίες χωρίς κατάσταση (stateless).

Το REST ορίζει τους εξής περιορισμούς:

- **Αρχιτεκτονική πελάτη - εξυπηρετητή**: Πρέπει, να υπάρχει διαχωρισμός των αρμοδιοτήτων πελάτη και εξυπηρετητή.
- **Έλλειψη κατάστασης (Statelessness)**: Ο εξυπηρετητής δεν θα πρέπει, να γνωρίζει και να αποθηκεύει την κατάσταση, στην οποία βρίσκεται ο πελάτης.

- **Δυνατότητα αποθήκευσης στην κρυφή μνήμη (Cacheability)**: Η απάντηση πρέπει να αναφέρει, αν τα δεδομένα μπορούν να αποθηκευτούν στην κρυφή μνήμη ή όχι (cachable).
- **Πολυεπίπεδο σύστημα (Layered system)**: Ο πελάτης δεν θα πρέπει, να γνωρίζει αν είναι συνδεδεμένος σε έναν τελικό εξυπηρετητή ή σε έναν ενδιάμεσο.
- **Κώδικας κατά παραγγελία (Code on demand)**: Ο εξυπηρετητής μπορεί να επεκτείνει την λειτουργικότητα του πελάτη, μεταφέροντας εκτελέσιμο κώδικα.
- **Ομοιόμορφη διεπαφή (Uniform interface)**: Οι αλληλεπιδράσεις μεταξύ πελάτη και εξυπηρετητή, βασίζονται στην ομοιομορφία των διεπαφών τους.

Για το REST, πολύ σημαντική έννοια είναι ο πόρος (resource). Ένας πόρος είναι οτιδήποτε μπορεί να ονομαστεί. Παραδείγματα από πόρους είναι χρήστες, προϊόντα, βίντεο και συνήθως σχετίζονται μεταξύ τους [43]. Κάθε πόρος έχει ένα URI ως αναγνωριστικό. Θα πρέπει να χρησιμοποιούμε ουσιαστικά για να αναπαριστάνουμε πόρους. Στον Πίνακα 5-1 δίνονται παραδείγματα από πόρους.

URI	Περιγραφή πόρου
https://api.example/products	Αναπαριστά μια συλλογή από προϊόντα – πόρους.
https://api.example/products/2	Αναπαριστά ένα προϊόν – πόρο με το αναγνωριστικό 2.

Πίνακας 5-1 Πόροι (resources)

Στην αρχιτεκτονική REST, ο πελάτης στέλνει ένα αίτημα για να πάρει ή να τροποποιήσει έναν πόρο και ο εξυπηρετητής στέλνει απαντήσεις σε αυτά τα αιτήματα. Το υποκείμενο πρωτόκολλο του REST είναι το HTTP, το οποίο είναι το βασικό πρωτόκολλο του διαδικτύου. Ένα αίτημα αποτελείται από [44]:

- ένα ρήμα HTTP (verb), το οποίο ορίζει τι λειτουργία θα πραγματοποιηθεί
- μια κεφαλίδα (header), η οποία επιτρέπει στον πελάτη να μεταφέρει πληροφορίες, σχετικά με το αίτημα
- την διαδρομή (path) του πόρου
- ένα προαιρετικό σώμα μηνύματος (message body) που περιέχει τα δεδομένα

Υπάρχουν 4 βασικά HTTP ρήματα, τα οποία χρησιμοποιούμε, για να αλληλεπιδράσουμε με τους πόρους (Πίνακας 5-2).

Ρήμα	**Λειτουργία**
GET	Ανακτούμε έναν πόρο ή μια συλλογή από πόρους.
POST	Δημιουργούμε ένα καινούριο πόρο.
PUT	Ενημερώνουμε ένα καινούριο πόρο.
DELETE	Διαγράφουμε ένα καινούριο πόρο.

Πίνακας 5-2 HTTP ρήματα

Οι RESTful πόροι είναι αφηρημένες οντότητες. Ο εξυπηρετητής πριν στείλει την απάντηση στον πελάτη, πρέπει να σειριοποιήσει (serialize) τα δεδομένα της απάντησης στην κατάλληλη αναπαράσταση. Οι πιο συνηθισμένες μορφές αναπαράστασης των πόρων είναι το JSON και η XML. Ένα προϊόν - πόρος μπορεί, να αναπαρασταθεί σε JSON ως εξής:

```
{
    "id" : 10,
    "name" : "iPhone-7",
    "price" : 1200
}
```

Ο ίδιος πόρος, μπορεί να αναπαρασταθεί σε XML μορφή ως εξής:

```
<product>
    <id> 10 </id>
    <name> iPhone-7 </name>
    <price> 1200 </price>
</product>
```

Τέλος, κάθε απάντηση του εξυπηρετητή περιέχει και έναν κωδικό κατάστασης (status code), ο οποίος ενημερώνει τον πελάτη, σχετικά με την επιτυχία της λειτουργίας. Οι πιο συνηθισμένοι κωδικοί περιγράφονται στον Πίνακα 5-3.

Κωδικοί κατάστασης	Περιγραφή
200 (OK)	Είναι η τυπική απάντηση για τα επιτυχημένα HTTP αιτήματα.
201 (CREATED)	Είναι η τυπική απάντηση, όταν ένα HTTP αίτημα είχε ως αποτέλεσμα τη δημιουργία ενός πόρου.
204 (NO CONTENT)	Είναι η τυπική απάντηση για τα επιτυχημένα HTTP αιτήματα, τα οποία έχουν κενό σώμα απάντησης (response body).
400 (BAD REQUEST)	Το αίτημα δεν μπορεί να επεξεργαστεί λόγω κακής σύνταξης του αιτήματος ή κάποιου σφάλματος του πελάτη.
403 (FORBIDDEN)	Ο πελάτης δεν έχει άδεια πρόσβασης σε αυτόν τον πόρο.
404 (NOT FOUND)	Ο πόρος δεν βρέθηκε. Μπορεί να διαγράφηκε ή να μην έχει δημιουργηθεί ακόμα.
500 (INTERNAL SERVER ERROR)	Είναι μια γενική απάντηση για κάποια αναπάντεχη αποτυχία του εξυπηρετητή, όταν δεν υπάρχουν πιο συγκεκριμένες πληροφορίες.

Πίνακας 5-3 Κωδικοί κατάστασης HTTP

5.5 Υλοποίηση εφαρμογής

Στην ενότητα αυτή, θα αναλυθεί ένα μικρό τμήμα του πηγαίου κώδικα της Spring Boot και Angular εφαρμογής.

Στο πακέτο springeshop.repositories της Spring Boot εφαρμογής, υπάρχουν κλάσεις – repositories με το annotation `@Repository`, οι οποίες πραγματοποιούν τις διάφορες λειτουργίες στη βάση δεδομένων (Πίνακας 5-4). Επεκτείνοντας τη διεπαφή `JpaRepository<T, ID>`, το Spring Data JPA μας παρέχει τις υλοποιήσεις των μεθόδων για την εισαγωγή, ανάκτηση, διαγραφή και ενημέρωση εγγραφών στον αντίστοιχο

πίνακα. Εάν θέλουμε, να πραγματοποιήσουμε ένα πιο σύνθετο ερώτημα στη βάση δεδομένων, χρησιμοποιούμε το annotation `@Query` στην υπογραφή της μεθόδου.

Διεπαφή	**Περιγραφή**
AuthorityRepository	Παρέχει CRUD λειτουργίες για τον πίνακα authorities
BillingInfoRepository	Παρέχει CRUD λειτουργίες για τον πίνακα billing_info
BrandRepository	Παρέχει CRUD λειτουργίες για τον πίνακα brands
CartRepository	Παρέχει CRUD λειτουργίες για τον πίνακα cart
CategoryRepository	Παρέχει CRUD λειτουργίες για τον πίνακα categories
DealImageReposiotry	Παρέχει CRUD λειτουργίες για τον πίνακα deal_images
DealRepository	Παρέχει CRUD λειτουργίες για τον πίνακα deals
InventoryRepository	Παρέχει CRUD λειτουργίες για τον πίνακα inventory
OrderProductRepository	Παρέχει CRUD λειτουργίες για τον πίνακα order_products
OrderRepository	Παρέχει CRUD λειτουργίες για τον πίνακα orders
ProductImageRepository	Παρέχει CRUD λειτουργίες για τον πίνακα product_images
ProductRepository	Παρέχει CRUD λειτουργίες για τον πίνακα products
ShippingInfoRepository	Παρέχει CRUD λειτουργίες για τον πίνακα shipping_info
UserRepository	Παρέχει CRUD λειτουργίες για τον πίνακα users

Πίνακας 5-4 Διεπαφές πακέτου springeshop.repositories

Στο πακέτο springeshop.service υπάρχουν κλάσεις με το annotation `@Service`, που περιέχουν τη λογική της εφαρμογής και τις διεπαφές, τις οποίες υλοποιούν (Πίνακας 5-5).

Διεπαφή, Κλάση	**Περιγραφή**
AmazonS3Clientservice, AmazonS3ClientServiceImpl	Ανεβάζουν τις φωτογραφίες στο AmazonS3
AuthorityService, AuthorityServiceImpl	Προσθέτουν τους ρόλους του χρήστη
BillingInfoService, BillingInfoServiceImpl	Εισάγουν τις πληροφορίες χρέωσης της παραγγελίας στη βάση
CartService, CartServiceImpl	Παρέχουν CRUD λειτουργίες για το καλάθι
CategoryService, CategoryServiceImpl	Παρέχουν μεθόδους, για την εισαγωγή και ανάκτηση κατηγοριών
DealImageService, DealImageServiceImpl	Παρέχουν μεθόδους, για την ανάκτηση των εικόνων των προσφορών
DealService, DealServiceImpl	Παρέχουν μεθόδους, για την ανάκτηση των προσφορών
EmailService, EmailServiceImpl	Στέλνουν email με τα στοιχεία της παραγγελίας στον χρήστη
InventoryService, InventoryServiceImpl	Εισάγουν και ανακτούν το απόθεμα του προϊόντος
OrderProductService, OrderProductServiceImpl	Εισάγουν τα προϊόντα της παραγγελίας στη βάση
OrderService, OrderServiceImpl	Εισάγουν την παραγγελία στη βάση
ProductImageService, ProductImageServiceImpl	Εισάγουν και ανακτούν τις εικόνες του προϊόντος
ProductService, ProductServiceImpl	Παρέχουν CRUD λειτουργίες για τον πίνακα products
SearchService, SearchServiceImpl	Πραγματοποιούν την αναζήτηση των προϊόντων
ShippingInfoService,	Εισάγουν τις πληροφορίες αποστολής της

ShippingInfoServiceImpl	παραγγελίας στη βάση
UserService, UserServiceImpl	Εισάγουν και ανακτούν τους χρήστες

Πίνακας 5-5 Διεπαφές και κλάσεις πακέτου springeshop.services

Στο πακέτο springeshop.controller υπάρχουν κλάσεις με το annotation `@Controller` και `@RestController`, οι οποίες δημιουργούν το RESTful API (Πίνακας 5-6).

Controller	**HTTP μέθοδος**	**Άκρο (Endpoint)**	**Περιγραφή**
AuthenticationController	*POST*	*/authentication/validateuser*	Ελέγχει εάν τα στοιχεία του χρήστη είναι σωστά
AuthenticationController	*GET*	*/authentication/session*	Δημιουργεί μια συνεδρία
AuthenticationController	*POST*	*/authentication/logout*	Αποσυνδέει τον χρήστη
AuthenticationController	*GET*	*/anonymous/session*	Δημιουργεί συνεδρία για τον ανώνυμο χρήστη
BrandApiController	*GET*	*/api/brands*	Επιστρέφει τις εταιρίες των προϊόντων της κατηγορίας
CartApiController	*POST*	*/api/carts*	Δημιουργεί το καλάθι του χρήστη
CartApiController	*GET*	*/api/carts/{userid}*	Επιστρέφει τα προϊόντα του καλαθιού
CartApiController	*GET*	*/api/carts/{userid}/count*	Επιστρέφει τον αριθμό των

			προϊόντων
CartApiController	*GET*	*/api/carts/{userid}/product/{productid}*	Επιστρέφει το προϊόν
CartApiController	*DELETE*	*/api/carts/{userid}/product/{productid}*	Διαγράφει το προϊόν
CartApiController	*PATCH*	*/api/carts/{userid}/product/{productid}*	Ενημερώνει τα στοιχεία του προϊόντος
CartApiController	*DELETE*	*/api/carts/{userid}*	Διαγράφει το καλάθι
CategoryApiController	*GET*	*/api/categories/{name}*	Επιστρέφει τα προϊόντα της κατηγορίας
CategoryApiController	*GET*	*/api/categories/{categoryname}/brands/{brandname}/products/count*	Επιστρέφει τον αριθμό των προϊόντων της εταιρίας στην κατηγορία
CategoryApiController	*GET*	*/api/categories/{name}/ranges/{rangeid}/products/count*	Επιστρέφει τον αριθμό των προϊόντων της κατηγορίας, σε ένα διάστημα τιμών
DealApiController	*GET*	*/api/deals*	Επιστρέφει τις προσφορές
OrderApiController	*POST*	*/api/orders*	Δημιουργεί μια παραγγελία
ProductApiController	*GET*	*/api/products*	Επιστρέφει τα προϊόντα
ProductApiController	*POST*	*/api/products*	Δημιουργεί ένα προϊόν
ProductApiController	*GET*	*/api/products/{name}*	Επιστρέφει το

			προϊόν
RegisterApiController	*POST*	*/api/register*	Πραγματοποιεί την εγγραφή του χρήστη
SearchApiController	*GET*	*/api/search*	Επιστρέφει τα προϊόντα της αναζήτησης
SearchApiController	*GET*	*/api/search/brands/{brand name}/products/count*	Επιστρέφει τον αριθμό των προϊόντων της εταιρίας στην αναζήτηση
SearchApiController	*GET*	*/api/search/ranges/{range id}/products/count*	Επιστρέφει τον αριθμό των προϊόντων της αναζήτησης, σε ένα διάστημα τιμών
SearchApiController	*GET*	*/api/search/brands*	Επιστρέφει τις εταιρίες των προϊόντων της αναζήτησης

Πίνακας 5-6 Κλάσεις του πακέτου springeshop.controller και το RESTful API, το οποίο δημιουργούν

Στην Angular, component είναι μια κλάση με το decorator `@Component` και ελέγχει ένα κομμάτι της οθόνης, το οποίο λέγεται view. Η κλάση περιέχει τα δεδομένα, την λογική της εφαρμογής και είναι συσχετισμένη με ένα HTML template.

Κάθε Angular εφαρμογή έχει τουλάχιστον ένα component, το οποίο λέγεται component «ρίζα» (root component). Αυτό μπορεί, να περιέχει και άλλα components, τα οποία λέγονται components «παιδιά» (child components). Κάθε component περιέχει τέσσερα αρχεία στο φάκελο του:

- **example.component.css**: είναι το αρχείο css του component και αφορά μόνο το συγκεκριμένο
- **example.component.html**: είναι το template του component, δηλαδή τι HTML πρέπει, να εμφανίσει η Angular όταν χρησιμοποιούμε αυτό το component
- **example.component.spec.ts**: μας επιτρέπει να ελέγξουμε το component
- **example.component.ts**: περιέχει τα δεδομένα και τη συμπεριφορά του component

Παρακάτω, δίνονται τα components της εφαρμογής, καθώς και οι οθόνες στις οποίες αντιστοιχούν.

