354TheStars Online Marketplace

SOFTWARE DESIGN DOCUMENT

Version: 1.0

Michael Rowe 26101267
Steven Smith 40057065
Nagib Al-Bouery 27153031
Gheith Abi-Nader 25703387
Claire Shaw 40022429
Gia Khanh Nguyen 24880803
Olivier Fradette-Roy 40074024
Mark Jarjour 40057853
Guillaume Rochefort-Mathieu 40058103
Amrou Abdelhadi 40045799
Ghislain Clermont 40057664
Mahdi Chaari 27219946

Gina Cody School of Engineering and Computer Science
Concordia University
Comp 354 Fall 2019
Friday, November 8, 2019

Contents

List of Figures

List of Tables

1. Introduction

This section of the Software Design Document will elucidate the purpose of the document, the scope of the system and provide an overview of this document and its organization [5].

1.1. Purpose

This software design document describes the design and system architecture of the e-commerce website 354TheStars. The document serves as a reference for developers and stakeholders who wish to understand the rationale behind the architectural design of the system, including its many components.

1.2. Scope

354TheStars in an online marketplace where users can buy and sell products. Its objective is to simplify the interaction between the buyer and the seller while maintaining an efficient and profitable model. The platform's objective is to increase its market share and, by extension, expand platform profitability. The system currently takes an eight percent fee per transaction completed on our platform. In return, we aim to provide users with a secure, dependable and accessible system.

1.3. Overview

This document will provide a general overview of the system's functionality, the system's architectural, data and component design as well as an elementary blueprint of the client's user interface. It is decomposed into three chapters:

- System Overview
- System Architecture
- User Interface Design

2. System Overview

Many online distribution services already exist. However, each one has its own set of advantages and disadvantages. 354TheStars aims to create a new simplified and efficient model for the sale of online goods.

354TheStars is designed as a closed commercial tool for distributors, large or small, to be able to sell their merchandise in a feature-rich and fair environment. Our revenues are user-driven. This means, that the software only generates an income when items are sold on the website. We only make money when the seller makes money.

The platform is compatible with the third-party payment processing software, PayPal. Due to its strong security and reliable status – the software is being designed to offer PayPal as the only available payment processing option, although, it is flexible enough to add more options later.

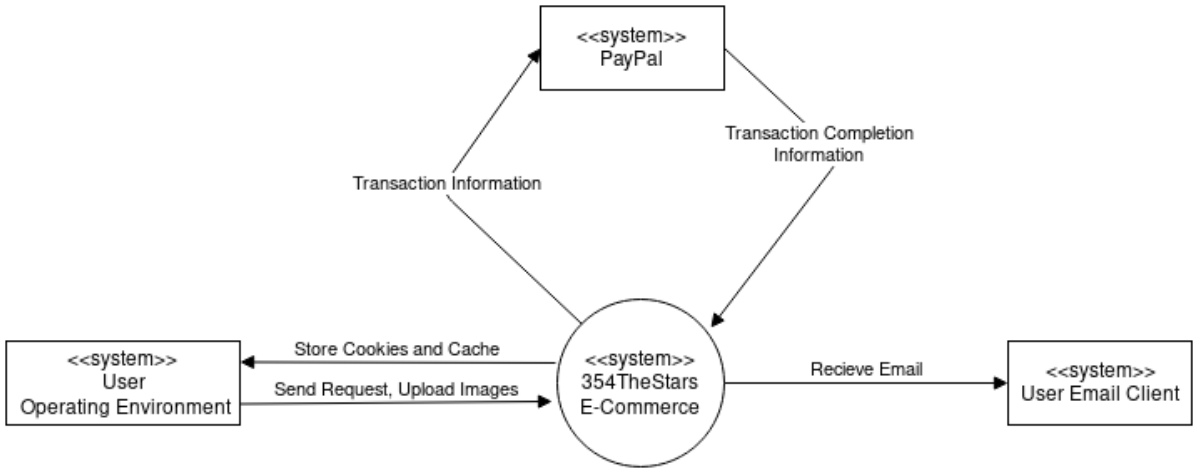
The primary advantage of our platform compared to the competitors is our subsidized shipping. It is our mission to ensure that every product has access to both standard and express shipping free of charge. This reduces buyers' and sellers' expenses. Besides this, our platform is designed to be more akin to a marketplace than a store. As such, any user can sell their merchandise on the platform without having to jump through any complicated hoops or procedures.

The software will have the following capabilities:

- Seller review.
- Unified buyer and seller account.
- Checkout using PayPal.
- Search, Sort and Filter listings on the website.
- Profile Management: this includes name, email, shipping address(es), phone number, past transactions, active listings, saved carts, etc.
- Create, edit or delete a product listing.
- Administrators can view monthly and weekly earnings as well as perform administrative tasks such as maintenance or removing inappropriate listings.

The context of the system is shown below:

Figure 2.1.: System Context Diagram



3. System Architecture

The objective of this section is to show a high-level overview and decomposition of the system's subsystems.

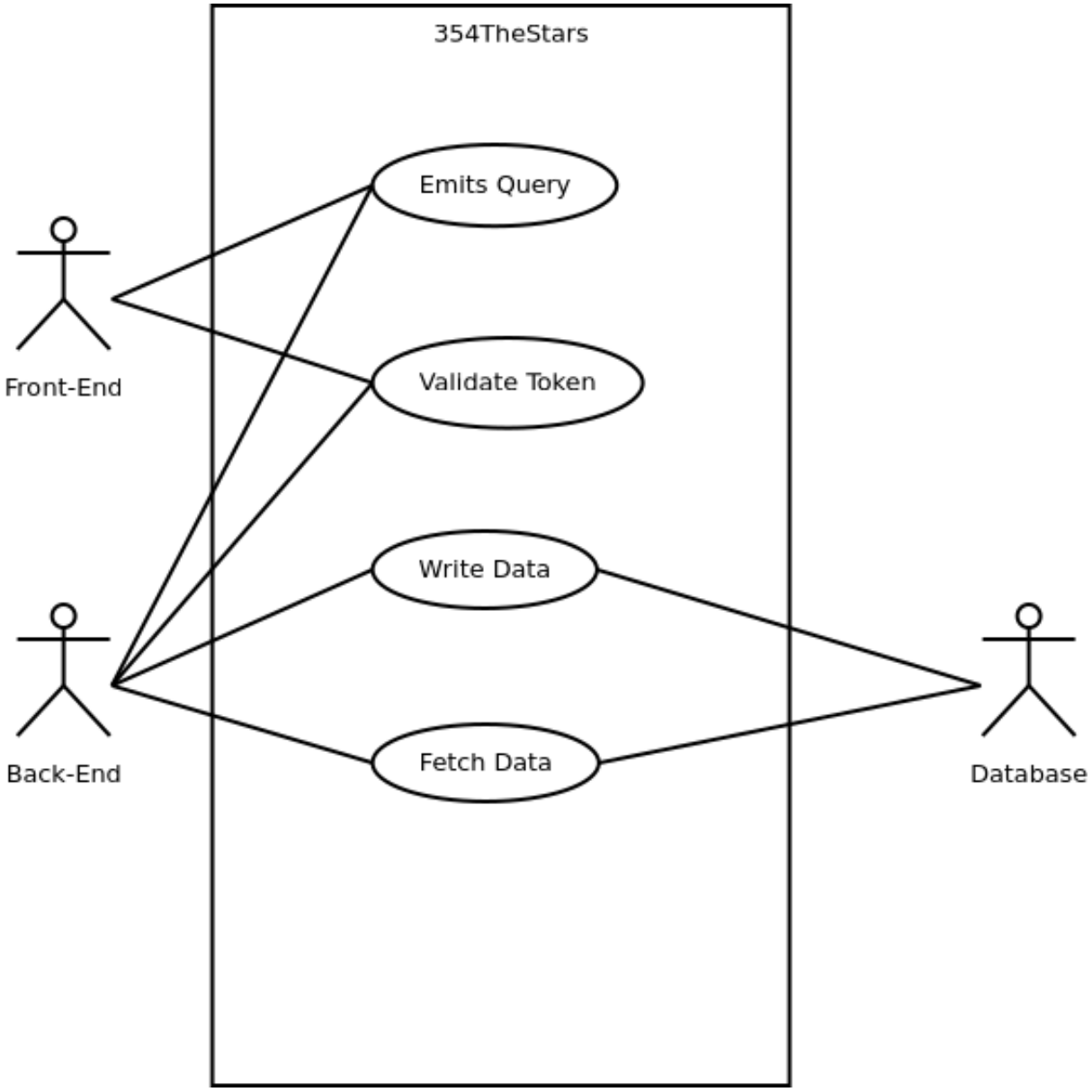
3.1. Architectural Design

354TheStars system is decomposed into three subsystems:

- Front-End
- Back-End Server
- Database

The system is an online marketplace. The three subsystems must perform in unison. The front-end is the user interface and the website. Its main purpose is to react to user input and send queries to the back-end when data must be read or written. The front-end is also communicating with the back-end to validate the authenticity of a user's session [10]. The back-end is a server that will link the database and the front-end. The queries made by the website will be interpreted by the server and it will process the information provided by the user. Afterward, the data is forwarded to the database where it is stored for an undetermined period. The high-level overview of the system is shown in **Figure 3.1**.

Figure 3.1.: 354TheStars System Overview



3.2. Decomposition Description

This section will provide an in-depth decomposition of the subsystems outlined in **Figure 3.1**.

