MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

AG COMPUTERS – AN ONLINE COMPUTER STORE

LICENSE THESIS

Graduate: **Andrei-Ioan GIURGIUMAN**

Supervisor: **Asst. eng. Cosmina IVAN**

2018

Table of Contents

Chapter 1. Introduction

1.1. Project context

Following the release of World Wide Web[1] for public use in 1991 the Internet's popularity has hugely grown changing how information is exchanged around the world.

Since 1994 when Pizza Hut became the first major business to offer online purchasing and until today, Internet affected the way business is conducted in the modern society.

Today we live in a digital world and according to Wikipedia's article on "Global Internet usage"[2] 51% of the world's population has access to the Internet. Every business, big or small, which sells products or services wants to take advantage of that and moves its activity online too, this is now known as E-commerce[3].

The computers business, PCs, laptops, notebooks and so on is a continuously growing business as these days they are used almost in every line of work. The solution to move this business (like any other business that sells products) in the online medium was to have built web applications resembling the experience that a client would have in an actual store, these are usually called online stores.

The business owners can expand their business easier online by building an online store than having to build physical stores. Also selling products through an online store means less personnel is required, maintenance is cheaper and the target audience becomes larger as there are theoretically no geographical limitations.

The clients can buy products from the comfort of their homes by accessing the online stores, the only requirements being: a computer, a web browser and of course Internet connection and a credit card, but some online shops allow also for payment on delivery. By enabling customers to buy from home, travel time and cost are eliminated which is very important in the modern society because people have less time to spare on activities like shopping, they can buy products whenever they want as online stores are open 24/7, the pollution is reduced as less customers have to drive their cars to the physical store and the customers have access to a lot more information about the product.

According to a study [1] made in Romania at the "Academia de Studii Economice din Bucuresti", even though Romania is the last in the European rankings at revenues gained from E-commerce, in 2017 there were 7,36 million online shoppers which made on average 8.7 acquisitions each. The study shows that online E-commerce has a positive impact on the state's budget, it advantages small businesses and, also leads to a change in the business model of delivery companies to accommodate the growing demands resulted from more products sold online. For the customers, the main benefits are time economy and product diversity compared to traditional forms of commerce and, also price transparency and ease of comparison between products. The study also shows that E-commerce contributes to a durable development of the Romanian society and has a positive impact on the environment: there were saved approximately 400.000 trees in Romania between 2014 and 2017 as an effect of reduced carbon monoxide emissions due to customers shopping online as opposed to customers driving in order to shop in a physical store. The study concludes

[1] https://en.wikipedia.org/wiki/World_Wide_Web
[2] https://en.wikipedia.org/wiki/Global_Internet_usage
[3] https://en.wikipedia.org/wiki/E-commerce

that E-commerce is a step forward in the optimizing the process of commercializing services and products.

The proposed application is a web store for a small business that sells computers to customers and wants to expand its activity online, this business falls under the Business-to-Customer[4] category of E-commerce. The application offers an easy and convenient way for owners to sell their products and for customers to browse and buy the products using their computers wherever they are. The application offers a secure way for customers to pay for their orders, it enables the store to manage products, special offers, keep track of suppliers, orders and registered users.

The stakeholders for this application are: the business owner that wants to have a working application resulting in more customers and increased revenue from online sales, the administrator which is an employee of the store who wants a working application allowing him/her to manage products, suppliers, view orders and registered users, registered users who want an easy way of ordering computers online avoiding the wasted time and effort associated with driving to a physical store and visitors who want to browse through computers, see their specifications and compare prices with other similar stores.

Similar applications exist, each tailored for the specific needs of each business. Personally, I have encountered poorly built websites that made the shopping process very ambiguous as I didn't know how to proceed next so I could complete my order. The proposed application must allow for a less ambiguous shopping experience by building an intuitive checkout process, informing users about the state they are in at each step and offering the possibility to get quick answers to questions related to store services by chatting with a FAQ bot.

1.2. Motivation

As discussed before E-commerce is a continuously growing business not only in Romania but worldwide, with revenues resulted from selling products and services online increasing each year as people can order everything online from their devices (smartphones, tablets, laptops, PCs and so on). There is also a fierce competition in this domain which results in evolution of such systems, competitors beginning to personalize how customers shop online by making use of Artificial Intelligence, Augmented Reality and Virtual Reality. A small business can't afford to bring such technologies to the table from the start.

The motivation for this project is to build a basic, easy-to-use, secure single-page web application serving as a store front which incorporates a simple chatbot which can be later upgraded to a personal shopping assistant, because such applications are very popular and in high-demand these days. A store front can help a small business grow by easier promoting its products, lifting geographical barriers and helping the environment by limiting the number of customers driving to the physical location which sells the products.

Besides the arguments mentioned in the paragraph above there is a personal motivation to make use and get a better understanding of web technologies learned while studying distributed systems, databases, user interface design and learn new ones, in order to build a single-page web application meeting all the requirements for a modern web application and to use this application as a starting point for learning to build better such applications and gain more knowledge in this domain and also in the

[4] https://virtocommerce.com/glossary/what-is-b2c-ecommerce

chatbot domain which is evolving very fast being adopted by more online businesses every day.

1.3. Project content

Chapter 1. Introduction – This chapter presents the project's context and motivation.

Chapter 2. Project Objectives – This chapter describes the main objective of the project and other secondary objectives on which the application's implementation depends.

Chapter 3. Bibliographic Research – In this chapter the concepts regarding E-commerce and E-business, chatbots and single page web applications are presented as well as a comparison between the proposed application and similar applications.

Chapter 4. Analysis and Theoretical Foundation – This chapter presents a conceptual architecture of the application, the technologies used for developing the proposed application and the reasons they were chosen, it also presents the functional requirements, non-functional requirements and use cases of the application.

Chapter 5. Detailed Design and Implementation – The details regarding the design and implementation of the proposed project are presented here. Diagrams (general architecture of the application diagram, client application architecture diagram, server application architecture diagram, database diagram, package diagram, class diagrams, database diagram, deployment diagram) and explanations about various parts of the implementation are provided here.

Chapter 6. Testing and Validation – In this chapter are described the tests performed on the application and the obtained results.

Chapter 7. User's manual – A guide for installing and using the application is presented here.

Chapter 8. Conclusions – A summary of what was achieved during the implementation of the proposed application and a description of how the application can be improved in the future are presented in this chapter.

Chapter 2. Project Objectives

In this chapter the main objective is presented along with the secondary objectives which need to be completed in order to fulfill the main objective.

2.1. Main objective

The proposed project's main objective is to design and implement a single-page web application serving as a store front to sell computers, which enables customers to easily shop online for computers, required being just a web browser, an Internet connection and a registered account. The application must allow for users to receive answers to questions regarding the services provided by the store in a human-like interactive way by using a simple bot. The web application also enables the store front administrator to keep track of registered users, placed orders, suppliers and product stock. Customer personal data must be secured after registration and the administrator is only allowed to see data associated with a placed order like name, username, e-mail, billing address, delivery address and the products ordered, the administrator is not allowed to see data like credit card details or the password associated to a customer's account. The registered customer is forbidden from accessing functionalities the administrator can perform and is also forbidden from accessing other customers data.

2.2. Secondary objectives

The secondary objectives are very important as they all must be achieved in order for the application to be complete and fulfill the main objective presented in the paragraph above.

The secondary objectives are the following:

- The application must allow visitors (unregistered customers) access to basic features like browsing through all the products available in order to see special offers or product details and chat with the frequently asked questions bot in order to receive answers to the questions regarding the services offered by the store.
- Account registration: the application must offer the visitors the possibility to create an account in order to gain full access to the application features a registered customer can use; the role of customer will be assigned to such an account. An administrator account is unique; it is pre-registered in the database having the role administrator assigned to it.
- Authentication and authorization: registered users must be able to authenticate (sign in) using the e-mail and password provided during registration, if these credentials are valid then the process of authorization follows which means the application gives users access to features and resources within the application based on the roles associated with the registered account (administrator, customer). The administrator account has full access to all the application features, customer features included, while the customer doesn't have access to the administrator features.

- E-mail notification: registered users must be automatically notified via e-mail after they register a new account, following a completed order, or after canceling an order (confirmation e-mail).
- Search products: users must be able to search products by typing in key words (model, brand), or by filtering products according to the existing categories (e.g. laptop, notebook)
- Logout: authenticated users must be able to log out of their personal account; at that point they become visitors, meaning they lose access rights associated with their account until logging in again.
- Product wish list: authenticated users must be able to add products they want to buy later, compare or show to friends to a wish list. The wish list associated with each users account and can be accessed at any time to view a list of the products inside, view each product's details or remove products from list.
- Review a product: any authenticated user can review a product. Reviewing a product means giving a rating to that product (from1 being lowest to 5 being highest) and leaving a comment based on the experience they had using the product. Authenticated users can delete only their reviews. The reviews must be displayed below each product so any user, visitor or registered is able to see them.
- Personal account: a registered and authenticated user must be able to edit personal information associated with their personal account such as: e-mail, address, phone number, name and must be able to reset their password if they forgot it.
- Shopping cart: the authenticated users must be able to place products they want to buy in a virtual shopping cart. When the users finished placing products in the shopping cart they must be able to see all products they placed in the shopping cart and be able to modify the quantity of each product, the application must always compute and display a total amount to be paid.
- Place order: an authenticated user must be able to order the items placed in the shopping cart. Placing an order means the user is asked to provide the application with data necessary for the delivery of the products (e.g. name, e-mail, phone number, billing address, delivery address) and then proceed to paying the products by filling in the credit card details and perform an electronic payment or by choosing to pay cash on delivery.
- Electronic payment: the authenticated users must be able to pay for their products using a credit card. The payment must be completed securely by using a payment service which checks the validity of the card and ensures credit card information privacy during this procedure.
- Cancel order: an authenticated user that chose "pay cash on delivery" as a payment method must be allowed to cancel an order during next 24 hours since the order was made.
- Administrator dashboard: the administrator must be able to view a list of all the orders that were placed by each user, a list of all users with their name, username, e-mail and phone numbers in case they need to be contacted, CRUD on products in store, CRUD on suppliers of the products, CRUD on special offers. All these features must be available

within the administrator dashboard and the access must be forbidden for visitors or other registered users (customers).

Other secondary objectives that must be met are related to mostly to the quality issues regarding the application such as scalability, security and usability. The scalability must be given mostly by the applications imposed architecture. Security must be ensured by limiting access to resources and features of the application based on credentials and roles associated with a user's account. Usability must come from how intuitive the application is and how easy it is for a user to learn to use it, resulting in a pleasant shopping experience and lesser shopping cart abandonments.

Chapter 3. Bibliographic Research

In this chapter are presented a series of relevant concepts regarding the domain where the current project will be used which form a base for designing and implementing the proposed application. More precisely the following topics are discussed: single-page web applications, E-commerce and E-business, similar applications that were studied in order to better understand what is required for an E-commerce application and a comparison between the studied applications and the proposed application.

3.1. E-business and E-commerce

The proposed project is a web store created with the purpose of selling computers to a larger customer base by making use of the Internet to reduce the inconveniences that a physical store imposes, like geographical barriers, time table, waiting in queues, comparing prices with other stores, finding information about a certain product. The before mentioned inconveniences require for the customer to drive to the physical store, compare prices with other stores he/she drove to before, asking someone in the staff more detailed information about certain products and then after deciding which product to buy, waiting in a queue along with other customers to purchase the chosen products. An online store helps customers get past these inconveniences. The proposed project falls under the B2C (Business to Consumer) type of E-commerce, E-commerce itself being a type of E-business. Next the concepts of E-business and E-commerce will be presented using information found on "wikibooks"[5] and "semanticscholar"[6].

E-commerce is defined in many ways, a few definitions will be presented for a better understanding of the concept[5] :

- Electronic commerce or E-commerce refers to a wide range of online business activities for products and services, it also refers to the many forms of transactions in which the sides involved are interacting electronically and not physically.
- The term E-commerce is mostly associated with the actions of buying and selling something on the Internet (which is the case here), or with transactions that imply the transfer of ownership or rights to use goods or services through a computer mediated network.
- The following is a more complete definition of E-commerce[6]: "E-commerce is the use of electronic communications and digital information processing technology in business transactions to create, transform and redefine relationships for value creation between or among organizations, and between organizations and individuals."

There are three primary processes in E-business: production processes (these processes include restocking, processing of payments, electronic links to suppliers for example), customer-focused processes (like promotional or marketing processes, selling over the internet, customers support etc.) and internal management processes (these include employee services, training, internal information-sharing, video-conferencing and recruiting).

[5]https://en.wikibooks.org/wiki/Introduction_to_Computer_Information_Systems/E-Commerce
[6] https://pdfs.semanticscholar.org/173c/819ce3fccafdc4f64af71fd6868e815580ad.pdf

We can see more clearly the differences between E-commerce and E-business in Figure 3.1, E-business refers more to the micro-environment of a company where E-commerce refers to micro-environment.

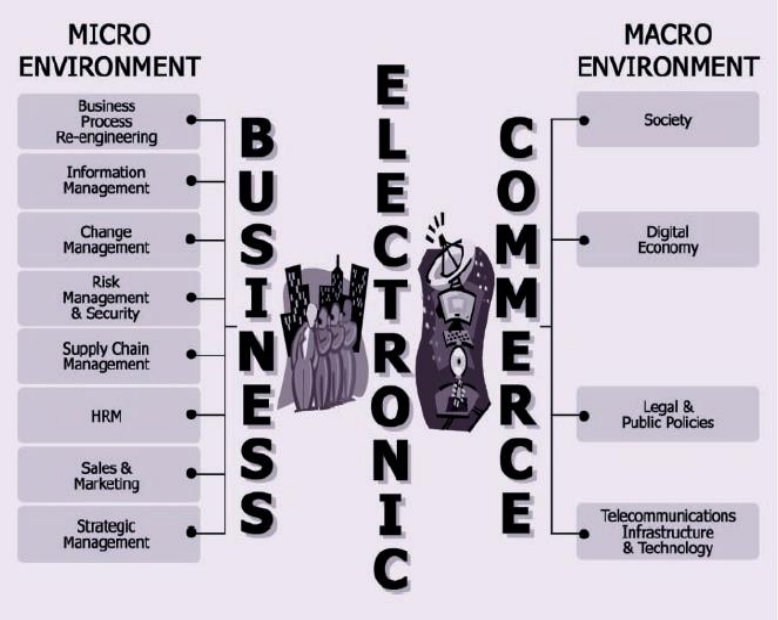


Figure 3.1 Differences between E-commerce and E-business (source: https://pdfs.semanticscholar.org)

Next the benefits and limitations of E-commerce will be presented as reminded in the "What are the benefits of E-commerce?" and "What about the limitations of E-commerce?" sections[7]. There are three major stakeholders affected by the benefits of E-commerce: organizations, consumers and society.

Benefits of E-commerce to organizations:

- International marketplace: when there was no Internet and no possibility of selling online, a business would have to build a physical store with the only possibility of expanding by building more such stores in different regions to increase the number of customers. Today a business can expand nationally or internationally using the Internet and the web making their products and services available to people all around the country or all around the globe, maximizing profit.
- Operational cost savings: the cost of creating, processing, distributing, sorting and retrieving paper-based information is drastically decreased.
- Enables reduced inventories: for large technology companies (e.g. Motorola) it would be very hard to keep components on stock with no real demand because they would rapidly become obsolete, so the benefit is that they can collect orders online and then deliver products through just-in-time manufacturing, meaning that the ordered product is assembled based on the specifications chosen by the customers and delivered within few days.
- Digitalization of products: software and music/video products can be easily delivered through Internet reducing a lot of costs.
- No 24-hour-time constraints: businesses can be contacted by customers or contact suppliers at any time.

Benefits of Ecommerce to consumers:

- 24/7 access: enables the customers to buy or make all kinds of transactions 24 hours a day, 7 days a week.
- More choices: customers have a bigger range of products and prices to choose from and even international suppliers.
- Price comparisons: customers can buy products from all around the world and compare prices to find the best one, either by visiting each website or by visiting a website that aggregates prices (e.g. Compari.ro).
- Improved delivery process: Digitalized products such as music or e-books can be immediately downloaded via Internet and other types of products can be tracked until their arrival.

Benefits of E-commerce to society:

- Benefits of E-commerce to society:
- Facilitates delivery of public services: health services are available over the internet making it easier for patients to receive professional assistance.

E-commerce also have several limitations, despite all the good benefits.

E-commerce limitations to organizations:

- E-commerce limitations to organizations:
- Businesses are always under pressure to innovate because business models can be easily copied via the Internet so it is hard to maintain a long-term advantage in face of competition.
- Problems with compatibility of old and new technologies. These problems arise when there is incompatibility between old business systems and new web and Internet infrastructure, resulting in big costs for developing new systems.

Limitations of E-commerce to consumers:

- Computing equipment is needed for customers who want to participate in the E-commerce business, meaning that there is an initial cost involved.
- Basic technical knowledge of operating a computer and using the Internet and the World Wide Web.
- Cost of access to the Internet: not just the initial cost of buying the computer but also staying up to date with the technology to make sure that is compatible with the changing requirements of Internet and applications.
- Lack of security and privacy of personal data: there doesn't exist a real control of data that are collected over the Internet and World Wide Web, because the laws on protection of user data differ from country to country.
- Lack of trust: people are interacting with a computer instead of a human being.

Limitations of E-commerce to society:

- Social division between those who have the required technical knowledge to work on this domain and those who don't.

There are several types of E-commerce, the most important are the following: B2B (Business to Business), B2G (Business to Government), C2C (Consumer to Consumer) and mobile commerce (m-commerce).

Since the current application falls under the B2C E-commerce, a definition of B2C will follow along with some advantages compared to buying products from physical stores.

Business to Consumer E-commerce is a type of E-commerce that involves companies selling products to customers over the internet. The products that are sold can be physical (books, computers, etc.) or information products (software, e-books etc.). B2C E-commerce reduces the search cost for consumers by giving them easier access to information and thus enabling them to find the best price for a product or service. For the business side it is more convenient cost-wise to put up and maintain a web site than build a physical store and when it comes to selling digitalized content (e.g. e-books) it is very convenient to deliver everything via the Internet cutting any costs that are delivery related.

Professors of Information Systems, Kalakota and Whinston first developed a framework that identifies the different business and technology components that make up E-commerce. They explain how different components fit with each other by using as an analogy the architecture of a building as seen in Figure 3.2.

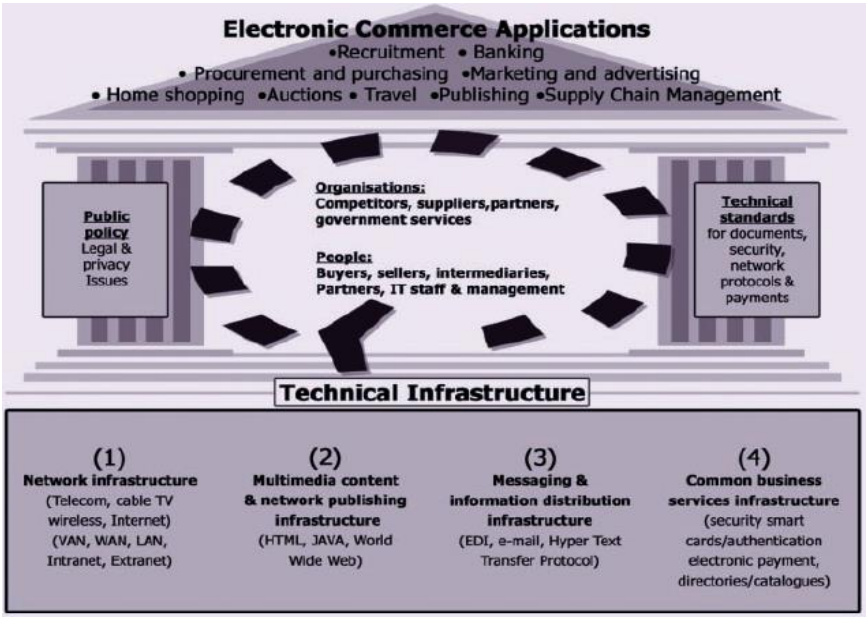


Figure 3.2 A framework for E-commerce

(source: https://pdfs.semanticscholar.org)

The network infrastructure takes a lot of forms like telephone wires, cables, wireless technology and is used to facilitate transport of information.

The publishing infrastructure transports electronic data and multimedia along the network. Multimedia content is created using tools like HTML or Java. For example, there is text only content, which is simple to transmit and more complex content like a computer game that requires more resources to be transmitted.

Messaging and information distribution infrastructure: after the multimedia content is created, there must be a way of sending and retrieving this information (e.g. E-mail, HTTP).

Once the content and data can be created, displayed and transmitted, business services are required for buying, selling or doing other kinds of transactions safely and reliably (e.g. smart cards, authentication, electronic payment etc.)

3.2. JavaScript single-page web applications

As the authors Michael S. Mikowski and Josh C. Powell explain in their book [2] about single page web applications, the cost of producing traditional websites is very high and this can have bad effects on a business. A slow website can make customers leave the website for a rival one costing the business a lot of money.

They give the following example on why traditional websites are slow, in the beginning of Part 1, "Introducing SPAs" of the book [2]:

"One reason traditional websites are slow is because popular MVC server frameworks are focused on serving page after page of static content to an essentially dumb client. When we click a link in a traditional website slideshow, for example, the screen flashes white and everything reloads over several seconds: the navigation, ads, headlines, text, and footer are all rendered again. Yet the only thing that changes is the slideshow image and perhaps the description text. Worse, there's no indicator when some element of the page becomes functional. For example, sometimes a link can be clicked as soon as it appears on a web page; other times we have to wait until the redrawing is 100% complete plus five seconds. This slow, inconsistent, and clunky experience is becoming unacceptable for an increasingly sophisticated web consumer."

Following is a definition for single-page application taken from Wikipedia:
"A single-page application (SPA) is a web application or website that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from the server. The approach avoids interruption of the user experience between successive pages, making the application behave more la a desktop application."[7]

This is very convenient for the user and makes the application respond faster as only the required resources are loaded as an effect of the actions performed by the user, this results in a less confusing and annoying experience for the user. On top of that, single-page web applications using JavaScript as the client language are faster due to the fact that they execute all client code on the user's machine, reliving the server from this task.

The advantages JavaScript single-page applications have over Java or Flash single-page applications are numerous and they contributed to its increasing popularity and to the reason such an application is used for the current project.

Some advantages of using JavaScript for building single-page applications as stated in the above-mentioned book [2], Part 1, subchapter "1.1.2 What took JavaScript SPAs so long" are the following: no plugins required, one client language, a more fluid and interactive page. The web browser is the most used application in the world and access to a JavaScript application is just one click away. JavaScript in the browser is one of the world's most widely distributed execution environments. Deployment of a JavaScript application is trivial compared to applications written in other languages. JavaScript is useful for cross-platform development, meaning that single-page applications can be created using different operating systems and can be deployed not only to desktops and laptops but also to smartphones and tablets.

Although the proposed project uses Java as the server-side language, it still benefits from these advantages because the client-side application is written in JavaScript and all its code executes on the client machine.

[7] https://en.wikipedia.org/wiki/Single-page_application

3.3. Chatbots in E-commerce

As the Artificial Intelligence (AI) evolved and became more accessible to the large public, many businesses small or big started to make use of this in order to better promote themselves and to offer a new and satisfying user experience. An example of how AI and NLP (Natural Language Processing) is used to promote an online and make a user's experience as interactive as possible is through chatbots. A chatbot is a kind of computer program that simulates a conversation with a human being. It can be seen as an assistant that communicates with users through text message and can be integrated into websites, applications and instant messengers.

Chatbots are used on E-commerce websites because they can speed up productivity and automate certain tasks, for example answering certain questions about the products sold or services delivered. The chatbot is always available in theory and can respond to the users as opposed to a live operator that has a working schedule and is available only during working hours. So, chatbots provide assistance or access to information quickly and efficiently, they amuse customers by giving funny tips or chatting with users using the small talk feature many of them have, they also attract the curiosity of customers.