Component	**Οθόνη / Λειτουργία**
index-page	Αρχική οθόνη
cart-page	Οθόνη καλαθιού
category-page	Οθόνη κατηγορίας προϊόντων
category-sidebar	Παρέχει τα φίλτρα κατηγορίας προϊόντων
checkout-page	Οθόνη ολοκλήρωσης παραγγελίας
discount-carousel	Εμφανίζει τις προσφορές
login-page	Οθόνη εισόδου
my-footer	Εμφανίζει πληροφορίες του καταστήματος
navigation-bar	Πλοήγηση στο κατάστημα
order	Οθόνη εμφάνισης παραγγελίας
product	Εμφανίζει την εικόνα και το όνομα του προϊόντος
product-page	Οθόνη προϊόντος
products-carousel	Εμφανίζει τα αγαπημένα ή καινούρια προϊόντα
register-page	Οθόνη εγγραφής
search	Οθόνη αναζήτησης προϊόντων
search-sidebar	Παρέχει τα φίλτρα αναζήτησης προϊόντων
shop-services	Εμφανίζει τις υπηρεσίες του καταστήματος

Πίνακας 5-7 Angular components – οθόνες

5.5.1 Αρχική Οθόνη

Δημιουργούμε την κλάση – entity Product, η οποία θα αντιστοιχιστεί με τον πίνακα products της βάσης δεδομένων από το JPA. Η αντιστοίχιση πραγματοποιείται με annotations, τα σημαντικότερα από τα οποία δίνονται παρακάτω:

- `@Entity`: Δηλώνει ότι είναι η κλάση είναι ένα entity
- `@Table`: Ορίζει τον πίνακα της βάσης δεδομένων, με τον οποίο θα αντιστοιχιστεί το entity.
- `@Column(name = "columnName")`: Ορίζει την στήλη του πίνακα, με την οποία θα αντιστοιχίστει το πεδίο του entity.
- `@Transient`: Ορίζει ότι αυτό το πεδίο δεν θα αποθηκευτεί στον πίνακα της βάσης.
- `@Id`: Ορίζει το πρωτεύον κλειδί του πίνακα.
- `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`: Δηλώνει τη συσχέτιση των entities

```
package springeshop.model;

@Entity
@Table(name = "products")
public class Product {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "id")
	private int id;

	@ManyToOne(optional = false)
	@JoinColumn(name = "brand_id")
	@NotNull
	private Brand brand;

	@ManyToOne(optional = false)
	@JoinColumn(name = "category_id")
	@NotNull
	private Category category;

	@Column(name = "name")
	@NotNull(message = "Please provide product name")
	@Size(min = 3, max = 200, message = "Name must be between 10 and 200
characters")
	private String name;

	@Column(name = "price", nullable = false)
	@NotNull(message = "Please provide product price")
	@Min(value = 0, message = "Price must be greater than zero")
	private double price;

	@Transient
	@NotNull(message = "Please provide product quantity")
```

```
    @Min(value = 0, message = "Quantity must be greater than zero")
    private int quantity;
    @Column(name = "description")
    private String description;

    @Column(name = "is_favorite")
    @NotNull
    private boolean is_favorite;
    @Column(name = "is_new")
    @NotNull
    private boolean is_new;

    @Transient
    private String smallImageUrl;

    @Transient
    private String largeImageUrl;

    @Transient
    private String verySmallImageUrl;

    public Product(){}

    // Getter -Setters ..
```

Αντίστοιχα, δημιουργούμε και τα άλλα entities της εφαρμογής. Μετά, δημιουργούμε το repository ProductRepository στο πακέτο springeshop.repositories. Η γλώσσα Java Persistence Query Language (JPQL) του JPA, μας επιτρέπει να ορίσουμε ερωτήματα πάνω στα entities.

```
package springeshop.repositories;

@Repository
public interface ProductRepository extends JpaRepository<Product, Integer>{

   @Query("select prod from Product prod where prod.is_favorite = true")
   Page<Product> findFavoriteProducts(Pageable pageable);

   @Query("select prod from Product prod where prod.is_new = true")
   Page<Product> findNewProducts(Pageable pageable);

   …
}
```

Η μέθοδος `findFavoriteProducts(Pageable pageable)` μας επιστρέφει τα αγαπημένα προϊόντα ανά τετράδες. Το JQPL ερώτημα `select prod from Product prod where prod.is_favorite = true`, αντιστοιχεί στο SQL ερώτημα `select * from products where is_favorite = true`.

Δημιουργούμε τα ProductService και ProductServiceImpl στο πακέτο springeshop.service. Τα annotations που χρησιμοποιούνται παρακάτω, έχουν τις εξής λειτουργίες :

- `@Service`: Δηλώνει ότι η κλάση είναι ένα service component, ώστε το αντικείμενο-bean να ανιχνευθεί από τον μηχανισμό αυτόματης ανίχνευσης του Spring Boot
- `@Transactional`: Δηλώνει ότι οι μέθοδοι θα εκτελεστούν στο πλαίσιο μιας συναλλαγής βάσης δεδομένων
- `@Autowired`: Δηλώνουμε ότι θέλουμε το Spring, να μας παρέχει ένα στιγμιότυπο του ProductRepository με dependency injection

```
package springeshop.service;

public interface ProductService{

        Page<Product> findFavoriteProducts(Pageable pageable);
        Page<Product> findNewProducts(Pageable pageable);

        …
}
```

```
package springeshop.service;

@Service("productService")
@Transactional
public class ProductServiceImpl implements ProductService{
        @Autowired
        private ProductRepository productRepository;

        @Override
        public Page<Product> findFavoriteProducts(Pageable pageable) {
           return productRepository.findFavoriteProducts(pageable);
        }

        @Override
        public Page<Product> findNewProducts(Pageable pageable) {
           return productRepository.findNewProducts(pageable);
        }
}
```

Αντίστοιχα, δημιουργούμε και τις κλάσεις / διεπαφές ProductImageRepository, ProductImageService, ProductImageServiceImpl, InventoryRepository, InventoryService, InventoryServiceImpl, για να ανακτήσουμε τις διευθύνσεις των εικόνων και του αποθέματος κάθε προϊόντος.

Δημιουργούμε τον ProductApiController στο πακέτο springeshop.controller. Ο πελάτης (Angular εφαρμογή) θα πραγματοποιεί ένα HTTP GET αίτημα στο *http://localhost:8080/api/products*, μαζί με δύο παραμέτρους filter και page. Για παράδειγμα, για να πάρει την πρώτη τετράδα των αγαπημένων προϊόντων, το αίτημα θα

πραγματοποιείται στο *http://localhost:8080/api/products?filter=favorite&page=0*. Τα annotations που χρησιμοποιούνται παρακάτω, έχουν τις εξής λειτουργίες :

- `@RestController`: Είναι ένα σύνθετο annotation, από τα `@Controller` και `@ResponseBody` και δηλώνει έναν controller, του οποίου κάθε μέθοδος κληρονομεί το annotation `@ResponseBody`, οπότε γράφει απευθείας στο σώμα της απάντησης και δεν επιστρέφει ένα HTML template.
- `@RequestMapping(value = "/products", method = RequestMethod.GET)`: Αντιστοιχίζει HTTP αιτήματα με μεθόδους του controller. Δηλαδή όταν πραγματοποιείται το HTTP GET αίτημα στο *http://localhost:8080/api/products*, θέλουμε να εκτελείται η μέθοδος getProductsByFilter().
- `@RequestParam(value = "filter") String filter`: Αντιστοιχίζει μια παράμετρο της μεθόδου με την παράμετρο του αιτήματος. Δηλαδή όταν πραγματοποιείται το HTTP GET αίτημα στο *http://localhost:8080/api/products?filter=favorite*, η παράμετρος String filter της getProductsByFilter() παίρνει την τιμή "favorite".

```
package springeshop.service;

@RestController
@RequestMapping("/api")
public class ProductApiController {

   @Autowired
   private ProductService productService;

   @Autowired
   private InventoryService inventoryService;

   @Autowired
   private ProductImageService productImageService;

   @RequestMapping(value = "/products", method = RequestMethod.GET)
   public ResponseEntity<?> getProductsByFilter(@RequestParam(value = "filter",
required = false) String filter, @RequestParam(value = "page", required =
false) int page){

    if(filter.equals("favorite")){
      Page<Product> favoriteProducts =
productService.findFavoriteProducts(PageRequest.of(page, 4));

      if(favoriteProducts.getTotalElements() == 0){
         return new ResponseEntity<>(HttpStatus.NO_CONTENT);
      }

      addImagesAndQuantityToProducts(favoriteProducts);
       return new ResponseEntity<Page<Product>>(favoriteProducts,
HttpStatus.OK);
   }else if(filter.equals("new")){
```

```
        Page<Product> newProducts =
productService.findNewProducts(PageRequest.of(page, 4));

        if(newProducts.getTotalElements() == 0){
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }

        addImagesAndQuantityToProducts(newProducts);
        return new ResponseEntity<Page<Product>>(newProducts, HttpStatus.OK);
    }else
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
   }
}
```

Το `ResponseEntity` μας επιτρέπει να επιστρέψουμε εκτός από το σώμα της απάντησης, και έναν κωδικό κατάστασης HTTP. Εάν το filter έχει τιμή "new", βρίσκουμε τα αγαπημένα προϊόντα με τη μέθοδο του `ProductService findFavoriteProducts(PageRequest.of(page, 4))` και τα επιστρέφουμε με την εντολή `return new ResponseEntity<Page<Product>>(favoriteProducts, HttpStatus.OK)` με κωδικό κατάστασης HTTP 200 OK. Η σειριοποίηση των αντικειμένων- entities σε μορφή JSON, πραγματοποιείται εσωτερικά από το Spring με τη βιβλιοθήκη Jackson. Εάν δεν υπάρχουν αγαπημένα προϊόντα, επιστρέφουμε κωδικό κατάστασης HTTP 204 NO CONTENT. Αντίστοιχα λειτουργούμε και για τα καινούρια προϊόντα.

Η διαδικασία για τη δημιουργία του άκρου *http://localhost:8080/api/deals*, το οποίο θα επιστρέφει τις προσφορές του καταστήματος είναι παρόμοια. Δημιουργούμε τα entities Deal, DealImage, τις κλάσεις / διεπαφές DealRepository, DealService, DealServiceImpl, DealImageRepository, DealImageService, DealImageServiceImpl και τον BrandApiController, ο οποίος θα περιέχει μια μέθοδο που θα χειρίζεται το endpoint.

Όσον αφορά την Angular, η αρχική οθόνη (index-page component), αποτελείται από πέντε components: το navigation-bar, discount-carousel, products-carousel (χρησιμοποιείται δύο φορές), shop-services και το my-footer (Κώδικας 5-1).

```
1  <navigation-bar></navigation-bar>
2  <discount-carousel></discount-carousel>
3  <products-carousel [carouselType] = "popularCarouselType" [carouselTitle] = "popularProductCarouselTitle" [carous
4  <products-carousel [carouselType] = "newCarouselType" [carouselTitle] = "newProductCarouselTitle" [carouselHtmlId
5  <shop-services></shop-services>
6  <my-footer></my-footer>
```

Κώδικας 5-1 index-page.component.html

Ενδεικτικά, δίνεται ο κώδικας HTML (Κώδικας 5-2) του shop-services, που περιέχει, τι θέλουμε να εμφανίζεται στην οθόνη, όταν χρησιμοποιούμε το template shop-services.

```
<!-- Services -->
<section  class="services">
  <div class="container">
      <div class="row">
          <div class="col-md-4">
              <h3><i class="fas fa-truck fa-lg mr-3"></i>ΓΡΗΓΟΡΗ ΚΑΙ ΔΩΡΕΑΝ ΔΙΑΝΟΜΗ</h3>
              <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
                  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
          </div>
          <div class="col-md-4">
              <h3><i class="fas fa-lock fa-lg mr-3"></i> ΑΣΦΑΛΕΙΣ ΠΛΗΡΩΜΕΣ</h3>
              <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
                  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
          </div>
          <div class="col-md-4">
              <h3><i class="far fa-credit-card fa-lg mr-3"></i>ΕΓΓΥΗΣΗ ΕΠΙΣΤΡΟΦΗΣ ΧΡΗΜΑΤΩΝ</h3>
              <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit,
                  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
          </div>
      </div>
  </div>
</section>
```

Κώδικας 5-2 shop-services.component.html

Για να πραγματοποιήσουμε HTTP αιτήματα στην Angular, χρησιμοποιούμε τα Services (κλάσεις με το decorator `@Injectable`) και τη βιβλιοθήκη HttpClient. Στον Κώδικα 5-3 δίνονται οι μέθοδοι του ProductService, οι οποίες πραγματοποιούν τα HTTP αιτήματα.

```
@Injectable({
  providedIn: 'root'
})
export class ProductService {

  private productListsApi = '/api/products';
  private favoriteParam = '?filter=favorite';
  private newParam = '?filter=new';
  private pageParam = '&page=';

  constructor(private http : HttpClient) { }

  getFavoriteProducts(pages : number[]) : Observable<ProductPage>{
    return from(pages).pipe(mergeMap(page => <Observable<ProductPage>>
      this.http.get<ProductPage>(this.productListsApi + this.favoriteParam + this.pageParam + page)));
  }

  getNewProducts(pages : number[]) : Observable<ProductPage>{
    return from(pages).pipe(mergeMap(page => <Observable<ProductPage>>
      this.http.get<ProductPage>(this.productListsApi + this.newParam + this.pageParam + page)));
  }

  getSingleProduct(name : string) : Observable<Product>{
    return this.http.get<Product>(this.productListsApi + '/' + name);
  }
```

Κώδικας 5-3 product.service.ts

Στη μέθοδο ngOnInit του products-carousel.component.ts, καλείται μία από τις παρακάτω μεθόδους, ανάλογα με την τιμή, που περνάμε στη μεταβλητή carouselType, για να πάρουμε είτε τα καινούρια, είτε τα αγαπημένα προϊόντα (Κώδικας 5-4).

```
  addFavoriteProductsToCarousel() : void{
    this.httpSubscription = this.productService.getFavoriteProductsInformation().subscribe(productPage => {
      this.productPagesInformation = productPage;
      this.initializePageNumber(this.productPagesInformation.totalPages);
      this.httpSubscription2 = this.productService.getFavoriteProducts(this.pageNumber)
                                .subscribe(productPage => this.productPages.push(productPage));
    });
  }

  addNewProductsToCarousel() : void{
    this.httpSubscription = this.productService.getNewProductsInformation().subscribe(productPage => {
      this.productPagesInformation = productPage;
      this.initializePageNumber(this.productPagesInformation.totalPages);
      this.httpSubscription2 = this.productService.getNewProducts(this.pageNumber)
                                .subscribe(productPage => this.productPages.push(productPage));
    });
  }
```

Κώδικας 5-4 Μέθοδοι addFavoriteProductsToCarousel() και addNewProductsToCarousel() του products-carousel.component.ts

5.5.2 Οθόνη προϊόντος

Στην οθόνη του προϊόντος, όταν ο πελάτης θέλει ένα προϊόν, κάνει ένα HTTP GET αίτημα στο URL *http://localhost:8080/api/products/{name}* και το Spring Boot του

επιστρέφει το προϊόν σε μορφή JSON. Προσθέτουμε τη μέθοδο που θα χειριστεί το αίτημα, στον ProductApiController.

```
@RequestMapping(value = "/products/{name}", method = RequestMethod.GET)
public ResponseEntity<?> getProductByName(@PathVariable("name") String name){

    Product product = productService.findByName(name);

    if(product == null){
        return new ResponseEntity<>(new ErrorMessage("Product with name " + name
+ " not found"),HttpStatus.NOT_FOUND);
    }

    ProductImage productImage =
productImageService.findByProductId(product.getId());
    product.setSmallImageUrl(productImage.getSmallImageurl());
    product.setLargeImageUrl(productImage.getLargeImageurl());
    product.setVerySmallImageUrl(productImage.getVerySmallImageurl());
    int productQuantity = inventoryService.findProductQuantity(product.getId());
    product.setQuantity(productQuantity);
    return new ResponseEntity<Product>(product, HttpStatus.OK);
}
```

Βρίσκουμε το προϊόν, προσθέτουμε τις διευθύνσεις των εικόνων και το απόθεμα του και το επιστρέφουμε με κωδικό κατάστασης 200 OK. Αν δεν υπάρχει, επιστρέφουμε 404 NOT FOUND.

Στη μέθοδο ngOnInit του products-page.component.ts, καλούμε τη μέθοδο getSingleProduct() του ProductService, για να πάρουμε το προϊόν (Κώδικας 5-5).

```
49  ngOnInit() {
50    this.isUserLoggedIn = this.authenticationService.isAuthenticated;
51    this.routeSubscription = this.route.params.subscribe(params => this.productNameParam = params['name']);
52    this.httpSubscription = this.productService.getSingleProduct(this.productNameParam)
53                                               .subscribe(product => {
54                                                 this.product = product;
55                                                 this.productCategory = product.category.name === 'Fish-Oils' ? thi
56                                                 this.categoryRoute = product.category.name === 'Fish-Oils' ? this.
57                                                 this.isProductAvailable = product.quantity > 0;
58                                                 this.productAvailability =  this.isProductAvailable ? this.availab
59                                               });
60
61  }
```

Κώδικας 5-5 Πραγματοποίηση HTTP αιτήματος, για να πάρουμε ένα προϊόν

5.5.3 Οθόνη κατηγορίας προϊόντων

Για την οθόνη της κατηγορίας, όταν ο πελάτης θέλει τα προϊόντα, κάνει ένα HTTP GET αίτημα στο URL *http://localhost:8080/api/categories/{name}?page=0&order=asc* και το Spring Boot

του επιστρέφει τα προϊόντα. Αν είναι επιλεγμένο κάποιο φίλτρο, τότε στέλνουμε τις τιμές του φίλτρου στις παραμέτρους. Για παράδειγμα, αν θέλουμε τα προϊόντα της κατηγορίας shampoos, που εταιρία τους είναι η Apivita και η τιμή τους βρίσκεται στα διαστήματα 0 - 9.99, 10 - 19.99 ευρώ, τα παίρνουμε από το URL *http://localhost:8080/api/categories/shampoos?page=0&order=asc&brand=Apivita&range=1&range=2* .