Front-End

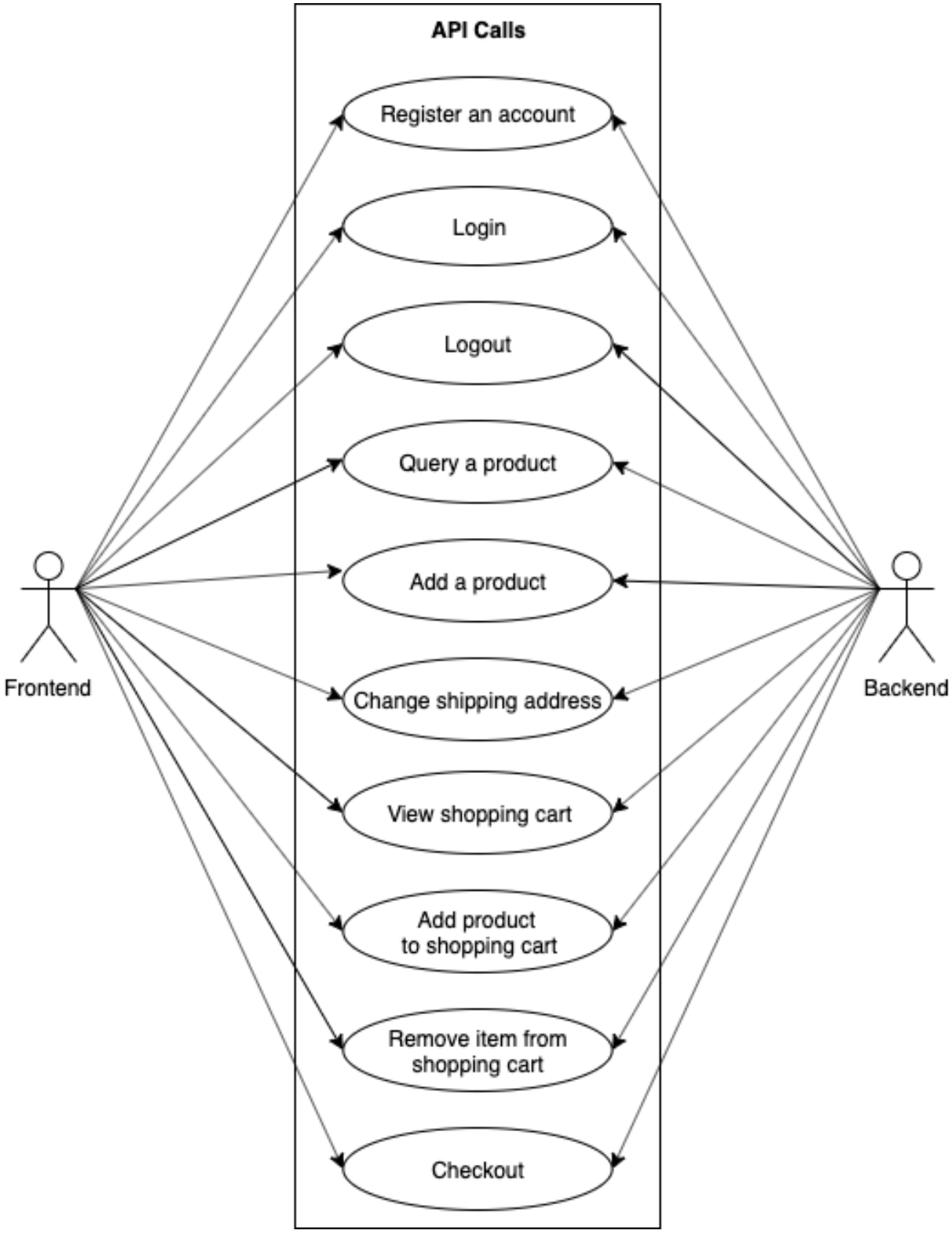
The front-end is the website subsystem for 354TheStars. The purpose of the front-end is to provide a clean and simplified model of our system that the users can interact with it. It is built using Vuetify which is based upon what is called a "single page presence" in which the entire front-end is a single page and it is dynamically updated to reflect the changes in its state. Furthermore, Vue has something called a <router-view/> element. This element displays a

component depending on the "route" of the web page. This allows us to quickly change what components are displayed on the page by dividing the subsystem's functionalities into multiple routes. For example, "354TheStars.com/login", where "/login" is the route for the login component.**Figure 3.2** shows a detailed representation of the front-end and its relationship with the back-end.

The front-end has ten primary component or functions:

- Register an account
- Login
- Logout
- Query a product
- Add a product
- Change shipping address
- View shopping cart
- Add product to shopping cart
- Remove item from shopping cart
- Checkout

Figure 3.2.: Front-End-Back-End API Use Case



Each function reacts to user input and emits information to the back-end. Whether it is for validation or querying information is irrelevant to our model. The front-end uses the back-end's

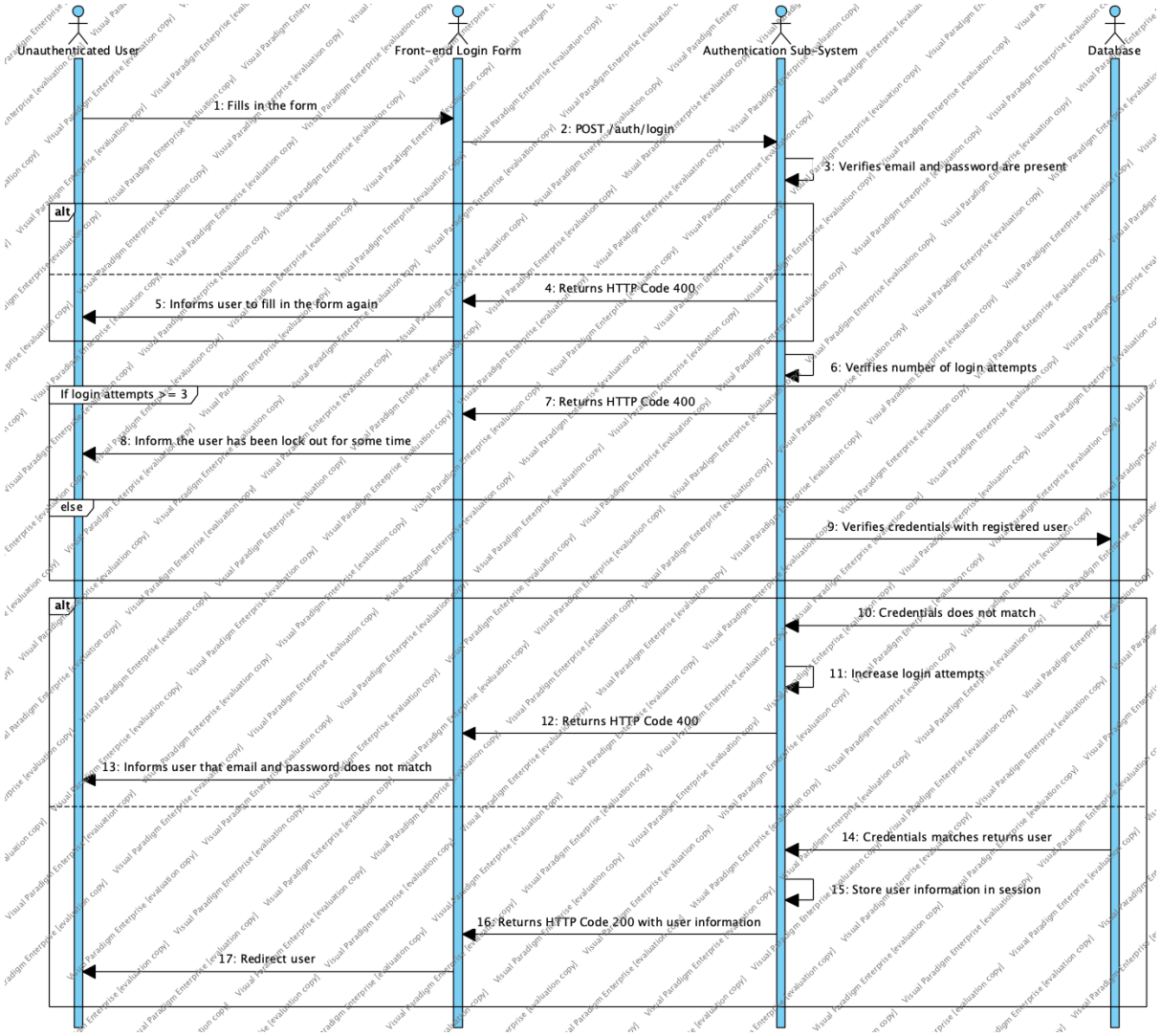
API to make the appropriate calls such that the expected data is return. This allows for greater efficiency and consistency when implementing the front-end. **Figure 3.2** is described further in the **Table 3.1**.

Table 3.1.: API Calls Use Case Description

Actors	**Front-End and Back-End**
Description	Front-end may communicate to the back-end by sending API query calls to the back-end for interpretation. This is used to transfer JSON data between the front-end and the back-end [4]. The front-end may also send a query to validate the login token for a user session.
Data	JSON, data request query, login token, cookies.
Stimulus	The user interacts with a button or toggle on the front-end that triggers a query/call to the back-end API. Example: User clicks the login button.
Response	The back-end receives the data, interprets it, executes the query and return data, if any.
Comment	Besides login and adding products to the shopping cart, all other systems require the user to have a valid login token.