Chatbots are used by big companies such as eBay or PizzaHut to give users a personalized experience while shopping or ordering food. The eBay shop bot can help you choose a gift for someone's birthday for example by completing a quiz. A simple FAQ bot can respond immediately to a user's question, so the users doesn't have to scroll through a list of frequently asked questions until he/she finds the right answer. There are also a lot of platforms available now that enable us to rapidly create and integrate a custom bot that fits our business best, like Dialogflow[8] or Chatfuel[9].

In conclusion a chatbot is a must have addition to any E-commerce business, no matter what products or services it sells.

3.4. Similar applications

This subchapter presents similar applications studied in order to better understand E-commerce applications, they also helped pin down some essential requirements such an application must meet. The main criteria for the chosen applications was to have a shopping cart functionality and an integrated electronic payment service.

The proposed application falls within the e-commerce applications category, more specifically computer e-commerce which is very popular nowadays given the fact that almost everybody has the need to buy a computer. The proposed application covers the business to client category of e-commerce, registered users can buy products from the store, there isn't the possibility to register as a corporation. There are a lot of such applications, basically they are online stores or webstores which sell computers.

3.4.1. Secondhand E-bookshop

This application developed by Talla Ngala Yiga for the bachelor's thesis is an electronic secondhand bookshop where users can sell and buy books. The specifications for the application as enumerated in the documentation [3], chapter "Requirements/Analysis" are the following: complete online shopping process should

[8] https://en.wikipedia.org/wiki/Dialogflow
[9] https://chatfuel.com/about-us.html

be developed, anonymous and registered user can buy books, the website should allow visitors to create, manage and log into their accounts, members can sell books they uploaded, they can delete a book if there is no current buy process going on that book, members can create and update their profiles, comments on buyers and seller books after a sale process, History of transactions of a member, administrator has a simple order management system, capability of handling financial transactions(Use another Provider API e.g. PayPal), mailing capabilities after selling or buying process, search for books on the website, a visitor should rapidly find a book he or she is looking for, latest books are shown on the main page, handle errors in a proper way.

This application has very similar functionality to the proposed application the only notable difference being that the secondhand E-bookshop allows for registered users to be sellers as well not only buyers.

3.4.2. Web shop system

This application developed by Shen Yeyin, for the bachelor's thesis [4], represents a more primitive approach to build a web shop in comparison with the Secondhand E-bookshop, it has the main functionalities like user registration and authentication, shopping cart and checkout process but it fails to integrate an electronic payment service.

The application has the following features: registration, login, product search, place order, track order, shopping cart, administrators can edit orders, edit member's accounts, logout.

3.4.3. Quickmobile.ro

Quickmobile.ro[10] is a pretty big Romanian online store. Quickmobile.ro sells smartphones, laptops, tablets, gadgets accessories and everything else that has the word "smart" in it. They say to be the first ones to bring to Romania all the latest gadgets. Quickmobile.ro allows both registered users and visitors to view all the products in store, view their details, reviews and add products to shopping cart. Because no authentication is required to add products to the shopping cart, customers can visit the site, add the products they like to the shopping cart, complete delivery details and upon completing the order the site creates an account for them so in addition they will be able to authenticate, have a profile where they can see their orders, add products to a wish list, cancel the order, write a review and rate a product. The customers can create an account in advance too, so they can use the additional functionalities from the start. On the landing page of this website there are always displayed the newest products in store and products on a sale for example. Registered users or visitors can search a product by applying filters or by typing in a keyword. Users are informed via a confirmation e-mail that their account has been created or that their order was placed. Users have four payments methods to choose from: debit card (Visa, MasterCard, Maestro), cash on delivery, payment order, credit card. Users can choose the delivery method too, meaning the courier company that delivers the products.

[10] https://www.quickmobile.ro/

3.4.4. Bobjohnson.com

Bobjohnson.com[11] is a small online store for a family business. Bobjohnson.com sells refurbished, rugged laptops and tables. This online store is much smaller than Quickmobile.ro, and the range of product categories too. Just like Quickmobile.ro, BobJohnson.com allows visitors as well as registered users to view product, their details, reviews and alongside reviews, questions other users asked about a particular product. A user that is not authenticated can add products to the shopping cart and an account is created upon completion of the order. What I consider a disadvantage is that no matter if the user is authenticated or not if he/she wants to write a review, he/she will have to provide again a name and an e-mail address, Quickmobile.ro doesn't allow for unregistered users to leave a review. The landing page is much simpler than that of Quickmobile.ro, having displayed only some featured products which might be on sale or not and some reviews made by the customers of the store. Users may pay for an order via PayPal, American Express, Discover and VISA. Bobjohnson.com also offers a "Watch this product" feature and they will send the user an email when the price on the "watched" product changes. This website doesn't offer the possibility to add items to a wish list. This website doesn't offer the possibility to create a corporate account, or to create an account using Google+/Facebook. It is a smaller website than Qucikmobile.ro with a less cluttered design, simple and it does the job it's meant to do.

3.5. Comparison

The table below (Table 3.1) presents a comparison between the similar applications described above and the proposed application. Studying these similar applications contributed to specifying the functional requirements for the proposed application.

Table 3.1 Comparison between similar systems

Functionality	Quickmobile.ro	Bobjohnson.com	Secondhand E-bookshop	Web shop system	AG Computers
Web application	YES	YES	YES	YES	YES
Account registration	YES	YES	YES	YES	YES
Google+/Facebook authentication	YES	NO	NO	NO	NO
Confirmation e-mail upon registration	YES	YES	NO	NO	YES
View products without authenticating	YES	YES	YES	YES	YES
Search for products by keyword or by applying different filters without authenticating	YES	YES	YES	YES	YES
Place items in the shopping cart without authenticating	YES	YES	YES	NO	NO
Create account for user at checkout	YES	YES	NO	NO	NO
Order products	YES	YES	YES	YES	YES

[11] https://www.bobjohnson.com/

Add products to wish list	YES	NO	NO	NO	YES
Modify shopping cart details	YES	YES	YES	YES	YES
Review and rate a product	YES	YES	YES	NO	YES
Modify personal account data	YES	YES	YES	YES	YES
Display newest items and products on a special offer	YES	NO	YES	YES	YES
Possibility to cancel an order	YES	YES	NO	NO	YES
Reset password	YES	YES	YES	NO	YES
E-mail confirmation upon completing an order	YES	YES	YES	NO	YES
View personal orders	YES	YES	NO	NO	YES
View all orders (Administrator)	UNK	UNK	YES	YES	YES
View, update, add suppliers(Administrator)	UNK	UNK	NO	NO	YES
View all registered accounts(Administrator)	UNK	UNK	YES	YES	YES
View, update, add special offers(Administrator)	UNK	UNK	NO	NO	YES

Chapter 4. Analysis and Theoretical Foundation

In this chapter the conceptual architecture of the application followed by a short description will be presented along with the technologies that are used to develop the web application, uses cases, functional and non-functional requirements. Some of the technologies are: Spring Boot Framework, Spring Security, JWT, Maven, Spring Data JPA, Hibernate, AngularJS Framework, MySQL Server.

4.1. Conceptual architecture of the system

The application has two big components, the Client Application and the Server Application as seen in Figure 4.1. The Client Application (frontend) component handles the GUI and all the user interaction, while the Server Application (backend, REST API) receives requests from the frontend component, processes the requests and sends back appropriate responses. The Server Application also handles the communication with the Database component.

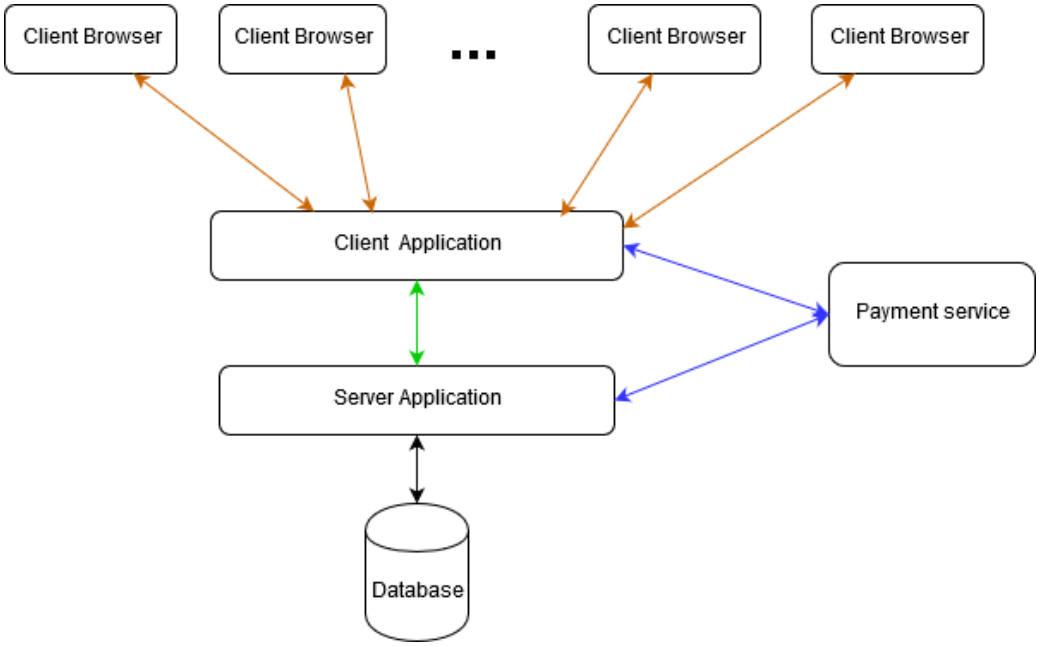


Figure 4.1 Technological perspective

4.1.1. Java and Spring Boot Framework

The programming language used to develop the REST backend is Java [5]. The Java SE 8[12] (Standard Edition) is used along with the Spring Boot Framework [6] to facilitate an easier process of building a web application by allowing an easier, more automatic and less error-prone configuration process and importing all the necessary dependencies required for developing such an application.

The Java programming language has the following features:

- Object oriented, platform independent, simple, secure, architecture-neutral and portable.

[12] https://www.tutorialspoint.com/java/java_overview.htm

Spring Boot[13] offers a solution for this time-consuming and error-prone process by skipping some configurations steps. Spring Boot offers easy dependency management: for example, by using "spring-boot-starter-data-jpa" dependency, Spring Boot will pull all the "spring-data-jpa" dependencies and also the Hibernate libraries because usually Hibernate is used as an implementation of the JPA while developing most applications. Spring Boot also offers auto configuration, meaning that it automatically configures commonly registered beans for the chosen "starter" dependency with sensible defaults (configurations that work best and remove unnecessary thinking to make the same choices again and again when starting a project). For developing this application the MySQL Server is used to store persistent data, the "spring-boot-starter-jdbc" dependency is used for connecting the application to the database, the connection details must be specified in the "application.properties" file and Spring Boot automatically read them while establishing the connection. If no details are provided on how and to what database to connect, Spring Boot will auto-configure an in-memory database. This auto configuration can be overridden whenever the case and replaced with a custom one.

Spring Boot also allows embedding of a server by specifying a dependency for it, in this case Tomcat. The application can be run independently as a runnable jar using its embedded server, Spring Boot makes all the required configurations so the developers don't need to worry about setting up a servlet container and deploying the application to it.

Spring Boot is used in developing the current application because it is a framework for Java and because it is less time-consuming by completely avoiding manual XML configurations, it offers the so called "starter" dependencies which are the most used, also automatically configures them and the server is embedded within the application, meaning there is no need to manually setup a new server and deploy the application on it.

4.1.2. Maven

According to the "Apache Maven Project" official page[14] Maven "is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information".

Maven tackles two aspects of building the software: it describes the way the software is built and it describes the software's dependencies.

Some of the arguments[15] for choosing Maven for developing this application are the following:

- Maven is very popular and it is easy to find answers on the internet to eventual problems.
- Maven is widely used for Java applications.
- Maven has a lot of existing plugins and you can also create your own
- Maven exists for some time now so the Documentation is good, there are a lot of resources about it, and you can find help in open community and forum.
- Setting up a project is very fast and it has a single way of defining dependencies so it's less prone to errors.

[13] https://dzone.com/articles/introducing-spring-boot
[14] http://maven.apache.org/
[15]http://people.apache.org/~ltheussl/maven-stage-site/benefits-of-using maven.html

- With many years of background, Maven has full support to almost each tool and every category.
- It allows to use dependencies from remote repositories, so open-source dependencies can also be used alongside others, we just need to specify them and Maven will pull them into the project.
- It uses the recommended standards and best practices for project layout and definition.
- It has a faster learning curve than Gradle but they are mostly used for the same things, Maven can be used in combination with Gradle but here it is not the case.

4.1.3. Spring Java Persistence API(JPA) with Hibernate

Java Persistence API with Hibernate as this[16] tutorial explaines is used because of the need to map Java objects to database tables. Java objects are designed using Object Oriented Programming concepts and databases are designed with tables and relations between tables.

Before the evolution of JPA the process involved writing SQL Queries by hand. JPA takes a different approach by mapping the Java objects directly to tables/relationships. This kind of mapping is called Object Relational Mapping (ORM). The mapping process is shown in Figure 4.2 below.

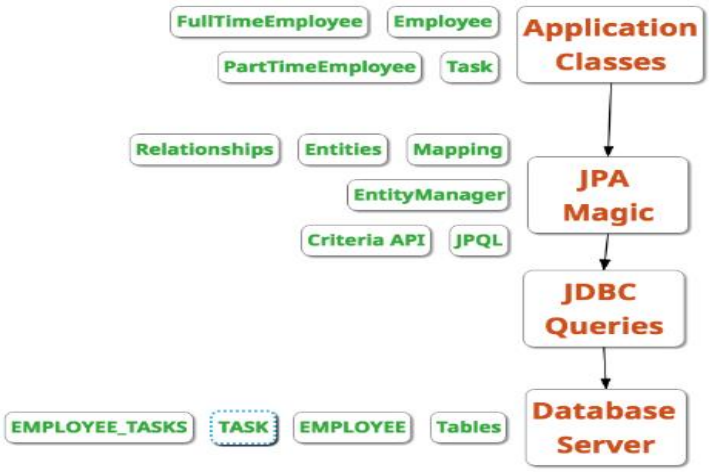


Figure 4.2 JPA mapping process
(source: http://www.springboottutorial.com/hibernate-jpa-tutorial-with-spring-boot-starter-jpa)

The Entity Manager: manages entities after the mappings are defined and handles all database interactions.

Java Persistence Query Language: provides a way to write queries in order to execute searches against entities. Unlike SQL queries, these queries already understand the created mappings.

Criteria API: Java based API to execute searches against databases.

JPA is and API and defines specifications like: how to define entities, how to map attributes, how to map relationships between entities and who manages entities.

Hibernate is an ORM framework. Hibernate is a popular implementation of JPA. Hibernate understands the mappings added between objects and tables and ensures that data is stored or retrieved from the database respecting the mappings.

[16]http://www.springboottutorial.com/hibernate-jpa-tutorial-with-spring-boot-starter-jpa

The current application uses JPA and Hibernate as its implementation to simplify the process of mapping application objects to database tables and relationships because they are popular, widely used, so it is easy to find books and a lot of tutorials on how to include them in a Spring Boot project. It is easy to define the mappings because the process only involves using some annotations which are fast to understand and very straightforward to use.

4.1.4. REST architectural style for web services

A web service is a client-server web application, in which a service provider server, here the REST [7] Backend, is accessible for other applications based on the URL address of the service. The main advantage of such a service is that it provides interoperability between different platforms and in the same time the applications are loosely coupled and the exchanged messages are self-contained, a standard of communication is imposed.

Next the REST and SOAP services are described based on this article[17] and a personal conclusion is drawn on why REST is more suitable for the current application.

SOAP (Simple Object Access Protocol) - SOAP relies heavily on XML and together with schemas, defines a very strongly typed messaging framework. Every operation the service provides is explicitly defined, along with the XML structure of the request and response for that operation. Each input parameter is similarly defined and bound to a type. Everything is codified in WSDL (Web Service Description Language) which is very similar to a contract between the provider and the consumer of the service, or it can be thought of as a method signature for the web service.

SOAP is platform and language independent so it represents a way of communication between different operating systems and different programming languages. This protocol can be a bit slow because it only uses XML to codify files. Using XML there is a supplementary number of information that needs to be transmitted between the two ends that are communicating.

REST (Representational state transfer) - REST is a software architecture style that relies on a stateless communications protocol, most commonly HTTP. REST structures data in XML, YAML, or any other format that is machine readable, but usually the JSON format is used. REST follows the object-oriented programming paradigm of noun-verb. REST is very data-driven, compared to SOAP, which is function-driven.

There is no standard for description format of REST services. The basic REST HTTP requests are: POST (to save something on server side), GET (to retrieve something from the server), PUT (to update something on the server) and DELETE (to delete something from the server).

The most suitable for the current application is REST because it communicates fast and easy with the AngularJS client due to the JSON format of messages, the learning curve is much smaller, the documentation is easier to understand, there is no need to write XML code, the application being build is not and enterprise one and there is no need for the WS* standards.

[17] https://www.soapui.org/learn/api/soap-vs-rest-api.html

4.1.5. JavaScript and AngularJS framework

JavaScript – JavaScript [8] has been adopted by all other major graphical web browsers. It made modern day web applications possible, applications with which the user can interact directly without doing a page reload. JavaScript is also used in more traditional websites to provide various forms of interactivity. JavaScript can be directly introduced inside a HTML page by writing code between "<script>" tags or it can be loaded from an external ".js" file by importing it in the HTML. JavaScript is one of the most efficient and versatile programming language that can be used by a developer. It is used in web applications, mobile web-sites and games.

Some of the advantages[18] of using JavaScript:

- Client-side execution: This means that the code is executed on the user's processor instead of the web server thus saving bandwidth and strain on the web server.
- JavaScript is an easy to learn language: The JavaScript language is relatively easy to learn and comprises of syntax that is close to English. It uses DOM model that provides plenty of prewritten functionality to the various objects on pages.
- Fast to the end user: Because the code is executed on the user's computer, results and processing are completed almost instantly depending on the task, as it does not need to be processed in the site's web server and sent back to the user consuming local and server bandwidth.
- Light programming language: JavaScript does not require a compiler or a special IDE, a developer is required to have a simple text editor at least and a browser in order to run JavaScript code.
- Extended functionality for web pages: Third party add-ons help JavaScript developers to write fragments of code that can be used on web pages when needed.

Arguments for choosing to use JavaScript in my application:

- Easy to learn, there are a lot of books, tutorials and threads on the internet.
- Easy to use, only a text editor and a browser are needed.
- Easy to test using "Developer Tools" available in every browser.

AngularJS - As stated in the framework's documentation, "AngularJS is a structural framework for dynamic web applications. It lets us use HTML as a template language and extend HTML's syntax to express the application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code we would have to write otherwise and everything happens within the browser, making it an ideal partner with any server technology."[19]

AngularJS works by first reading the HTML page, which has embedded into it additional custom tag attributes, those attributes are interpreted as directives telling Angular to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of the JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources.

[18] https://www.quora.com/What-are-the-Advantages-of-Javascript
[19] https://docs.angularjs.org/guide/introduction

AngularJS [9] takes a different approach than the older web MVC frameworks such as Apache Struts or Spring MVC. These older web MVC frameworks reside entirely on the server, all functions such as database access, business logic, display logic and UI activities happen on the server and use server memory and resources. An AngularJS application on the other hand uses business logic that is exposed through REST web services. The REST services can run anywhere and be written in any programming language. The entire AngularJS application runs on the user's hardware, resulting in a much better user experience, meaning that the application runs faster and is much more responsive. This approach frees the server to handle nothing but business logic and data access. Using REST services that send and receive JSON helps to greatly simplify AngularJS applications: JSON is a data-interchange format for REST services that is easy to read and understand. The difference between a web MVC framework based application and an AngularJS application can be seen in the figures below: Figure 4.3 "A Conventional web MVC framework" and Figure 4.4. "AngularJS application design".

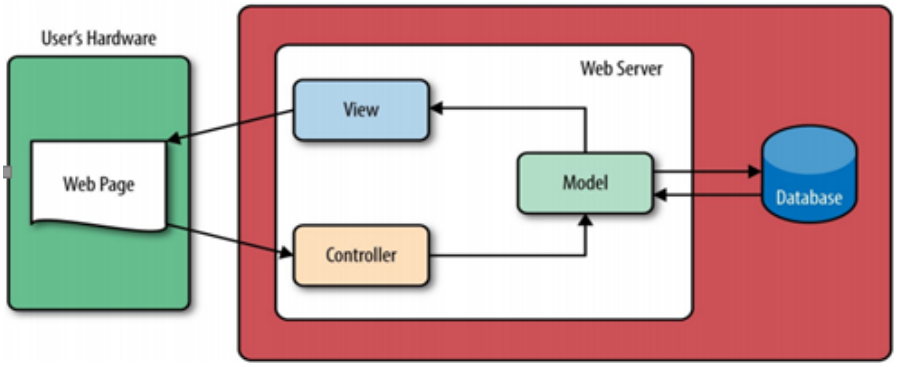


Figure 4.3 Conventional web MVC framework
(Source: Ken Williamson, "Learning AngularJS" [9], Chapter 3. MVC and AngularJS)

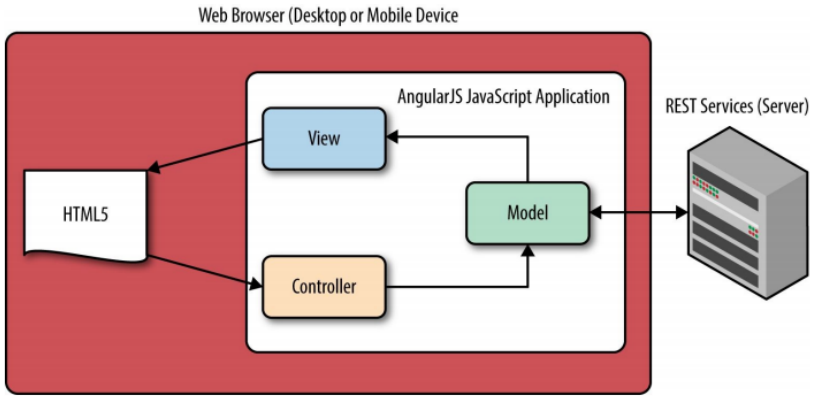


Figure 4.4 AngularJS application design
(Source: Ken Williamson, "Learning AngularJS" [9], Chapter 3. MVC and AngularJS)

Some of the most important advantages of AngularJS are:

- It provides the capability to create Single Page Applications in a very clean and maintainable way.
- It provides data binding capability to HTML, giving the user a rich and responsive experience.

- We can write unit tests for the AngularJS code.
- AngularJS uses dependency injection and makes use of separation of concerns.
- AngularJS uses dependency injection and makes use of separation of concerns.
- With AngularJS we can achieve more functionality with shorter code.

I chose to use AngularJS in my application because of the following:

- AngularJS handles all the DOM and AJAX glue code that otherwise we would have to write by hand and puts it in a well-defined structure.
- It was built for creating CRUD applications and offers everything there is needed for such applications: data-binding, basic templating directives, form validation, routing, deep-linking, reusable components and dependency injection.
- Unit-testing and end-to-end testing can be performed.
- Offers a "Seed" application with directory layout and test scripts as a starting point.

4.1.6. CSS

CSS[20] (Cascading Style Sheets) is a style language that defines the layout of the HTML (Hypertext Markup Language) documents. The difference between HTML and CSS is that HTML is used to structure the content of a page while CSS is used for formatting structured content.

CSS [10] allows us to create a set of rules for specifying how the content of a HTML element appears to the user (e.g. the background of the page is black, all paragraphs appear white and are positioned in the middle of the page).