Επειδή δεν γνωρίζουμε ποια φίλτρα είναι επιλεγμένα εκ των προτέρων, αλλά μόνο κατά τη διάρκεια εκτέλεσης της εφαρμογής, θα χρησιμοποιήσουμε το JPA Criteria API. Για κάθε CriteriaQuery αντικείμενο, το entity ρίζα του ερωτήματος, από το οποίο ξεκινάμε να διασχίζουμε, λέγεται ερώτημα ρίζα (query root). Είναι αντίστοιχο με το FROM ενός JPQL ερωτήματος. Μπορούμε, να δημιουργήσουμε συνθήκες χρησιμοποιώντας τη μέθοδο CriteriaQuery.where, η οποία περιορίζει τα αποτελέσματα ενός ερωτήματος στο αντικείμενο CriteriaQuery. Η μέθοδος where αντιστοιχεί με το WHERE ενός JPQL ερωτήματος.

Στην παρακάτω μέθοδο, εάν κάποιες εταιρίες είναι επιλεγμένες στα φίλτρα, δημιουργούμε συνθήκες WHERE `prod.brand = :brand` για κάθε εταιρία. Εάν κάποιο διάστημα τιμών είναι επιλεγμένο, δημιουργούμε συνθήκες WHERE `prod.price between :range1 and :range2` για κάθε διάστημα τιμών. Επίσης, προσθέτουμε την συνθήκη `prod.category = :category` για την κατηγορία. Το τελικό ερώτημα, το οποίο πραγματοποιείται στη βάση είναι :

- Σε JPQL : `SELECT prod FROM Product prod WHERE prod.category = :category AND (prod.brand = :brand1 OR prod.brand = :brand2 …) AND ( prod.price BETWEEN :range1 and :range2 OR prod.price BETWEEN :range3 and :range4 …) ORDER BY prod.price ASC`
- Σε SQL : `SELECT * FROM products WHERE category_id = cid AND (brand_id = bid1 OR brand_id = bid2) AND ( price BETWEEN range1 and range2 OR price BETWEEN range3 and range …) ORDER BY price ASC`

Ο EntityManager εκτελεί το ερώτημα και επιστρέφει μια εξάδα προϊόντων.

```
@Override
public ProductPage findByCategoryIdWithBrandAndPriceRange(Category category,
List<Brand> brands , List<double[]> priceRanges, int page, String order) {
            ProductPage productPage = new ProductPage();
```

```
     CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
     CriteriaQuery<Product> criteriaQuery =
criteriaBuilder.createQuery(Product.class);
     Root<Product> productsRoot = criteriaQuery.from(Product.class);
     List<Predicate> brandPredicateList = new ArrayList<>();
     List<Predicate> priceRangePredicateList = new ArrayList<>();

     Predicate categoryPredicate =
criteriaBuilder.equal(productsRoot.get("category"), category);
     Order orderCriterion = order.equals("asc") ?
criteriaBuilder.asc(productsRoot.get("price")) :
criteriaBuilder.desc(productsRoot.get("price"));

     if(!brands.isEmpty()){
         for(int i=0; i < brands.size(); i++){
       brandPredicateList.add(criteriaBuilder.equal(productsRoot.get("brand"),
brands.get(i)));
         }
     }

     if(!priceRanges.isEmpty()){
         for(double[] range : priceRanges){
       priceRangePredicateList.add(criteriaBuilder.between(productsRoot.get("pri
ce"), range[0], range[1]));
             }
          }

      Predicate[] brandsPredicateArray = new
Predicate[brandPredicateList.size()];
      brandPredicateList.toArray(brandsPredicateArray);

      Predicate[] priceRangePredicateArray = new
Predicate[priceRangePredicateList.size()];
           priceRangePredicateList.toArray(priceRangePredicateArray);

      Predicate brandsPredicate = criteriaBuilder.or(brandsPredicateArray);
      Predicate priceRangePredicate =
criteriaBuilder.or(priceRangePredicateArray);
      Predicate brandsPriceRangeAndPredicate =
criteriaBuilder.and(brandsPredicate, priceRangePredicate);

      if(!brands.isEmpty() && !priceRanges.isEmpty()){
         criteriaQuery.where(criteriaBuilder.and(categoryPredicate,
brandsPriceRangeAndPredicate));
      }else{
          if(brands.isEmpty()){
              criteriaQuery.where(criteriaBuilder.and(categoryPredicate,
priceRangePredicate));
          }else{
              criteriaQuery.where(criteriaBuilder.and(categoryPredicate,
brandsPredicate));
          }
      }

      criteriaQuery.orderBy(orderCriterion);
      int totalProducts =
entityManager.createQuery(criteriaQuery).getResultList().size();
      int startProductPosition = page * 6;
      productPage.setTotalElements(totalProducts);
      productPage.setTotalPages(getProductPages(totalProducts));
      List<Product> wantedProducts =
entityManager.createQuery(criteriaQuery).setFirstResult(startProductPosition).s
etMaxResults(6).getResultList();
      productPage.setContent(wantedProducts);
      productPage.setNumber(page);
      productPage.setNumberOfElements(wantedProducts.size());
```

```
        return productPage;
}
```

Δημιουργούμε τον CategoryApiController και προσθέτουμε την παρακάτω μέθοδο getCategoryProducts. Εάν δεν είναι επιλεγμένα φίλτρο, οι παράμετροι brand και range είναι null. Εάν δεν υπάρχουν προϊόντα επιστρέφουμε κωδικό κατάστασης 204 NO CONTENT, αλλιώς επιστρέφουμε τα προϊόντα με κωδικό κατάστασης 200 OK.

```
@RequestMapping(value = "/categories/{name}", method = RequestMethod.GET)
public ResponseEntity<?> getCategoryProducts(@PathVariable("name") String name,
@RequestParam(value = "page", required = true) int page,
@RequestParam(value = "order", required = true) String order,
@RequestParam(value = "brand", required = false) String[] brands,
@RequestParam(value = "range", required = false) String[] ranges){

              Page<Product> products;
              ProductPage filteredProductPage = new ProductPage();
              Direction sortDirection = order.equals("asc") ? Sort.Direction.ASC
: Sort.Direction.DESC;

              Category category =
categoryService.findByName(getCorrectCategoryName(name));

              if(category == null){
                     return new ResponseEntity<>(new ErrorMessage("Category with
name " + name + " not found"),HttpStatus.NOT_FOUND);
              }

              if(brands == null && ranges == null){
                     products =
productService.findByCategoryId(category.getId(), PageRequest.of(page, 6,
sortDirection, "price"));
                     addImagesAndQuantityToProducts(products.getContent());

                     if(!products.hasContent()){
                            return new ResponseEntity<>(HttpStatus.NO_CONTENT);
                     }

                     return new ResponseEntity<>(products, HttpStatus.OK);
              }else{
                     List<Brand> brandList = new ArrayList<>();
                     List<double[]> priceRangeList = new ArrayList<>();

                     if(brands != null){
                            brandList = brandService.findSpecificBrands(brands);
                     }

                     if(ranges != null){
                            for(String range : ranges){
                                   double[] rangeValues = new double[2];
                                   rangeValues[0] = getRangeMin(range);
                                   rangeValues[1] = getRangeMax(range);
                                   priceRangeList.add(rangeValues);
                            }
                     }

                     filteredProductPage =
productService.findByCategoryIdWithBrandAndPriceRange(category, brandList,
priceRangeList, page, order);

                     if(filteredProductPage.getContent().isEmpty()){
```

```
                    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
                }

        addImagesAndQuantityToProducts(filteredProductPage.getContent());

        return new ResponseEntity<>(filteredProductPage, HttpStatus.OK);
        }

}
```

Παρόμοια, δημιουργούμε τα άκρα, που θα χρειαστούν τα φίλτρα:

- Για την αρχικοποίηση των φίλτρων εταιριών, ο πελάτης θα ζητάει τον αριθμό των προϊόντων της κάθε εταιρίας, σε μια κατηγορία. Αν επιλεχθεί κάποιο φίλτρο από τα διαστήματα τιμών μετά, τότε στέλνει μια παράμετρο range για να πάρει τον αριθμό των προϊόντων της κάθε εταιρίας, σε ένα διάστημα τιμών.
    - αρχικό αίτημα, για την εταιρία Solgar: HTTP GET *http://localhost:8080/api/categories/vitamins/brands/Solgar/products/count*
    - αριθμός προϊόντων εταιρίας Solgar, στα διαστήματα 0 - 9.99, 10 - 19.99 : HTTP GET *http://localhost:8080/api/categories/vitamins/brands/Solgar/products/count?range=1&range=2*
- Για την αρχικοποίηση των φίλτρων των διαστημάτων τιμών, ο πελάτης ζητάει τον αριθμό των προϊόντων ανά διάστημα τιμών. Αν επιλεχθεί κάποιο φίλτρο από τις εταιρίες μετά, τότε στέλνει μια παράμετρο brand για να πάρει τον αριθμό των προϊόντων στο συγκεκριμένο διάστημα τιμών, που ανήκουν σε αυτές τις εταιρίες.
    - αρχικό αίτημα, για το διάστημα 0 - 9.99 : HTTP GET *http://localhost:8080/api/categories/vitamins/ranges/1/products/count*
    - αριθμός προϊόντων εταιριών Solgar και Solaray, στο διάστημα 0 - 9.99 : HTTP GET *http://localhost:8080/api/categories/vitamins/ranges/1/products/count?brand=Solgar&brand=Solaray*

Παρακάτω, δίνεται το τμήμα του κώδικα της Angular, το οποίο πραγματοποιεί τη λήψη των προϊόντων (Κώδικας 5-6).

```
115       if( oldCategory === this.category && (this.currentPage != oldPage || didBrandParametersChange || didRangePar
116         if(isClickFromNavigationBarParam && (didBrandParametersChange || didRangeParametersChange) && (this.brandF
117           this.sidebar.updateSidebar();
118         }
119
120         this.httpSubscription2 = this.categoryService.getCategoryProductsPage(this.category, this.currentPage, thi
121           this.pageNumbers = [];
122           this.productPage = productPage;
123           this.products = productPage.content;
124           this.setProductRange(this.productPage.number, this.productPage.numberOfElements);
125           this.isGetCategoryProductsRequestDone = true;
126           this.initializePageNumberArray(this.pageNumbers, this.productPage.totalPages);
127         },
128         error => {
129           console.log(error);
130           this.isGetCategoryProductsRequestDone = true;
131         });
132       }
133     });
134   }
```

Κώδικας 5-6 Λήψη προϊόντων – category-page.component.ts

Όταν ο χρήστης κάνει κλικ σε κάποια εταιρία (event), εκτελείται η μέθοδος onSelectedBrand(), η οποία ενημερώνει τον αριθμό των προϊόντων σε κάθε διάστημα τιμών (Κώδικας 5-7). Αντίστοιχα, όταν κάνει κλικ σε κάποιο διάστημα (event), εκτελείται η μέθοδος onSelectedPriceRange(), η οποία ενημερώνει τον αριθμό των προϊόντων για κάθε εταιρία.

```
155   onSelectedBrand() : void{
156     this.selectedBrands = this.numberOfProductsPerBrand
157                           .filter(brandOption => brandOption.checked)
158                           .map(brandOption => brandOption.brand);
159
160     this.router.navigate(['/category', this.category],
161           {queryParams : { page : 0, fn : 'no', brand : this.selectedBrands, range : this.selectedPriceRanges}});
162     this.updatePriceRanges();
163   }
164
165   onSelectedPriceRange() : void{
166     this.selectedPriceRanges = this.numberOfProductsPerPriceRange
167                           .filter(priceRangeOption => priceRangeOption.checked)
168                           .map(priceRangeOption => priceRangeOption.rangeId);
169
170     this.router.navigate(['/category', this.category],
171           {queryParams : { page : 0, fn : 'no',  brand : this.selectedBrands, range : this.selectedPriceRanges}});
172     this.updateBrands();
173   }
```

Κώδικας 5-7 Ενημέρωση φίλτρων – category-sidebar.component.ts

5.5.4 Οθόνη εγγραφής

Για την πραγματοποίηση της εγγραφής, ο πελάτης κάνει ένα HTTP POST αίτημα στο URL *http://localhost:8080/api/register*, στέλνοντας ένα entity User στο σώμα του αιτήματος. Προσθέτουμε την παρακάτω μέθοδο στον RegisterApiController. Αν υπάρχει ήδη χρήστης με αυτό το email ή username, επιστρέφουμε κωδικό κατάστασης 409

CONFLICT. Αν η διαδικασία είναι επιτυχής, επιστρέφουμε κωδικό κατάστασης 201 CREATED.

```
@RequestMapping(value = "/register", method = RequestMethod.POST)
public ResponseEntity<?> registerUser(@Valid @RequestBody User user){

        if(userService.doesUserExist(user)){
            return new ResponseEntity<>(new ErrorMessage("Unable to create. A
user with username " + user.getUsername() + " already exists."),
HttpStatus.CONFLICT);
        }

        if(userService.doesEmailExist(user)){
            return new ResponseEntity<>(new ErrorMessage("Unable to create. A
user with email " + user.getEmail() + " already exists."),
HttpStatus.CONFLICT);
        }
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setIs_active(true);

        if(userService.addUserAndIsSuccess(user)){
               Authority authority = new Authority();
               authority.setUser(user);
               authority.setRole("ROLE_USER");
               if(authorityService.saveAuthorityAndIsSuccess(authority)){
                   return new ResponseEntity<>(user, HttpStatus.CREATED);
                }
               }
          return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
```

Όταν ο χρήστης πατάει το κουμπί «Εγγραφή», καλείται η μέθοδος onSubmit() και πραγματοποιείται η εγγραφή (Κώδικας 5-8).

```
28    onSubmit(): void {
29      this.httpSubscription = this.registerService.registerUser(this.user).subscribe(
30        user => this.router.navigate(['/login'], {queryParams : { redirect : 'register'}}),
31
32        errorResponse => {
33        this.setProperFieldError(errorResponse.error.errorMessage);
34        });
35    }
```

Κώδικας 5-8 Πραγματοποίηση εγγραφής – register.component.ts

5.5.5 Οθόνη εισόδου

Όταν ο πελάτης κάνει το πρώτο αίτημα του στον εξυπηρετητή, το servlet container δημιουργεί μια συνεδρία HttpSession, με την εντολή `HttpSession session = request.getSession()`, δημιουργεί ένα μοναδικό ID και το αποθηκεύει στη μνήμη του server. Στην κεφαλίδα Set-Cookie της HTTP απάντησης του, ο εξυπηρετητής δημιουργεί ένα cookie με όνομα SESSION και του δίνει την τιμή του μοναδικού ID. Ο πελάτης

αντίστοιχα στέλνει το cookie στα επόμενα αιτήματα του, ώστε να μπορεί να τον αναγνωρίσει ο εξυπηρετητής και να πάρει την αντίστοιχη συνεδρία HttpSession από την μνήμη.