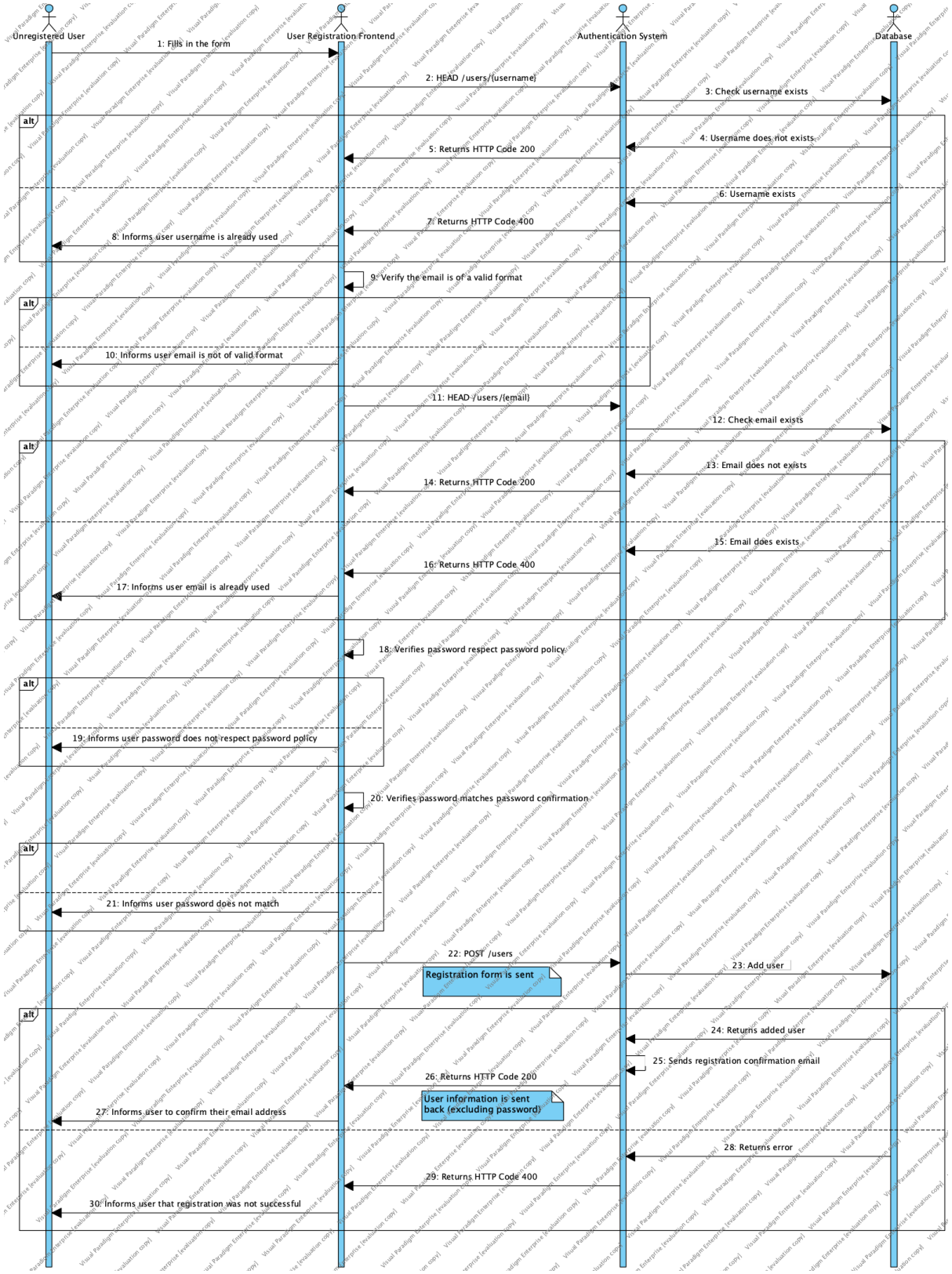
Figure 3.3 and **Figure 3.4** show the Login and Registration sequence diagrams respectively. The sequence diagrams further illustrate the relationships between the various components of the system, both internal and external boundaries, and their corresponding inputs, outputs and timings. Furthermore, they also shows the conditional branches in the system's decision-making which are important to understanding the various input and output combinations and how they might affect the related functions.

Figure 3.3.: User Login Sequence Diagram



The user login sequence diagram shows the interaction between the user, front-end, back-end and the database. In **Figure 3.3**, the authentication subsystem is one of the components of the back-end. It shows the input and outputs of each system over time and the choices that can be made when alternative options are available.

Figure 3.4.: User Registration Sequence Diagram



The user registration interacts with much of the same actors as the login feature, however, it is much more involved. There are a lot more inputs to validate and this creates a greater margin for error over time. Consequently, the sequence diagram in **Figure 3.4** is a lot longer than the one of login. This is the case because we need to handle various situations where some input is entered incorrectly or is unavailable.

Figure 3.5.: Logout Sequence Diagram

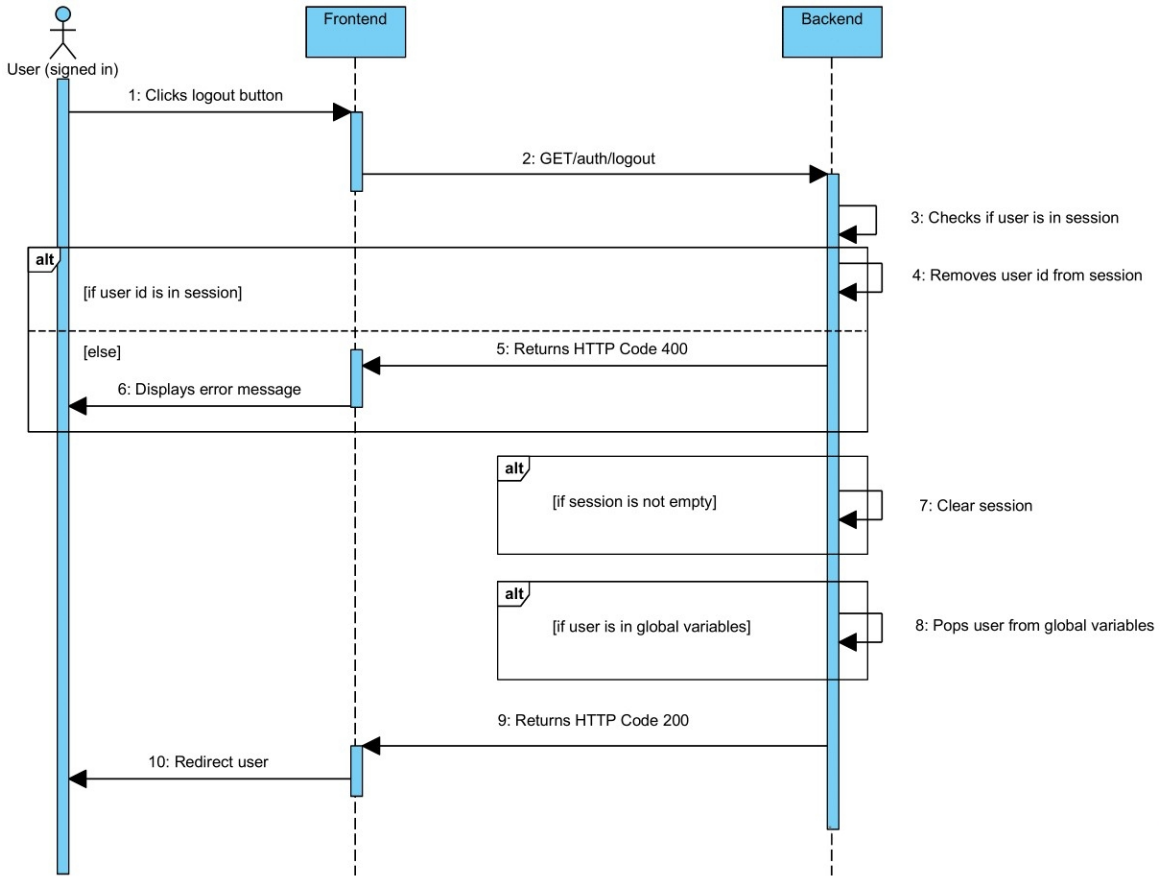


Figure 3.5 portrays the sequence diagram for the logout function and the interaction required between the user, front-end and back-end. It shows the flow of data over time and how each subsystem depends on each other for the completion of tasks.

Figure 3.6.: Add Product to Cart Sequence Diagram

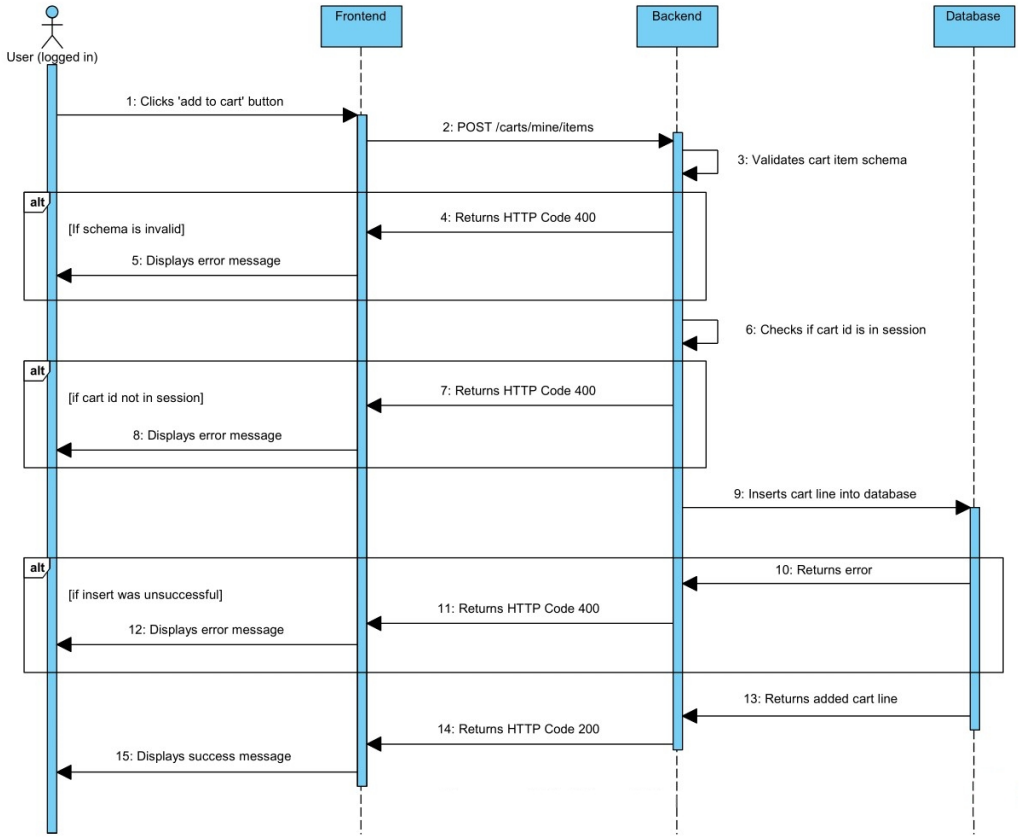


Figure 3.6 shows the sequence diagram for adding a product to the shopping cart. This diagram shows the interaction of the user, front-end, back-end and the database. We clearly see that the front-end reacts to the input from the user which gets interpreted by the back-end. The back-end then returns either an error code or processes the query and returns the required data. The database is only accessed when information must be fetched or written. The sequence diagram shown in **Figure 3.7** is similar in concept. It shows the combination of input and output required over time for the component to work as expected.