Some of the benefits of using CSS are: control over the layout of a page, control the layout of several pages using a single style sheet, custom layout for different screen sizes.

Arguments for using CSS for the current application:

- We can give style to any element from our web pages in a large variety of ways.
- Using CSS classes, style for an element can be created once and then reused whenever needed on other elements that are alike.
- It is easy to learn and use in a project.
- Even though there are styling frameworks like Bootstrap, CSS core is necessary when we want to create our own styles.

4.1.7. Bootstrap

According to Wikipedia, "Bootstrap is a free and open-source front-end framework for designing websites and web applications. It contains HTML and CSS based design templates for typography, form, buttons, navigation and other interface components, as well as optional JavaScript extensions."[21]

Bootstrap gives us the ability to create a responsive layout with little effort. One of the biggest advantages Bootstrap offers is that it comes with free toolset for creating responsive and flexible layouts.

[20] http://html.net/tutorials/css/lesson1.php
[21] https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)

Some of the advantages[22] of using Bootstrap are presented next:

- It saves a lot of development time: use predefined classes and templates of Bootstrap.
- Bootstrap responsive greed: Bootstrap offers a twelve-column grid system. The grid system is responsive, meaning that it adjusts itself depending on the user's device resolution, without us having to do anything special.
- Consistent design: All Bootstrap components share the same design templates and styles from a central library, that is Bootstrap's results are uniform across platforms (we will see the same design of the website on Internet Explorer, Chrome and Firefox for example). The code is easier to understand for other developers that might work later on the project.
- Ease of use: Anyone with basic knowledge of HTML and CSS can easily use Bootstrap.
- Cross-browser compatibility: Bootstrap is compatible with all modern browsers like: Mozilla Firefox, Google Chrome, Safari, Internet Explorer and Opera.

The arguments for choosing Bootstrap are the following:

- It is very easy to use and provides pre-built layout examples on their website.
- It has a good Documentation so it is easy to learn.
- It is easy and fast to create responsive designs with the twelve-column grid.
- It ensures cross-browser compatibility.
- It is quick and easy to prototype new designs.

4.1.8. Database

According to the following definition, a database is "a separate application that stores a collection of data externally (on a storage device). Databases are manipulated with the help of database management system. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds. Other kinds of data stores can also be used, such as files on the file system or large hash tables in memory, but data fetching and writing would not be so fast and easy with those types of systems."[23]

A relational database is used for developing the current application. The database for the current application is stored and managed using MySQL [11] relational database management system.

Relational database management systems (RDBMS) are used to store and manage huge volume of data, there are other types of systems called non-relational, but they are new on the market and are not suitable for every case. This is called relational database because all the data is stored into different tables and relations are established using Primary Keys or other keys known as Foreign Keys.

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. The MySQL architecture[24] is basically a client-server system. MySQL

[22]https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-introduction.php
[23] https://www.tutorialspoint.com/mysql/mysql_tutorial.pdf
[24] https://www.rathishkumar.in/2016/04/understanding-mysql-architecture.html

database server is the server and the applications which are connecting to MySQL database server are clients. The MySQL architecture contains the following major components: Application Layer, MySQL services and utilities, SQL interface, Parser, Optimizer Caches, Storage Engine Layer.

The arguments for choosing MySQL Server for the current application are the following:

- MySQL was designed and optimized for Web applications.
- It is quite old on the market and it has a well-developed community and a good documentation.
- It is open-source, free and very popular so a lot of tutorials are found online.

4.1.9. Security

JSON Web Tokens (JWT) - JWT[25] is a token based authentication system. It is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is digitally signed using JSON Web Signature. The JWT is a self-contained token which has authentication information, expire time information and other user defined claims digitally signed.

Because HTTP is a stateless protocol it doesn't store any state from request to response. If we login for one request, we will have to login again in order to make another request.

The token-based authentication systems, like JWT (Figure 4.5), allow users to enter their username and password in order to obtain a token which allows them to fetch a specific resource without entering their user and password at each request. Once their token has been obtained, the user can use the token to access specific resources for a set time period.

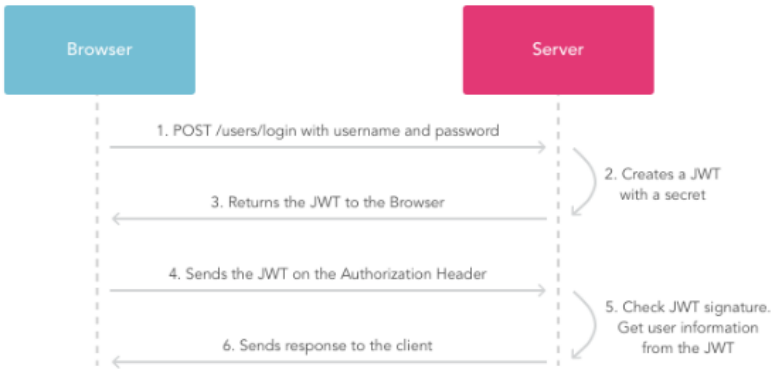


Figure 4.5 Token-based authentication

(JWT, source: https://jwt.io/introduction/)

Reasons for using JWT with the current application:

- No "session" to manage: the JWT is a self-contained token.
- Portable: A single token can be used with multiple backends if the case.
- Good performance: It reduces the network round trip time.

[25] https://dzone.com/articles/jwtjson-web-tokens-are-better-than-session-cookies

- Decoupled/Decentralized: The token can be generated anywhere. Authentication can happen on the resource server or easily separated into its own server.

Spring Security - According to the documentation, „Spring Security is a powerful and highly customizable authentication and access-control framework. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements. „[26]

Spring Security offers the following features: comprehensive and extensible support for both Authentication and Authorization, protection against attacks like cross site request forgery, session fixation, and many more.

Spring Security [12] provides us with means of limiting access to particular Java methods based on roles for example and with means of encoding a user's password so it will not be vulnerable to attackers.

Password encryption - "A password hash is an encrypted sequence of characters obtained after applying certain algorithms and manipulations on user provided password, which are generally very weak and easy to guess."[27]

Once the password is hashed and stored in the database it can't be converted back to the original password. Every time a user logs in the password hash has to be generated again and match it with the hash stored in the database in order to check the validity of the password. Examples of algorithms, used in Java to encrypt passwords are MD5 which is considered to generate weak hashes and SHA (1, 256, 384, 512) which generates stronger hashes than MD5 but is still weak compared to other encryption algorithms. The two algorithms mentioned before are fast and easy to implement but they are an easy target for brute-force attacks[28], also different passwords can eventually result in the same hash.

A solution for reducing the risk against the types of attacks mentioned above was to append to the password a randomly generated text before hashing it, the randomly generated text is called "salt".

A "salt" is defined as a "random data that are used as an additional input to a one-way function that hashes a password or pass-phrase"[29].

Even after applying the "salt" method to better secure passwords using MD5 and SHA algorithms, the problem is that today hardware is very fast and brute force attacks using dictionaries and rainbow tables can crack such a password eventually. To solve this problem, the solution was to make the hash function slow enough to minimize the damage done by a brute force attack but fast enough so the user won't notice the delay. This was implemented using some CPU intensive algorithms such as PBKDF2, Bcrypt or Scrypt. There exists an implementation in Java for each of these three algorithms, more information can be found here[24].

Bcrypt + "salt" is used for the proposed project because it is easy to implement, and safer than the MD5 or SHA + "salt" approach.

Bcrypt works in the following way to prevent attacks: "It uses Blowfish to encrypt a magic string, using a key derived from the password. Later, when a user enters a password, the key is derived again, and if the ciphertext produced by

[26] https://spring.io/projects/spring-security
[27]https://howtodoinjava.com/security/
[28] https://crackstation.net/hashing-security.htm
[29] https://en.wikipedia.org/wiki/Salt_(cryptography)

encrypting with that key matches the stored ciphertext, the user is authenticated. The ciphertext is stored in the password table, but the derived key is never stored. In order to break the cryptography here, an attacker would have to recover the key from the ciphertext. This is called a "known-plaintext" attack, since the attack knows the magic string that has been encrypted, but not the key used. Blowfish has been studied extensively, and no attacks are yet known that would allow an attacker to find the key with a single known plaintext."[30]

4.1.10. Payment Service

A comparison[31] between two payment services, Stripe[32] and Braintree[33] is presented next.

Braintree - Braintree is mostly a payment service for enterprise customers. It provides all the payment services of Stripe while also delivering PayPal and low currency conversion rates. Provides integration with PayPal; we can accept PayPal payments and it also works with Venmo, a digital wallet that belongs to PayPal. We can accept in addition Android Pay and Apple Pay payments and it also has support for Bitcoin payments.

No monthly fee, fees are imposed per transaction (2.9% + $0.30), the advantage would be that if we are not selling anything for some reason, we don't have to pay for the service. It is available in 46 countries. It has the advantage when it comes to operating at a larger scale and also has a more responsive support team.

Braintree offers a pre-built payment form that we can bring into our website just by copy pasting the provided code: Braintree Drop-in UI. It also allows for a custom-tailored checkout experience as well. Braintree requires our server to retrieve a one-time client token from Braintree's API before displaying a payment form, whether using its Drop-in UI or hosted fields (its custom form solution). It offers sandbox for testing with a low barrier to entry and helpers for automated tests, it also has a quick start guide.

Stripe - Stripe is an online payment service. Stripe is a good option for a digital startup it helps establish the business and grow it. Works with AliPay, a Chinese payment platform. In addition, it can accept Android Pay and Apple Pay, it also has support for Bitcoin payments like Braintree. Like Braintree Stripe doesn't have any monthly fee, fees are imposed per transaction (2.9% + $0.30), the advantage would be that if we are not selling we don't have to pay for the service. Stripe is available in 25 countries. Stripe has a fast setup because of its easy to use API and easy to understand documentation which is found to be more precise than that of Braintree. It also allows trial without a credit card. It has support for percentage discount.

Stripe has a pre-built payment form that we can bring into our website: Stripe Checkout. It also allows for a custom-tailored checkout experience as well. Stripe doesn't require our server to retrieve a one-time token before the client renders the payment form, saving a step and likely cutting load time by 1–2 seconds. Stripe has robust webhook status and logs, available from its dashboard. Stripe also offers sandbox for testing with a low barrier to entry and helpers for automated tests. Like Braintree, Stripe also has a quick start guide.

[30] https://stackoverflow.com/questions/1561174/sha512-vs-blowfish-and-bcrypt
[31] https://www.process.st/stripe-vs-paypal-vs-square-vs-braintree/
[32] https://stripe.com/
[33] https://www.braintreepayments.com/en-r

Because the current application is a webstore that is targeting a small business, all the readings point towards using Stripe, because it is easier to integrate into the application, the documentation is more precise than that of Braintree (which can be vague sometimes), it offers a trial period without submitting card information, it has robust webhook status and logs available from its dashboard. PayPal[34] doesn't really make sense to be integrated separately as it supported by Braintree, which is much easier to integrate and it's also highly scalable. Braintree and Stripe have very similar functionalities, the big differences being that Braintree is more suitable for enterprise use and it facilitates PayPal payment while Stripe is better for startup and facilitates payment with AliPay[35].

4.1.11. Dialogflow

Dialogflow is a platform owned by Google. It works using natural language processing and machine learning. It helps users to build chatbots that facilitate computer-human interaction by giving the impression to the user that he/she is speaking with a human. Dialogflow is the choice because it has a friendly and easy to use interface and because it runs on Google's infrastructure it can be later scaled to millions of users if the case. It also has support for more than fourteen languages and for more than twenty different platforms (e.g. Facebook, Twitter and Slack) and a getting started guide which takes you through the steps of creating a simple chatbot. More information can be found in their documentation[36].

4.2. Requirements

4.2.1. Functional requirements

According to Wikipedia: „In software engineering, a functional requirement defines a function of a system or its component. A function is described as a set of inputs, the behavior and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system or application is supposed to accomplish."[37]

Table 4.1 Functional requirements

FR#	Functional requirement	User
FR 1	Sign up	Visitors
FR 2	View products	Visitors, registered users
FR 3	Search products by applying filtering based on categories	Visitors, registered users
FR4	Search products by brand and model	Visitors, registered users
FR5	Chat with FAQ bot	Visitors, registered users

[34] https://www.paypal.com/ro/home
[35] https://intl.alipay.com/
[36] https://dialogflow.com/
[37] https://en.wikipedia.org/wiki/Functional_requirement

FR 6	Review a product (comment +rating)	Registered users
FR 7	Edit personal profile	Registered users
FR 8	Add products to shopping cart	Registered users
FR 9	Edit shopping cart details	Registered users
FR 10	Order products from shopping cart	Registered users
FR 11	Cancel an order that was not payed for by card	Registered users
FR 12	Reset password	Registered users
FR 13	Log out	Registered users
FR 14	View order history	Registered users
FR 15	CRUD on product suppliers	Administrator
FR 16	View each registered user's orders	Administrator
FR 17	CRUD on special offers	Administrator
FR 18	View registered accounts and orders associated with these accounts	Administrator
FR 19	CRUD on products	Administrator

4.2.2. Non-functional requirements

According to Wikipedia: "In software engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors, contrary to functional requirements that define the specific behavior or functions. "[38]

Next, the non-functional requirements that were taken into consideration while developing the application are presented.

Usability - following is a definition of usability as found on Wikipedia: "In software engineering usability is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency and satisfaction in a quantified context of use."[39]

In this context the current application aims to minimize the time and number of steps needed to complete and order, add a product to wish list, review and rate a product for example, by clearly letting the user know through appropriate messages about the current state of the application and about what he/she should do next to be able to achieve the desired goal. If it's not clear for the user how he/she should proceed next or what will happen after they complete a step, they might abandon the shopping cart for example or other tasks that they were doing inside the application.

As a summary, usability should be achieved through efficient use (e.g. on the landing page show the newest products or the products that are on a special offer so users can see them immediately when accessing the application), ease of use (e.g. the application should have well-structured help information, easy to understand and guiding error messages), interface consistency (e.g. the system is intuitive, the user

[38] https://en.wikipedia.org/wiki/Non-functional_requirement

[39] https://en.wikipedia.org/wiki/Usability

interface remains consistent on every page, so a "Next" button should have the same style everywhere in the application).

Performance - The performance[40] of a system, in most cases, is resumed to the maximum response time needed by a system to give the necessary feedback to users as a result of their actions. Depending on the context, performance may involve one or more of the following: short response time for a request, high throughput (rate of processing a request), low utilization of computing resources, fast compression and decompression of data, short data transmission time.

The general advice[41] on response time is: 0.1 second is the limit for having the user feel that the system is reacting instantaneously, meaning no special feedback is necessary but to display the result, 1.0 second is the limit for the user's flow of thought to be uninterrupted even if the user notices the delay. Normally no special feedback is required for delays between 0.1 and 1.0 seconds. The limit for keeping the user's attention focused on the application is about 10 seconds for keeping the user's attention focused on the application. For longer delays, users will want to perform other tasks while waiting for a response, so they should be given feedback indicating when the response is expected to be ready.

Scalability - According to Wikipedia, scalability „is the capability of a system, network or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth."[42]

A system is considered scalable, if it is capable of increasing its total output under an increased load when hardware resources are added.

The application should accommodate more users, products and processed orders without changing the application as the business if shifting. This means that we have to take into account that the system will have to perform more transactions and store more information as time passes, this could be achieved by using a cloud platform for the application to reside into because such a platform allows for easy horizontal or vertical scaling.

Availability – According to Wikipedia, availability „means the probability that a system is operational at a given time, i.e. the amount of time a device is actually operating as the percentage of total time it should be operating. High-availability systems may report availability in terms of minutes or hours of downtime per year."[43]

A web store should be available 24/7 in theory because otherwise the business could lose potential customers, but unexpected errors might occur and the system might be brought down for maintenance. The availability is also given by the machine on which the application is running. For a higher availability the application can be deployed using a cloud hosting service like Amazon's EC2[44] (Elastic Cloud Computing) as they claim offering 99.99% availability for each Amazon EC2 Region.

Maintainability – It is defined as „the probability of performing a successful repair action within a given time"[45]. In other words, maintainability measures the ease and speed with which a system can be restored to operational status after a failure occurs.

Maintainability is also a measure of how easily code can be understood and modified, so architecture and code standards should be imposed and respected.

[40] https://en.wikipedia.org/wiki/Computer_performance
[41] http://www.1202performance.com/atricles/how-to-write-performance-requirements
[42] https://en.wikipedia.org/wiki/Scalability
[43] https://en.wikipedia.org/wiki/Reliability,_availability_and_serviceability
[44] https://aws.amazon.com/ec2
[45] https://www.reliasoft.com/products/reliability-analysis/blocksim/maintainability-analysis

For example, systems that respect the low coupling and high cohesion properties are easier to maintain. Coupling is a measure of the degree to which a component depends upon other components of the system and cohesion is a measure of how strongly related the various functions of a single component are. So, for the current application to be maintainable it should have components that can be used and modified independently from each other (low coupling), components that embody once concept, that are focused on fulfilling a specific responsibility (high cohesion).

Portability - Portability[46] in high-level computer programming, according to Wikipedia is the usability of the same software in different environments.

When a software with the same functionality is produced for several computing platforms, portability is the key issue for development cost reduction. The proposed application can run on any computing platform as long as there is a browser installed and there is an internet connection.

Security - The definition for security in the information systems context as found on Techopedia: „Information systems security, refers to the processes and methodologies involved in keeping information confidential, available and assuring its integrity. It also refers to: access controls, which prevent unauthorized personnel from entering or accessing the system; protecting information no matter where that information is, i.e. in transit or in a storage area"[47]

The application stores user personal data (name, address, email etc.) and also exchanges personal data between the client side and server side. A user's personal data should not be available to other users of the application or to attackers and also users must not be allowed to access certain pages of the application if they don't have permission to do that. Three things are taken into consideration here:

- Authentication: JSON Web Tokens and Spring Security are used for this, the user logs in by providing the username and password, a token is generated and associated with that user, each request the user makes inside the application will include that token so the application can identify the user and execute the request.
- Authorization: The application limits the resources a user is allowed to access based on two roles, "ROLE_CUSTOMER" and "ROLE_ADMIN", the token received during authentication. For example, a user having the role "ROLE_CUSTOMER" is not allowed to access the administrator page and perform tasks like managing products, or managing other users accounts.
- Information Exchange: This is also based on using the JSON Web Tokens, as the information sent between the client-side of the application and server-side must be verified to make sure that the sender is a registered user of the application and not a possible attacker.

4.3. Use cases

A use case[48] is a methodology used in system analysis to identify, clarify and organize system requirements. Use cases specify the expected behavior and not how that behavior is achieved.

[46] https://en.wikipedia.org/wiki/Software_portability
[47] https://www.techopedia.com/definition/24840/information-systems-security-infosec
[48] https://www.quora.com/What-are-use-cases

4.3.1. Types of users

The application supports three types of users:

- Visitor: a user that doesn't have a registered account. This type of user can browse through the products on the web site by applying filters can see a products details reviews and can choose to create an account and then authenticate, in order to use other functionalities of the application.
- Registered user: a user that owns a registered account. This type of user in addition to the visitor user can add products to the shopping cart, can order products, can add products to a wish list, can rate, review a product and cancel an order.
- Administrator: this type of user is a registered user, in addition to what a registered user can do, the administrator can view registered users accounts, product suppliers, products, special offers and view order history of registered users.

4.3.2. Use cases

Next the use cases corresponding to each type of users will presented in Figure 4.7 "Visitor use case diagram", Figure 4.7 "Registered user and administrator use case diagram.

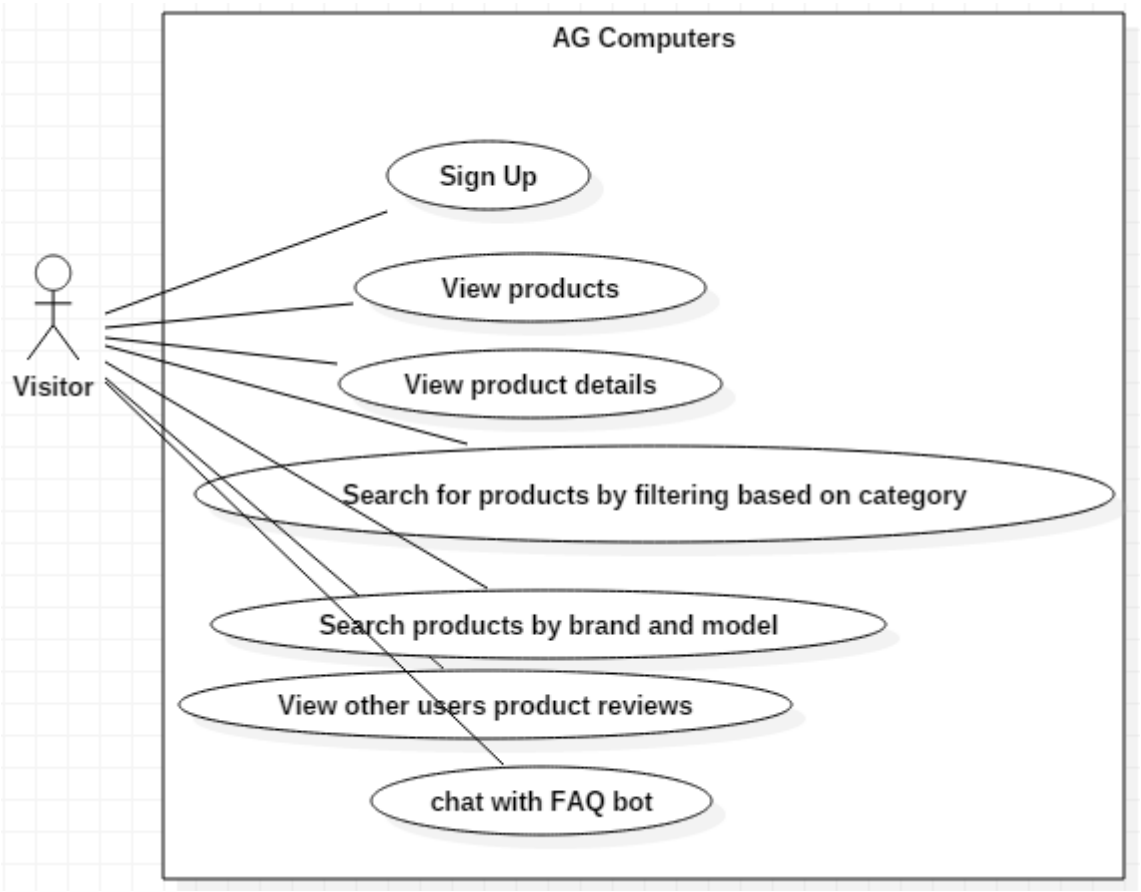


Figure 4.6 Visitor use case diagram

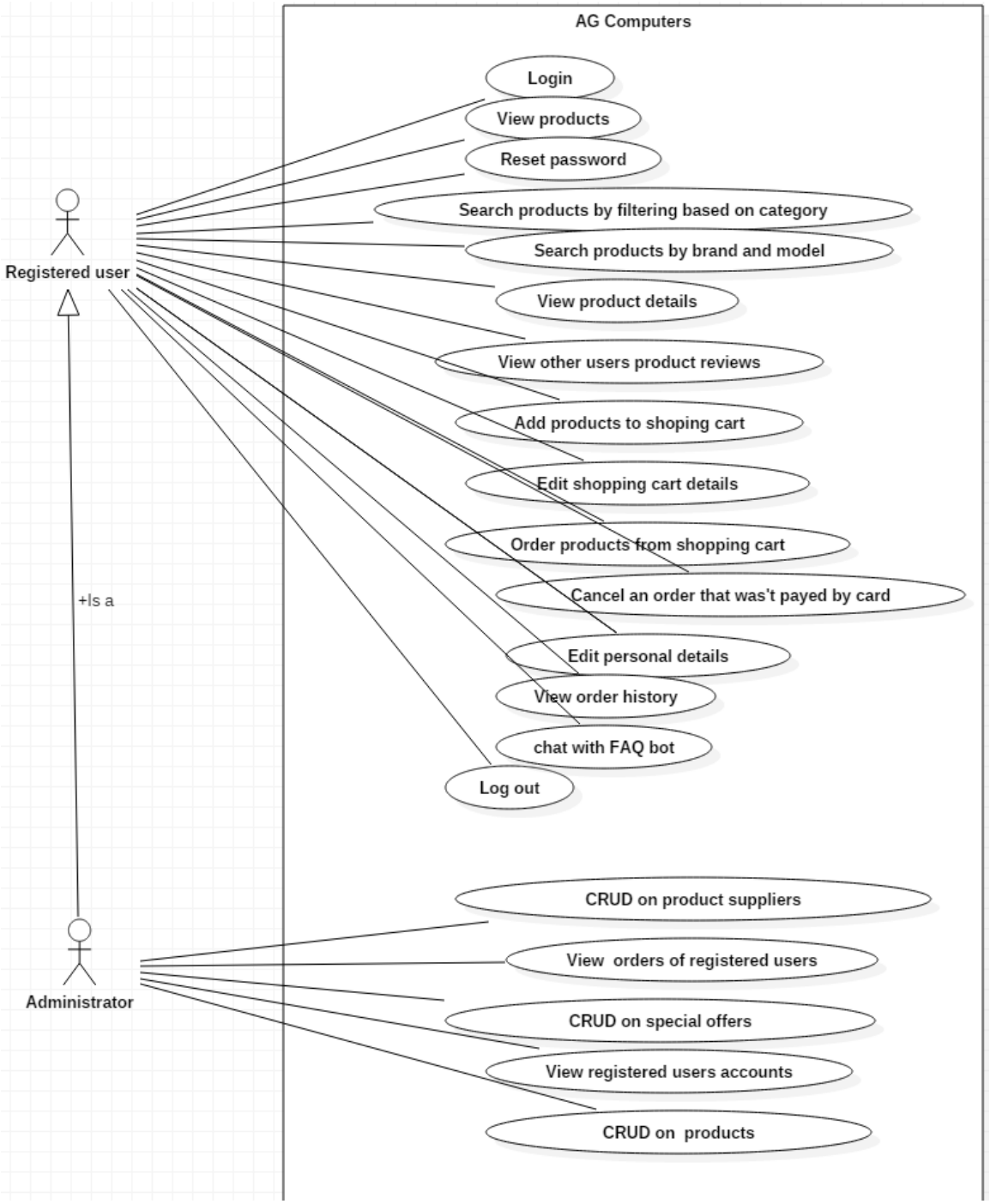


Figure 4.7 Registered user and Administrator use case diagram

Detailed description of use cases

Next a few of the use cases seen in the figures above will be detailed.