Για να εισέλθει ο χρήστης στην εφαρμογή, η Angular εφαρμογή θα κάνει ένα HTTP POST στο άκρο *http://localhost:8080/authentication/validateuser*, στέλνοντας το username και τον κωδικό του χρήστη. Αν είναι σωστά, τότε θα κάνει ένα HTTP GET στο *http://localhost:8080/authentication/session*, βάζοντας τα στοιχεία στο Authorization: Basic <credentials>, να πάρει το Session cookie και να μην τα ξαναστέλνει σε κάθε αίτημα.

```
@RequestMapping(value = "/authentication/validateuser", method =
RequestMethod.POST)
public ResponseEntity<?> login(@RequestBody UserCredentials userCredentials){
       if(!userService.doesEmailExist(userCredentials.getEmail()))
           return new ResponseEntity<>(new ErrorMessage("Δεν υπάρχει λογαριασμός
με αυτό το email"),HttpStatus.NOT_FOUND);

       User user = userService.findByEmail(userCredentials.getEmail());
      boolean doPasswordsMatch =
passwordEncoder.matches(userCredentials.getPassword(), user.getPassword());

       if(!doPasswordsMatch)
           return new ResponseEntity<>(new ErrorMessage("Ο κωδικός δεν είναι
σωστός"),HttpStatus.BAD_REQUEST);
       return new ResponseEntity<>(user, HttpStatus.OK);
}


@RequestMapping(value = "/authentication/session", method = RequestMethod.GET)
public ResponseEntity<?> getSession(Principal principal, HttpSession session){
       User user = userService.findByUsername(principal.getName());
       Session userSession = new Session();
       userSession.setId(session.getId());
       userSession.setUsername(principal.getName());
       String userId= user != null ? Integer.toString(user.getId()) : "";
       userSession.setUserid(userId);
       String type = principal == null ? "anonymous" : "user";
       userSession.setType(type);
       return new ResponseEntity<>(userSession, HttpStatus.OK);
}
```

Για να αποσυνδεθεί ο χρήστης, ο πελάτης κάνει ένα HTTP POST στο άκρο *http://localhost:8080/authentication/logout* και ακυρώνεται η συνεδρία του.

```
@RequestMapping(value = "/authentication/logout", method = RequestMethod.POST)
public ResponseEntity<?> logout(HttpSession session){
       session.invalidate();
       return new ResponseEntity<>(HttpStatus.OK);
}
```

Όταν ο χρήστης πατάει το κουμπί «Είσοδος», καλείται η μέθοδος login() και πραγματοποιείται η είσοδος (Κώδικας 5-9).

```
33    async login() {
34      let authenticationMessage = await this.authenticationService.login(this.userCredentials);
35      if(authenticationMessage === this.emailDoesNotExistErrorMessage){
36        this.emailDoesNotExist = true;
37        this.passwordIsNotCorrect = false;
38      }else if(authenticationMessage === this.passwordIsNotCorrectErrorMessage){
39        this.emailDoesNotExist = false;
40        this.passwordIsNotCorrect = true;
41      }else if(authenticationMessage === this.genericError){
42        this.emailDoesNotExist = false;
43        this.passwordIsNotCorrect = false;
44      }else{
45        this.router.navigate(['']);
46      }
47    }
```

Κώδικας 5-9 Πραγματοποίηση εισόδου – login.component.ts

Όταν ο χρήστης πατάει το κουμπί «Αποσύνδεση», καλείται η μέθοδος logout() και πραγματοποιείται η αποσύνδεση (Κώδικας 5-10).

```
73    logout(){
74      this.authenticationService.logout();
75    }
```

Κώδικας 5-10 Πραγματοποίηση αποσύνδεσης – navigation-bar.component.ts

5.5.6 Οθόνη καλαθιού

Στην οθόνη του καλαθιού ο χρήστης μπορεί να δει τα προϊόντα του καλαθιού του, να διαγράψει κάποιο προϊόν ή να ενημερώσει την ποσότητα του. Η προσθήκη προϊόντων γίνεται στην οθόνη του προϊόντος. Παρακάτω, δίνονται οι μέθοδοι του CartRepository και τα ερωτήματα, τα οποία πραγματοποιούν στον πίνακα cart της βάσης δεδομένων.

```
@Repository
public interface CartRepository extends JpaRepository<Cart, Integer>{

   @Query("select cartProduct from Cart cartProduct where cartProduct.user.id =
:userid")
   List<Cart> findUserCartProducts(@Param("userid") int userid);

   @Query(value = "delete from Cart cartProduct where cartProduct.user.id =
:userid")
   @Modifying
void deleteUserCart(@Param("userid") int userid);

   @Query("select cartProduct from Cart cartProduct where cartProduct.user.id =
:userid and cartProduct.product.id = :productid")
```

```
    Cart findUserCartProductRow(@Param("userid") int userid, @Param("productid")
int productid);

    @Query(value = "delete from Cart cartProduct where cartProduct.user.id =
:userid and cartProduct.product.id = :productid")
    @Modifying
    void deleteUserCartProductRow(@Param("userid") int userid,
@Param("productid") int productid);

    @Query(value = "insert into cart (user_id, product_id, quantity, expiration)
values ( :userid, :productid, :quantity, :expiration )", nativeQuery = true)
    void addProductToCart(@Param("userid") int userid, @Param("productid") int
productid, @Param("quantity") int quantity, @Param("expiration") Timestamp
expiration);

    @Query("update Cart cartProduct set cartProduct.quantity = :quantity where
cartProduct.user.id = :userid and cartProduct.product.id = :productid")
    @Modifying
    void updateCartProduct (@Param("userid") int userid, @Param("productid") int
productid, @Param("quantity") int quantity);
}
```

Ενδεικτικά, θα αναλύσουμε μόνο τη μέθοδο του CartApiController, για την προσθήκη προϊόντων. Όταν ο πελάτης θέλει να προσθέσει ένα προϊόν στο καλάθι, κάνει ένα HTTP POST αίτημα στο URL *http://localhost:8080/api/carts*, στέλνοντας ένα CartProduct στο σώμα του αιτήματος (request body), το οποίο περιέχει τα στοιχεία του προϊόντος, σε ποιο χρήστη ανήκει και την ποσότητα του.

Εάν δεν υπάρχει το προϊόν ή ο χρήστης στη βάση, ή η ποσότητα είναι αρνητική επιστρέφουμε κωδικό κατάστασης 400 BAD REQUEST. Δημιουργούμε το σύνθετο πρωτεύον κλειδί, το οποίο αποτελείται από το id του χρήστη και του προϊόντος, προσθέτουμε το προϊόν στη βάση και επιστρέφουμε κωδικό κατάστασης 201 CREATED.

```
@RequestMapping(value = "/carts", method = RequestMethod.POST)
public ResponseEntity<?> createCartProduct(@RequestBody CartProduct
cartProduct){
       User user = userService.findById(cartProduct.getUserid());
       Product product = productService.findById(cartProduct.getProductid());
       if(user == null){
             return new ResponseEntity<>(new ErrorMessage("Unable to create.
User does not exist"), HttpStatus.BAD_REQUEST);
       }
       if(product == null){
             return new ResponseEntity<>(new ErrorMessage("Unable to create.
Product does not exist"), HttpStatus.BAD_REQUEST);
       }

       if(cartProduct.getQuantity() < 0){
              return new ResponseEntity<>(new ErrorMessage("Unable to create.
Quantity cannot be negative"), HttpStatus.BAD_REQUEST);
       }
        CartPrimaryKey cartProductCompositeId = new
CartPrimaryKey(cartProduct.getUserid(), cartProduct.getProductid());
       Cart cartRow =  new Cart();
       Timestamp expiration = new Timestamp(System.currentTimeMillis());
```

```
        cartRow.setId(cartProductCompositeId);
        cartRow.setUser(user);
        cartRow.setProduct(product);
        cartRow.setQuantity(cartProduct.getQuantity());
        cartRow.setExpiration(expiration);

        if(cartService.addProductToCartAndIsSuccess(cartRow)){
            CartProduct cartProductCreated = new CartProduct();
            cartProductCreated.setUserid(cartRow.getId().getUserId());
            cartProductCreated.setProductid(cartRow.getId().getProductId());
            cartProductCreated.setName(cartRow.getProduct().getName());
          cartProductCreated.setBrand(cartRow.getProduct().getBrand().getName());
            cartProductCreated.setQuantity(cartRow.getQuantity());
            cartProductCreated.setPrice(cartRow.getProduct().getPrice());
            return new ResponseEntity<>(cartProductCreated, HttpStatus.CREATED);
        }
        return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
}
```

Όταν ο χρήστης πατάει το κουμπί «Προσθήκη στο καλάθι», η μέθοδος addProductToCart(), προσθέτει το προϊόν (Κώδικας 5-11). Εάν ο χρήστης είναι ανώνυμος, τότε το καλάθι του αποθηκεύεται στο LocalStorage του προγράμματος περιήγησης ως αλφαριθμητικό.

```
addProductToCart() : void{
  if(this.isUserLoggedIn && !this.isLocalStorageEmpty){
    let userId = localStorage.getItem('userid');
    this.isAddProductRequestDone = false;
    this.cartService.getCartProduct(Number(userId), this.product.id).toPromise()
                    .then(cartProduct => {...
                    })
                    .catch( errorRespons
                      this.cartService.addProductTocart(Number(userId), this.product.id, this.wantedQuantity)
                                      .toPromise()
                                      .then(cartProduct => {
                                       this.navigationBar.setCartCount(this.navigationBar.userId);
                                       this.isAddProductRequestDone = true;
                                       })
                                      .catch(errorResponse => {...
                                       });
                    });
  }else if(!this.isUserLoggedIn){
    let cartProduct = {} as CartProduct;
    cartProduct.productid = this.product.id;
    cartProduct.name = this.product.name;
    cartProduct.brand= this.product.brand.name;
    cartProduct.price = this.product.price;
    cartProduct.quantity = this.wantedQuantity;
    cartProduct.imageUrl = this.product.verySmallImageUrl;
    this.cartService.addProductToAnonymousUserCart(cartProduct);
    this.navigationBar.setAnonymousUserCartCount();
  }
}
```

(method) CartService.addProductTocart(userId: number, productId: number, quantity: number): Observable<CartProduct>

Κώδικας 5-11 Προσθήκη προϊόντος – product-page.component.ts

5.5.7 Οθόνη ολοκλήρωσης αγοράς

Στην οθόνη του καλαθιού ο χρήστης πληκτρολογεί το email του, τις πληροφορίες χρέωσης και αποστολής. Για την ολοκλήρωση της παραγγελίας, ο πελάτης κάνει ένα HTTP POST αίτημα στο URL *http://localhost:8080/api/orders*, στέλνοντας ένα OrderDetails στο σώμα του αιτήματος, το οποίο περιέχει το καλάθι του και τα παραπάνω στοιχεία.

Εάν το email δεν έχει συμπληρωθεί, επιστρέφουμε κωδικό κατάστασης 400 BAD REQUEST. Εισάγουμε τις πληροφορίες χρέωσης, διεύθυνσης, την παραγγελία και τα προϊόντα στη βάση και επιστρέφουμε 201 CREATED. Εάν δεν εισάχθηκε κάτι σωστά επιστρέφουμε 500 INTERNAL SERVER ERROR.

```
@RequestMapping(value = "/orders", method = RequestMethod.POST)
public ResponseEntity<?> createOrder(@Valid @RequestBody OrderDetails
orderDetails, Principal principal){

       boolean isAnonymous = principal == null ? true : false;
       if(isAnonymous && orderDetails.getEmail() == null){
          return new ResponseEntity<>(new ErrorMessage("Email was not
provided"), HttpStatus.BAD_REQUEST);
       }

       if(shippingInfoService.saveShippingInfoAndIsSuccess(orderDetails.getShipp
ing_info()) &&

       billingInfoService.saveBillingInfoAndIsSuccess(orderDetails.getBilling_in
fo())){

            Order order = new Order();
            if(!isAnonymous){
               User user = userService.findByUsername(principal.getName());
                order.setUser(user);
            }

            order.setEmail(orderDetails.getEmail());
            order.setStatus("Created");
            Timestamp currentDate = new Timestamp(System.currentTimeMillis());
            order.setOrder_date(currentDate);
            order.setShippingInfo(orderDetails.getShipping_info());
            order.setBillingInfo(orderDetails.getBilling_info());

            if(orderService.saveOrderAndIsSuccess(order)){
                List<OrderProduct> orderProducts = new ArrayList<>();
                for(CartProduct cartProduct : orderDetails.getCartProducts()){
                    OrderProduct orderProduct = new OrderProduct();
                     OrderProductPrimaryKey id = new
OrderProductPrimaryKey(order.getId(), cartProduct.getProductid());
                    orderProduct.setId(id);
                     orderProduct.setOrder(order);
       orderProduct.setProduct(productService.findById(cartProduct.getProductid(
)));
                     orderProduct.setQuantity(cartProduct.getQuantity());
                     orderProducts.add(orderProduct);
                }
```

```
                    if(orderProductService.saveOrderProducts(orderProducts)){
                emailService.sendEmail(order, orderDetails.getCartProducts());
                return new ResponseEntity<Order>(order,  HttpStatus.CREATED);
        }
    }
}
    return new ResponseEntity<>( HttpStatus.INTERNAL_SERVER_ERROR);
        }
```

Όταν ο χρήστης πατάει το κουμπί «Ολοκλήρωση αγοράς», η μέθοδος onSubmit(), καλείται και πραγματοποιεί την παραγγελία (Κώδικας 5-12).

```
51  onSubmit(): void {
52    this.isAddOrderRequestDone =  false;
53    this.orderDetails.cartProducts = this.cartProducts;
54    this.orderDetails.shipping_info.method = this.isCourierChecked ? 'Courier' : 'Takeaway';
55    this.orderDetails.billing_info.method =  'Cash On Delivery' ;
56    if(this.orderDetails.isShippingAddressSameWithBillingAddress)
57      this.copyShippingInfoToBillingInfo();
58
59    this.checkoutService.addOrder(this.orderDetails).toPromise()
60                        .then(order => {
61                          this.isAddOrderRequestDone =  true;
62                          this.checkoutService.order = order;
63                          this.checkoutService.setOrderProducts(this.cartProducts);
64                          this.deleteUserCart();
65                          this.router.navigate(['/order', order.id]) ;
66                          })
67                        .catch(errorResponse => {
68                          console.log(errorResponse)
69                          this.isAddOrderRequestDone =  true;});
70  }
```

Κώδικας 5-12 Πραγματοποίηση παραγγελίας – checkout-page.component.ts

5.5.8 Οθόνη αναζήτησης προϊόντων

Στην οθόνη αναζήτησης προϊόντων ο χρήστης πληκτρολογεί έναν όρο και εμφανίζονται τα σχετικά προϊόντα. Για την πραγματοποιήση της αναζήτησης, ο πελάτης κάνει ένα HTTP GET αίτημα στο URL *http://localhost:8080/api/search?keyword=shampoo&page=0&order=asc*, όπου η παράμετρος keyword είναι οι όροι αναζήτησης. Πραγματοποιούμε το ερώτημα `Select prod from Product prod where prod.name like "%shampoo%"` στη βάση δεδομένων.

Οι μέθοδοι του SearchServiceImpl για την εύρεση των σχετικών προϊόντων και την ανανέωση των φίλτρων είναι παρόμοιες τη μέθοδο `ProductPage findByCategoryIdWithBrandAndPriceRange(Category category, List<Brand>`

brands , List<double[]> priceRanges, int page, String order), που χρησιμοποιούμε το JPA Criteria API, οπότε θα παραλειφθούν.

Η παρακάτω μέθοδος του SearchApiController επιστρέφει τα προϊόντα της αναζήτησης με κωδικό κατάστασης 200 OK. Εάν δεν βρέθηκαν σχετικά προϊόντα, τότε επιστρέφουμε κωδικό κατάστασης 204 NO CONTENT.

```
@RequestMapping(value = "/search", method = RequestMethod.GET)
public ResponseEntity<?> getSearchProducts(@RequestParam(value = "page") int
page, @RequestParam(value = "keyword") String[] keywords,
@RequestParam(value = "brand", required = false) String[] brands,
@RequestParam(value = "range", required = false) String[] ranges,
@RequestParam(value = "order", required = false) String order){

	ProductPage productPage = new ProductPage();
	List<Brand> brandList = new ArrayList<>();
	List<double[]> priceRangeList = new ArrayList<>();
	if(brands != null){
		brandList = brandService.findSpecificBrands(brands);
	}

	if(ranges != null){
		for(String range : ranges){
			double[] rangeValues = new double[2];
			rangeValues[0] = getRangeMin(range);
			rangeValues[1] = getRangeMax(range);
			priceRangeList.add(rangeValues);
		}
	}

	productPage = searchService.findBySearchTerms(keywords, page, brandList,
priceRangeList, order);
	if(productPage.getContent().isEmpty()){
		return new ResponseEntity<>(HttpStatus.NO_CONTENT);
	}
	addImagesAndQuantityToProducts(productPage.getContent());
	return new ResponseEntity<>(productPage, HttpStatus.OK);
}
```

Παρόμοια, η παρακάτω μέθοδος επιστρέφει τον αριθμό των προϊόντων, ανά διάστημα τιμών.

```
@RequestMapping(value = "/search/ranges/{rangeid}/products/count", method =
RequestMethod.GET)
public ResponseEntity<?>
getSearchProductsNumbersByRange(@PathVariable("rangeid") String rangeid,
@RequestParam(value = "brand", required = false) String[] brands,
@RequestParam(value = "keyword") String[] keywords){

	ProductsPerPriceRange ppNumber = new ProductsPerPriceRange();
	List<Brand> brandList = new ArrayList<>();

	if(brands != null){
		brandList = brandService.findSpecificBrands(brands);
	}

	ppNumber.setNumber(searchService.findSearchProductsNumberByRange(keywords
, getRangeMin(rangeid), getRangeMax(rangeid), brandList));
```

```
        ppNumber.setMin(getRangeMin(rangeid));
        ppNumber.setMax(getRangeMax(rangeid));
        ppNumber.setRangeId(Integer.parseInt(rangeid));
      return new ResponseEntity<ProductsPerPriceRange>(ppNumber, HttpStatus.OK);
}
```

Όταν ο χρήστης πατάει το κουμπί «Αναζήτηση» (Κώδικας 5-13), μεταφέρεται στην οθόνη αναζήτησης , όπου εμφανίζονται τα προϊόντα (Κώδικας 5-14).