Figure 3.7.: View Shopping Cart Sequence Diagram

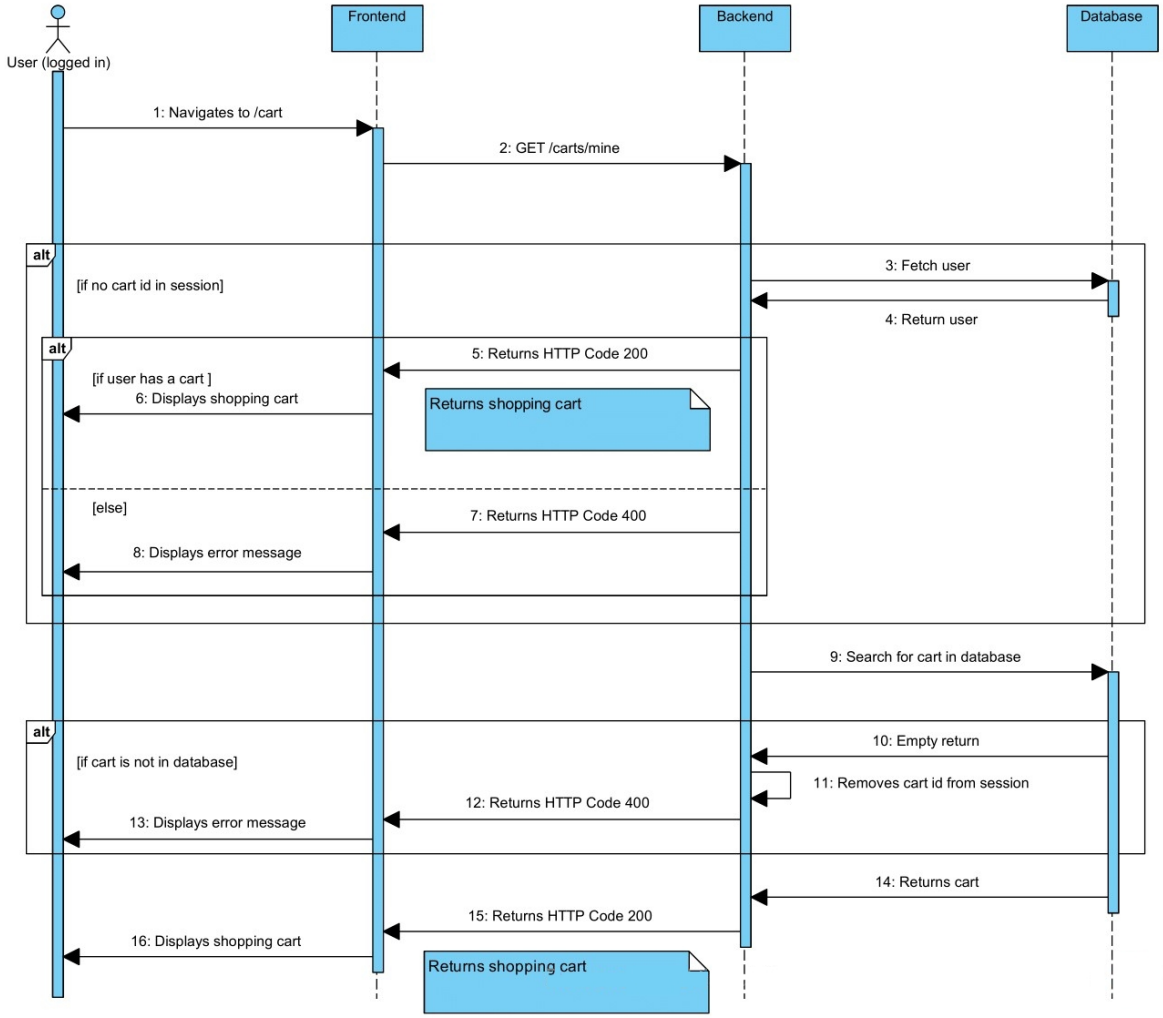
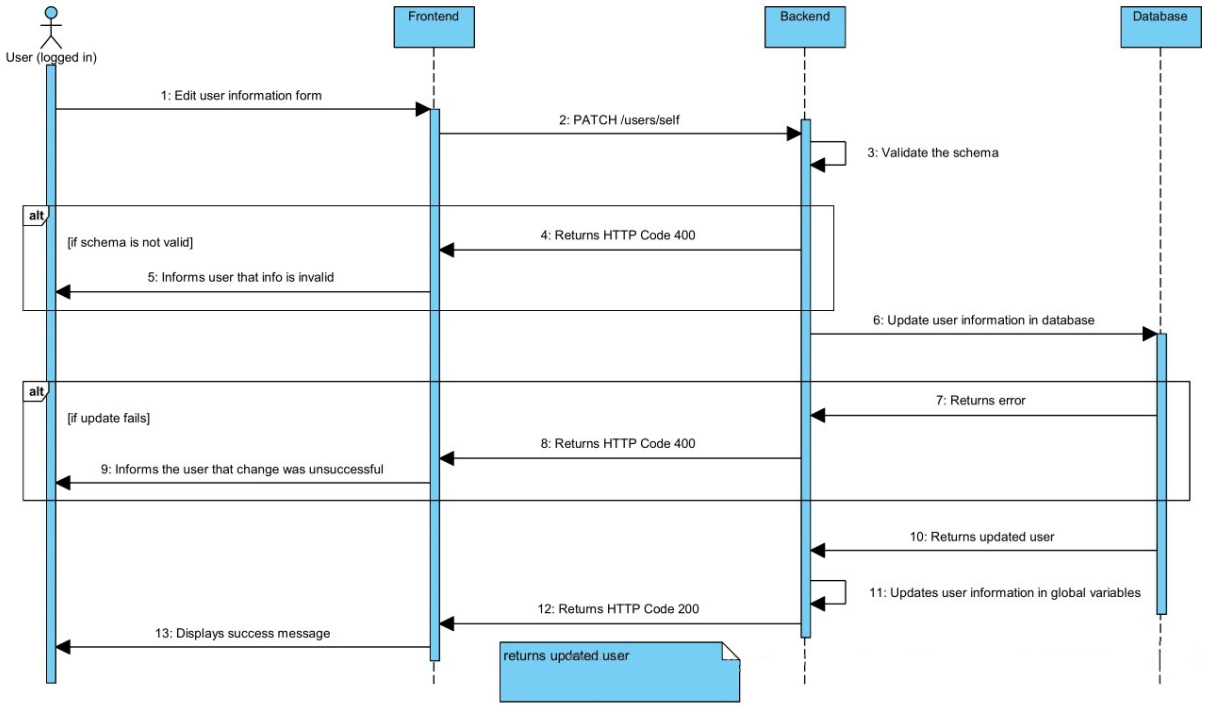
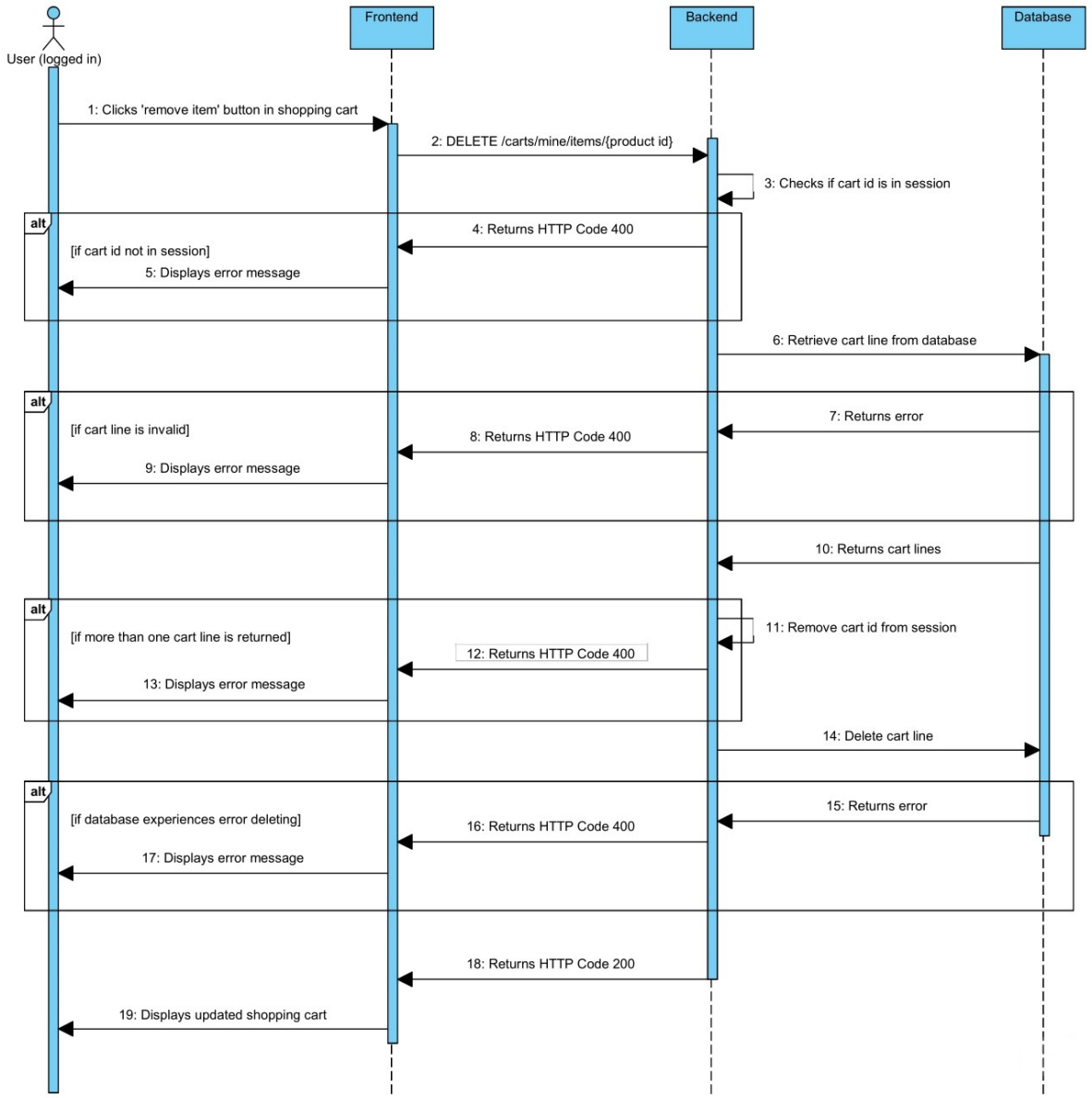