Use Case 1

Use case name: Order product

Primary actor: Registered user

Stakeholders and interests:

- Store owner - interested in having virtual means (user friendly, intuitive, fast) to provide the customers (common users) similar experience of buying a product to the real world without having to take a trip to the physical store.

- Common user – interested in having user friendly and intuitive means to complete the product order process.

Preconditions: The actor is authenticated and authorized.

Postconditions:

- The order is placed.
- The user is notified via an e-mail that the order is placed.

Basic flow:

<step1>The actor adds a product to the shopping cart.

<step2>System updates shopping cart to contain products chosen by the customer.

<step3>The actor opens the shopping cart.

<step4> The actor checks how many units of the same product will be ordered.

<step5> The actor checks if the delivery details are correct.

<step6> The actor chooses a delivery method.

<step7> The actor chooses a payment method.

<step8> System validates the payment method.

<step9> The actor sends the order.

Alternative flows:

Edit number of units for the product to be ordered.

<step 4a> The user chooses how many units of the same product wants to order:

1.System saves the number of units to be ordered.

Edit delivery details.

<step 5a> The user changes the delivery details:

1.System saves the new delivery details.

<step 7a> The user chooses credit card:

1.The user fills in credit card details.

<step 7b> The system validates card details:

1.Details correct, user can continue to <step9>.

2.Details not correct, system sends user to <step7a>.

Use Case 2

Use case name: Sign up

Primary actor: Visitor

Stakeholders and interests:

- Store owner - interested in having more registered customers so they can buy online products from the store.
- Visitor – interested in having a registered account to be able to order product, save products to wish list for later purchase or review a product.

Preconditions: There isn't an already registered account with the same e-mail address and username of the actor.

Postconditions:

- The account is created.
- The data provided by the actor upon registration is stored.
- The actor receives a confirmation e-mail saying the registration was successful.

Basic flow:

<step1> The actor chooses to Sign up.

<step2> The system displays a form regarding actor's information like e-mail, phone number, username, password etc.

<step3> The actor fills in the form with all the required information.

<step4> The system validates the form.

<step5> The actor finishes the registration.

<step6> The application displays a success message.

Alternative flows:

Actor doesn't fill the form with all the required information

<step4a>The system invalidates the form and the actor can't register:

1. The actor is prompted with an error message that lets him/her know what information is wrong.

<step5a>The actor can't register:

1. The actor is prompted with a corresponding error message that lets him/her know what went wrong.

Use Case 3

Use case name: Write a review

Primary actor: Registered user

Stakeholders and interests:

- Store owner – interested in getting positive reviews for the sold products.
- Other users – interested in reading other user's opinion on the products before buying them.

Preconditions: The actor is authenticated, authorized and is on the details page of the product

Postconditions:

- The review is immediately displayed.
- The review data is persisted in the database.
- The average rating of the reviewed product is updated and displayed.

Basic flow:

<step1> The actor chooses a rating from 1 to 5.

<step2> The actor fills in the text input with a comment about the product.

<step3> The system validates the input.

<step4> The actor submits the review.

Alternative flows:

Actor doesn't fill the text input with a comment.

<step2a>The system invalidates the review and the actor can't submit it:

1. The actor is prompted with an error message that lets him/her know what went wrong and the review is not submitted.

Chapter 5. Detailed Design and Implementation

This chapter presents the most relevant parts of how the application was designed and implemented. The application has three main components: Client Application, Server Application and the Database. Details like the application's architecture diagram, database diagram, package diagram, class diagram, deployment diagram are presented, as well as implementation details of most important components for a better understanding of how the application is designed and built.

5.1. General architecture of the application

The general architecture of the application is presented in Figure 5.1 "General Architecture of the Application" with all its components and in the following subchapters the architecture of each application (Server Application and Client Application) is presented in detail along with the most relevant parts of the implementation.

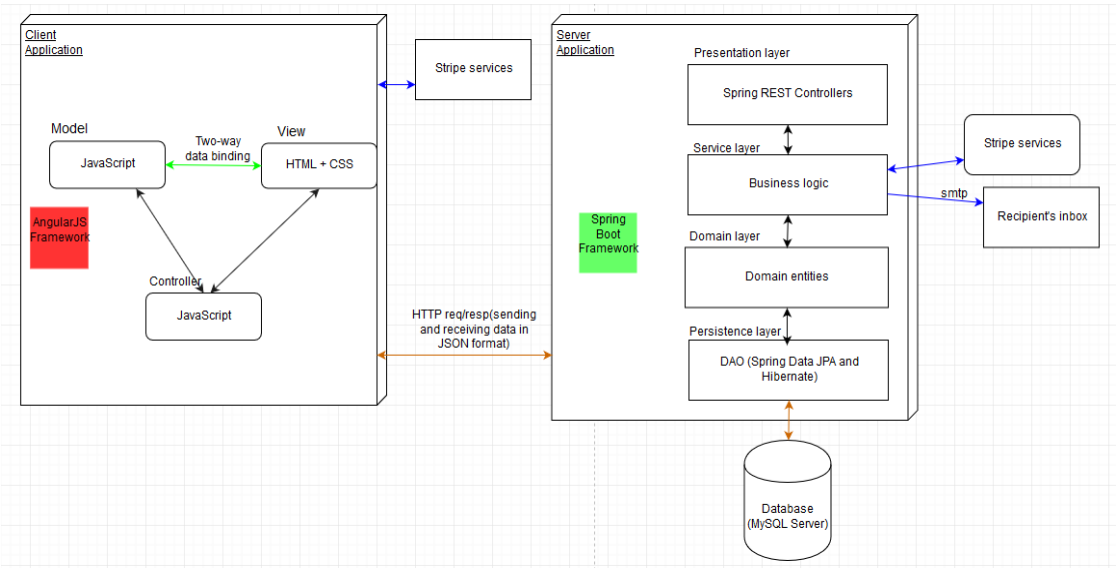


Figure 5.1 General Architecture of the Application

The whole application consists of three major components: Client Application, Server Application and Database. The Client Application handles the GUI[49] and user interaction and makes HTTP[50] requests to the Server Application by sending data in in JSON[51] format, the Server Application processes the data, if needed it retrieves or stores data from/to the Database and sends back a response also containing the data in JSON format which the Client Application processes and displays it to the user in a more readable form. Both the client and server applications use Stripe[52] for creating a safe payment process so the card information is not sent between the two applications, a request is made from the client to Stripe, in order to validate card information, a token is received which is passed to the server and the server makes a payment request to Stripe using that token because Stripe can ensure a better security of the

[49] https://en.wikipedia.org/wiki/Graphical_user_interface
[50] https://ro.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[51] https://www.json.org/
[52] https://stripe.com/

card information that our applications. E-mails are sent using JavaMail[53] through the SMTP[54] protocol from the Server Application each time a user registers, completes an order or cancels an order. The Client Application is implemented using AngularJS Framework which imposes the MVC design pattern as seen in figure 5.1 above, while the Server Application is implemented Spring Framework and consists of multiple layers each having their own responsibility. More details about the design and implementation of each component are presented in the subchapters that follow.

5.2. Client Application

5.2.1. High - level overview

The Client Application component, is the component responsible for the GUI (Graphical User Interface) and the one that deals with user interaction. This component reacts to user requests and further passes them to the Server Application, if the case, in order to give the user an appropriate response. This application was developed using the AngularJS Framework (presented in Chapter 4) which imposes the MVC (Model View Controller) design pattern which offers separation of concerns and decoupling of code, meaning that each component does a specific job. In Figure 5.2 "Angular MVC", we can see how the different components communicate with each other, with the user and the server application, making use of the two-way databinding feature that Angular offers.

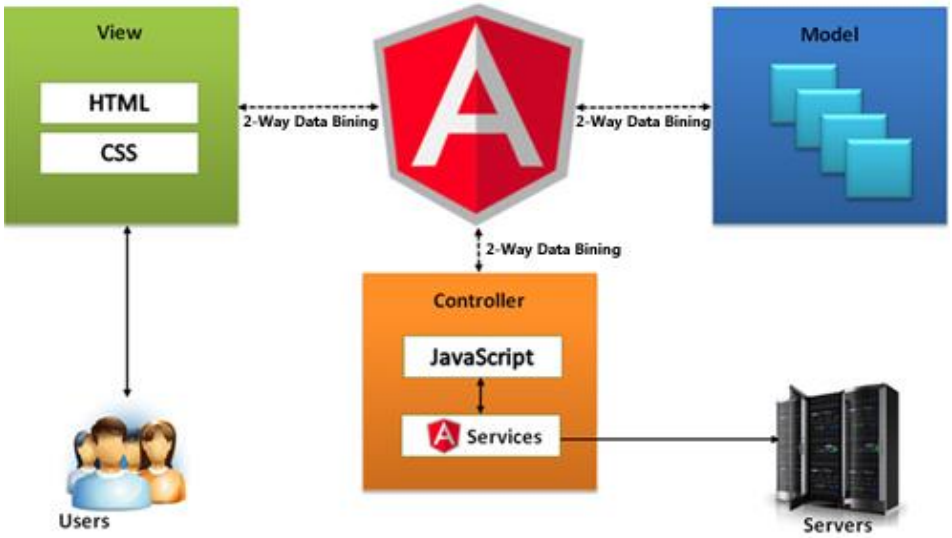


Figure 5.2 Angular MVC

(Source: https://codeburst.io/angularjs)

The View is rendered from a Template which implemented using HTML, CSS, Bootstrap (presented in Chapter 4) and makes use of Angular directives to bind to the model like "ng-model", or we can write our own directives and embed them in html tags. The directives are a way of extending HTML with new attributes. The model contains the JavaScript objects that are rendered in the view, they are a JS representation of the models from the Server Application. The Controller is responsible with the communication between Model and View and it also communicates with the Server Application by making HTTP requests when needed

[53] https://en.wikipedia.org/wiki/JavaMail
[54] https://ro.wikipedia.org/wiki/SMTP

through Angular services. AngularJS has a feature called two-way data binding meaning that when the model changes the view is updated and vice versa.

5.2.2. Detailed description of the components

In Figure 5.3 we can see the AngularJS architecture in more detail and how the components mentioned above are connected in the developed application.

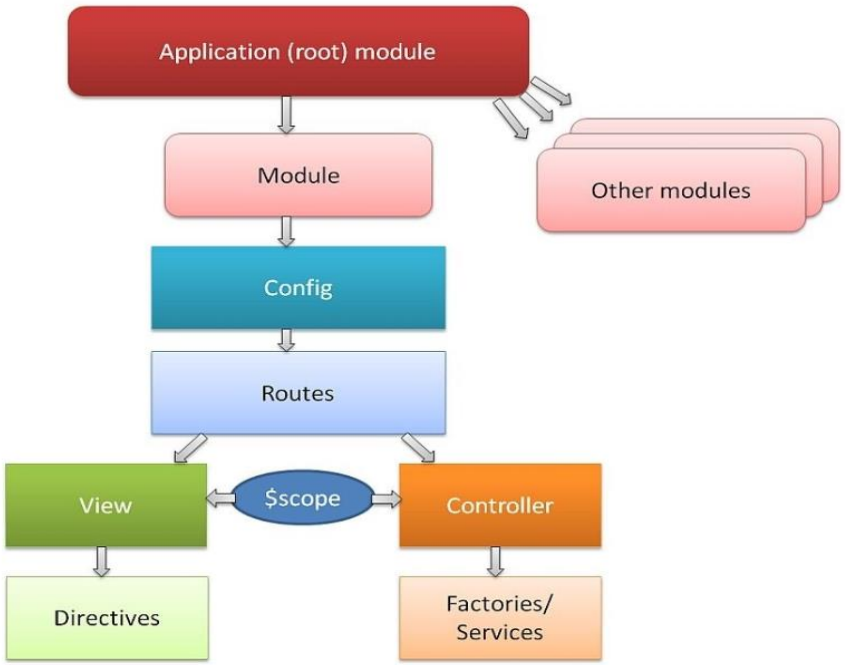


Figure 5.3 AngularJS Architecture

(source: https://codesjava.com/)

Module - The module is a container for the different parts of the application, the module in the current application is named "app" and it hold mostly all the components, like templates, controllers, directives, services and factories.

Config, Run, Routes (states) and Views- Config is a phase, during the config phase the providers[55] have been registered but they haven't been called yet. In the current application the config phase is used to configure all available states and their routes using provider "$stateProvider.state". Then during run phase all providers are already called and their services are registered so we can use the "$state" service to navigate between different views. For a better understanding Figure 5.4 shows an example of what can be done in the config phase and in the run phase using "ui-router" library which is also used in the proposed application. We can see that during the config phase we can't inject any service (e.g. "$state") and during run phase we can't inject any provider (e.g. "$stateProvider").

[55] https://docs.angularjs.org/guide/providers

```
angular.module("myApp", ["ui.router"])
       .config(function($stateProvider) {
           // Configure all available states during the config phase
           // using the $stateProvider!
           $stateProvider.state("home", { ... });
           $stateProvider.state("about", { ... });
           $stateProvider.state("contact", { ... });
       })
       .run(function($state) {
           // Use the configured $state service afters to go to the
           // initial state! You could do the same in a service.
           $state.go("home");
       });
```

Figure 5.4 AngularJS UI-Router used in config and run phases

(source: https://tuhrig.de/angularjs-provider-and-app-configuration/)

In the config phase using "ui-router" all the possible states are declared together with the templates they should render, the controller which manages every action in that state and if the case a property named "resolve" that can contain one or more promises that must resolve successfully before the route or state is changed. Next, we will refer to Figure 5.5 below to explain what a promise is regarding the current implementation and why it is needed.

```
function config($urlRouterProvider, $stateProvider) {

  $urlRouterProvider.when('', '/home/landingpage');
  $stateProvider
    .state('home', {
      url: '/home',
      abstract: true,
      templateUrl: 'mainapp/templates/home.html',
      controller: 'HomeCtrl as home',
      resolve: {
        cartItems: getComputerOrderItemsHome,
        categories: getAllComputerCategories
      }
    })
    .state('home.login', {
      url: '/login',
      templateUrl: 'mainapp/templates/log-in.html',
      controller: 'LogInCtrl as login',

    })
```

Figure 5.5 Config phase, using ui-router to configure the states

A promise in our case is a HTTP request to the Server Application through the "getCmputerOrderItemsHome" function for example to bring all the items in a user's cart (cartItems) if there are any before loading the home state. The items inside the cart are require din order to display a badge over the shopping cart, showing how many items are inside the shopping cart. The "cartItems" array can be used inside the "HomeCtrl" Controller by injecting the "cartItems" variable in the controller's header.

The "url" parameter is the URL that will be displayed in the browser while in that state. The "templateUrl" parameter specifies the path to the HTML and CSS template that is rendered when the user goes to that state, in less words, it is what the user sees when the user is in that state. The "controller" parameter specifies the controller to be used for that state and also provides and alias so we don't have to type

its full name when referring to it while writing The “abstract: true” parameter specifies that the state is a parent state and it can never be called separately, instead it is called whenever a child state that inherits it is called, for example as seen in Figure 5.4, the “home.login” state is a child state of the “home” state, so whenever we are in the “.login” state the “home” state is also called. All the states in the Client Application are children of the “home” state because the Template and Controller of this state are used everywhere due to the fact that they represent the navigation bar with all its actions and the footer of the application. The “$urlRouterProvider.when” is used to always direct the user to the starting page of the application when the application is started. Another parameter specified while configuring the state can be “data”, used like in Figure 5.6 to limit a user’s access to that state based on the role that user has, as a measure of security, because that state contains information about other users of the application which should not be displayed to everyone. This is just a portion of the security measures implemented using JWTs (JSON Web Tokens) which will be detailed during the paragraphs that follow for the Client Application and in the next Subchapter for the Server Application.

Figure 5.6 Limit user access to state based on role

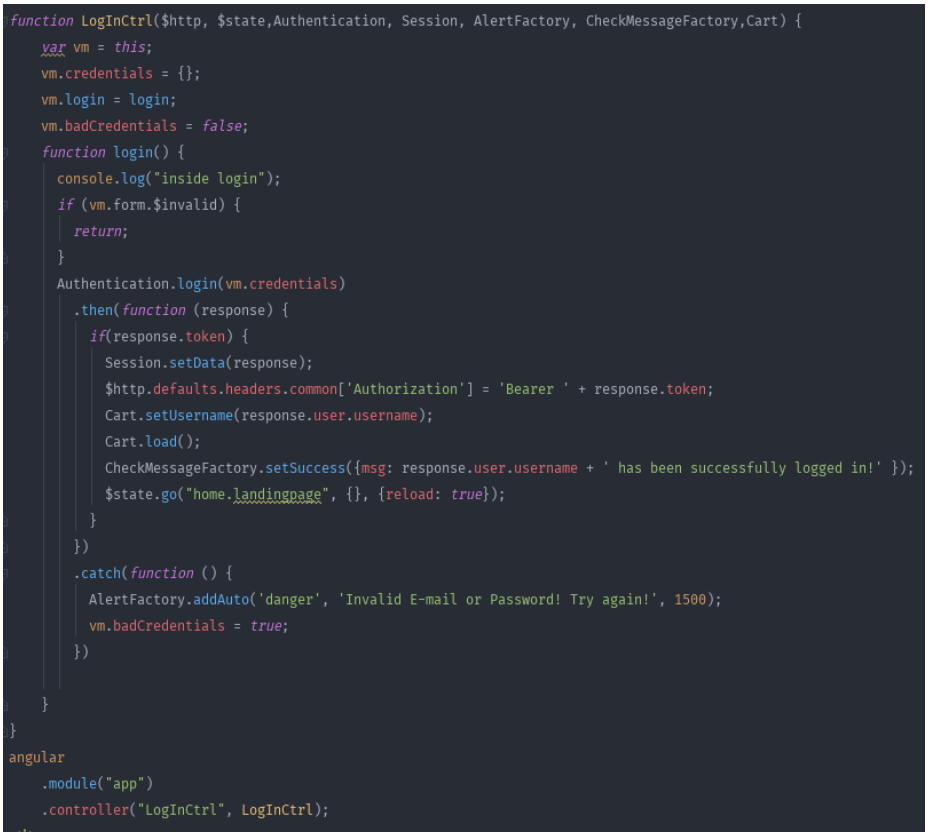
In the run phase we can use the services that were configured during the config phase as seen in Figure 5.7 below. At point 1 in the figure, the publishable key for the Stripe services is set. Based on this key, we will receive a token when the card payment method is used. The token will be sent to the Server application along with the amount that has to be paid so the card can be charged. This method is used so the card information is not passed between our two applications because of security threats. The Server Application will charge the card based on the token and amount received and also based on a secret token that is known only by the Server Application. At point 2 in the figure, the application listens for every state change the “ui-router” broadcasts. At point 3, the data present on the current Session which is actually the user data (username, e-mail, role, etc.) is saved in the “session” variable. At point 4, the state a user wants to go to is checked if it needs authentication, if it does and the user is not authenticated the user is redirected to the application main page, else the user’s role is checked to see if the user has authorization to go to that state, if the user has the necessary authorization, the state is changed, if not the user is sent again to the main page of the application.

```
function run($http,$window, Session, $rootScope, $state) {
  //Setting Stripe Test Account Publishable Key
  $window.Stripe.setPublishableKey('pk_test_9vDcmYidHzKfgq8n4ty0SKmr'); 1

  $rootScope.$on('$stateChangeStart', function(event, toState) { 2
    var session = Session.getData(); 3
    if (!session) {
      if (toState.name != 'home.landingpage'&&toState.name != 'home.login'&&toSta
        event.preventDefault();
        $state.go('home.landingpage', {}, {reload: true});
      }
    } else {
      $http.defaults.headers.common['Authorization'] = 'Bearer ' + session.token;
      if (toState.data && toState.data.role) {  5
        var hasAccess = false;
        var role = session.user.role;
        if (toState.data.role == role) {
          hasAccess = true;
        }
        if (!hasAccess) {
          event.preventDefault();
          $state.go('home.landingpage', {}, {reload: true});
        }
```

Figure 5.7 Run phase and run function executed during this phase

Controllers – The AngularJS controllers are the most useful of the components in an AngularJS Application. AngularJS controllers are composed of JavaScript functions and objects which perform the majority of UI related work and also make use of services injected in their heather to make requests to the Server Application through them. The controllers are like a gateway between the model (the data in our application) and the view (what a user sees on the screen and interacts with). Next the log-in-controller is presented as seen in Figure 5.8 below together with its functionalities, to serve as an example for a better understanding how the controllers in the application are implemented. The other controllers used in the application are declared similarly and each has its particular purpose to serve. Each state has its own controller. The controllers of the application that are worth mentioning are: home-controller, landingpage-controller, computer-details-controller, order-details-controller, login-controller, singup-controller and cart-details-controller.



```
function LogInCtrl($http, $state,Authentication, Session, AlertFactory, CheckMessageFactory,Cart) {
    var vm = this;
    vm.credentials = {};
    vm.login = login;
    vm.badCredentials = false;
    function login() {
      console.log("inside login");
      if (vm.form.$invalid) {
        return;
      }
      Authentication.login(vm.credentials)
        .then(function (response) {
          if(response.token) {
            Session.setData(response);
            $http.defaults.headers.common['Authorization'] = 'Bearer ' + response.token;
            Cart.setUsername(response.user.username);
            Cart.load();
            CheckMessageFactory.setSuccess({msg: response.user.username + ' has been successfully logged in!' });
            $state.go("home.landingpage", {}, {reload: true});
          }
        })
        .catch(function () {
          AlertFactory.addAuto('danger', 'Invalid E-mail or Password! Try again!', 1500);
          vm.badCredentials = true;
        })

    }
}
angular
    .module("app")
    .controller("LogInCtrl", LogInCtrl);
```

Figure 5.8 Log-In Controller

We can see the implementation of the Log-In Controller in Figure 5.8 above. The controller is declared as a function having the name (“LogInCtrl”) and registered in the module “app” as a controller with the same name, we can see this in the bottom of the picture, below the “angular” key word. The controller has in its heather injected the services and factories that are further used. The services injected are: $http, $state, Session. The factories injected are: AlertFactory, CheckMessageFactory and Cart. The difference between services and factories is discussed in the paragraphs that follow.