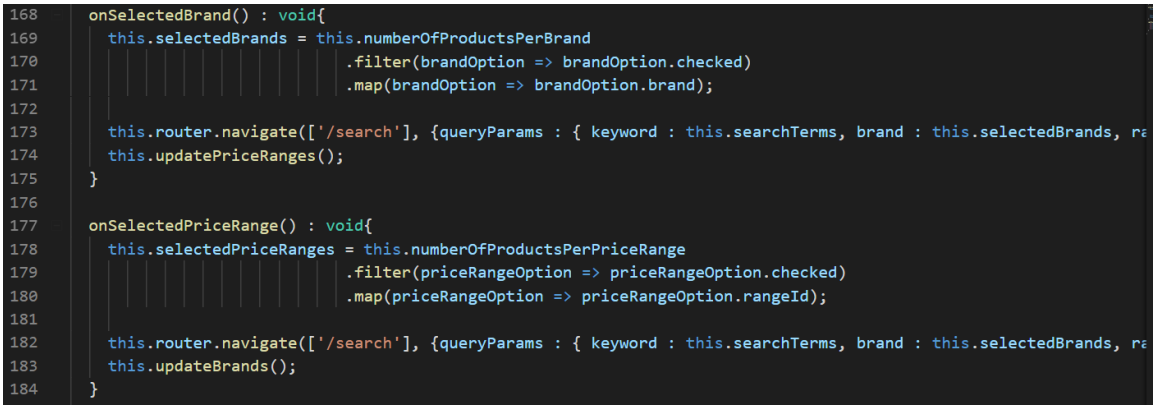
```
87  onSearchClicked() : void{
88    var regularExpression = /[^0-9^a-z]+/;
89    var keywords = this.searchText.split(regularExpression);
90    this.router.navigate(['/search'], {queryParams : { keyword : keywords, brand : this.selectedBrands, range : th
91  }
```

Κώδικας 5-13 Μεταφορά στην οθόνη αναζήτησης – navigation-bar.component.ts

```
69        this.httpSubscription = this.searchService.getSearchProducts(this.keywords, this.currentPage, this.selected\
70          if(productPage !== null){
71            this.pageNumbers = [];
72            this.productPage = productPage;
73            this.products = productPage.content;
74            this.setProductRange(this.productPage.number, this.productPage.numberOfElements);
75            this.isGetSearchProductsRequestDone = true;
76            this.didSearchReturnAnyProducts = true;
77            this.initializePageNumberArray(this.pageNumbers, this.productPage.totalPages);
78          }else{
79            this.isGetSearchProductsRequestDone = true;
80            this.didSearchReturnAnyProducts = false;
81          }
82        },
83        error => {
84          this.isGetSearchProductsRequestDone = true;
85          this.didSearchReturnAnyProducts = false;
86          console.log(error);
87        });
```

Κώδικας 5-14 Πραγματοποίηση αναζήτησης – search-page.component.ts

Στα φίλτρα, όταν ο χρήστης κάνει κλικ σε κάποια εταιρία (event), εκτελείται η μέθοδος onSelectedBrand(), η οποία ενημερώνει τον αριθμό των προϊόντων σε κάθε διάστημα τιμών (Κώδικας 5-15). Αντίστοιχα, όταν κάνει κλικ σε κάποιο διάστημα (event), εκτελείται η μέθοδος onSelectedPriceRange(), η οποία ενημερώνει τον αριθμό των προϊόντων για κάθε εταιρία.



```
168   onSelectedBrand() : void{
169     this.selectedBrands = this.numberOfProductsPerBrand
170                           .filter(brandOption => brandOption.checked)
171                           .map(brandOption => brandOption.brand);
172
173     this.router.navigate(['/search'], {queryParams : { keyword : this.searchTerms, brand : this.selectedBrands, ra
174     this.updatePriceRanges();
175   }
176
177   onSelectedPriceRange() : void{
178     this.selectedPriceRanges = this.numberOfProductsPerPriceRange
179                           .filter(priceRangeOption => priceRangeOption.checked)
180                           .map(priceRangeOption => priceRangeOption.rangeId);
181
182     this.router.navigate(['/search'], {queryParams : { keyword : this.searchTerms, brand : this.selectedBrands, ra
183     this.updateBrands();
184   }
```

Κώδικας 5-15 Ενημέρωση φίλτρων – search-sidebar.component.ts

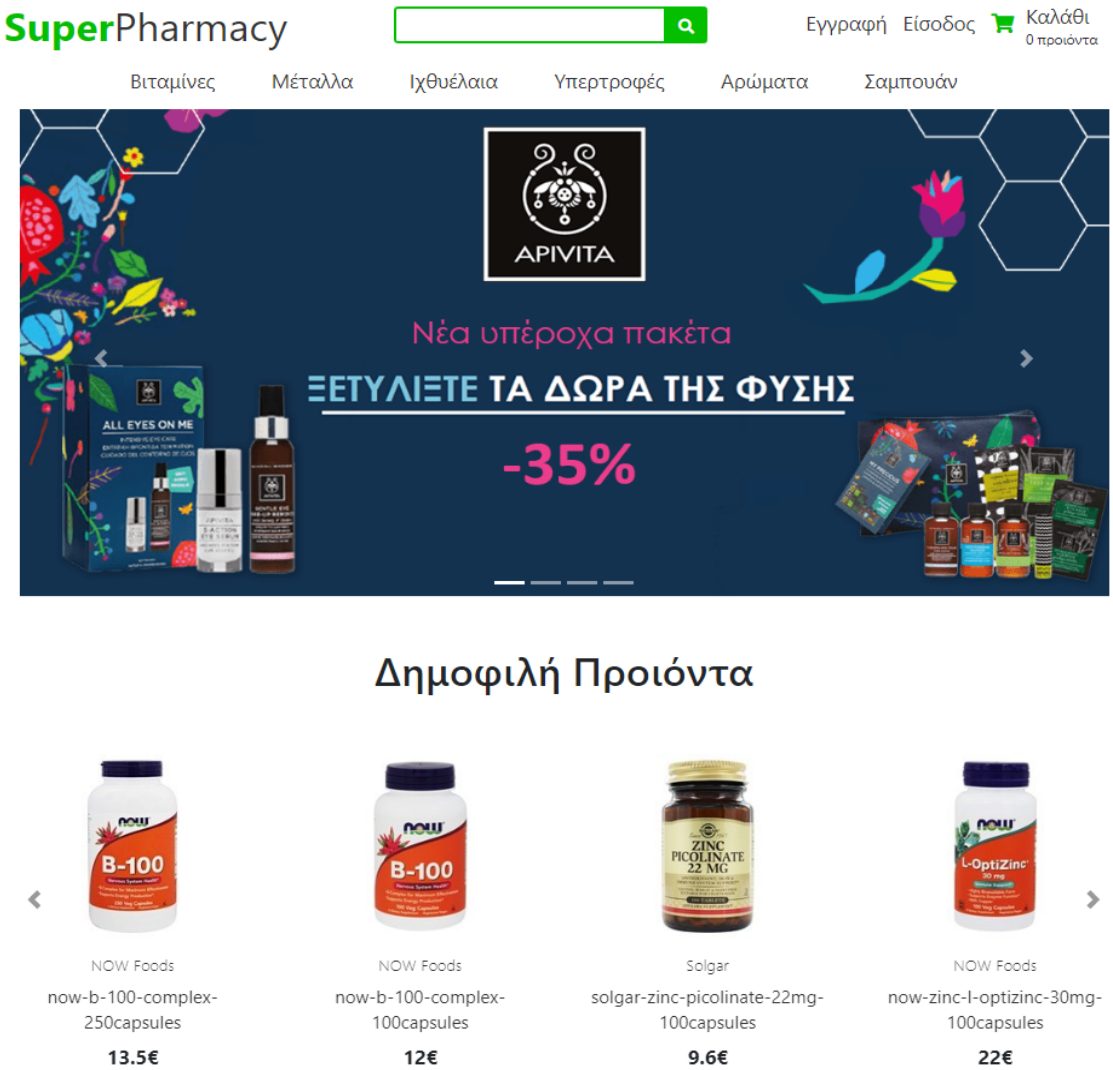
6 Παρουσίαση της εφαρμογής

Σε αυτό το κεφάλαιο, θα παρουσιαστούν οι οθόνες και οι λειτουργίες της εφαρμογής.

6.1 Αρχική οθόνη

Στο πάνω μέρος της οθόνης εμφανίζονται το πεδίο αναζήτησης, το καλάθι του χρήστη, οι κατηγορίες των προϊόντων, οι προσφορές του καταστήματος και τα δημοφιλή προϊόντα (Εικόνα 6-1). Στο κάτω μέρος εμφανίζονται τα νέα προϊόντα, οι υπηρεσίες και άλλες πληροφορίες του καταστήματος (Εικόνα 6-2).

Ο χρήστης μπορεί, να αναζητήσει προϊόντα, πληκτρολογώντας τον όρο στο πεδίο αναζήτησης. Αν πατήσει πάνω σε μια κατηγορία, μεταφέρεται στην οθόνη κατηγορίας προϊόντων, ενώ αν πατήσει πάνω σε ένα προϊόν, μεταφέρεται στην οθόνη προϊόντος.

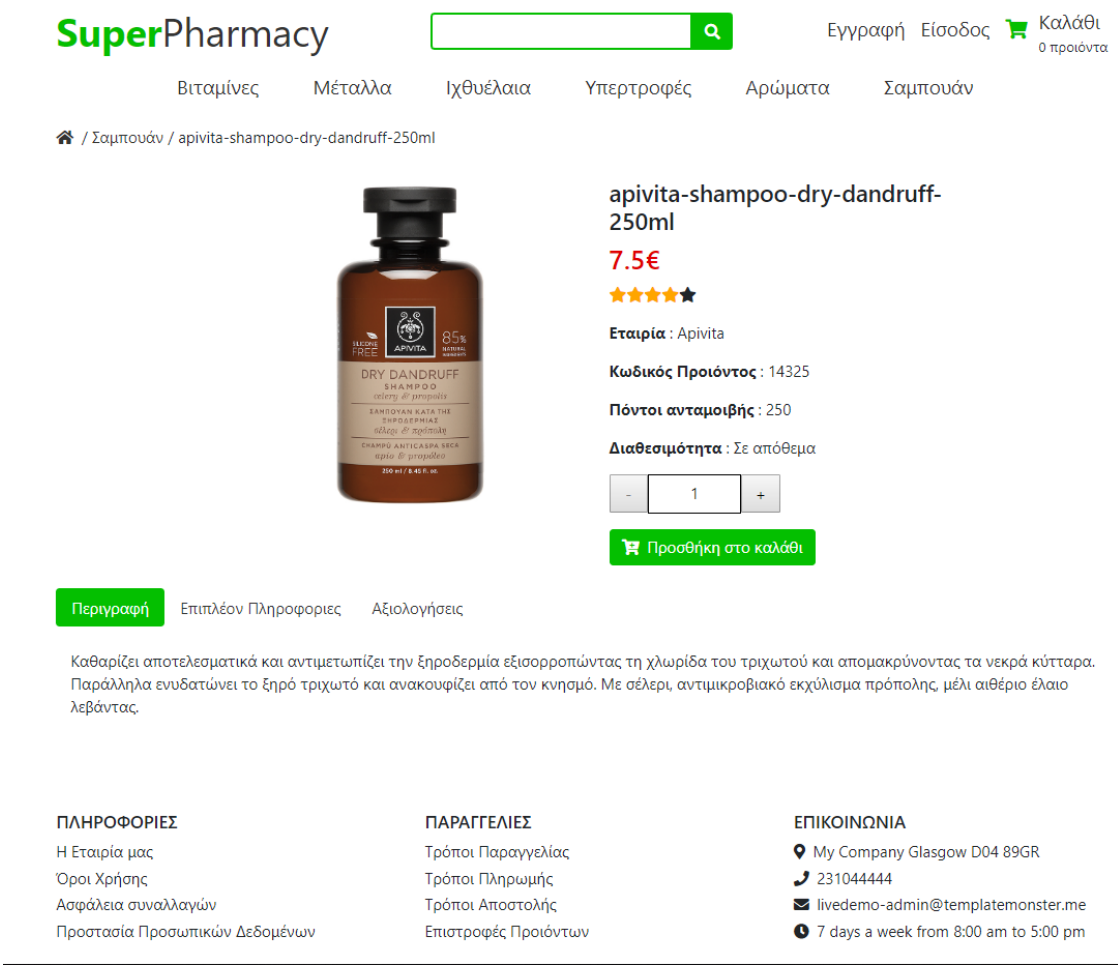


Εικόνα 6-1 Αρχική οθόνη – πάνω μέρος

Εικόνα 6-2 Αρχική οθόνη – κάτω μέρος

6.2 Οθόνη λεπτομερειών προϊόντος

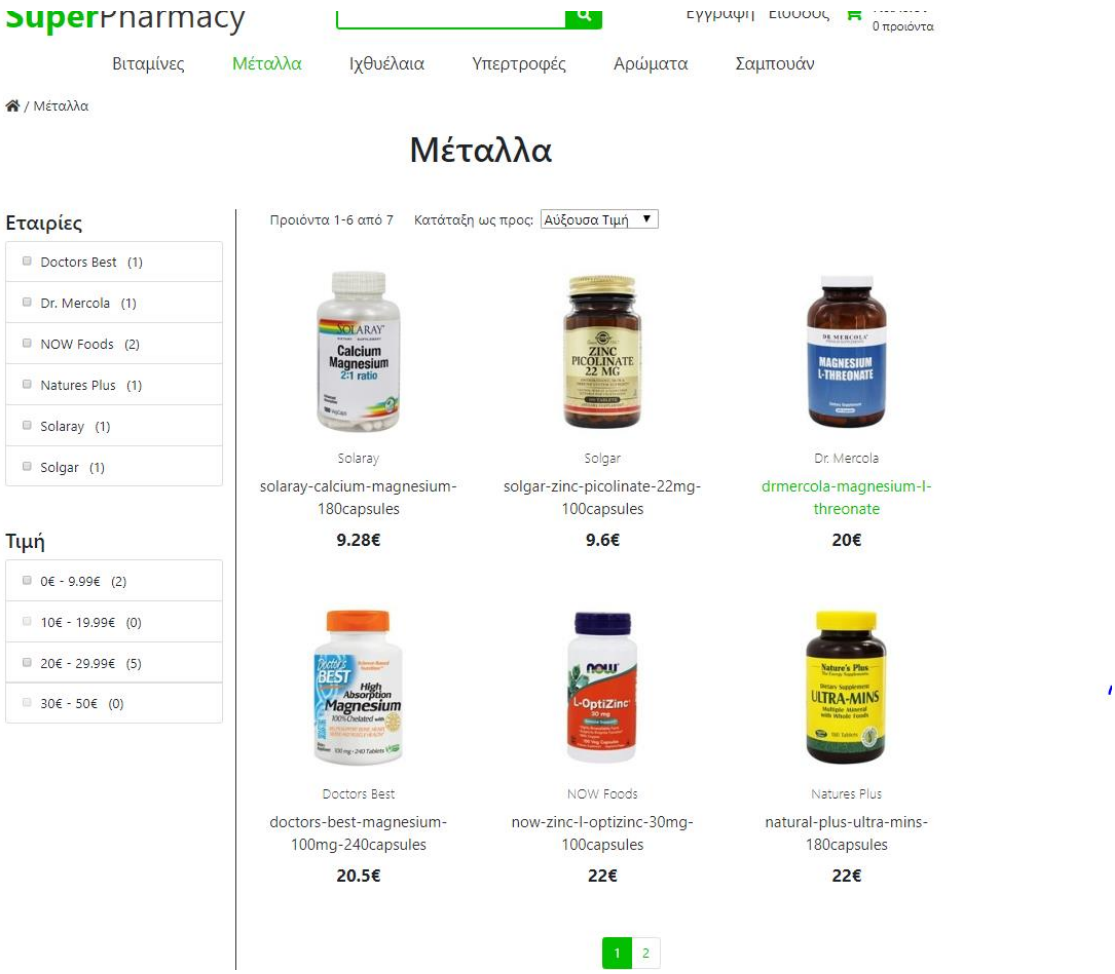
Στην οθόνη εμφανίζονται οι λεπτομέρειες του προϊόντος, όπως το όνομα, η τιμή του, η εταιρία και η περιγραφή του (Εικόνα 6-3). Ο χρήστης μπορεί, να επιλέξει την ποσότητα που επιθυμεί και να το προσθέσει στο καλάθι του.