Figure 3.8.: Update User Sequence Diagram



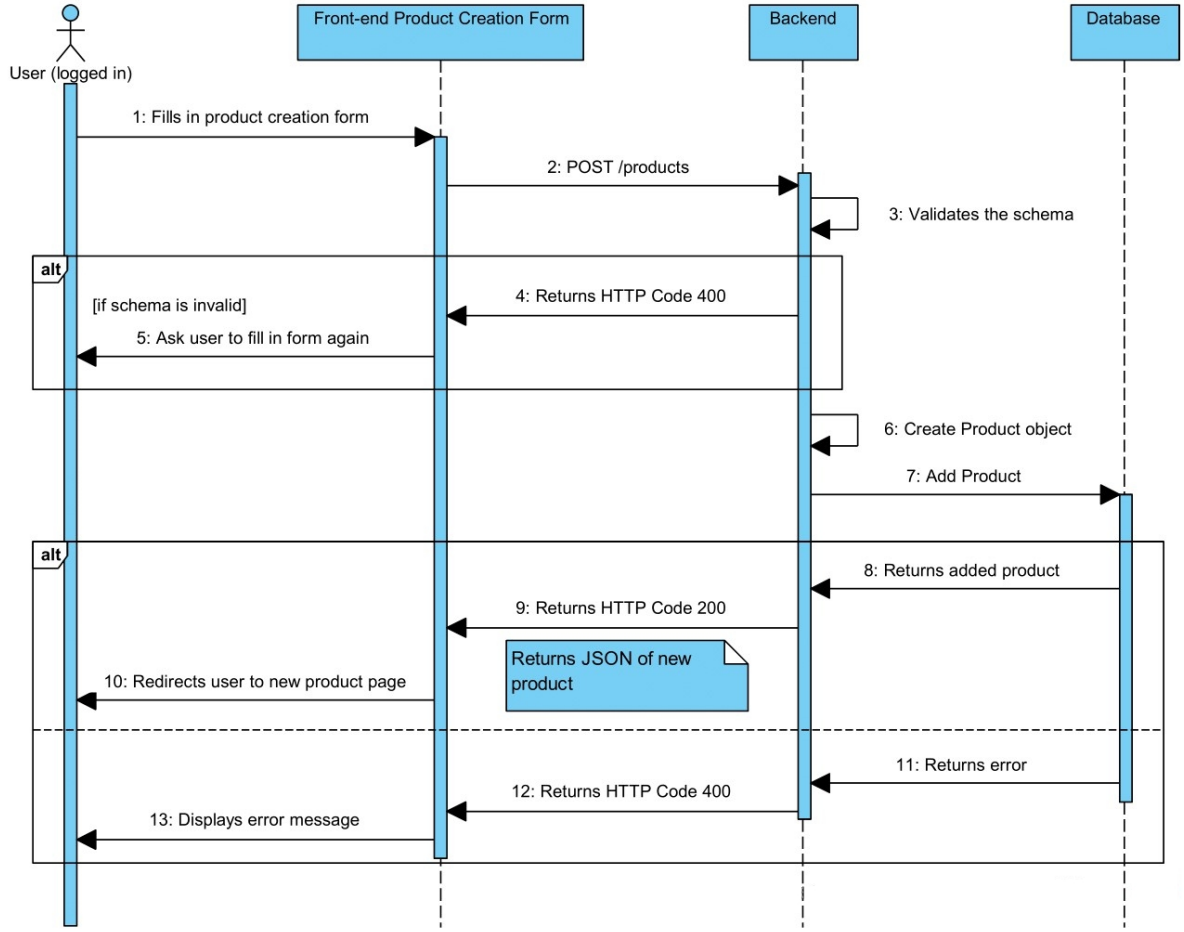
The update user sequence diagram shows the flow of data over time between the user, front-end and the back-end. Furthermore, it illustrates the possible input that will result in errors and a generalized representation of how such errors are handled by the front-end. For this scenario an error will result in one of two messages; either the user will see an "Information invalid" or "Changes were unsuccessful" message informing the user that something happened that was not expected. This is normal since all inputs are verified to ensure they conform to the models that will be discussed in greater depth in the following sections.

Figure 3.9.: Remove Item From Cart Diagram



The remove item from cart sequence diagram shown in **Figure 3.9** displays the exchange of data between the user, front-end and the back-end over time and all the possible errors that may occur from an incorrect input. For example, if the cart does not exists then a database failure will occur and HTTP Code 400 will be returned to the front-end where it will display an error message for the user.

Figure 3.10.: Product Creation Sequence Diagram



Product creation is a much more complex task and its sequence diagram is shown in **Figure 3.10**. It is a process that requires a lot of information. This implies that there are a lot of opportunities for errors to occur. Even more so since it interacts with the front-end, back-end and the database, therefore it increases the possibility of an input not being correct for one of the three subsystems. Luckily, the back-end exists partially to verify the commands sent from the front-end. As such we can come to expect certain types of errors that can occur due to improper inputs.

Back-End

The back-end is designed using Python and Flask. Flask is essential as it contains a development server and debugger as well as support for RESTful request dispatching [3][1]. Similarly, to how the front-end works, the back-end also functions using routing. With these functionalities, Flask functions as an API between the front-end and the server. The Python functions have predefined routes that are used to call the API from the front-end. Any function that is defined and that can be used from the website will need to contain a *@bp.route("/route_name")* instruction, where *bp* is a Flask blueprint. Since it is necessary to work with many data and have it coherent between the front-end and back-end - JSON objects are used and their schemas are stored for data input validation.

Figure 3.11.: High Level Architecture Overview

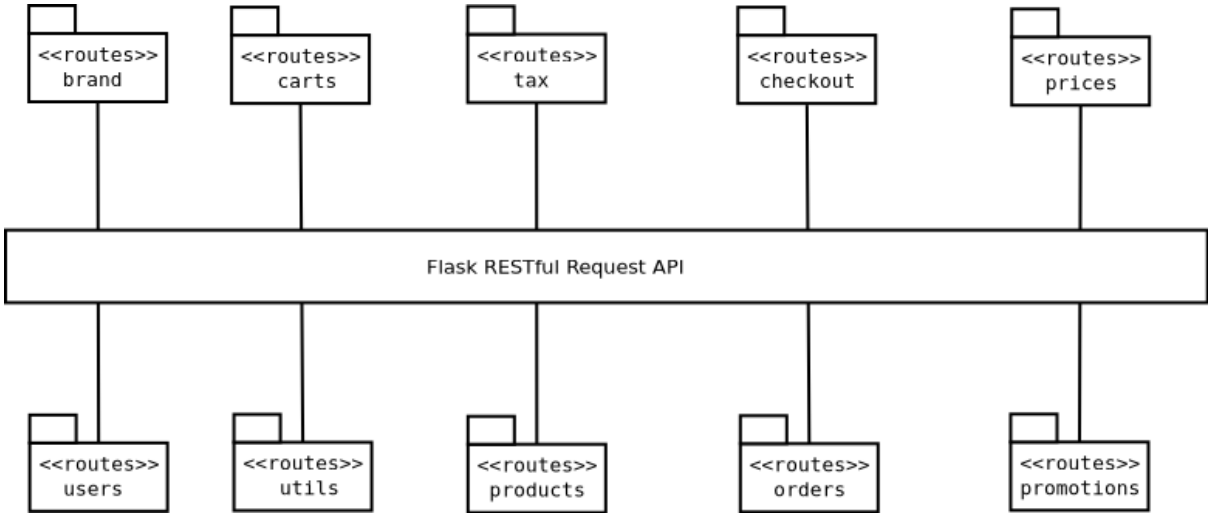


Figure 3.11 elaborates on the topic of routing. Flask's RESTful request dispatching allows the front-end to dispatch queries to the back-end using its related routes. Each function has a unique route defined that the front-end can interface with. Furthermore, individual functions also interact separately with the database to acquire the information that they need. **Figure 3.1** already portrays a generalization of the use case between the back-end and the database. **Figure 3.12** shows a more in-depth use case and is described in **Table 3.2**.