When a user requests to be logged in, the “login” function is called inside this controller. If the provided credentials are valid, the “login” function inside the “Authentication” service is called, which in turn makes a HTTP request to the Server Application to check if the credentials provided are associated to a registered account and also to generate a token that from now on its associated with the authenticated user and will be passed on each request between Client and Server applications to check the identity of the user. If the response from the Server contains a token, it means that the credentials were correct, we set the user data for the current session using “Session.setData()”, the token is then set on the “headers.common” to be passed at each request, the shopping cart items are loaded if the user has any, a success message is set using “CheckMessageFactory” to be later displayed on the page the user is redirected and then the user is redirected to the main page of the application (i.e. “home.landingpage” state). If the credentials provided by the user are not associated with an existing account, the user is kept on the current page (i.e. “home.login” state) and an error message is displayed to let the user know that the credentials are wrong. The application has a controller resembling this one for each state, having different implementation of course.

Services – AngularJS services are functions or objects that are used to do specific tasks. Services provide a way to store data during the lifetime of the app and communicate between AngularJS controllers in a consistent manner. Services are also used to fetch data from the database by calling the server application, because if we use controllers for this task, once we leave that controller the data is lost, while a service stores that and can pass it to multiple controllers by injection.

The AngularJS services can be defined by using the keywords "factory" or "service". The difference is that by using "service" keyword, the registered function is invoked as a constructor when called and this method is used when we want the service to pass the same data between controllers. If we use "factory" keyword, the registered function is invoked as a simple function when called and is generally used when we want to pass different data between several controllers.

AngularJS services are lazily instantiated, meaning that AngularJS services are instantiated only when an application component depends on them. Services are singletons, that means, every component dependent on a particular service gets a reference to the single instance generated by the service factory.

There exist about 30 built-in services in AngularJS. Such services that are used in the implementation of the application are: $http, $window, $timeout, $state, $scope, $rootScope. Besides these built-in services, custom implemented services are used, such as: "Authentication", used for user login to communicate with the Server Application or "Computer", for bringing product related data; these were defined suing "service" keyword. There are custom services defined using "factory" keyword such as: "AlertFactory", that displays a given message as an alert on the screen and also "CheckMessageFactory", used to send an alert message from a state to another state to be displayed using the before mentioned "AlertFactory".

Directives – AngularJS directives allow us to extend HTML with custom attributes and elements. There exist built-in directives such as "ng-if" that allow us to condition when an HTML block is displayed for example, or "ng-model" that let us bind a certain data model to HTML inputs, such an example can be seen in Figure 5.9 below.

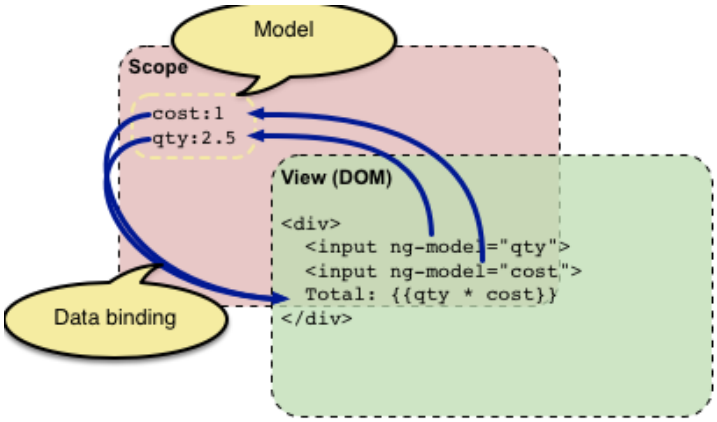


Figure 5.9 AngularJS ng-model directive

(source: https://docs.angularjs.org/guide/concepts)

A custom directive that was used in the current application is "matchPassword" directive, it is implemented for the sign-up form. There are two password inputs, one for the password and one for password confirmation, if the passwords inside the two inputs don't match, the user is prompted. Such a directive consists of a function that specifies the template and controller used that are going to

be sued when called. It also has the controller in the same place with this function. The controller is implemented like any other controller used inside this application. The directive is placed inside the HTML input tag like “match-password”, and when it is executed it actually executes the controller inside the directive.

The Client Application runs on top of a server that is configured inside the “angular-seed”[56] project that was used as a starting point to write the application.

5.2.3. Frequently asked questions chatbot

The chat box containing the frequently asked questions bot is embedded into the client application using and iframe HTML tag with code provided on the Dialogflow website. Iframe means inline frame and it is used to embed another document within the current HTML document, so basically by introducing the iframe provided by Dialogflow in our code we load into our website a piece of their website.

The frequently asked questions bot works by using the custom written intents which are processed by the Dialogflow tool using NLP (natural language processing) and ML (machine learning). An intent represents a mapping between what a user says and what action should be taken by the program. Besides intents the tool supports differentiation of user inputs based on context and entities present in their phrases and based on those the endpoints of a custom written API can be used to take certain action depending on the input provided to the bot.

The chatbot embedded into our client application only uses the intents to answer questions asked by the user. It has intents for the following cases: a user wants a refund, a user wants to cancel an order, a user want to know how to find order history, a user wants to speak with an employee, a user asks how the bot can be of help, a user asks how can he/she search for a product. The bot also benefits from the small talk feature of Dialogflow which helps the bot react to predefined phrases like “Where do you work?”, “How old are you”, “Are you real?” or “Where were you born?”.

The bot offers a great way of interacting with customers and can be easily upgraded by using webhooks to connect it to a custom API and trigger certain actions based on user input coming from the Dialogflow bot.

5.3. Server Application

The Server Application runs of Tomcat Server which is embedded in the container of the application by using Spring Boot and Spring Framework as described in. This application serves data in JSON format to the Client Application via HTTP responses to the HTTP requests made by the Client Application. It also has the responsibility of retrieving/storing data from/in the database. Next a high-level overview of this application is presented, followed by explanations regarding the implementation of its most relevant components.

5.3.1. High-level overview

For developing the Server Application, the Spring Framework is used along with Spring Boot as described in section 4.2.1 of chapter 4, for a faster configuration process of the project. This application is built using a layered architecture[57] as depicted in Figure 5.10 below. This architecture organizes the project structure into

[56] https://github.com/angular/angular-seed
[57] https://dzone.com/articles/layered-architecture-is-good

four main categories: presentation, service, domain and persistence, each with its corresponding layer. This architecture is chosen because it enforces the separation of concerns, each layer has its own tasks, so tasks that are specific for the persistence layer should not be placed in the presentation layer and so on (e.g. querying the database should not be done in the presentation layer). The separation of concerns results in low coupling and high cohesion between components, favorizing a higher maintainability and reusability of the code.

Next, each layer is described with regards to its concerns.

Presentation layer – This layer consists of mainly the classes responsible for receiving a request from the client and sending back a response. In the current application these classes are the Spring REST controllers which receive the request from the client and further pass it to a corresponding service in the service layer to handle it, then they send the data received from the service back to the client.

Service layer – This layer consists of services that contain all the logic that is needed to fulfill the functional requirements. These services are accessed by the controllers in the layer above to handle a request and they make use of the domain and persistence layers below in order to fulfill their job.

Domain layer – It is represented by the domain entities or models that are used in the application.

Persistence layer - This layer contains the repository interfaces that extend the Spring JPA CrudRepository and have the task of persisting data by communicating with the database. In less words the interfaces in this layer perform CRUD (Create, Read, Update, Delete) operations on the database when requested to do so. The HTTP request/response and the database are depicted in the picture below just to facilitate this explanation; the Server Application is represented only by the four layers.

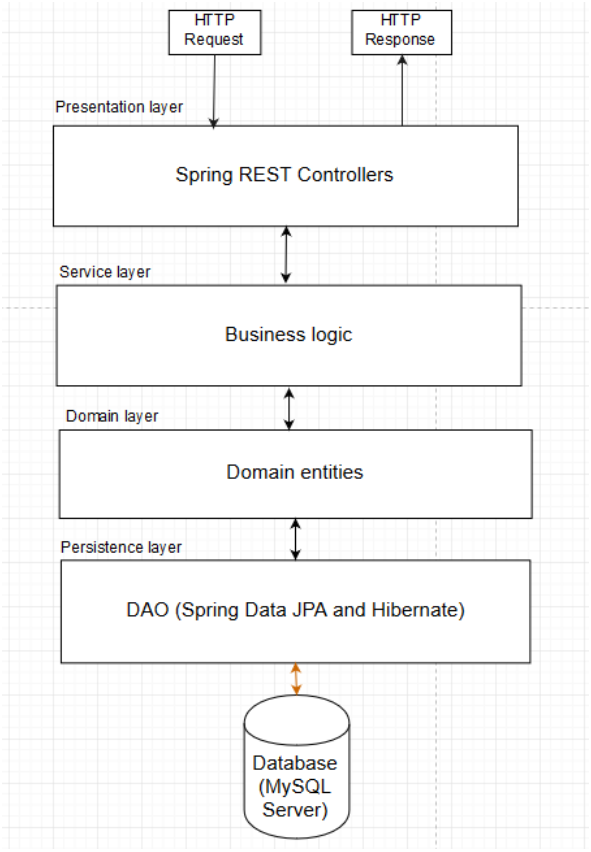


Figure 5.10 Layered Architecture of the Server Application

5.3.2. Detailed description of the components

In this subchapter, first, a package diagram, which represents the relationships between the high-level components of the Server Application is presented and explained and then, components from each package, such as controllers, services, repositories and so on are described in more detail.

The most important packages from the Server Application are: webstore.controller, webstore.security.config, webstore.exception, webstore.service, webstore.service.Impl, webstore.model, webstore.payment_component and webstore.repository.

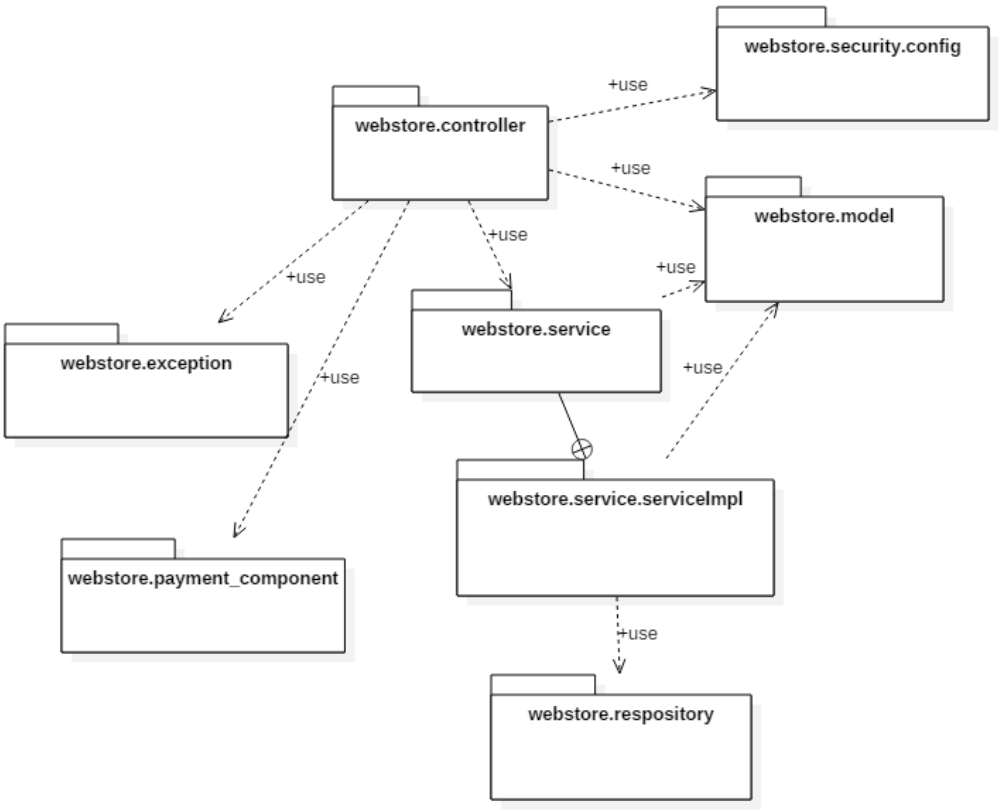


Figure 5.11 Server Application package diagram

An overview of the concerns of each package depicted in Figure 5.11 above are described next:

Webstore.security.config - This package contains the logic responsible for the security of the application. It contains the logic for authentication and authorization using JWT tokens and Spring Security.

Webstore.controller – This package contains the logic necessary for receiving requests and sending them to the appropriate service to handle and then for sending back the appropriate response.

Webstore.service – This package contains interfaces that are implemented by the actual services found in the webstore.service.serviceImpl package. These interfaces are used by controllers in the webstore.controller package to call specific services that implement certain functionalities.

Webstore.service.serviceImpl - This package contains most of the application's logic which is necessary to fulfill the functional requirements. It holds the services that are called by controllers in webstore.controller package through the

service interfaces. It makes use of interfaces present in the webstore.repository package when a certain service needs to handle persistent data.

Webstore.repository – This package contains the interfaces that facilitate the communication with the database, it makes use of the models (entities) present in the webstore.model package.

Webstore.model – This package contains all the models (entities) of the application which are mapped on database tables facilitating persistence.

Webstore.exception – This package contains the logic used to throw custom exceptions when needed.

Webstore.payment_component – This package contains the logic that is needed to charge the credit card via Stripe services.

Next are described implementation details of the components found in the packages presented earlier.

Controllers

The controller classes of the current application are found in the webstore.controller package. The controllers make the presentation layer and are a way of exposing the server application's resources as a RESTful web services to other applications, in this case the client application. So, basically, what the controller classes do is: they receive a HTTP request in this case from our client application, they pass the request to a specialized class (i.e. service) to handle it and then they send back the response via a HTTP response for the client to display to the user. This is made possible by using several Spring annotations, one of the most important being "@RestController", which also gives the name of Spring Restful Controllers to these classes. Some examples of controllers in our application are: UserController, PaymentController and ComputerController. Following the most used annotations and how a controller class is defined will be discussed based on Figure 5.12 below.

```
@RestController
@RequestMapping("/computer")
public class ComputerController {
    @Autowired
    private ComputerService computerService;

    @RequestMapping(value = "/getByCategory/{computer_category}", method = RequestMethod.GET)
    public ResponseEntity<?> getAllComputersByCategory(@PathVariable("computer_category") String category) {
        try {
            List<Computer> computers = computerService.getAllComputersByCategory(category);
            return new ResponseEntity<>(computers, HttpStatus.OK);
        } catch (Exception e) {
            FrontendMessage fem = new FrontendMessage();
            fem.setMessage(e.getMessage());
            return new ResponseEntity<>(fem, HttpStatus.NOT_FOUND);
        }
    }
```

Figure 5.12 Spring Restful Controller

The Spring annotations give the controller the desired behavior.

- @RestfulController – used at the class level to mark a class as a controller where every method returns a domain object instead of a view like in the case of the older MVC Frameworks. Such a class can now receive/send HTTP requests/responses.
- @RequestMapping – this annotation can be used both at class level and method level as we can see in figure 5.12 above. It is used to ma web requests onto specific handler classes and methods. When used at class level, it creates a base URI for which that controller is used and when used on a specific method it will give the URI for which that method will be executed. This annotation also supports as parameter the HTTP request method, in the figure

above we can see a GET request; other types of requests used in the current application are DELETE, PUT, POST.

- @Autowired – this annotation can be applied on fields, setter methods and constructors. In our case it is applied on fields only and it injects object dependency implicitly, so when we want to use a service or repository class we annotate it like this and Spring will do a scan in the project and inject what is required where the annotation is found.
- @PathVariable, @RequestBody are used to annotate the arguments for the request handler methods. As we can see in the figure spring will bind the URI parameter "computer_category" to a Java String named "category" which is further used when calling the service.

The "ResponseEntity" which is the return type for our handler method in Figure 5.13 represents the entire HTTP response. As we can see the response entity encapsulates an object and a status code which allows the client application to determine if the request was a success (OK – status 200), or if it failed in this case the requested data was not found (NOT_FOUND – status 400).

Services

Services are the components that make up the service layer, they are found in the webstore.config.service.serviceImpl package. Services hold all the business logic required to meet the functional requirements of the application. In Spring services are annotated with "@Service". Services are directly used by the controller handler methods to fulfill a request.

The "@Service" annotation is used at a class level and is used to mark a Java class that performs services as a bean so the component-scanning mechanism of Spring can find it and pull it into the application context when the application is run. It has the same behavior as the "@Component" annotation, they are in fact interchangeable but @Service is used in the service layer because it specifies intent better.

Some of the main service classes of our application are: UserServiceImpl, ComputerServiceImpl, OrderServiceImpl, ComputerOrderServiceImpl.

Domain models

Models or entities are found in the package webstore.model and they make up the domain layer. They serve as skeletons for the objects used in the whole application, each has specific attributes. The models are mapped to database tables to facilitate persistence of data using Hibernate annotations and they are also used for sending data between the client and server applications.

Following is a description of the main Java models created for the current application:

1. User – The user model is concerned with all the user related data, used to identify a user and used for account creation and later for authentication and authorization. It has attributes for holding personal data about a user required for placing and order and creating an account, data like: first name, last name, username, phone number, E-mail address, billing address, delivery address, account creation date, last login and role. The role attribute is used to differentiate between the two different types of users: administrator and registered user (or customer).
2. Order – This model is concerned with all the data details regarding an order to be received, manipulated then persisted. It defines attributes such as the customer's: first name, last name, addresses, phone number, E-mail address, the date the order was placed, payment type etc.

3. ComputerOrder – This model was implemented because of the need to associate computer (product) model with a certain order model, used further in the shopping cart implemented on the client application. It defines attributes like ids of order and products and the quantity of each product present in the shopping cart so the cart can be loaded with the correct items and their quantities each time a user authenticates.

Other models defined in our application are: Category, FrontendMessage, Invoice, Mail, PasswordResetToken, Review, SpecialOffer, Supplier, Wishlist. They will not be described because they are similar with the ones presented above, these models are just Java classes with specific attributes and each have getter and setter methods. Their attributes are annotated using Spring annotations so they are ready to be mapped to database tables and persisted. There are annotations that allow us to specify the corresponding database data types, relationships between entities (@OneToOne, @OneToMany and @ManytoMany), primary key and so on.

Repository Interfaces

The repository interfaces are found in the persistence layer of the application and inside the webstore.repository package in our application. The repository interfaces help reduce the amount of code that we need to write in order to perform CRUD (Create, Read, Update, Delete) operations on the database tables and prevents us from writing duplicate code. Such a repository interface is depicted in Figure 5.13 below.

```
public interface UserRepository extends CrudRepository<User, Integer> {

    User findByUsername(String username);

    User findByEmail(String email);

    @Query(value = "SELECT users.* FROM users where users.role = :userRole", nativeQuery = true)
     List<User> findAllCustomers(@Param("userRole") String userRole);

}
```

Figure 5.13 repository Interface for User

By making our custom interface extend JPA's CrudRepository interface we immediately have access to basic methods like: delete, findAll, findOne and save so we don't need to write any extra code. We just have to specify the type of the entity or model the repository is used for (e.g. User) and the type of the entities id field (e.g. Integer).

If we want we can use query methods based on certain keywords like "findByUsername" and the interface knows to look up a user based on the username provided as a parameter or we can write customer queries by using "@Query" annotation like the one used for "findAllCustomers" method in the figure above. All the other repository interfaces are defined the same for all the models in our application, except for the Mail, FrontEndMessage and Category models which are used as helpers and are not persisted in the database.

Security

The classes responsible for the security of the application are placed in the webstore.security.config package. There are two classes "JWTFilter" and "WebSecurityConfig". The "JWTFilter" class makes use of JWT tokens and Spring Security to facilitate the authentication and authorization process of a user based on credentials and tokens. The "WebSecurityConfig" class specifies the configuration of the authorization and authentication process. Things like endpoints that can be

accessed to retrieve data for display to unregistered, the cross-origin resource sharing configuration and the password encoder configuration are defined here.

Following is a short explanation of how the authentication and authorization process works: When a user logs in, their credentials are verified against the existent credentials in the database, if they match the credentials of an existing registered user, a JWT token is generated and response containing that token is sent to the client application together with certain information associated with the authenticated user like username, role, E-mail but not the password. The client application receives the response, authenticates the user and saves the token and role of the user and based on them it authorizes the user to access certain pages of the application. From now on every request made by this user will contain the token generated initially inside the server application and based on that token and role the application decides if the user has the authority to access certain endpoints. When the user logs out of their account, the token is destroyed. When logging back in a new token is created and so on. The information inside the token is signed using the HMAC-SHA256 algorithm (HS256) and a secret key. An explanation of how the HMAC algorithm works can be found here[58] and for the SHA256 which is a hashing algorithm an explanation can be found here[59].

Payment

The application uses the Stripe payment service described in section 4.10 of chapter 4. The Server side of the payment works in the following way: after the client application obtains a token for the card information that were introduced, the token and the amount that needs to be charged are sent to the payment controller which passes the request to the "StripeClient" component depicted in Figure 5.15.

```
@Component
public class StripeClient {
    @Autowired
    StripeClient() {
        Stripe.apiKey = "secret_key";
    }

    public Charge chargeCreditCard(String token, double amount) throws Exception {
        Map<String, Object> chargeParams = new HashMap<String, Object>();
        chargeParams.put("amount", (int)(amount * 100));
        chargeParams.put("currency", "USD");
        chargeParams.put("source", token);
        Charge charge = Charge.create(chargeParams);
        return charge;
    }

}
```

Figure 5.14 Stripe Client component

The card is charged based on the secret "api_key" received upon creating a Stripe account and it comes in pair with a publishable key that was used in the client application to obtain the token. The "Stripe" and "Charge" classes are provided in the "java-srtipe" library imported in the project using Maven. The create method of the "Charge" class receives as parameter a map containing string, object pairs. The currency is set as USD and because Stripe charges in cents we multiplied the amount that needs to be charged by 100. The token received from the client application is set as the source. After the card is charged we can see that the card was really charged in in the test console in our Stripe account. This token can be used to make only one charged and afterwards is destroyed.

[58]https://en.wikipedia.org/wiki/HMAC
[59] https://en.wikipedia.org/wiki/SHA-2

Mail

Personalized E-mails are sent upon account creation, order completion and order cancellation using JavaMail[60] API, the SMTP[61] protocol, Mail model and the Apache FreeMarker[62] template engine. The JavaMail configuration is created in the "application.properties" file located in the resources folder as depicted in Figure 5.15 where the FreeMarker template also resides.

```
spring.mail.default-encoding=UTF-8
spring.mail.host=smtp.gmail.com
spring.mail.username=username
spring.mail.password=password
spring.mail.port=587
spring.mail.protocol=smtp
spring.mail.test-connection=false
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Figure 5.15 Java Mail configuration

The FreeMarker template engine enables us to write HTML and CSS in a file with ".ftl" extension and allows us to embed in the HTML customer variables declared by using the following notation: "${variable_name}" . Before the E-mail is sent we set those variables in Java and then the template engine compiles the ".ftl" file and replaces all those variables with the appropriate values, so when the mail is sent it will be personalized with data particular to the user that it is sent. A part of the template used can be seen in Figure 5.16 below.