Εικόνα 6-3 Οθόνη λεπτομερειών προϊόντος

6.3 Οθόνη κατηγορίας προϊόντων

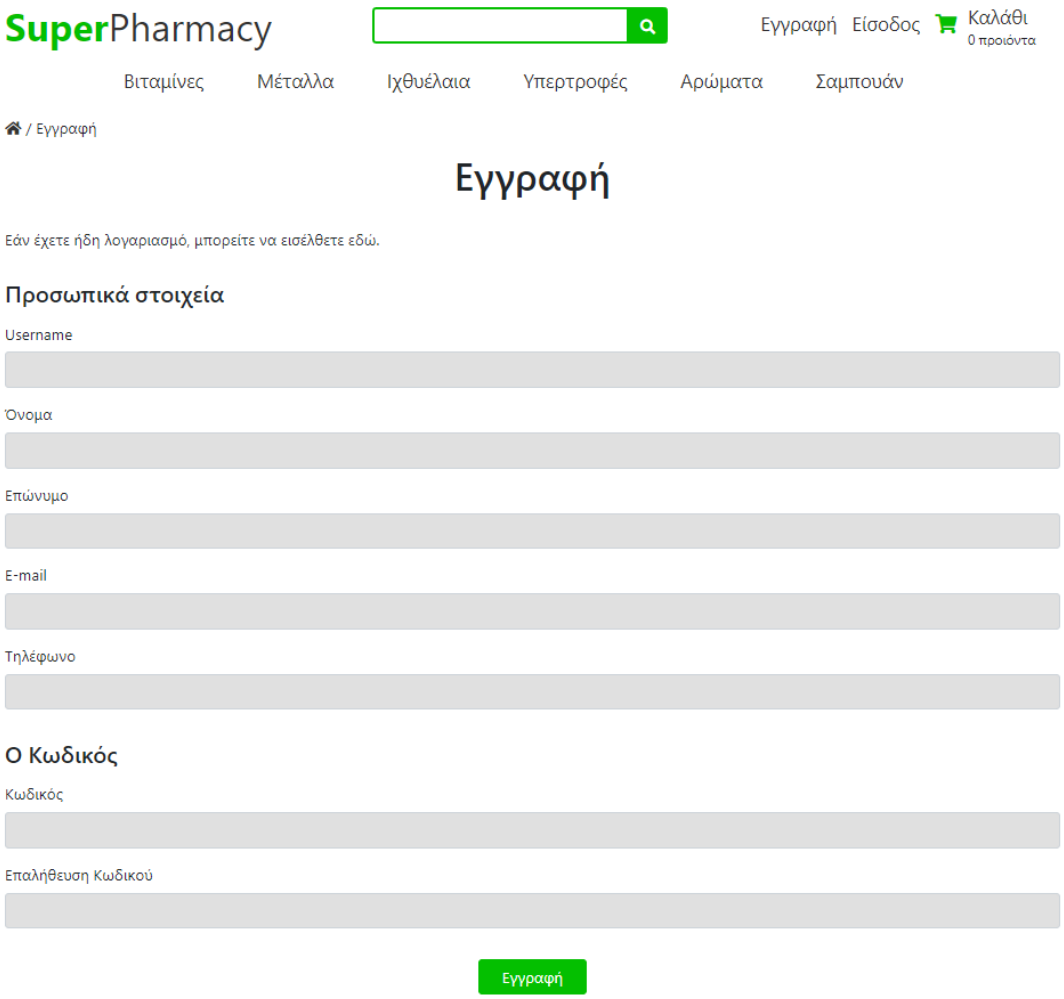
Στην οθόνη εμφανίζονται τα προϊόντα της κατηγορίας (Εικόνα 6-4). Ο χρήστης μπορεί, να περιορίσει τα προϊόντα, επιλέγοντας εταιρίες ή διαστήματα τιμών από τα φίλτρα και να επιλέξει την ταξινόμηση τους με βάση την τιμή (αύξουσα / φθίνουσα). Αν πατήσει σε κάποιο προϊόν, μεταφέρεται στη οθόνη του προϊόντος.



Εικόνα 6-4 Οθόνη κατηγορίας προϊόντων

6.4 Οθόνη εγγραφής

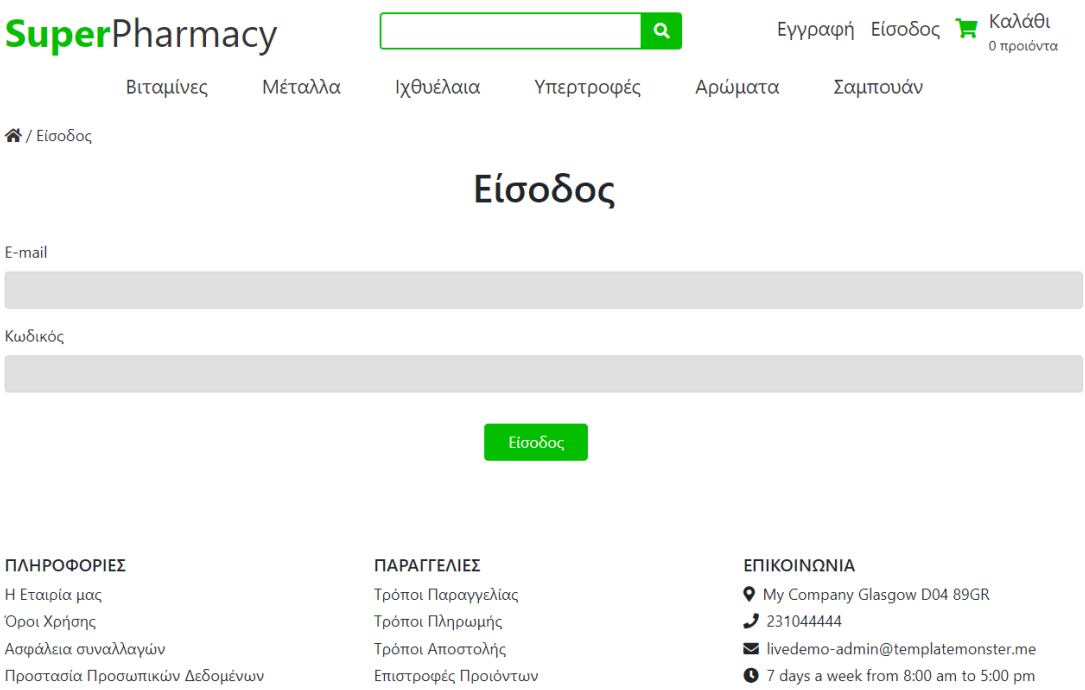
Στην οθόνη πραγματοποιείται η εγγραφή του χρήστη (Εικόνα 6-5). Ο χρήστης συμπληρώνει τα στοιχεία και παταέι το κουμπί «Εγγραφή». Εάν είναι σωστά, μεταφέρεται στην οθόνη εισόδου, αλλιώς εμφανίζεται μήνυμα σφάλματος κάτω από το αντίστοιχο πεδίο.



Εικόνα 6-5 Οθόνη εγγραφής

6.5 Οθόνη εισόδου

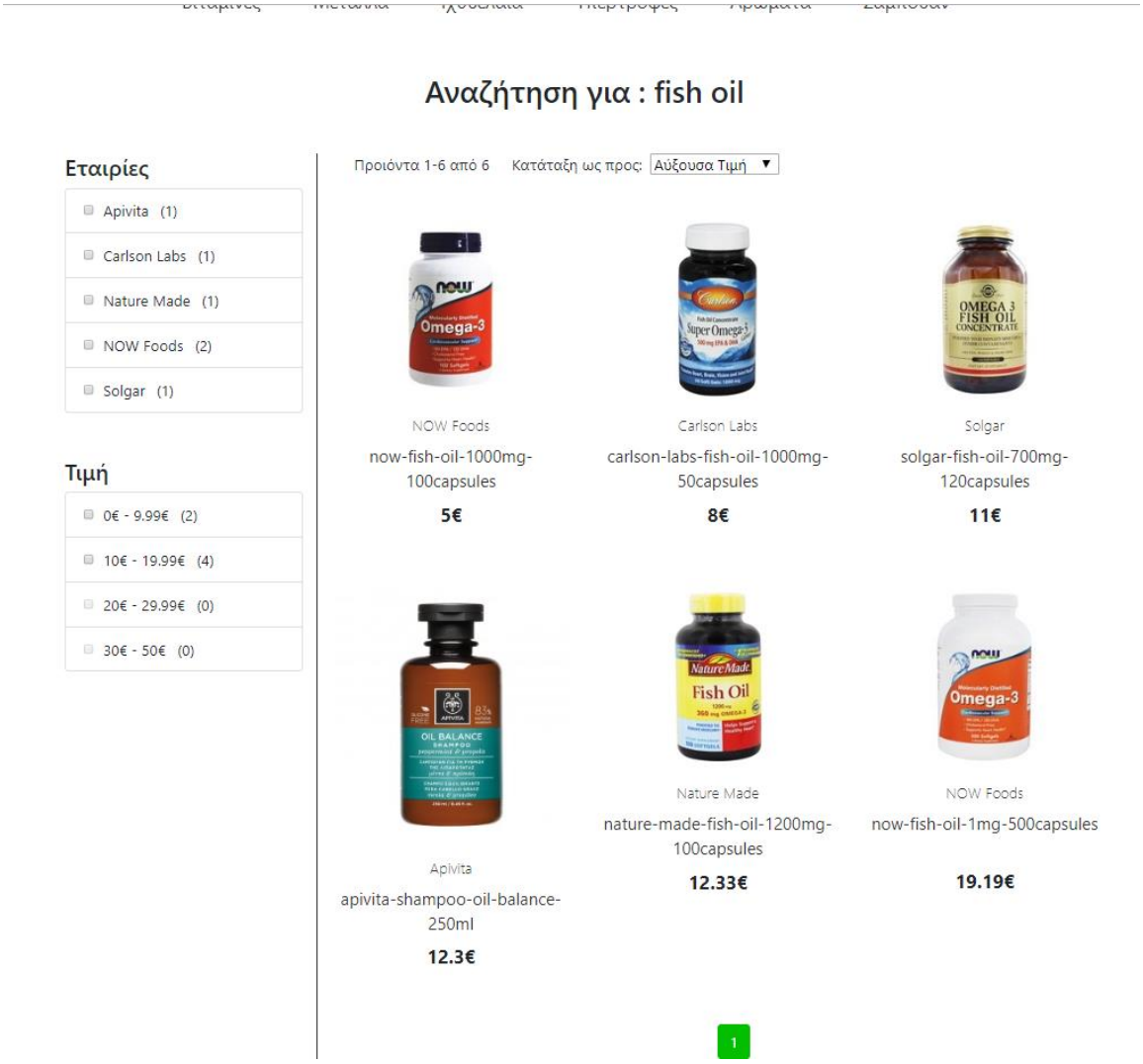
Στην οθόνη ο χρήστης συνδέεται στην εφαρμογή (Εικόνα 6-6). Συμπληρώνει το email και τον κωδικό του και πατάει το κουμπί «Είσοδος». Εάν είναι σωστά, μεταφέρεται στην αρχική οθόνη, αλλιώς εμφανίζεται μήνυμα σφάλματος κάτω από το αντίστοιχο πεδίο.



Εικόνα 6-6 Οθόνη εισόδου

6.6 Οθόνη αναζήτησης

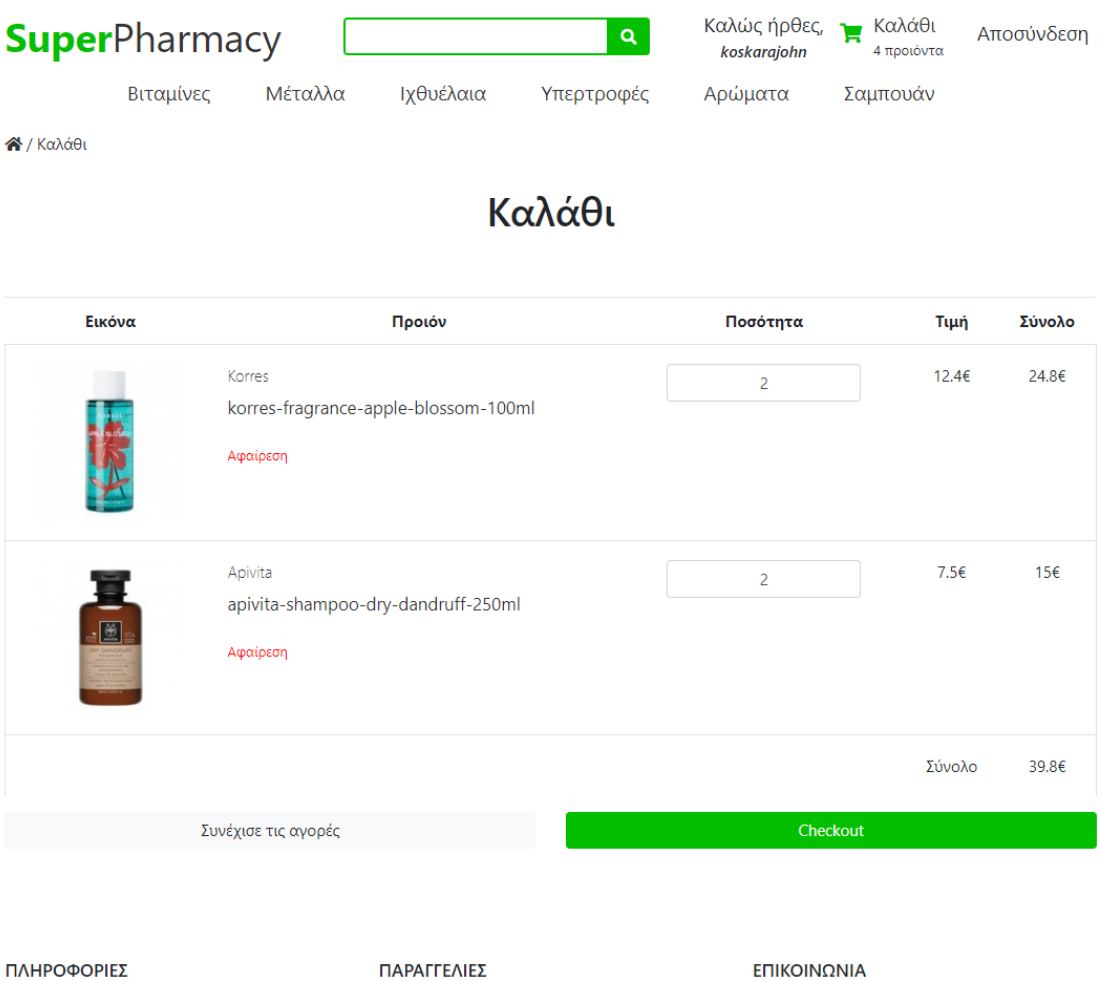
Στην οθόνη της Εικόνας 6-7 εμφανίζονται τα προϊόντα της αναζήτησης. Ο χρήστης μπορεί, να περιορίσει τα προϊόντα, επιλέγοντας εταιρίες ή διαστήματα τιμών από τα φίλτρα και να επιλέξει την ταξινόμηση τους με βάση την τιμή (αύξουσα / φθίνουσα). Αν πατήσει σε κάποιο προϊόν, μεταφέρεται στη οθόνη του προϊόντος.



Εικόνα 6-7 Οθόνη αναζήτησης

6.7 Οθόνη καλαθιού

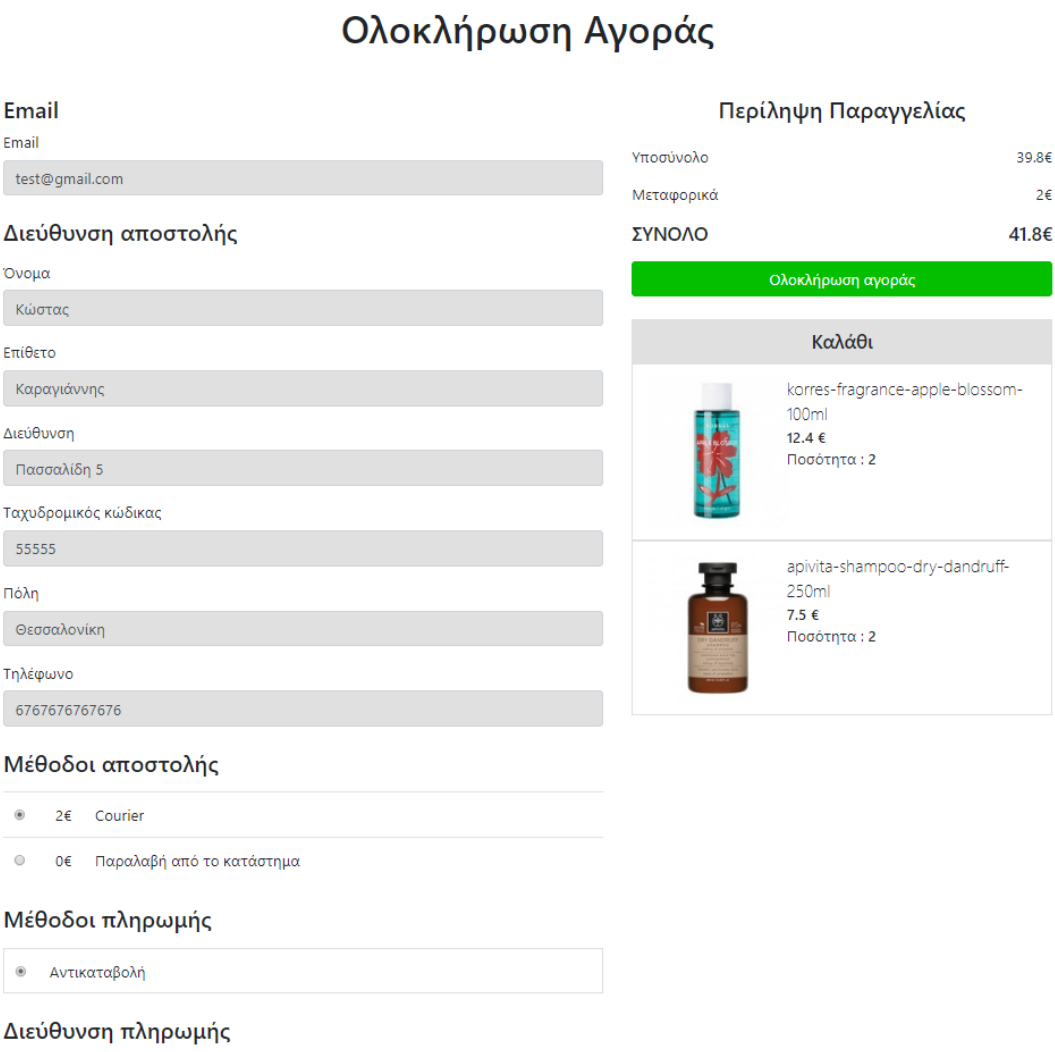
Στην οθόνη της Εικόνας 6-8 εμφανίζονται τα προϊόντα του καλαθιού του χρήστη. Ο χρήστης μπορεί να ενημερώσει την ποσότητα κάποιου προϊόντος ή να το αφαιρέσει από το καλάθι, πατώντας το κουμπί «Αφαίρεση». Εάν πατήσει στο κουμπί «Checkout», μεταφέρεται στην οθόνη Ολοκλήρωσης αγοράς, ενώ αν πατήσει στο κουμπί «Συνέχισε τις αγορές» μεταφέρεται στην Αρχική οθόνη.