Figure 3.12.: Back-End Routing Use Case

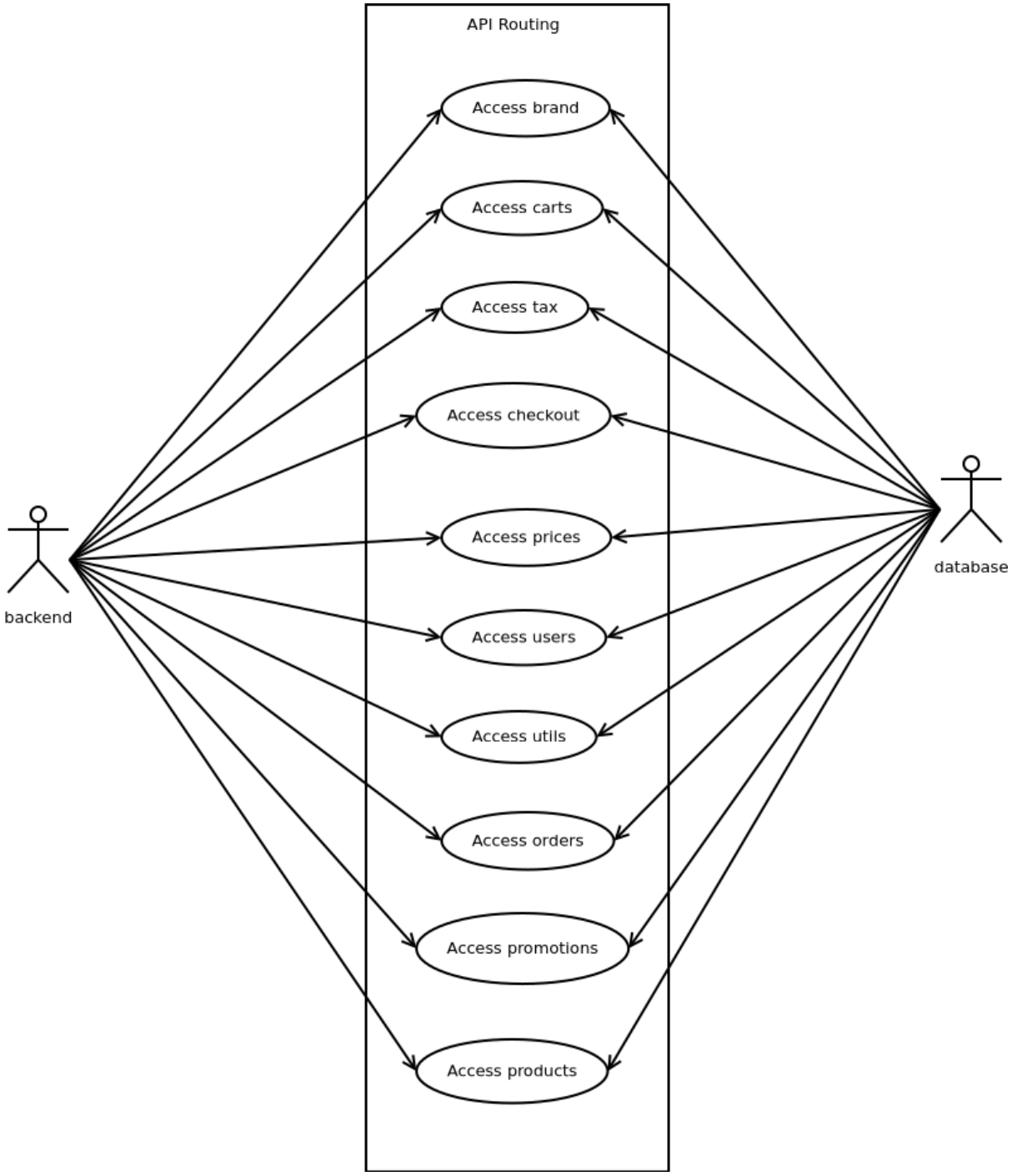
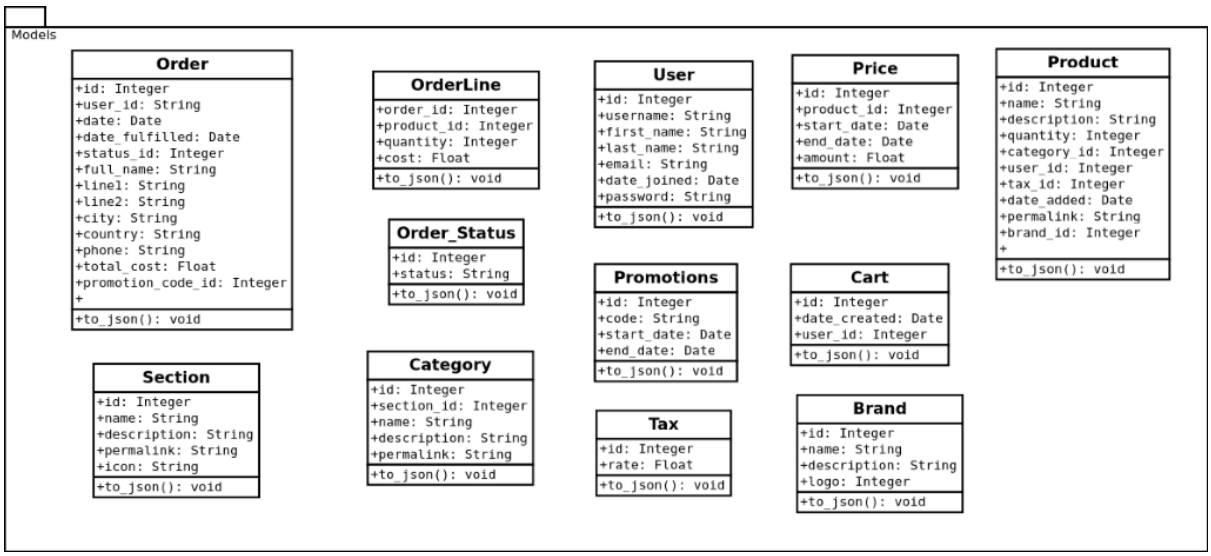


Table 3.2.: Back-End Use Case Explanation

Actors	**Front-End and Database**
Description	Back-end API routing use case. When invoked by the front-end and executed by the back-end each API route requires a unique connection to the database to read or write data.
Data	JSON objects, error messages, queries, cookies.
Stimulus	API call invoked from front-end (**Figure 3.2**).
Response	Back-end processes the query and retrieves information from the database, if any.
Comment	A valid login token is required for all tasks other than viewing products or adding products to shopping cart.

The back-end is composed of several key classes that are shown in **Figure 3.13**. Although our architecture is not object-oriented in design, we cannot circumvent the use of classes. These classes are the backbone of our data management and communication. They are models that map attributes to a JSON representation and allows us to more easily manipulate incoming and outgoing information. Please refer to the SRS for further information on the tools and libraries used for the development of the back-end. Class relationship can be observed from the database entity component diagram in the next section. To avoid repetition, class relationships will only be discussed when covering the databases entity-relationship diagram. The reason is that the class diagram are back-models used to verify input and ensure that it follows the database table structures. In other words, they aren't really objects. Rather they are models that represent the database tables within the back-end, as such all relationship between the classes will be discussed alongside the ERD.

Figure 3.13.: UML Class Diagrams for Models' Package



The back-end contains roughly twelve different models. We have the user models which represents the user table within the database and holds information such as id, username, password, first and last name, etc. You also have the product model which holds the product id, name, description, quantity, seller, brand, etc. The product model also mimics its database counterpart, because that is how our design works. Without it, using SQLAlchemy would be a lot more

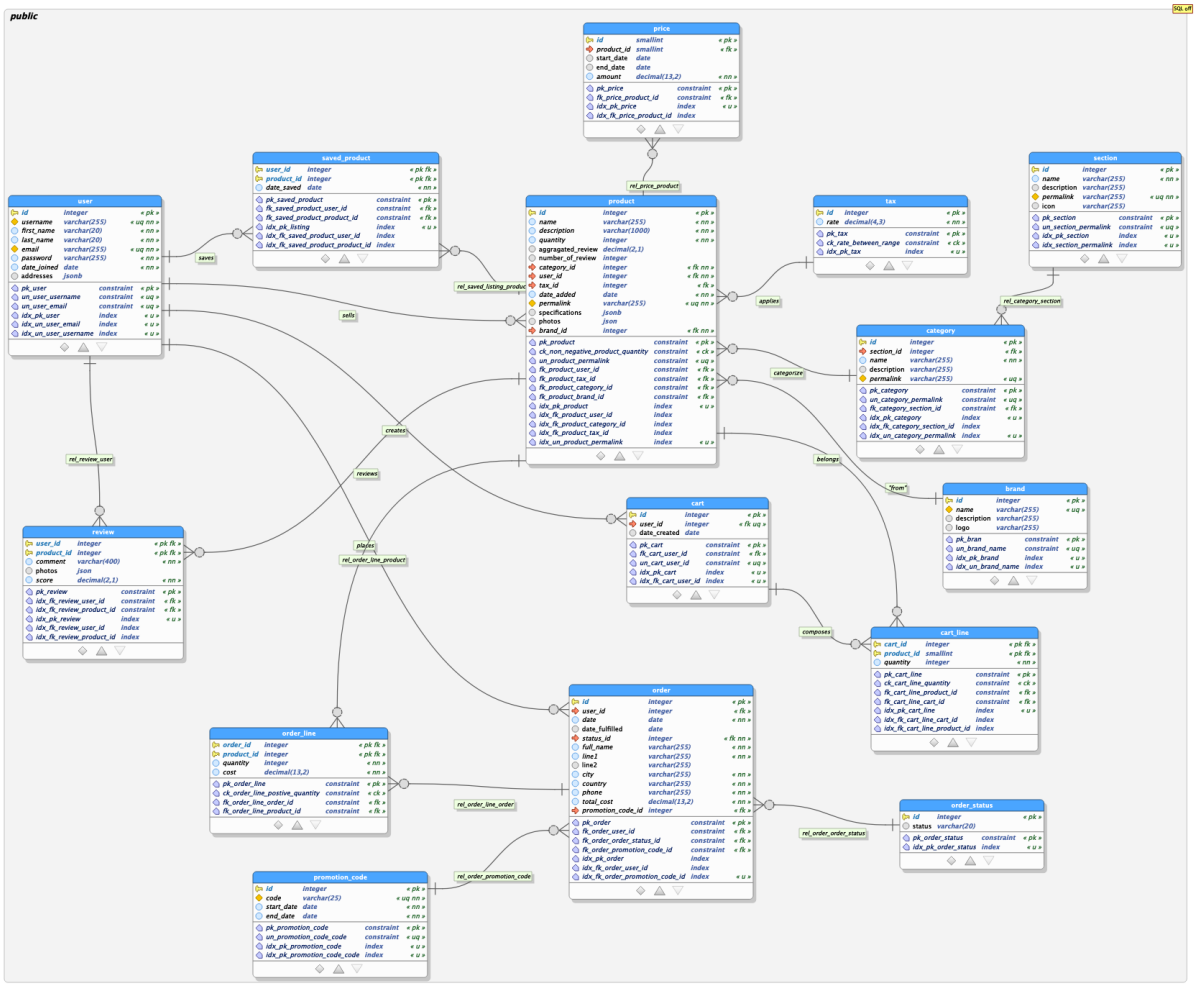
complicated and would ruin the purpose of using the framework.

You also have models like Order, which has a one-to-one relationship with Order Status as each order has only one status. This one-to-one relationship is a composition relationship, again this is outlined in the entity-relationship diagram. Likewise we also have the Category and Section models which keep track of the various product categories and their sub sections. For example, clothes would be a category, but children would be a section of clothes. This is a one-to-many relationship which is also repeated for the analysis of the entity-relationship diagram.