```
<table align="center" border="0" cellpadding="0" cellspacing="0" width="600
    <tr>
        <td align="center" bgcolor="#777777" style="padding: 40px 0 30px 0;
            <h1 style="color: deepskyblue;">AG-Computers</h1>
        </td>
    </tr>
    <tr>
        <td bgcolor="#eaeaea" style="padding: 40px 30px 40px 30px;">
            <p>Dear Mrs./Mr. ${name},</p>
            <p><b>${emailContent}</b></p>
            <p>${productList}</p>
            <p>${orderTotal}</p>
        </td>
    </tr>
    <tr>
        <td bgcolor="#3bc5f7" style="padding: 30px 30px 30px 30px;">
            <p>${signature}</p>
            <p>Contact us at: sales@AG-Computers.ro OR +40751092154</p>
            <p>${location}</p>
        </td>
```

Figure 5.16 FreeMarker template

Class diagrams

The class diagram of the main models used for the Server application is depicted in Figure 5.17 below. The reason why it is presented separately is because all the classes of the application would not fit into one photo.

[60] https://en.wikipedia.org/wiki/JavaMail
[61] https://ro.wikipedia.org/wiki/SMTP
[62] https://freemarker.apache.org/index.html

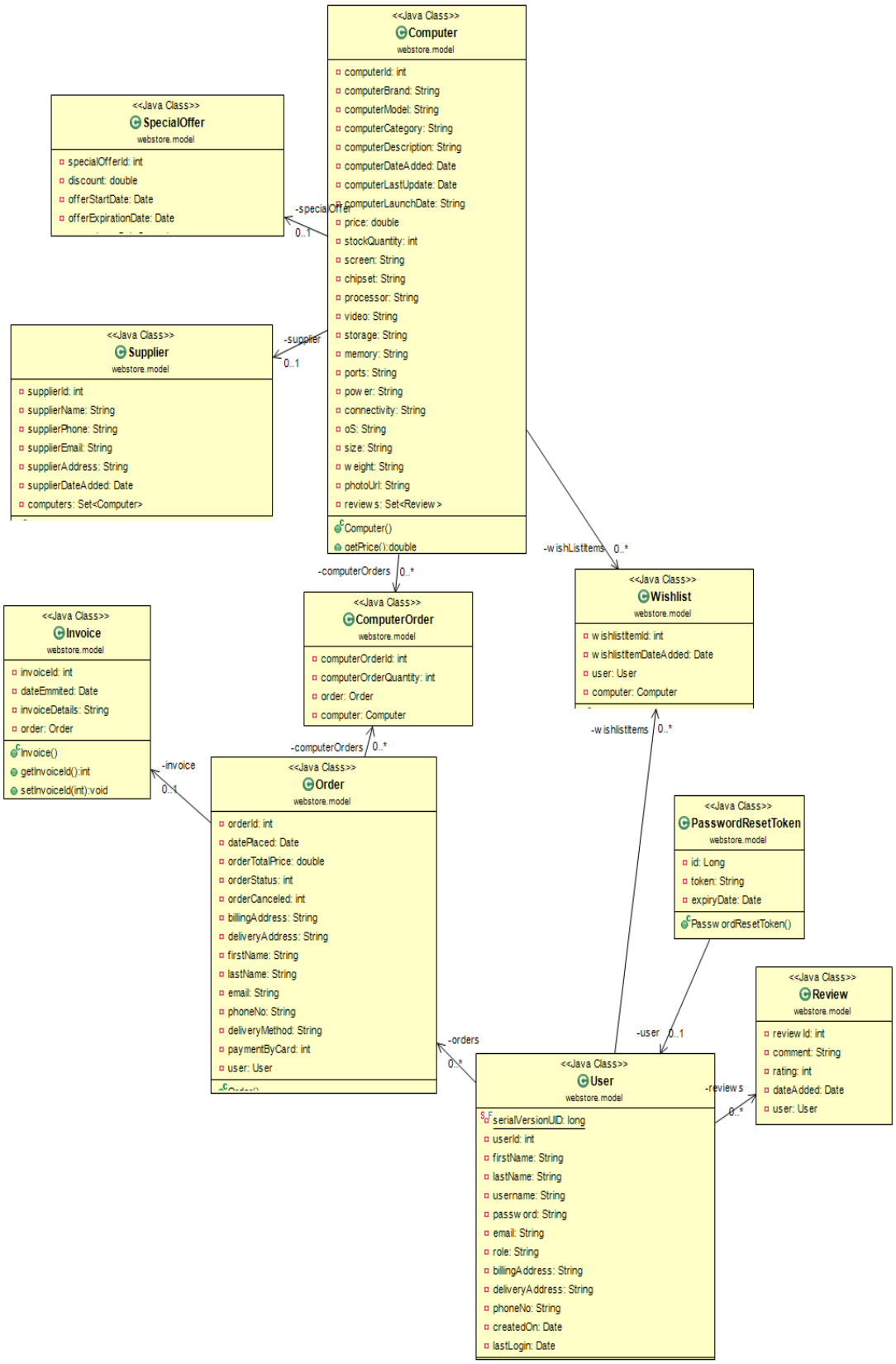


Figure 5.17 Model Class Diagram

Next the class diagram containing the main controllers and their relationships with the services used to handle a request is depicted in Figure 5.19 below. Their attributes and methods are not shown because they could not fit in a single picture.

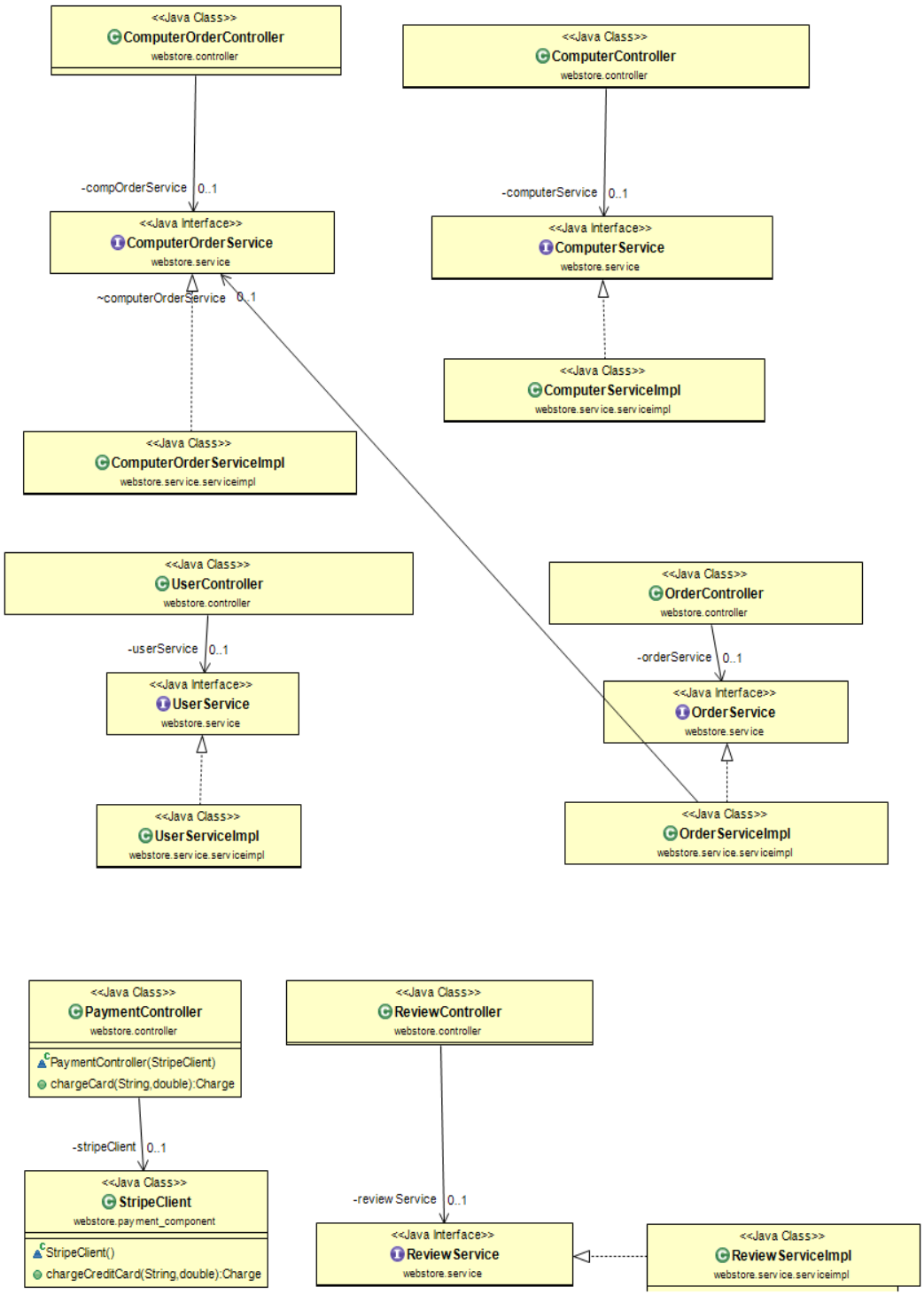


Figure 5.18 Controllers and services Class Diagram

5.4. Checkout sequence diagram

The sequence diagram for the checkout process is depicted in Figure. 5.19 below, followed by a short description. The preconditions to proceed to checkout are: a user must be authenticated, authorized and have at least one item in the shopping cart.

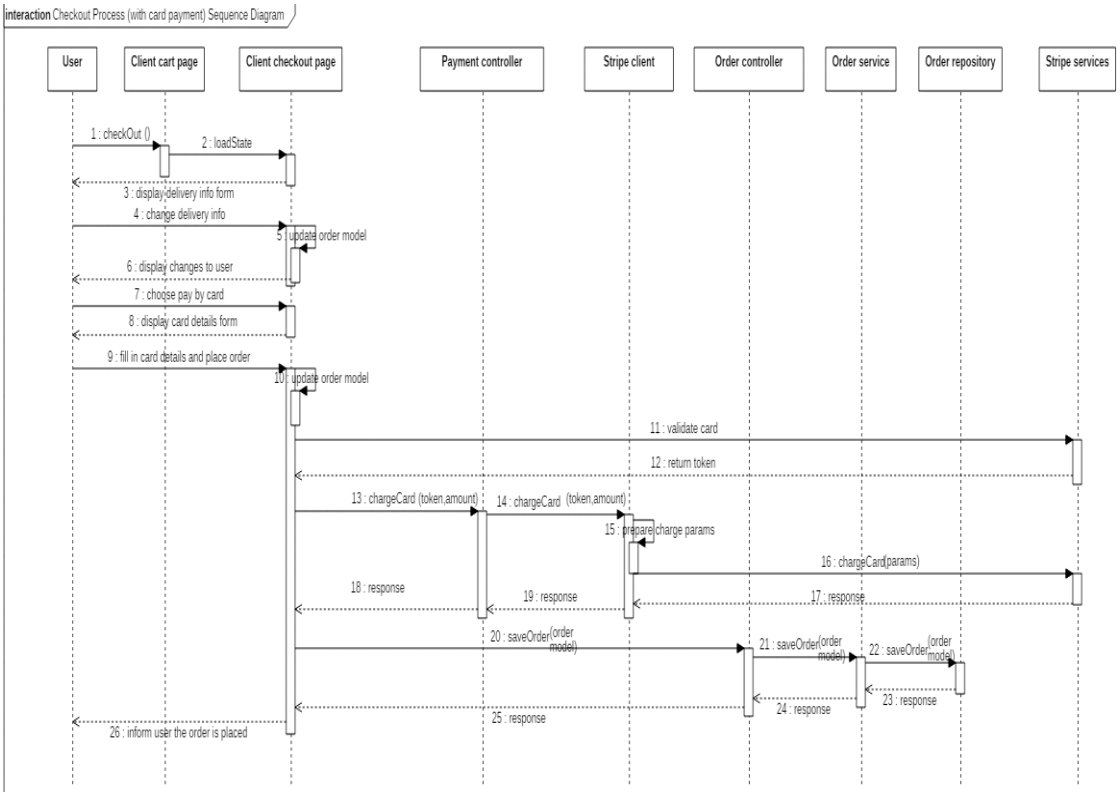


Figure 5.19 Checkout Sequence Diagram

The sequence diagram of the checkout process as seen in Figure 5.19 illustrates the success scenario of a user placing an order and paying by card, there is also the option for the user to choose cash on delivery as a payment method in which case the steps that correspond to calling the payment controller and Stripe services are skipped.

The process starts with the user being on the shopping cart page. At least a product must be in the shopping cart. The total is already computed when the customer chooses the checkout action. The checkout action takes the user to the checkout page. First thing displayed on the checkout page is a form with the delivery information like: user's first name, last name, e-mail, billing address, phone number. This delivery information form is already filled in with the details the user provided when creating an account. The user can modify it, in which case the controller corresponding to this page updates the order model with the new information. The user chooses next the pay by card action and he/she is sent to another tab on the same page. The new tab contains a form that needs to be filled by the user with credit card information. After the user fills in the credit card information he chooses the place order action. The page controller sends a request to Stripe services to validate the card information. The card information is validated and as response the controller receives a token. The page controller makes a call, having the token and the total amount that needs to be paid as parameters, to the server application's payment controller. The payment controller calls the stripe client service which in turn make a call to the Stripe services to charge the card based on the token and the order total. A response is sent back to the client application which makes a call to the order controller on the server application. The order controller calls the order service that makes use of the order repository to persist the order details. A response is sent back to the client application and the user is informed that the order has be registered successfully.

5.5. Database diagram

In this subchapter the database is described based on the illustration from Figure 5.20 below.

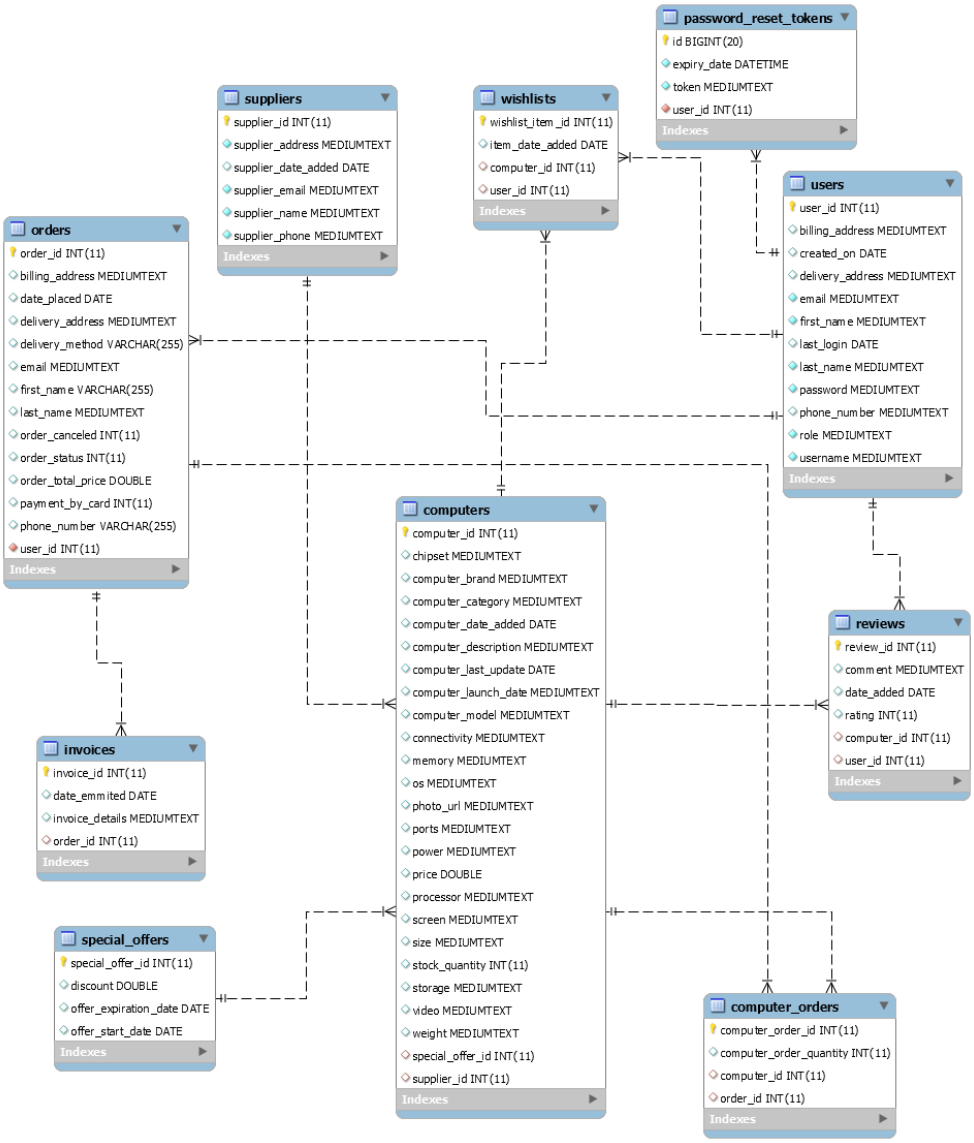


Figure 5.20 Database Diagram

The database used for storing persistent data for our application is a relational one and as a Relational Data Base Management System or RDBMS MySQL Server 5.7 is used. Relational database management systems and relational databases are described in chapter 4. A RDBMS helps us store and organize data in tables and inside tables in rows and columns and also allows us to have relationships between this tables by using primary and foreign keys. The relationships between tables allow us to refer to a whole table inside another table preventing us from storing duplicate data and allowing us to retrieve data faster. MySQL server is open-source so it is free to use.

Next, each table is described shortly:

- The computers table, stores all the information about a computer product such as brand, model, launch date, memory, storage etc. It also holds references to the special_offers and suppliers table.

- The users table stores all the information about a user's account such as: username, email, password, role, first name, last name. This information is stored when a user creates an account and is later used for delivery details when placing an order or when logging in. This table is updated if the user requests to edit personal information.
- The computer_orders table represents the items present in a user's shopping cart and stores a reference to the orders table and to the computers table along with a quantity value. When a user adds a new product to the cart or modifies the quantity of a product present in the cart this table is updated. This table connects the computers and orders tables through a Many-to-many relationship. This means that a computer can be in more shopping carts and that a shopping cart can have more computers inside.
- The orders table store information about details of an order such as delivery information (addresses, E-mail, phone number, name of the user etc.) and information about how the payment was made or when the order was placed. The orders table also acts as a shopping cart for the user by having the items from computers table associated with it via the computer_orders intermediary table.
- The wishlists and reviews table both hold references to the users and computers table. The wishlists table represents the wish list of a user where the user can store one of each product in the computers table. The reviews table stores information such as comment and rating a user posted for a computer.
- The suppliers table stores contact information regarding the supplier of certain computers.
- The password_reset_tokens table store a reference to the user, a string which represents the token and an expiry date of the token. It is used for the password reset process.
- The invoices table stores the date when an invoice was issued and details about the order such as the products, total price etc. A single invoice is issued for each order and when this happens the information is stored here.
- The special offers table stores information needed to apply a discount to a computer (product). The information stored is the discount that has to be applied and the dates when the special offer starts and when it ends.

The relationships used for our database are: One-to-many and Many-to-many. These relationships are achieved by using the primary key of a table(PK), which is in our case the id of each table, as a foreign key(FK) in another table.

An example of One-to-many relationship is that between special-offers table and computers table. The PK "special offer id" is used with the same name as a FK in the computers table. This means that more computers can have the same discount.

An example of Many-to-many relationship is the one between computers and orders in which the auxiliary table "computer_orders" is used. Both computers table and orders table are in a One-to-many relationship with the "computer_orders" table, as the one described above. Those to One-to-many relationships result in a Many-to-many relationship between computers table and orders table.

The database is normalized in accordance to the three normal forms in order to reduce data redundancy and improve data integrity.

The tables respect the first normal form by not having multivalued columns or repeating groups.

All the tables in the database have only single-field primary keys, this means they respect the second normal form which says that every non-key field must depend on the entire primary key and not on part of a composite primary key.

The tables in the database respect the third normal form because there aren't any non-key fields that depend on other non-key fields.

5.6. Deployment diagram

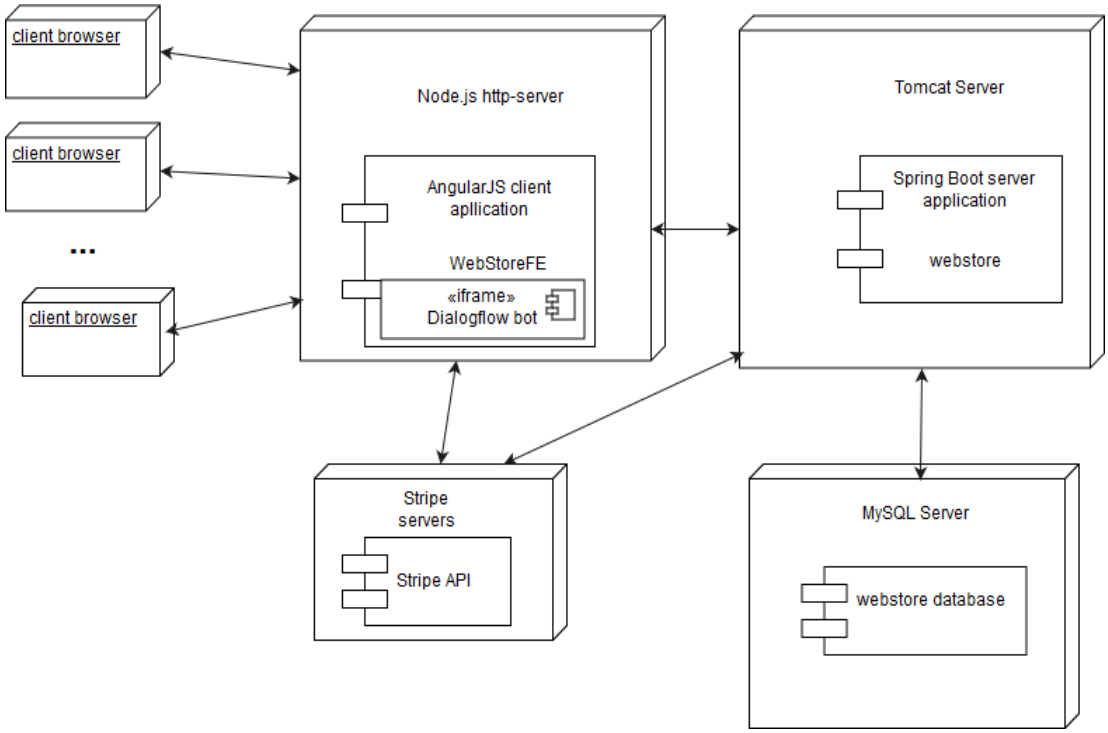


Figure 5.21 Deployment Diagram

In Figure 5.21 above the deployment diagram of the application is depicted. Each component runs on its own server. The client application embeds the Dialogflow bot and both the server and client applications make calls to the Stripe API which runs on Stripe servers.

Chapter 6. Testing and Validation

This chapter describes the methods used for testing and validation of the developed application. These methods help us determine if the system works as expected and satisfies the specified requirements. For testing the proposed application, the manual or black box testing method was used in the form of writing test cases and checking if every specified action holds the expected result. Other methods used for testing the application imply using tools such as Postman[63] which automate the process of testing in order to verify if the server application's endpoints meet the required functionality and if they return the correct data so we know how to receive and process the data in the client application. Other tools like Google Chrome's DevTools[64] which are built in the Chrome browser are used to debug the client application, both the JavaScript code and the HTML and CSS templates.

All the functionalities were tested, but here the test cases for the two most relevant features of the online store are presented: shopping cart and placing an order which includes using an online payment service. Also, a test case to check if the administrator account works as expected, meaning that a customer shouldn't see the link for the administrator dashboard on the navigation bar and should not have access to the pages that are meant only for an administrator account.

6.1. Test cases

The test cases fall under the manual testing category. The person that tests doesn't have access to the source code so the testing is rather made from the perspective of an end-user, this is also called black-box testing. The functionalities of the application are verified through the GUI, some actions are taken and there is an expected result for each action that the application should produce.

The test cases are sets of conditions or variables which are used by the person that performs the tests to determine if the application satisfies the requirements and works correctly. This process of writing test cases can sometimes help spotting problems in the design or requirements of the application. Test cases must be written in such a way that we test only one thing at a time, also positive and negative scenarios must be both covered. The test cases should be written in a simple and easy to understand language and contain exact and consistent names for forms, fields and so on.

Following is an explanation of the terms used in writing the test cases:

- Test case: the name of the functionality that we are testing.
- Preconditions: conditions that need to be fulfilled before executing the test case.
- Postconditions: the state we get if the test case steps are executed successfully.
- Actions: the steps that need to be executed in order to complete a test case.
- Expected result: the result that we should get after each action executed.
- Result: the actual result that we get after executing an action. To avoid duplicating the expected results columns the result columns are marked

[63] https://www.getpostman.com/
[64] https://developers.google.com/web/tools/chrome-devtools/

with “OK” if the result coincides with the expected result, else the actual result obtained is written.

6.1.1. Test case: view contact information for registered users.

Preconditions: User must have a registered administrator account.
Postconditions: For this test case there are no postconditions.

Table 6.1 Test case: view contact information for registered users

No.	Action	Expected result	Result
1	Open the application	The landing page is displayed	OK
2	On the navigation bar, select Log in from the ACCOUNT dropdown	The log in page is displayed containing a log in form with E-mail, password inputs and a log in button	OK
3	Fill in the inputs with wrong credentials and press the log in button	An error message informing the user that wrong E-mail or password was introduced is displayed	OK
4	Fill in the inputs with correct administrator credentials and press the log in button	The user is redirected to the landing page with a success message and the admin dashboard link should appear on the navigation bar	OK
5	Click on the admin dashboard link on the navigation bar	The user is redirected to the admin dashboard and a table with registered users and their information is displayed	OK

6.1.2. Test case: add item to shopping cart

Preconditions: User is authenticated, authorized and the product is not in the shopping cart already.

Postconditions: The application saves the shopping cart state in the database so, it is persisted even after logout.

Table 6.2 Test case: add item to shopping cart

No.	Action	Expected result	Result
1	On the landing page click on the item you wish to add to cart	The product details page is displayed with all the details about the product	OK
2	Click the add to cart button present under the description of the item	A success message is displayed and the badge showing the number of items in cart is updated	OK
3	On the navigation bar, click the shopping cart icon the choose shopping cart details	The shopping cart is displayed with all the items and the added item should be here	OK

6.1.3. Test case: edit shopping cart details

Preconditions: User is authenticated, authorized and there exists at least one product in the shopping cart.

Postconditions: The application saves the new quantities and the delivery method chosen so the shopping cart is persisted in case the user logs out.

Table 6.3 Test case: edit shopping cart details

No.	Action	Expected result	Result
1	On the navigation bar click the shopping cart icon and choose shopping cart details	The shopping cart page is displayed with all the items added	OK
2	From the dropdown under the quantity row choose 2	The quantity of the item should be updated, also the total price and item subtotal should be updated	OK
3	On the order summary tab choose premium delivery	The total price and the delivery method should be updated	OK
4	Remove an item	The item disappears and the item subtotal and the total price should be updated	OK

6.1.4. Test case: place order

Preconditions: User is authenticated, authorized and there exists at least one product in the shopping cart.

Postconditions: The application charges the card. The application saves the order details in the database so they can be used later.

Table 6.4 Test case: place order

No.	Action	Expected result	Result
1	On the navigation bar click the shopping cart icon and choose shopping cart details	The shopping cart page is displayed with all the items added	OK
2	From the bottom of the page click the checkout button	The user is redirected to the order details page and a tab that has a form containing delivery information fields already filled in with the information provided at sign up is displayed	OK
3	Modify the information in the billing address field	The field is updated with the new value	OK
4	Under the delivery information form click the pay by card button	The user is sent to the pay by card tab and a card information form is displayed having the inputs: card number, expiry-date, CVC	OK

5	Fill in the inputs with the test data. For card number: “4242 4242 4242 4242”. For expiry date: any date that is greater than the current date. For CVC: “123”	The inputs are immediately updates with the new values	OK
6	Click the place order button below the card information form	An information message saying that the card is verified should be displayed. The user is redirected to the landing page and the shopping cart is emptied.	OK

6.2. Testing using tools

There exists the possibility of automating tests like the ones mentioned before in the presented test cases by using tools or frameworks.

Some of the popular testing tools which can be used for automating tests on a Java web application are described next:

- JUnit – allows developers to write unit tests. This requires access to the source code, and each unit of code, such as a method is tested separately to see if it meets its requirements. Junit can be easily integrated with Spring Tool Suite, the IDE used for developing the server application.
- Mockito – it is a mocking framework for Java classes that enables the developer to test the code in isolation without any dependencies, it is best used with Junit.
- REST Assured – is a tool that lets us perform integration tests on REST API’s.

Popular testing tools for AngularJS:

- Karma – provides easy debugging directly from WebStorm IDE that was used for developing the client application.
- Angular Mocks – provides support for injecting and mocking Angular services in unit tests.
- Protractor – an end to end test framework for AngularJS. It launches the application is a browser and interacts with it using Selenium, which in turn is a testing framework for running automated test on the UI (user interface)

The tool used for testing the proposed application is DevTools integrated directly into the Google Chrome browser. DevTools can be opened by pressing F12 while in the browser or by right clicking and choosing, inspect. DevTools allows us to place breakpoints, run the code line by line, use the console to access values of certain variables, or see how much time it takes for a request to the server application’s endpoints to receive a response. This tool was used while developing the client application.

The tool used to test the server application’s endpoints is Postman. Postman lets us make requests to a RESTful API’s endpoints, it only requires that we provide the endpoint URL and the request body or parameters if the case. It supports the types of HTTP requests used for our application (POST, DELETE, PUT, GET) and many more. Postman was used to test each individual endpoint and it enables us to see the

response in JSON format so we know how to deal with it when received by the client application, it also enables us to see the status of the response, the time waited to get a response and the size of the response.

A request for getting all the computers (products) at a special offer from the database using Postman is depicted in Figure 6.1 below. We can see the URL of the endpoint marked with 1, marked with 2, we can see the status of the request, the time took to get a response and the size of the response and at 3 we can see the response body which represents the requested items in JSON format.

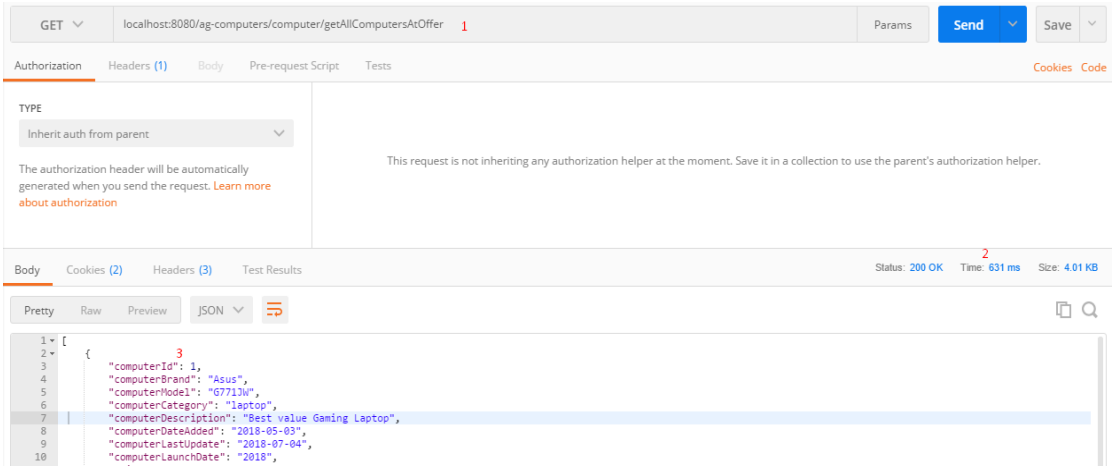


Figure 6.1 Postman request

Chapter 7. User's manual

This chapter presents the hardware and software resources needed for installing and running the application and a step by step guide of how the application can be installed and run. Then a user's guide is presented which describes how the main features of the application can be used, offering instructions accompanied by images for a better understanding.

7.1. Hardware and software resources

7.1.1. Hardware resources

For running the software that is presented in the next sections the machine used should meet the following minimum requirements:

- Free storage: 3 GB
- CPU: Intel Core i3 4th Gen, 2.4 GHz, or an AMD equivalent
- GPU: Integrated graphics card
- Memory: 4 GB RAM

7.1.2. Software resources

The application was developed and tested using the Windows operating system, but the application and other software used for installing and starting it should work without problems on Linux or Mac OS X

The machine used for running the application should have installed one of the following browsers: Google Chrome, Mozilla Firefox. The application was tested only on these two browsers but it should work without problems also with Safari, Opera or Microsoft Edge.

Other software resources needed to be installed for getting the application up and running are:

- Java SE Development Kit[65], the link is in the footer, choose the version corresponding to your operating and your operating system's architecture.
- Spring Tool Suite[66] (STS), which is an IDE (Integrated Development Environment) for developing Java Spring applications, the link can be found in the footer, download the version corresponding to your operating system. Other IDEs such as Eclipse or IntelliJ IDEA should work as well.
- MySQL Server 5.7 and MySQL Workbench, which correspond to the server on which the database resides and a GUI tool for managing the database. These two programs can be installed using MySQL Installer[67], the link is provided in the footer, choose the version corresponding to you operating system and then follow the instructions provided by the installer.

[65] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
[66] https://spring.io/tools/sts/all
[67] https://dev.mysql.com/downloads/mysql/5.7.html#downloads

- Node.js[68] which is needed for the client application that is developed using JavaScript.

7.2. Installing and running the application

The first steps are to ask for access rights to the Bitbucket repositories where the client and server applications are uploaded and clone each one in a separate directory on the local machine using GIT Bash[69] for Windows by typing in the console "git clone repository_url" or other tools from this list[70] which better suites your operating system or preferences.

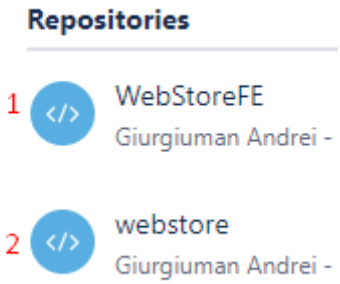


Figure 7.1 Bitbucket repositories

In Figure 7.1 we can see two repositories, the one marked with "1" is the repository that contains the client application and the one marked with "2" is the repository that contains the server application. Clone each repository to a directory having the same name using one of the git tools mentioned above.

7.2.1. Client application setup

Using the command line navigate to the directory "WebStoreFE" which contains the client application source code and run the following commands in order to install all the required dependencies: "npm install", press enter then type "bower install".

7.2.2. Server application setup

Open the STS IDE or one of the IDE's mentioned in section 7.1.2 and import the server application project from the "webstore" directory where the project was cloned. To import the project into the STS IDE, follow these steps: from the menu choose File, then choose Import, then choose Existing Maven Projects, then select Browse and go to the location of the "webstore" directory, select the directory and choose Finish. Now to make sure there will not be any issues regarding the dependencies right click on the imported project and choose Maven, then Update Project and then click Ok.

Next the database and JavaMail configuration must be modified. The configuration is found under the directory "src/main/resources" and the file which hold the configuration is named "application.properties". If it is opened the file should look like in Figure 7.2 below.

[68] https://nodejs.org/en/
[69] https://gitforwindows.org/
[70] https://git-scm.com/download/gui/windows