Εικόνα 6-8 Οθόνη καλαθιού

6.8 Οθόνη ολοκλήρωσης αγοράς

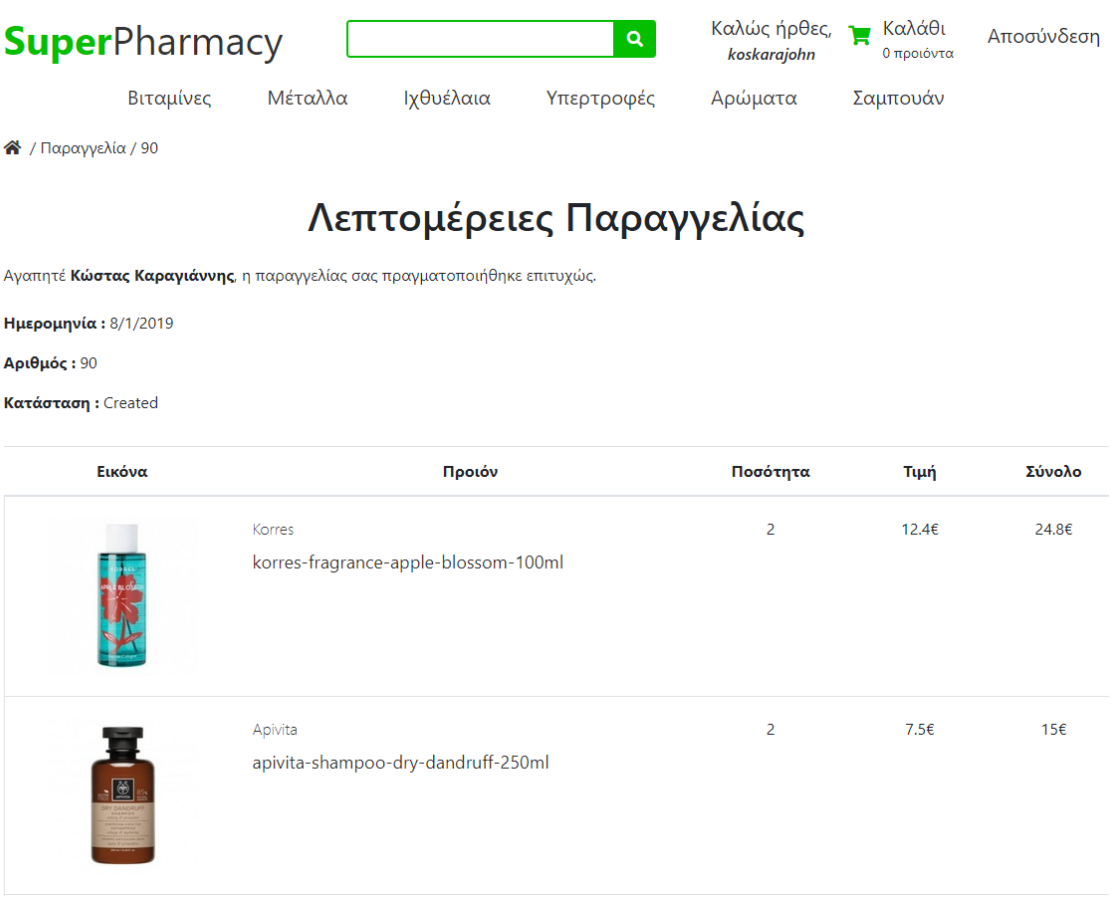
Στην οθόνη αυτή (Εικόνα 6-9), ο χρήστης συμπληρώνει το email και τις διευθύνσεις αποστολής και πληρωμής. Υπάρχει επιλογή, ώστε να μην συμπληρώσει τα στοιχεία πληρωμής, όταν η διεύθυνση αποστολής είναι ίδια με αυτή της πληρωμής. Επίσης, πρέπει να επιλέξει μέθοδο αποστολής. Για να ολοκληρώσει την αγορά του, ο χρήστης πατάει το κουμπί «Ολοκλήρωση αγοράς» και μεταφέρεται στην Οθόνη εμφάνισης παραγγελίας. Εάν συμπλήρωσε κάποιο πεδίο λανθασμένα, τότε εμφανίζεται μήνυμα σφάλματος κάτω από το αντίστοιχο πεδίο.



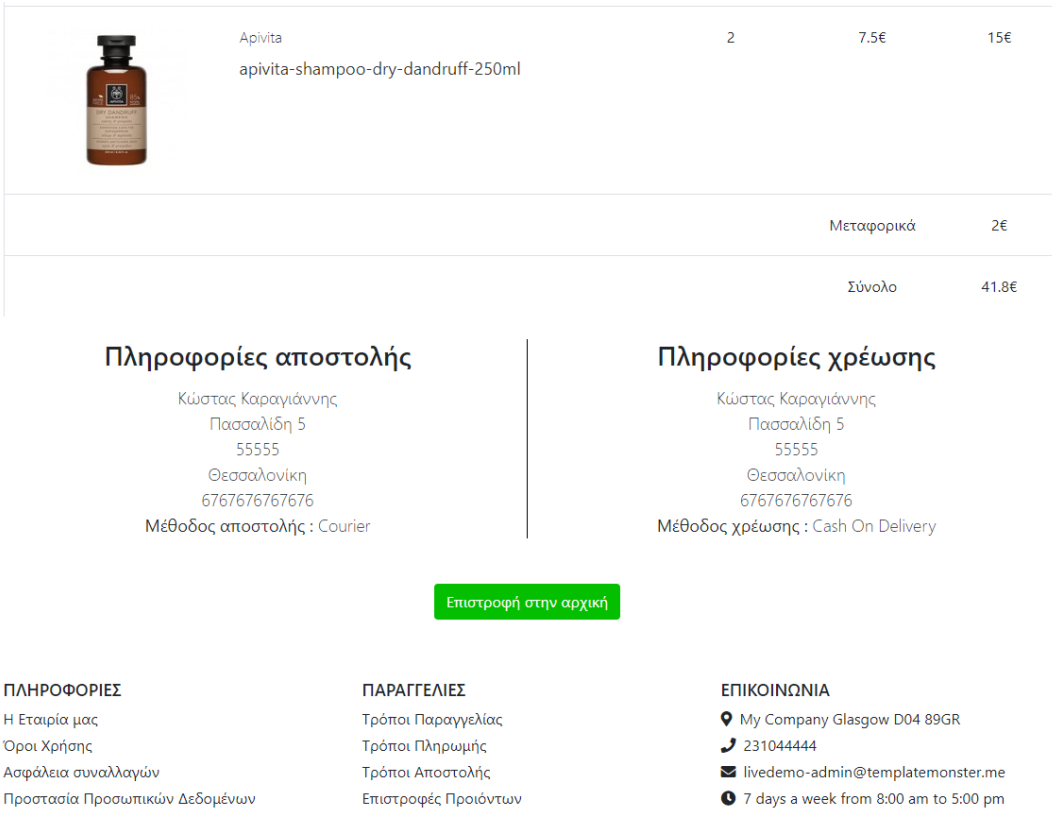
Εικόνα 6-9 Οθόνη ολοκλήρωσης αγοράς

6.9 Οθόνη εμφάνισης παραγγελίας

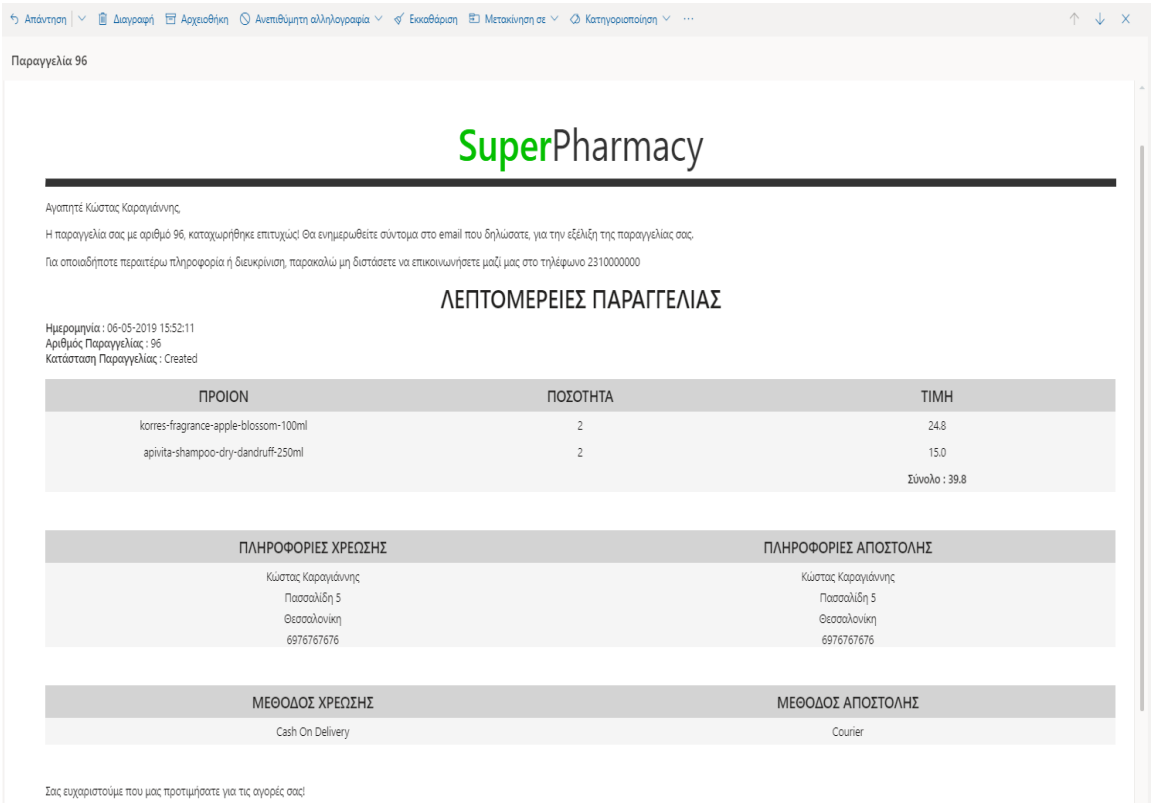
Στην οθόνη των Εικόνων 6-10, 6-11 εμφανίζονται τα στοιχεία της παραγγελίας. Επίσης, στέλνεται email στον χρήστη, που περιέχει τα στοιχεία της παραγγελίας του (Εικόνα 6-12). Αν πατήσει στο κουμπί «Επιστροφή στην αρχική οθόνη», τότε μεταφέρεται στην Αρχική οθόνη.



Εικόνα 6-10 Οθόνη εμφάνισης παραγγελίας – πάνω μέρος



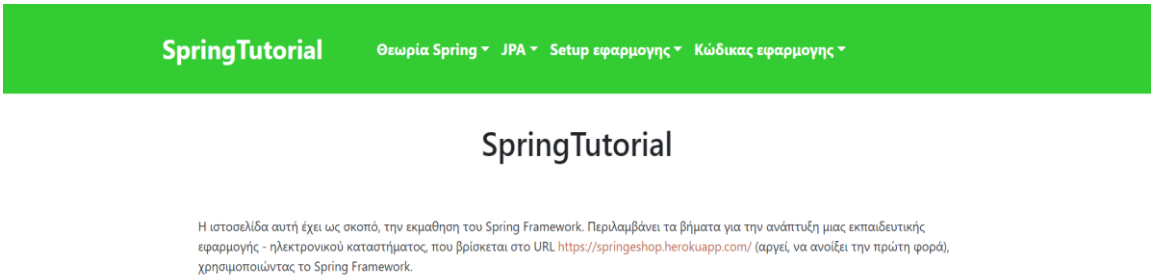
Εικόνα 6-11 Οθόνη εμφάνισης παραγγελίας – κάτω μέρος



Εικόνα 6-12 Email επιβεβαίωσης παραγγελίας

7 Περιεχόμενο ιστότοπου

Ο εκπαιδευτικός ιστότοπος περιλαμβάνει τη διαδικασία ανάπτυξης της εφαρμογής και θεωρία, σχετικά με το Spring framework, το JPA και την Angular και βρίσκεται στην τοποθεσία https://koskarajohn.github.io/SpringTutorial/. Το υλικό του ιστότοπου μπορεί να αξιοποιηθεί από κάθε ενδιαφερόμενο, για την εκμάθηση της ανάπτυξης διαδικτυακών εφαρμογών σε Java με το Spring Framework.



Εικόνα 7-1 Εκπαιδευτικός ιστότοπος

Ο ιστότοπος έχει πέντε ενότητες:

1. **Θεωρία Spring:** Σε αυτή την ενότητα εξηγείται τι είναι το Spring Framework και το Spring Boot, πως πραγματοποιείται η έγχυση εξαρτήσεων από το Spring Container και πως λειτουργεί το μοντέλο αρχιτεκτονικής MVC.
    - *Τι είναι το Spring*: Αναλύεται το Spring Framework, τα χαρακτηριστικά του και τα τμήματα του
    - *Spring IoC Container*: Εξηγείται τι είναι το Spring IoC Container και οι τρόποι δημιουργίας κόκκων (beans).
    - *Spring Beans*: Περιγράφεται τι είναι τα Spring Beans και το εύρος τους.
    - *Dependency injection*: Περιγράφεται τι είναι η έγχυση εξαρτήσεων και πως πραγματοποιείται από το Spring IoC Container.

- *Annotations*: Περιέχει τα annotations του Spring Framework και του JPA που χρησιμοποιούνται.
- *Spring Boot*: Εξηγείται τι είναι το Spring Boot.
- *Spring MVC*: Αναλύονται το Spring MVC Framework και παραδείγματα χρήσης του.
- *Spring Security*: Περιέχει την ανάλυση του Spring Security Framework, το οποίο παρέχει τη δυνατότητα αυθεντικοποίησης και εξουσιοδότησης.
- *REST API*: Εξηγείται τι είναι το REST API.

2. **JPA:** Στη δεύτερη ενότητα περιγράφεται η διαδικασία της αντιστοίχισης πινάκων – κλάσεων με το JPA και πως λειτουργούν τα Spring Data JPA repositories.
    - *JPA*: Περιγράφονται το JPA, η διαδικασία αντιστοίχισης πινάκων – κλάσεων, η γλώσσα ερωτημάτων JPQL και το JPA Criteria API.
    - *Spring Data JPA*: Εξηγούνται το Spring Data JPA Framework και τα JPA repositories.

3. **Στήσιμο εφαρμογής:** Εδώ περιγράφεται η δημιουργία της Spring Boot εφαρμογής, του εξυπηρετητή στο Heroku, της βάσης δεδομένων και η διαδικασία ενσωμάτωσης της Angular εφαρμογής στην Spring Boot εφαρμογή.
    - *Αρχιτεκτονική εφαρμογής*: Περιγράφεται η αρχιτεκτονική της εφαρμογής.
    - *Spring Tool Suite*: Περιγράφει τα βήματα για την εγκατάσταση του Spring Tool Suite.
    - *Heroku*: Αναλύονται η δημιουργία της αρχικής μορφής της εφαρμογής και η εγκατάσταση της στο Heroku.
    - *Postgres*: Αναλύονται η δημιουργία της βάσης δεδομένων. PostgreSQL στο Heroku και η πραγματοποιήση απομακρυσμένης σύνδεσης, χρησιμοποιώντας την εφαρμογή pgAdmin4.
    - *Amazon S3*: Εξηγούνται η διαδικασία δημιουργίας λογαριασμού και κάδου στο Amazon S3 για την αποθήκευση των φωτογραφιών.
    - *Angular*: Εξηγούνται συνοπτικά η Angular και τα πιο σημαντικά στοιχεία της.

- *Δημιουργία Frontend*: Περιγράφεται η δημιουργία της Angular εφαρμογής και η διαδικασία ενσωμάτωσης της στην Spring Boot εφαρμογή.
- *Visual Studio Code*: Περιγράφει τα βήματα για την εγκατάσταση του Visual Studio Code.

4. **Κώδικας εφαρμογής:** Στη ενότητα αναλύεται ο κώδικας, ο οποίος αναπτύχθηκε για τη δημιουργία της εφαρμογής.
    - *Ρύθμιση βάσης δεδομένων και JPA*: Περιγράφει την κλάση JpaConfiguration, για τη σύνδεση με τη βάση δεδομένων.
    - *Ρύθμιση Amazon S3* : Περιέχει τη δημιουργία των μεταβλητών ρυθμίσεων στο Heroku και της κλάσης AmazonS3Configuration.
    - *Ανέβασμα εικόνων στο Amazon S3*: Περιέχει τις απαραίτητες κλάσεις για το ανέβασμα εικόνων στο Amazon S3.
    - *Δημιουργία πινάκων brands, categories, product_images, inventory*: Περιγράφει τη δημιουργία των πινάκων αυτών στη βάση και των αντίστοιχων οντοτήτων – κλάσεων.
    - *Δημιουργία repositories, services για τους πίνακες products, categories, brands, product_images, inventory*: Περιέχει τις κλάσεις για την πραγματοποίηση των CRUD λειτουργιών στη βάση.
    - *Δημιουργία controller, repository, service για την εισαγωγή προϊόντων* : Αναλύεται η κλάση ProductApiController, η οποία ανεβάζει τις εικόνες στο Amazon S3.
    - *Δημιουργία πινάκων deals, deal_images*: Περιγράφει τη δημιουργία των πινάκων αυτών στη βάση και των αντίστοιχων οντοτήτων – κλάσεων.
    - *Δημιουργία controller, repository, service για την ανάκτηση προσφορών*: Περιέχει τις κλάσεις για την πραγματοποίηση των CRUD λειτουργιών στη βάση.
    - *Angular – Routes*: Περιγράφει τη δημιουργία διαδρομών στην Angular.
    - *Angular - Components my-footer, shop-services, navigation-bar, product*: Περιγράφει τη δημιουργία των παραπάνω components.

- *Angular - index-page*: Αναλύεται η διαδικασία δημιουργίας του index-page component (αρχικής οθόνης).
- *Angular - product-page*: Αναλύεται η διαδικασία δημιουργίας του product-page component (οθόνη προϊόντος).
- *Angular - category-page*: Αναλύεται η διαδικασία δημιουργίας του category-page component (οθόνη κατηγορίας προϊόντος).
- *Angular - category-sidebar*: Αναλύεται η διαδικασία δημιουργίας του category-sidebar component (φίλτρα κατηγορίας προϊόντος).
- *Angular - ανανέωση προϊόντων κατηγορίας*: Περιέχεται το τμήμα του κώδικα για την ανανέωση των προϊόντων κατηγορίας.
- *Spring Security - Spring Session*: Δίνεται ο κώδικας της κλάσης SecurityConfiguration για την ενεργοποίηση του HTTP Basic Authentication.
- *Angular - register-page component*: Αναλύεται η διαδικασία δημιουργίας του register-page component (οθόνη εγγραφής).
- *Angular - login-page, navigation-bar components (είσοδος και αποσύνδεση)*: Αναλύεται η διαδικασία δημιουργίας του login-page component (οθόνη εισόδου).
- *Angular - cart-page component*: Αναλύεται η διαδικασία δημιουργίας του cart-page component (οθόνη καλαθιού).
- *Angular - checkout-page component*: Αναλύεται η διαδικασία δημιουργίας του cart-page component (οθόνη ολοκλήρωσης αγοράς).
- *Angular - order-page component*: Αναλύεται η διαδικασία δημιουργίας του order-page component (οθόνη εμφάνισης παραγγελίας).
- *Spring Email - αποστολή E-mail*: Δίνεται ο κώδικας για την αποστολή email.
- *Angular - search-page*: Αναλύεται η διαδικασία δημιουργίας του search-page component (οθόνη αναζήτησης προϊόντων).
- *Angular - search-sidebar*: Αναλύεται η διαδικασία δημιουργίας του search-sidebar component (φίλτρα αναζήτησης προϊόντων).

5. **Επεκτάσεις:** Στην τελευταία ενότητα – σελίδα δίνονται οι προτεινόμενες επεκτάσεις της εφαρμογής.
    - *Επεκτάσεις*: Δίνονται οι προτεινόμενες επεκτάσεις της εφαρμογής.

Ο πιο σύντομος τρόπος μελέτης του υλικού για την κατανόηση του κώδικα της εφαρμογής περιλαμβάνει τα εξής: τις σελίδες *Τί είναι το Spring*, *Spring IoC Container*, *Spring MVC* και *REST API* της πρώτης ενότητας, την δεύτερη ενότητα *JPA*, την σελίδα *Angular* της ενότητας *Στήσιμο εφαρμογής* και ολόκληρη την τέταρτη ενότητα *Κώδικας εφαρμογής*. Εναλλακτικά, εάν ο ενδιαφερόμενος θέλει να εμβαθύνει περισσότερο και να μελετήσει αναλυτικά την διαδικασία δημιουργίας της εφαρμογής, μπορεί να μελετήσει με τη σειρά τις ενότητες *Θεωρία Spring*, *JPA*, *Στήσιμο εφαρμογής* και *Κώδικας εφαρμογής*.

8 Συμπεράσματα και προτάσεις για μελλοντική επέκταση

8.1 Σύνοψη και συμπεράσματα

Σκοπός της μελέτης είναι η ανάπτυξη εκπαιδευτικού υλικού, για την εκμάθηση του Spring Framework. Αναπτύχθηκε μια διαδικτυακή εφαρμογή (ηλεκτρονικό κατάστημα), με τη χρήση του Spring Framework. Επίσης, δημιουργήθηκε ένας ιστότοπος, ο οποίος περιέχει τη θεωρία του Spring, του JPA και τη διαδικασία ανάπτυξης της εφαρμογής. Το εκπαιδευτικό υλικό του ιστότοπου μπορεί να αξιοποιηθεί από κάθε ενδιαφερόμενο, για την εκμάθηση της ανάπτυξης διαδικτυακών εφαρμογών σε Java με το Spring Framework.

Όσον αφορά την εφαρμογή, η διαδικασία της ανάπτυξης της εφαρμογής ήταν ευκολότερη και πιο γρήγορη με τη χρήση του Spring Boot και του Spring Data JPA. Χρησιμοποιώντας τα Spring Boot starters, το Spring Boot μας παρέχει τις κατάλληλες βιβλιοθήκες για τη δημιουργία του REStful API και εξασφαλίζει ότι οι εκδόσεις τους είναι συμβατές μεταξύ τους. Επίσης, για τη ρύθμιση της βάσης δεδομένων και της ασφάλειας της εφαρμογής, χρησιμοποιήθηκαν κώδικας Java και annotations και όχι πολύπλοκη XML, η οποία ήταν η μοναδική επιλογή σε παλιότερες εκδόσεις του Spring. Τέλος, η βιβλιοθήκη Spring Data JPA παρείχε τις υλοποιήσεις των μεθόδων, για την πραγματοποίηση των πιο συχνών λειτουργιών στη βάση δεδομένων, χωρίς να χρειαστεί να γράψουμε τον αντίστοιχο κώδικα.

8.2 Όρια και περιορισμοί της έρευνας

Το ηλεκτρονικό κατάστημα έχει εκπαιδευτικό χαρακτήρα και δεν δημιουργήθηκε, για να εξυπηρετεί εκατοντάδες χρήστες ταυτόχρονα. Επίσης, η εργασία δεν ασχολήθηκε με το κομμάτι της ασφάλειας των συναλλαγών, το οποίο είναι κρίσιμο για ένα πραγματικό ηλεκτρονικό κατάστημα, που πραγματοποιούνται πραγματικές συναλλαγές και πληρωμές.

Όσον αφορά τον εκπαιδευτικό ιστότοπο, το περιεχόμενο του καλύπτει τα τμήματα (modules) του Spring Framework, που ανήκουν στις ομάδες Spring Core Container, Data Access/Integration και Web.

8.3 Προτάσεις για μελλοντική επέκταση

Ο ιστότοπος θα μπορούσε, να επεκταθεί, προσθέτοντας ερωτηματολόγια ελέγχου κατανόησης του Spring Framework και ασκήσεις για την προσθήκη λειτουργιών στην εφαρμογή.

Στην διαδικτυακή εφαρμογή θα μπορούσαν, να γίνουν τα εξής:

- Προσθήκη λειτουργίας αξιολόγησης των προϊόντων και εμφάνιση της βαθμολογίας και των αξιολογήσεων στη σελίδα κάθε προϊόντος.
- Προσθήκη λειτουργίας, ώστε ο χρήστης να μπορεί να δει την κατάσταση της παραγγελίας του και τις παραγγελίες, τις οποίες έχει πραγματοποιήσει.
- Προσθήκη φόρμας επικοινωνίας, ώστε να μπορούν οι χρήστες να πραγματοποιούν ερωτήσεις σχετικά με τις παραγγελίες τους ή για τυχόν άλλες απορίες τους
- Προσθήκη διαχειριστικού πίνακα, ώστε ο διαχειριστής του καταστήματος να μπορεί, να προσθέτει προϊόντα και να βλέπει τις παραγγελίες, που έχουν πραγματοποιηθεί.

9 Βιβλιογραφία

[1] Wikipedia, *Java (programming language)* [Online] Available from : https://en.wikipedia.org/wiki/Java_(programming_language) [Accessed 30th Jan 2019]

[2] Wikipedia, *Java Platform, Enterprise Edition* [Online] Available from : https://en.wikipedia.org/wiki/Java_(programming_language) [Accessed 30th Jan 2019]

[3] Antonio Goncalves, (2013). *Beginning Java EE 7*, Apress

[4] Wikipedia, *Software Framework* [Online] Available from : https://en.wikipedia.org/wiki/Software_framework [Accessed 30th Jan 2019]

[5] tutorialspoint, *Spring Framework - Overview* [Online] Available from : https://www.tutorialspoint.com/spring/spring_overview.htm [Accessed 30th Jan 2019]

[6] Craig Walls, (2015). *Spring in Action*, Manning

[7] tutorialspoint, *JSF - Overview* [Online] Available from : https://www.tutorialspoint.com/jsf/jsf_overview.htm [Accessed 30th Jan 2019]

[8] Wikipedia, *JavaServer Faces* [Online] Available from : https://en.wikipedia.org/wiki/JavaServer_Faces [Accessed 30th Jan 2019]

[9] Niriksha B. K. , Sharmila S., (2015). *Rich Internet Web Application Development using Google Web Toolkit*

[10] Wikipedia, *Google Web Toolkit* [Online] Available from : https://en.wikipedia.org/wiki/Google_Web_Toolkit [Accessed 30th Jan 2019]

[11] tutorialspoint, *GWT - Overview* [Online] Available from : https://www.tutorialspoint.com/gwt/gwt_overview.htm [Accessed 30th Jan 2019]

[12] techopedia, *Struts Framework* [Online] Available from : https://www.techopedia.com/definition/3984/struts-framework [Accessed 30th Jan 2019]

[13] Baeldung, *A Quick Struts 2 Intro* [Online] Available from : https://www.baeldung.com/struts-2-intro [Accessed 30th Jan 2019]

[14] hotframeworks, *Java* [Online] Available from : https://hotframeworks.com/languages/java [Accessed 30th Jan 2019]

[15] Tony C Shan , Winnie W Hua (2006). *Taxonomy of Java Web Application Frameworks*

[16] Wikipedia, *Rich web application* [Online] Available from : https://en.wikipedia.org/wiki/Rich_web_application [Accessed 30th Jan 2019]

[17] cjoint, *The curious coder's java web frameworks comparison* [Online] Available from : https://www.cjoint.com/15mi/EEvlkExkV1t_comparaison_framework.pdf [Accessed 30th Jan 2019]

[18] Spring Io, *Introduction to Spring Framework* [Online] Available from : https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html [Accessed 30th Jan 2019]

[19] Spring Io, *Container Overview* [Online] Available from : https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-basics [Accessed 30th Jan 2019]

[20] tutorialspoint, *Spring Boot - Introduction* [Online] Available from : https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm [Accessed 30th Jan 2019]

[21] Wikibooks, *Java Persistence/What is JPA?* [Online] Available from : https://en.wikibooks.org/wiki/Java_Persistence/What_is_JPA%3F [Accessed 30th Jan 2019]

[22] Spring Io, *Spring Data JPA - Reference Documentation* [Online] Available from : https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ [Accessed 30th Jan 2019]

[23] PostgreSQL, *HTML* [Online] Available from : https://www.postgresql.org/about/ [Accessed 30th Jan 2019]

[24] Wikipedia, *What is PostgreSQL?* [Online] Available from : https://en.wikipedia.org/wiki/HTML [Accessed 30th Jan 2019]

[25] Wikipedia, *Cascading Style Sheets* [Online] Available from : https://en.wikipedia.org/wiki/Cascading_Style_Sheets [Accessed 30th Jan 2019]

[26] tutorialspoint, *What is CSS?* [Online] Available from : https://www.tutorialspoint.com/css/what_is_css.htm [Accessed 30th Jan 2019]

[27] Wikipedia, *Javascript*[Online] Available from : https://en.wikipedia.org/wiki/JavaScript [Accessed 30th Jan 2019]

[28] MDN web docs, *JavaScript* [Online] Available from : https://developer.mozilla.org/en-US/docs/Web/JavaScript [Accessed 30th Jan 2019]

[29] tutorialspoint, *TypeScript - Overview* [Online] Available from : https://www.tutorialspoint.com/typescript/typescript_overview.htm [Accessed 30th Jan 2019]

[30] Wikipedia, *TypeScript* [Online] Available from : https://en.wikipedia.org/wiki/TypeScript [Accessed 30th Jan 2019]

[31] Angular, *Architecture overview* [Online] Available from : https://angular.io/guide/architecture [Accessed 30th Jan 2019]

[32] Angular, *Introduction to components* [Online] Available from : https://angular.io/guide/architecture-components [Accessed 30th Jan 2019]

[33] Wikipedia, *Heroku* [Online] Available from : https://en.wikipedia.org/wiki/Heroku [Accessed 30th Jan 2019]

[34] Heroku, *Heroku Dynos* [Online] Available from : https://www.heroku.com/dynos [Accessed 30th Jan 2019]

[35] Heroku, *Dynos and the Dyno Manager* [Online] Available from : https://devcenter.heroku.com/articles/dynos#ephemeral-filesystem [Accessed 30th Jan 2019]

[36] Wikipedia, *Amazon S3* [Online] Available from : https://en.wikipedia.org/wiki/Amazon_S3 [Accessed 30th Jan 2019]

[37] OpenLearn, *Client–server architecture* [Online] Available from : https://www.open.edu/openlearn/science-maths-technology/introduction-web-applications-architecture/content-section-1.1# [Accessed 30th Jan 2019]

[38] grokonez, *How to use Spring JPA with PostgreSQL | Spring Boot* [Online] Available from : https://grokonez.com/spring-framework/use-spring-jpa-postgresql-spring-boot [Accessed 30th Jan 2019]

[39] Wikipedia, *Multitier architecture* [Online] Available from : https://en.wikipedia.org/wiki/Multitier_architecture [Accessed 30th Jan 2019]

[40] Spring Framework Documentation - Web on Servlet Stack, *DispatcherServlet* [Online] Available from : https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc-servlet [Accessed 30th Jan 2019]

[41] tutorialspoint, *Spring - MVC Framework* [Online] Available from : https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm [Accessed 30th Jan 2019]

[42] Wikipedia, *Representational state transfer* [Online] Available from : https://en.wikipedia.org/wiki/Representational_state_transfer [Accessed 30th Jan 2019]

[43] Fielding R., (2000). *Architectural Styles and the Design of Network-based Software Architectures* [Online] Available from : https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf [Accessed 30th Jan 2019]

[44] codeacademy, *What is REST?* [Online] Available from : https://www.codecademy.com/articles/what-is-rest [Accessed 30th Jan 2019]