```
spring.datasource.url:jdbc:mysql://localhost/webstore?useSSL=false
spring.datasource.username:your_username
spring.datasource.password:your_password
spring.datasource.driver:com.mysql.jdbc.Driver

spring.jpa.properties.hibernate.dialect:org.hibernate.dialect.MySQL5Dialect
logging.level.org.hibernate.SQL:debug
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update

server.port=8080
server.address=127.0.0.1
server.session-timeout=120

spring.config.name=application
spring.main.web-environment=true
server.contextPath=/ag-computers

# mail properties
spring.mail.default-encoding=UTF-8
spring.mail.host=smtp.gmail.com
spring.mail.username=your_email
spring.mail.password=your_email_password
spring.mail.port=587
spring.mail.protocol=smtp
spring.mail.test-connection=false
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Figure 7.2 Application properties

The properties that start with "your_" should be replaced with valid E-mail and database credentials in order for the application to work as it should. You will choose the credentials for the database when creating a new connection as specified in section 7.2.3 below.

7.2.3. Database setup

Open MySQL Workbench which should be downloaded and installed as specified in section 7.1.2. Choose the "+" as depicted in Figure 7.3 in order to create a new connection, choose a name for it, a username and a password, these have to be provided in the "application.propoerties" file mentioned earlier in order to establish a connection between the server application and the database.

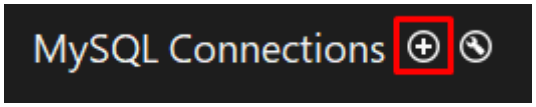


Figure 7.3 New connection in MySQL Workbench

Next select the new connection, provide the credentials and click "OK". Now from the menu choose, create new schema as depicted in Figure 7.4 below and give it the name "webstore" and the choose "Apply". The database tables are created automatically when the server application is run, after that we just have to right click on the database name and choose refresh all and the tables will be displayed. MySQL Workbench can be further use to view the data in the tables and to manage the database tables.

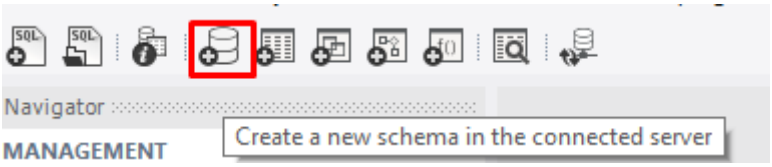


Figure 7.4 New schema

7.2.4. Running the application

To start the server application, we have to open the STS IDE, right click on the "webstore" project and choose Run As -> Spring Boot App. The server application will start on port 8080 and the database server will be started automatically too. The database server runs by default on port 3306.

To start the client application, open the terminal, navigate to the "WebStoreFE" directory and type the command: npm start. The client server will start on port 8000.

Next, open a new window in the browser and type in the following URL: "http://localhost:8000". The landing page of the web application should be displayed.

7.3. Using the application

Instructions for using the most important features of the online store are presented next, each accompanied by relevant images.

When a user accesses the application, the landing page of the application is displayed. The landing page as depicted in Figure 7.5 contains some of the products in the store, and a menu with the actions a visitor (non-registered user) can choose.

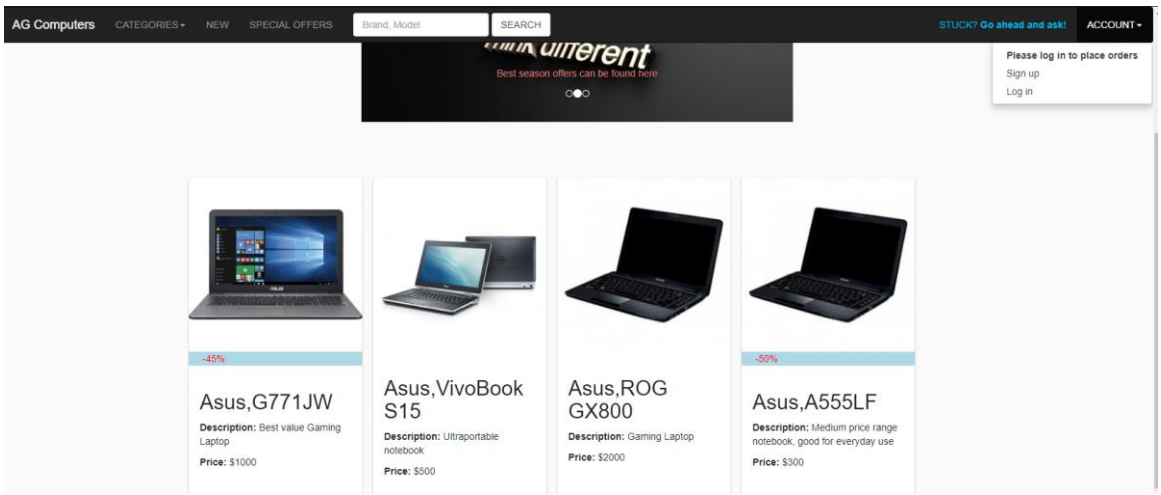


Figure 7.5 Landing page

Next, the features that can be used by a visitor (user which doesn't have a registered account) are presented.

View products based on category

From the menu the user must click on "CATEGORIES", a dropdown displaying all the available categories is displayed as depicted in Figure 7.6 and choose a category for which he/she wants to see the products. The user is redirected to a page containing only the products from the chosen category.

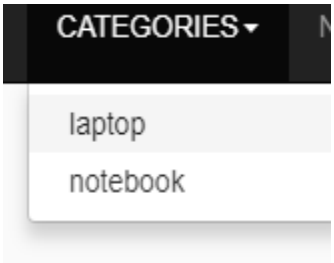


Figure 7.6 Product categories

View new products or products on a special offer

From the menu the user must click on "NEW" in order to see a list of the products that are not older than a month in store. If the user clicks on "SPECIAL OFFERS" a page containing all the products at a special offer is displayed. Figure 7.7 shows these option on the navigation bar.

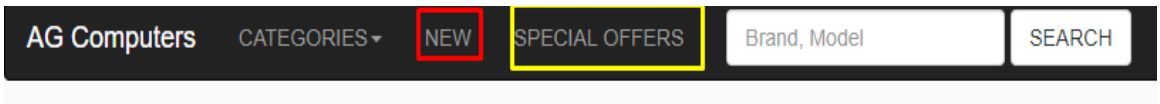


Figure 7.7 Menu options

Search products based on brand and model

In order for the user to search for the desired products he/she must enter the brand or both the brand and model names in the search bar as depicted in Figure 7.7 and then press "SEARCH". A page containing all the matched items is displayed.

Ask a question related to the services of the store

A user can ask a question related to the services of the store and get a quick answer from the customer helper bot by clicking on the "STUCK? Go ahead and ask!" link in the top menu. The chat in which a user can ask questions is shown in Figure 7.8.

A user can ask questions related to: getting a refund, searching products, cancelling an order, viewing order history etc.

Sign up

A user has the possibility to register an account by choosing sign-up from the "ACCOUNT" dropdown on the application menu. The user is redirected to a sing-up page as depicted in Figure 7.9 and must fill in the form with the required information then choose sign-up. The user is signed in and redirected to the landing page. Now the user can choose to sing in at any time using the credentials associated with the created account and benefit from other features of the application like adding products to the shopping cart or to a wish list.

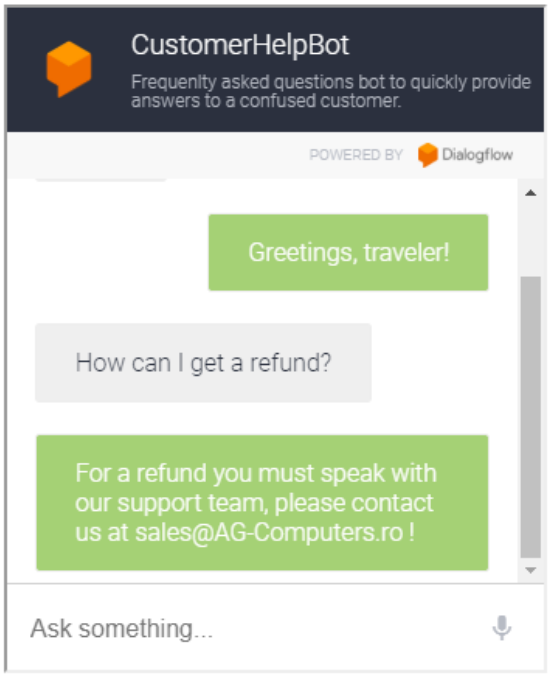


Figure 7.8 Customer help bot

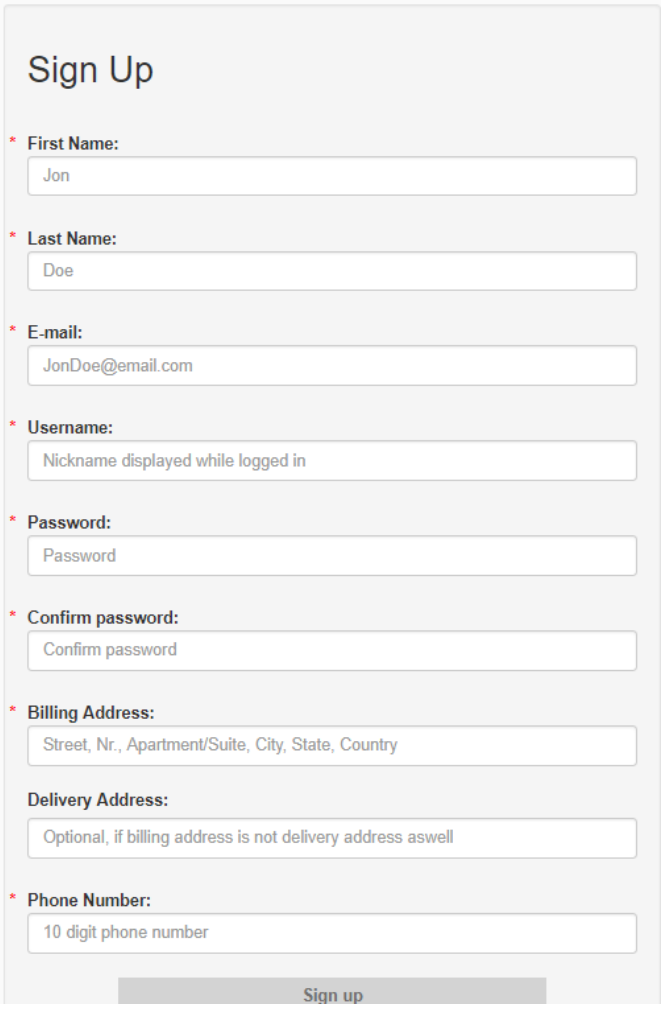


Figure 7.9 Sign up page

Next, the main features that can be used by a registered and authenticated user are presented in addition to the features presented earlier which are available to all users.

Add products to shopping cart to wish list or place a review

A registered user can add a desired product to the shopping cart, to a wish list or can write a review for a product. These can be done by clicking on the product, as a response the user is sent to the product details page and can choose one of the previously mentioned actions as seen in Figure 7.10 below. The user can also remove his/her own review and view other user's reviews. A registered user can add only one review for a particular product.

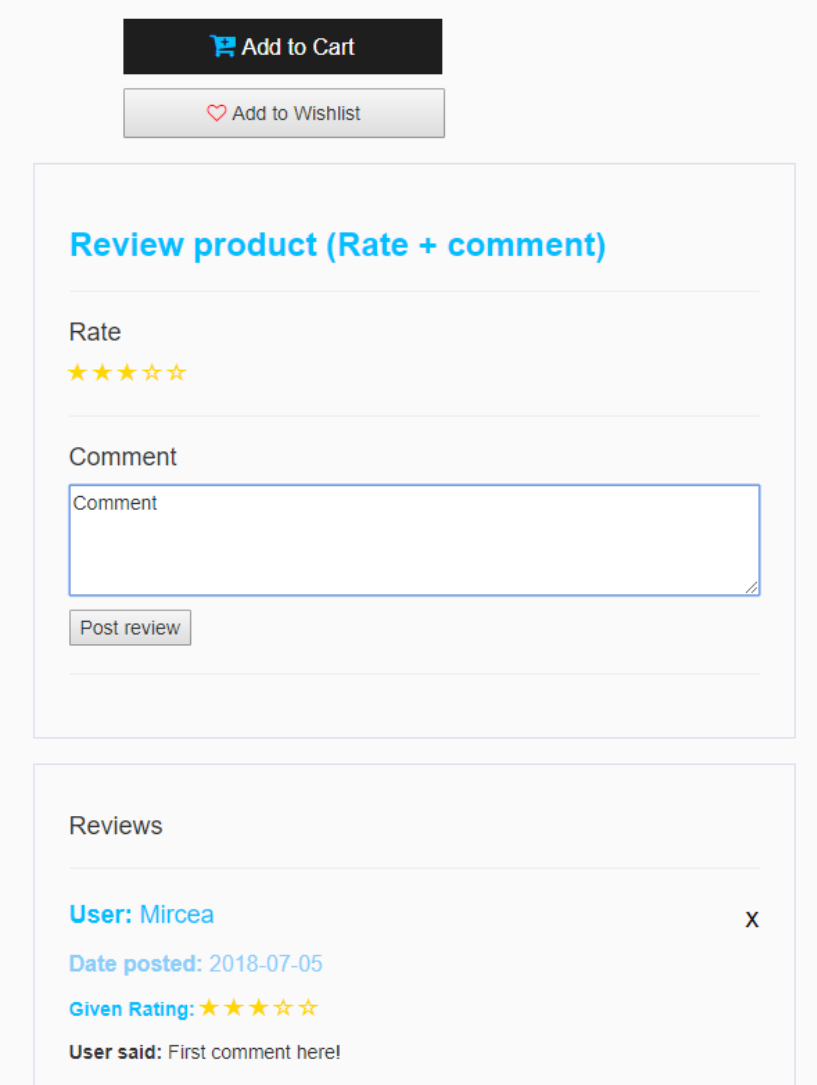


Figure 7.10 Product details

Sign in

A registered user can sign in using the E-mail and password provided at sign up by choosing the log in option in the "ACCOUNT" dropdown from the application menu bar. The user is sent to the log-in page as depicted in Figure 7.11 and after flling in the credentials and pressing the "Log in" button he/she is redirected to the landing page of the application.

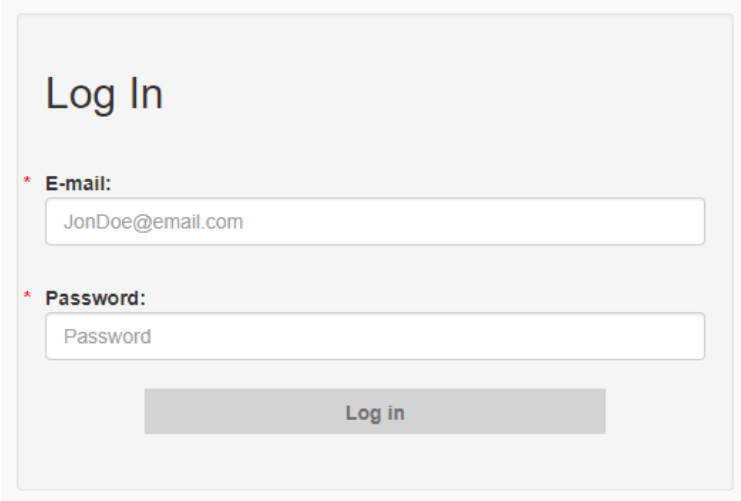


Figure 7.11 Log in page

Edit shopping cart

A registered user can edit shopping cart details, he/she can edit the quantity of each product, can remove a product or can choose a delivery method. The applciation updates the cart after these actions so the user is able to immediately see the changes. To see shopping cart details as depicted in Figure 7.12 the user must click on the cart icon and choose "CART DETAILS".

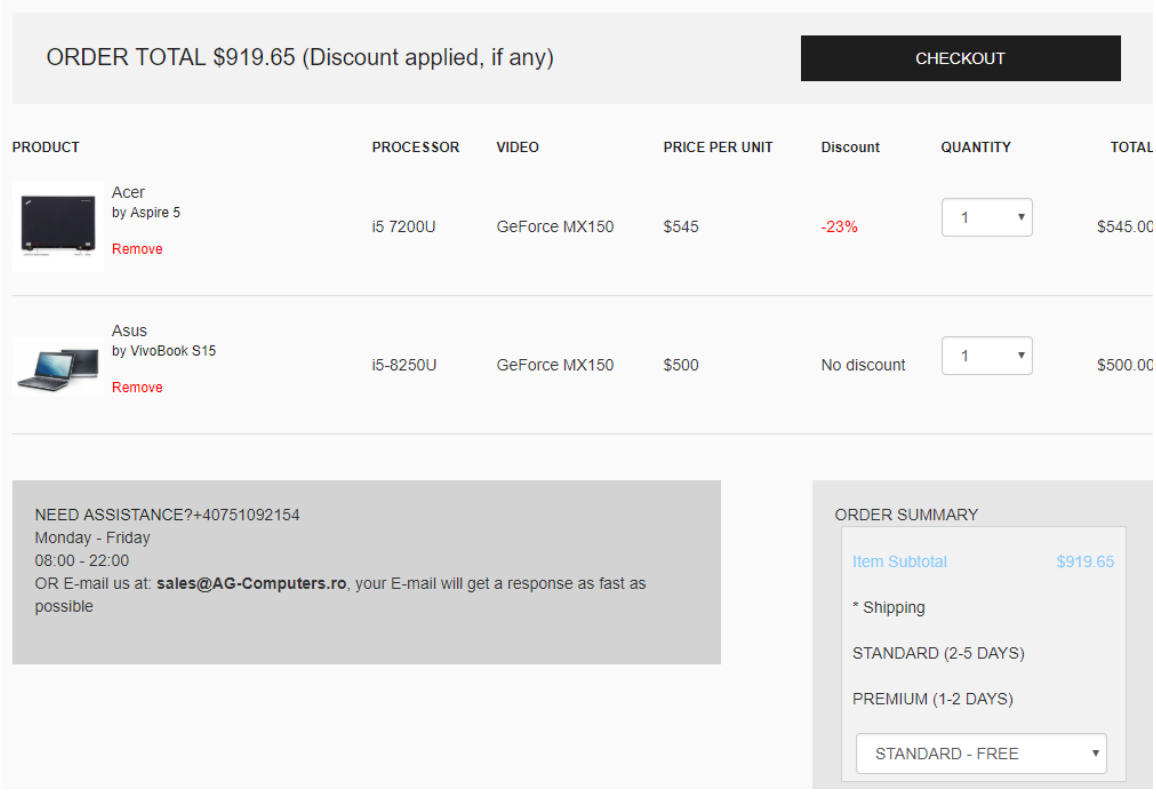


Figure 7.12 Shopping cart details page

Checkout process

A registered user can place an order by choosing "Checkout" from shopping cart details page. The user is redirected to the order details page, here he/she can update the delivery information provided at sign-up and then choose one of the two actions: pay by card or order and pay cash on delivery as depicted in Figure 7.13. If the user chooses order and pay by cash on delivery the order is placed directly and the

user is redirected to the landing page of the application and the shopping cart is emptied. If the user chooses pay by card, he/she is sent to the pay by card tab (this page is illustrated in Figure 7.14) and has to fill in his/her credit card details and choose place order, the user is the redirected to the landing page of the application and the shopping cart is emptied.

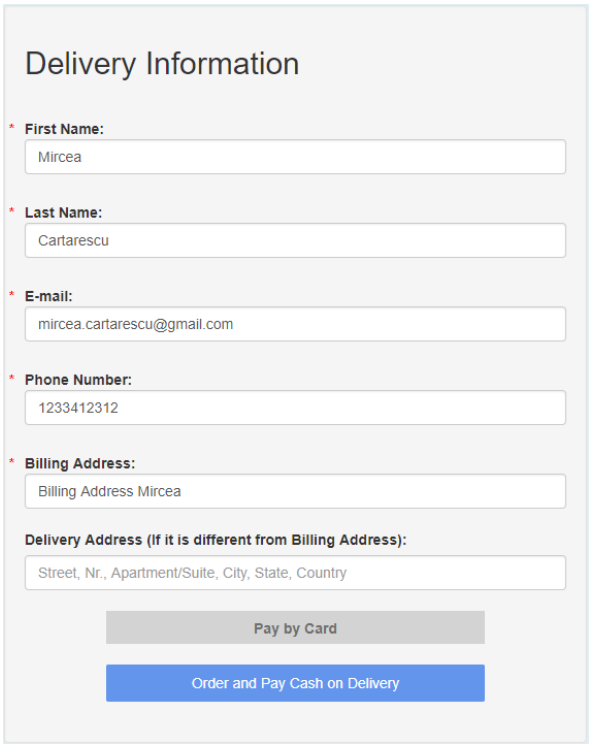


Figure 7.13 Delivery information tab

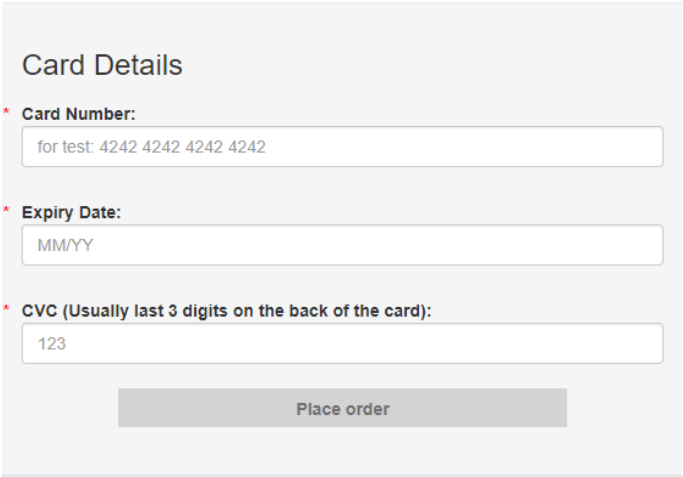
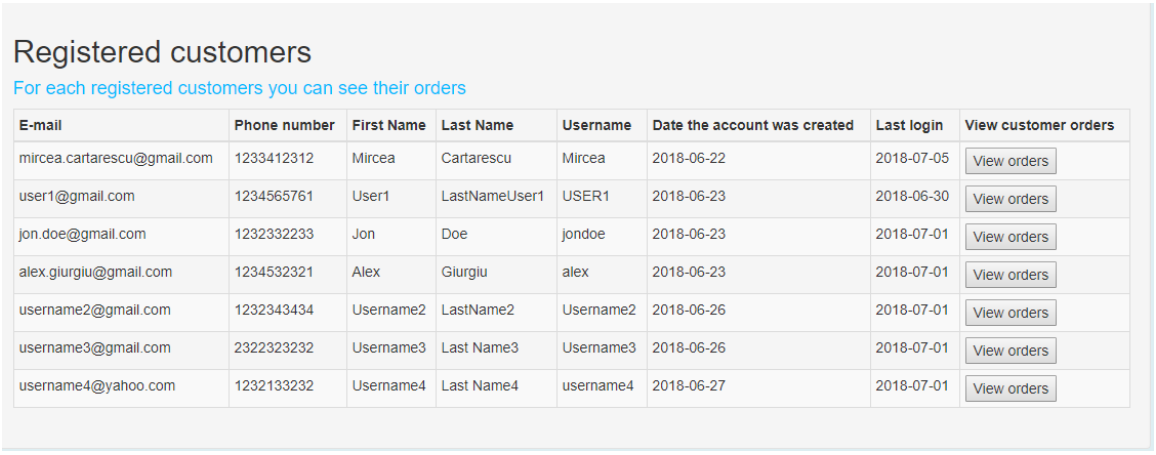


Figure 7.14 Card details tab

View registered user's contact infromation and order history

The administrator has access to all the features above and some additional ones through the admin dashboard like viewing al registered user's contact information which are presented in Figure 7.15, repectively Figure 7.16 below. When a user logs in with valid aministrator credentials a new link is available in the application menu called "ADMIN DASHBOARD". If the user clicks on that link he/she is redirected to the admin dashbiard page and the first thing displayed is a table

of all registered users and information associated with them as well as a “View orders” button wllowing the administrator to see all the past orders of a registered user.

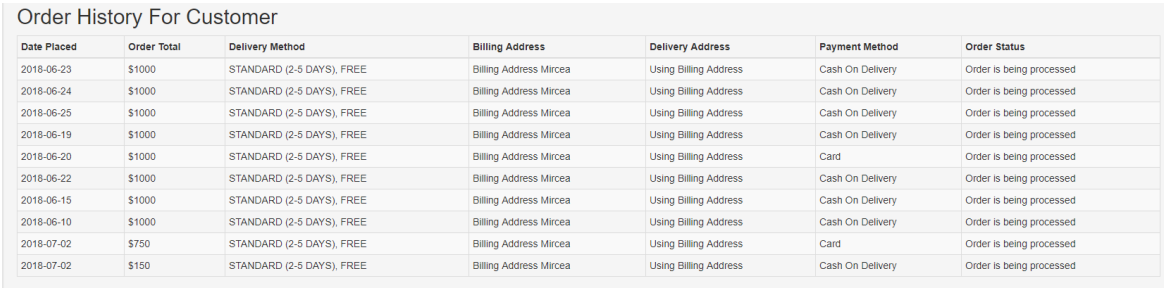


Registered customers

For each registered customers you can see their orders

E-mail	Phone number	First Name	Last Name	Username	Date the account was created	Last login	View customer orders
mircea.cartarescu@gmail.com	1233412312	Mircea	Cartarescu	Mircea	2018-06-22	2018-07-05	View orders
user1@gmail.com	1234565761	User1	LastNameUser1	USER1	2018-06-23	2018-06-30	View orders
jon.doe@gmail.com	1232332233	Jon	Doe	jondoe	2018-06-23	2018-07-01	View orders
alex.giurgiu@gmail.com	1234532321	Alex	Giurgiu	alex	2018-06-23	2018-07-01	View orders
username2@gmail.com	1232343434	Username2	LastName2	Username2	2018-06-26	2018-07-01	View orders
username3@gmail.com	2322323232	Username3	Last Name3	Username3	2018-06-26	2018-07-01	View orders
username4@yahoo.com	1232133232	Username4	Last Name4	username4	2018-06-27	2018-07-01	View orders

Figure 7.15 Registered cutomers table



Order History For Customer

Date Placed	Order Total	Delivery Method	Billing Address	Delivery Address	Payment Method	Order Status
2018-06-23	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed
2018-06-24	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed
2018-06-25	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed
2018-06-19	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed
2018-06-20	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Card	Order is being processed
2018-06-22	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed
2018-06-15	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed
2018-06-10	$1000	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed
2018-07-02	$750	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Card	Order is being processed
2018-07-02	$150	STANDARD (2-5 DAYS), FREE	Billing Address Mircea	Using Billing Address	Cash On Delivery	Order is being processed

Figure 7.16 Order history for a customer

Chapter 8. Conclusions

In this chapter are presented conclusions regarding the implemented application by describing the achievements and fulfilled objectives, followed by ideas for further development.

8.1. Achievements

The aim of the proposed project was to design and implement an intuitive, easy to use and secure web application to serve as a store front for selling computers which benefits both the owner of the store and the customers. The customers benefit from the fact that they can shop for the desired products from home, they have access to product information and prices and can compare them with offers from other stores, they also spare time by not driving to the physical store, also older users can benefit greatly from this because for them it is harder to commute from home to a bunch of physical stores and compare products and prices. The owner benefits from the fact that the online store is always open, thus enabling the business to sell its products at any time, increasing the number customers from different geographical regions as well eliminating the costs of physically expanding the business by building brick and mortar stores. Other benefits brought to the owner are that the store now has an automated process of keeping track of users and their orders as well as the products in stock. We can say that the aim of the project was achieved as the implemented application has all the important features of a modern web store. The customers are able to add the desired items to a shopping cart and proceed to check out, they can edit the quantity of items in the shopping cart, get discounts, leave a review for a product as well as read reviews posted by other customers, search for products, view a product's details, choose the delivery method, online payment and also ask questions and get immediate answers for the customer help bot. The administrator of the store has access to all the before mentioned features plus features regarding registered users, their orders and the products in store.

Next how specific objectives are met is presented. The application supports three types of users: visitor, registered user, administrator (which is also a registered user). An important aspect is that visitors which don't own a registered account can benefit from the basic features of the application like: view products in store, search for particular products based on category, brand and model, view products that are new in store or at a special offer and ask the customer helper bot questions regarding the store services. It is important to have these features available by anyone using the application without registering an account because customers might just read reviews for a product's quality or inform on certain aspects regarding a product like specifications and price. If visitor considers he/she wants to purchase an item, an account can be registered at any time.

After a visitor registers an account he/she is automatically authenticated and authorized. A registered user can sign in at any time using the E-mail and password provided during registration. A registered user, in addition to the features mentioned earlier, has access to the following features: have access to a personal account page where they can edit personal account information that was provided at registration like, phone number, E-mail, delivery/billing address, last/first name and can view the history of their orders, use the shopping cart (add items, edit items in cart), place an order and choose to pay for it online or by cash on delivery, add items to a wish list (e.g. if the user doesn't have money for the moment to buy a product and has already

decided to buy it later it can be placed here so no additional searches need to be made next time), leave a review for a product.

The store administrator can use all the functionalities of the application that a registered user can and in addition he/she can view information about registered users accounts and their order history, view products in stock and information associated with them and edit them, view and edit supplier's information, view and edit special offers. These functionalities help the store to better manage the selling process and keep track of everything they have in store, eliminating the need of keeping important information on paper which is more likely to be lost or destroyed.

The usability requirements are met by having an intuitive user interface and constantly providing the user with feedback when performing an action.

The security requirements were met using Spring Security and JWT tokens for authentication and then authorization. A visitor can't access pages destined for registered users or administrator and a registered user can't access the pages on which only the administrator should have access. The passwords are encrypted before being stored in the database and because after logging in the authorization is based on JWT tokens the password is not forwarded between the server and client applications minimizing the risk of an attacker intercepting it. Also, the payment process is secured by first making a request from the client to Stripe services with the card details (so the card details never go to the server application) and a token is received which is then sent to the server to charge the card based on it. That token is destroyed after the card is charged and can't be used by attackers to steal money from the clients based on it.

8.2. Further development

The application is built based on the separation of concerns concept so the implementation respects the low coupling and high cohesion properties between components making the job of adding additional functionalities very easy and making the application's components easy to maintain and reuse.

In this section are presented ideas for improving the current functionalities and develop new ones.

8.2.1. Improvements

- The shopping cart can be improved to display a summary of items inside when clicked and allow for changing the quantity of each item without having to access shopping cart details page.
- The overall user interface can be improved as well as it's responsiveness to better fit all sizes of screens.
- The search functionality can be improved to support more parameters.

8.2.2. New features

- The customer help bot can be upgraded to offer a personalized experience for each user by recommending certain products based on a quiz answered by the user (which would allow users to choose a product that fits their needs the best). An upgrade to the bot would also be to allow customers to shop for products directly from the chat, offering more personalized and interactive way of shopping which would increase the number of customers that use the application.

- Besides the client and server applications a third application can be developed to serve as an API for the customer help bot and deal with all its requests. This would help develop the personalized experience for each user as mentioned above.
- The shopping cart can be further developed to check for how long the items of a user have been in the shopping cart and if for example 24 hours have passed and the user did not check out the user is informed by mail that he has products waiting in the shopping cart. A special offer for the items in the cart can be sent to the user via mail to increase the chances that a customer will return to his/her cart and complete the order. Also, a questionnaire can be sent to that user to find out why he/she did not complete the order. These ideas can help in minimizing shopping cart abandonment.
- Another functionality that can be integrated in the existing application is E-mail advertising. With this functionality the application should inform registered users when a new product appears in the store or about certain special offers via E-mail.
- Allow customers to place a product on watch, meaning that they will get an E-mail alert when that product's stock is supplemented or when the product is on a special offer.
- Allow users to like or dislike a review so users that read the reviews for a product would know if the review is reliable or not. Give a special discount to user's that have the most likes on a review, that would encourage people to give reliable and well taught reviews, bringing more customers to the store.
- Payment process: while the user introduces the card number, offer real-time validation and display the type of card that is being used. Enable the user to check a "remember card details" box and create for that user a Stripe user that has these card details stored on the Stripe servers so when the user checks out and wants to pay by card, the card details are already completed.
- Login based on a social media account like Google+ or Facebook.

Bibliography

[1] Eduard Dinu, Mihaela Mosora, Adrian Anica-Popa, Ana Maria Popescu, "Evaluarea impactului E-commerce asupra consumatorului, economiei si societatii", Case Study, "Academia de Studii Economice din Bucuresti", Romania, 2018. [Online]. Available: http://dmci.ase.ro/images/proiecte/Studiu_Final_e_commerce.pdf

[2] Michael S. Mikowski and Josh C. Powell, "Single Page Web applications: JavaScript end-to-end", Manning Publications Co. , USA, 2014.

[3] Talla Ngala Yiga, „Design and implementation of a Secondhand E-bookshop", Bachelor's Thesis, Akademin för innovation, design och teknik(IDT), Mälardalens högskola, Feb. 2011. [Online]. Available: http://www.diva-portal.se/smash/get/diva2:398899/FULLTEXT01.pdf

[4] Shen Yeyin, "Design and Implementation of a Web Shop System", Bachelor's Thesis, Kemi-Tornio University, Nov. 2010. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/22377/Shen_Yeyin.pdf

[5] Paul Deitel and Harvey Deitel, "Java™ SE 8 for programmers", Third Edition, Deitel and Associates, Inc. , Michigan, USA, Mar. 2014.

[6] K.Siva Prasad Reddy, "SpringBoot: Learn By Example", Leanpub, 2016.

[7] Deepak Kumar, "Best Practices for Building RESTful Web services", Infosys Limited, Bengaluru, India, 2017. [Online] Available: https://www.infosys.com/digital/insights/Documents/restful-web-services.pdf

[8] Marijn Haverbeke, "Eloquent Jvascirpt", 3RD Edition, No Starch Press, 2018. [Online]. Available: https://books.google.ro/books

[9] Ken Williamson, "Learning AngularJS", O'Reilly Media, Inc. , USA, 2015.

[10] Jon Duckett, "HTML&CSS: Design and Build Websites", John Wiley & Sons, Inc. , USA, 2011.

[11] Paul DuBois, "MySQL Cookbook", Third Edition, O'Reilly Media, Inc. , USA, 2014. [Online]. Available: https://books.google.ro/books

[12] Ben Alex, Luke Taylor, "Spring Security Reference Documentation", [Online].Available:https://docs.spring.io/springsecurity/site/docs/3.0.x/reference/springsecurity.pdf

Appendix 1 – List of tables

Appendix 2 – List of figures

Appendix 3 – Glossary

Abbreviation	Meaning
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
HTTP	Hypertext Transfer Protocol
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
DOM	Document Object Model
AJAX	Asynchronous JavaScript and XML
XML	Extensible Markup Language
CRUD	Create Read Update Delete
MVC	Model View Controller
JS	JavaScript
JSON	JavaScript Object Notation
IDE	Integrated Development Environment
RDBMS	Relational DataBase Management System
JWT	JSON Web Token
SHA	Secure Hash Algorithms
HMAC	Hash-based Message Authentication Code
SMTP	Simple Mail Transfer Protocol
UI	User Interface
GUI	Graphical User Interface
JDK	Java Development Kit
SE	Standard Edition
WS	Web Services
YAML	YAML Ain’t Markup Language
JPA	Java Persistence API