Database

354TheStars is an e-commerce website. This implies that a lot of information will need to be stored and quickly accessed. Consequently, a database is required for our design. Information can be stored on the server's file system using files. However, reading and writing to files are long and inefficient. Therefore, the database chosen is PostgreSQL and SQLAlchemy is used to handle communication with the back-end [8]. **Figure 3.14** shows the entity-relationship diagram model for our database. The ERD also shows the relationship between each table of the database. Each table also represents one of the classes in **Figure 3.7**.

Figure 3.14.: Database Entity-Relationship Diagram



The entity-relationship diagram shows the connection between every table and the type of relationship that they entail. For example, the Order and OrderLine table share a one-to-many composition relationship, because one order could contain many orderlines. Similarly, the same relationship applies for the Cart and CartLine tables.

The Products table possesses many relationships with other tables. For example, it has a one-to-one relationship with the User table, because a particular product listing is being sold by one seller and thus a unique user. Furthermore, a product can have more than one price, however, only one price can ever be active at any given moment. As such, the Product table also has a one-to-many composition relationship with the Price table. Every product has a category, as such, Product has a one-to-one composition relationship with Category. Meanwhile, Category has a one-to-many relationship with Section since one category can have many sections or subcategories.

The User table is perhaps one of the most important. A user can be a buyer and a seller, consequently, a seller can receive reviews concerning their service quality. This is a one-to-many relationship, because a User can receive many reviews. In addition, the user can buy products from other sellers. Everything a User buys is stored within their profile. Therefore, a User has a one-to-many relationship with Order since the user can have many orders. Remember, how a product has a one-to-one relationship with user, well a user has a one-to-many relationship with Product. A product can only have one seller, however, a seller can have multiple products for sale. In addition, a user has only one shopping cart so that is a one-to-one relationship. Although a user can only have one cart, they can save multiple products so User has a one-to-many relationship with Saved Products. These relationship was an important consideration to our design rationale.

3.3. Design Rationale

The system is composed of three subsystems – the front-end, back-end, and database. The reason why this architecture was selected is because of the nature of the requirements and security. It is not impossible to develop a website without the use of a back-end server. A website could relay data directly to a cloud database. JavaScript does allow for that functionality, however, there are a significant number of issues that can arise from this. Firstly, it can lead to security compromises since the server is a second independent system that is needed to relay and interpret information between the front-end and the database. This also allows for decryption to occur on the back-end for sensitive data. Therefore, it was obvious to us that a back-end server would be mandatory to fulfill the requirements.

Furthermore, it improves efficiency as the website will not have to do as much work. A database was also mandatory as the only other alternative to store client data information is using files on the back-ends' file system. Reading and writing to and from files is a very slow process, one that would compromise the user experience. Consequently, our current architecture model, using the tools mentioned in the SRS documentation, better fits our Agile Development method and our team's competence, while being more robust, secure and efficient than other options.

4. Human Interface Design

This section outlines all the external requirements of the interfaces needed for the software to execute properly.

4.1. Overview of User Interface

The user functionality offered by the system will be quite robust for the final product, as we will be employing "Material Design"[2], as mentioned in the SRS, to allow for dynamic interaction and superior user experience. The following section entails the current functionality accessible to the user in the prototype.

In the prototype, the user begins by accessing the Login page, where they can proceed to enter their email and password to continue as a registered user, or they can browse anonymously as a guest. They can also access the account Registration page to create a new account.

The Registration page allows the user the option to enter their first and last names, username, email, password, and confirm the password. Then clicking 'create' grants the user an account, assuming they entered the information in the proper formats. They can also cancel registration and return to the Login page.

Upon creating an account and logging in, the user is automatically redirected to the Home page, which offers the vast majority of the site's functionality. This page allows the user access to the Shopping Cart page containing their items, to categories to view listings of items, to the Account Settings page (not fully implemented for the prototype so this page hasn't been included in the screenshots), go back (which returns to the previous page), logout, or shop now (to do general shopping).

Clicking on shop now or a category brings the user to the Search page, which offers the user the ability to filter items from high to low price or vice versa, filter by price range, or add an item to the shopping cart (not shown in the screenshot).

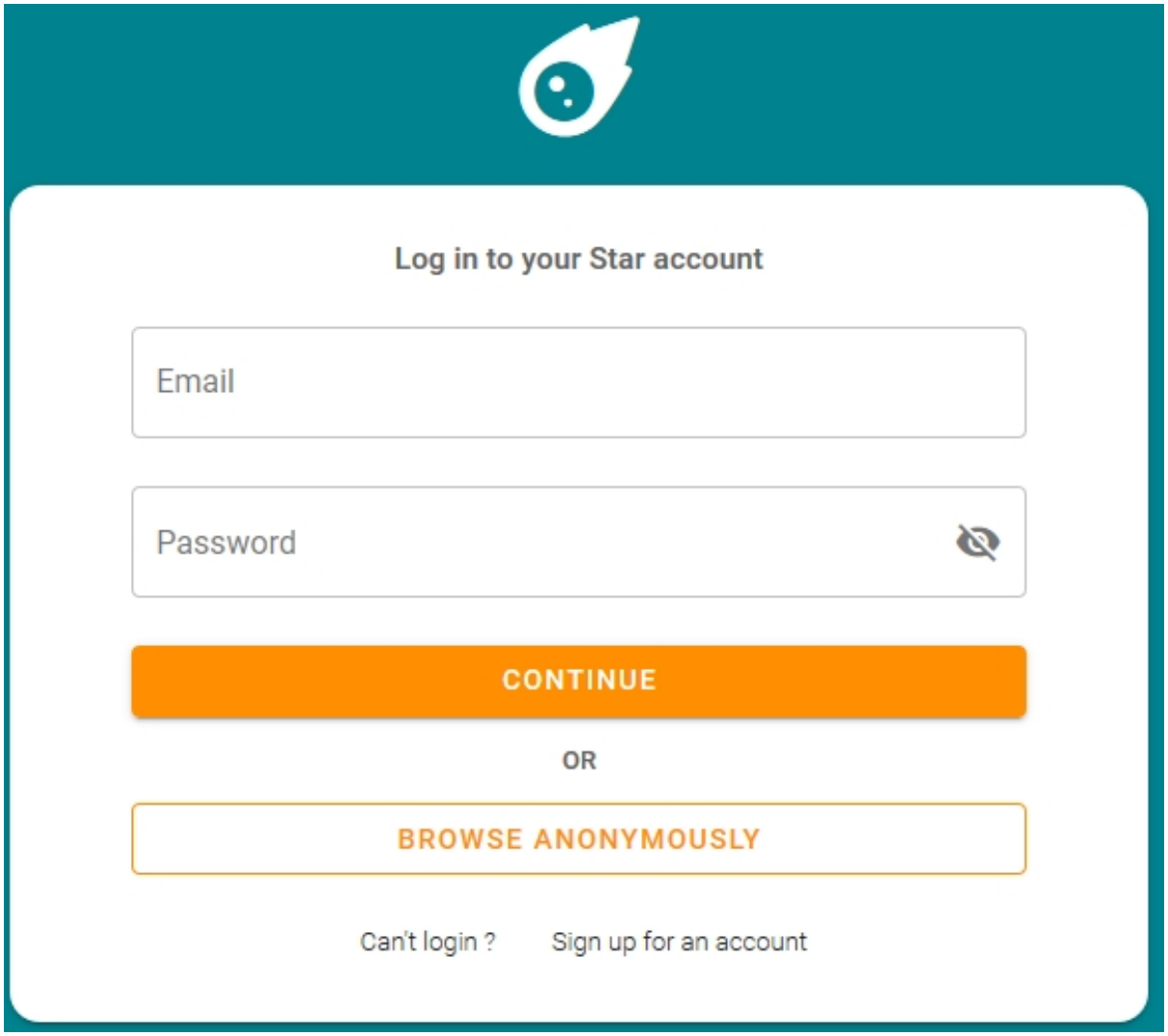
The Shopping Cart page allows the user to view the total cost of the items, as well as delete any items they do not want.

Note: The overview of these pages is limited only to the prototype, and will satisfy all requirements, by offering more robust functionality for the final product.

4.2. Screen Images

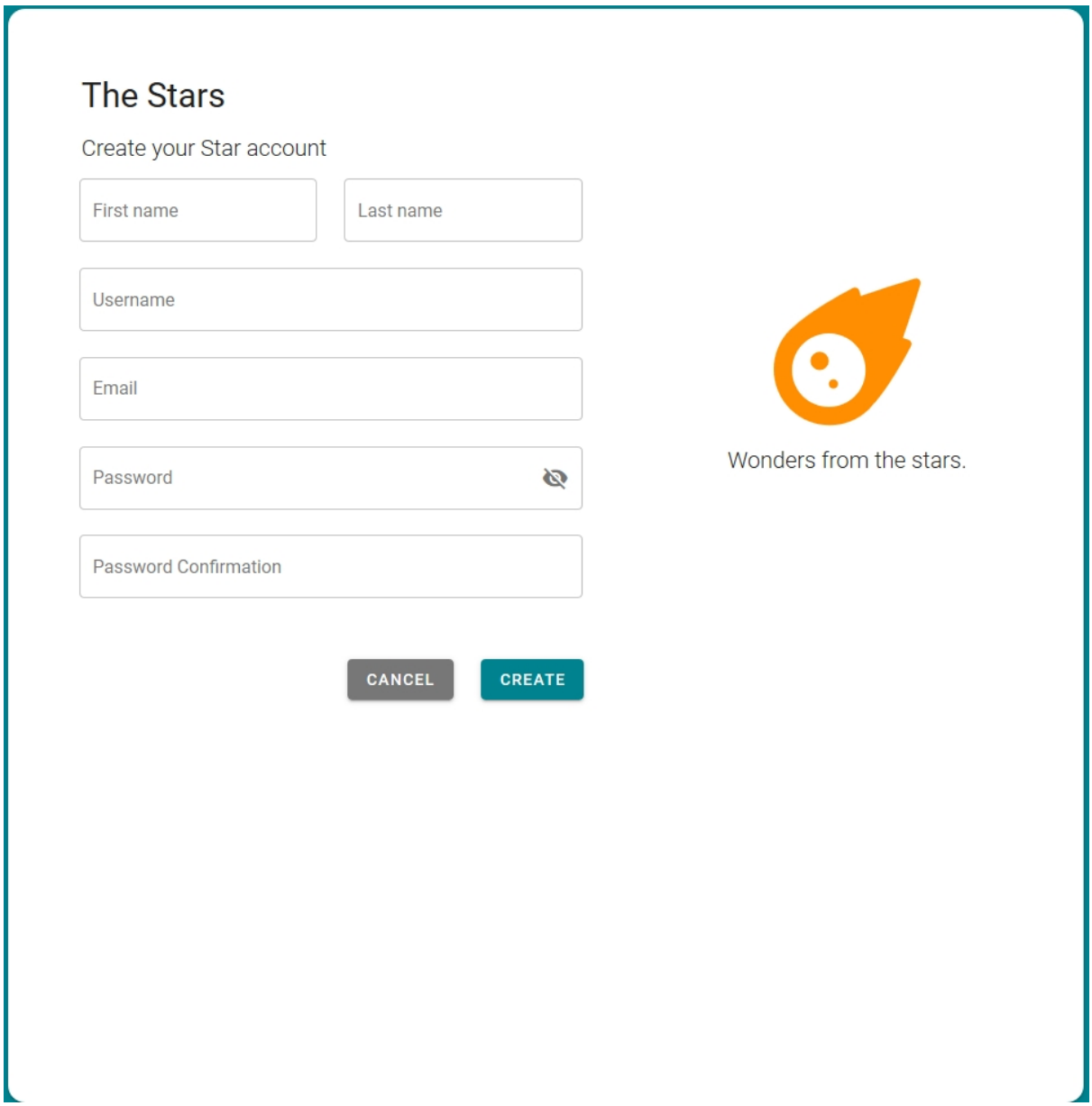
Below is an external interface screenshot of the login page:

Figure 4.1.: Login website page



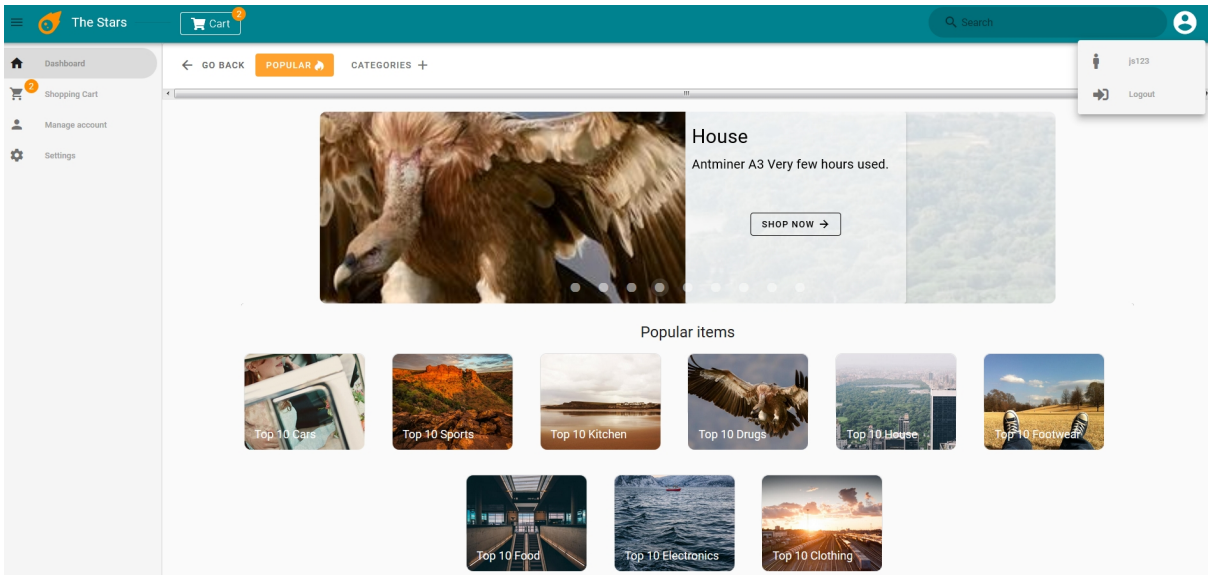
Below is an external interface screenshot of the registration page:

Figure 4.2.: Registration website page



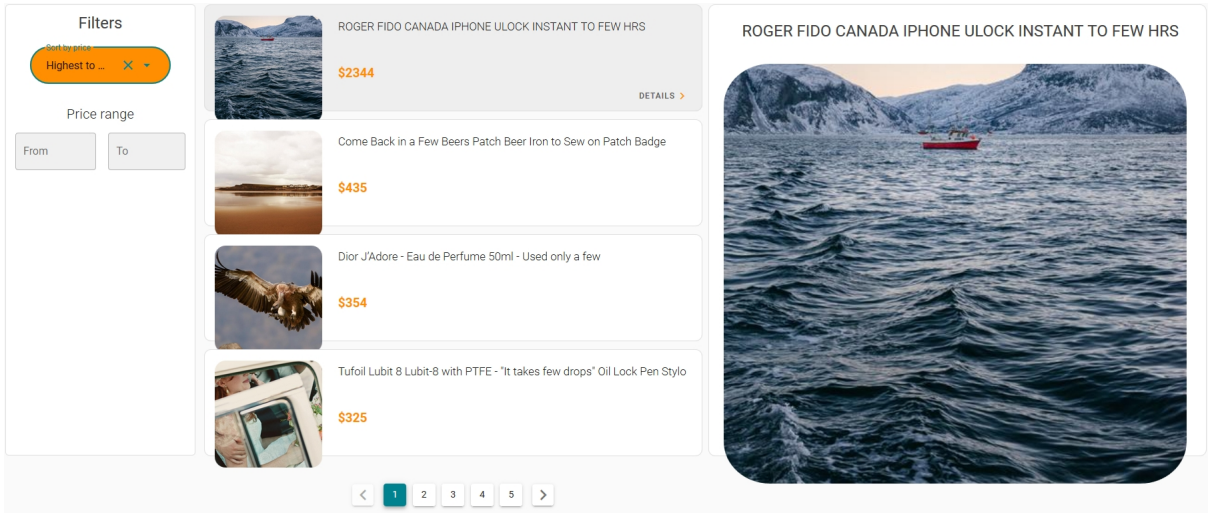
Below is an external interface screenshot of the home / landing page:

Figure 4.3.: Home / Landing website page



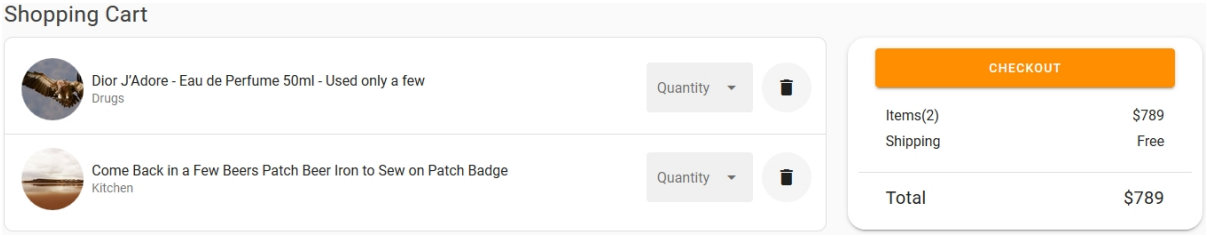
Below is an external interface screenshot of the search page:

Figure 4.4.: Search website page



Below is an external interface screenshot of the shopping cart page:

Figure 4.5.: Shopping Cart website page



4.3. Screen Objects and Actions

The following table (**Table 4.1**) documents the inputs and outputs of each of the previous front pages:

Table 4.1.: Front Pages Inputs and Outputs.

Page	**Input(s)**	**Output(s)**
Login	Fields: Email and password. Buttons: Continue, browse anonymously, and sign up for an account	Redirects to: Registration and home.
Registration	Fields: First name, last name, username, email, password, and password confirmation. Buttons: Show/hide password, cancel and create.	Redirects to: Login and home. Messages: 'A name is required', 'a username is required', 'an email is required', 'a password is required', 'a minimum of 8 characters is required', 'password content is not valid' and 'passwords must match.'
Home	Buttons: Cart, go back, popular, categories, 'account name', logout, dashboard, shopping cart, manage account, settings, shop now, top 10 cars, top 10 sports, top 10 kitchen, top 10 drugs, top 10 house, top 10 footwear, top 10 food, top 10 electronics, and top 10 clothing. Drop-down Menus: Menu and account.	Redirects to: Login, search, account settings, and shopping cart. Buttons Displayed on Click: Cars, sports, kitchen, drugs, house, footwear, food, electronics and clothing.
Search	Select Box: Sort by price. Fields: From and to. Buttons: Details.	Lists: Items sorted by price range. Mouseover: Display item basic info and picture.
Shopping Cart	Select Box: Quantity. Buttons: Remove and Checkout.	Calculates: Total cost of items dynamically.

5. Conclusion

In conclusion, 354TheStars is built on top of a solid foundation of reputable and known industry tools that power our front-end and back-end subsystems. The system communicates externally with the PayPal payment processing system and the system's user. The system is developed using an Agile development method that involves iterative progression. Our user interface is designed using material design which provides clean and modern aesthetics that we believe our users will love. The purpose of the software is to offer a new solution for online shoppers and expand our market share such that we can maximize our profit margins.

Due to the changing nature of software development, we have enclosed an appendix which tracks the functional and user interface requirements changes that we have encountered in this project. This is to be considered as an update to the SRS document. Refer to these SRS changes in the following section.

Appendices

A. SRS Changes

A.0.1. User Interface Design Changes

The user interface design changes are:

- First page / homepage should be attractive.
- Display items which are cheap and attractive or with different variety.
- Display advertisements with help of cookies.

A.0.2. Functional Requirements Changes

The functional requirements change for Account Creation is:

Table A.1.: Account Creation System feature (FR-ACS)[7].

Feature ID	Feature Name	Description
FR-ACS2	Validate Password	Verify that the entered passwords contain at least one upper case, one special character and a minimum length of eight characters. The password is encrypted and then saved in the database.

The functional requirements change for Product Listing is:

Table A.2.: Product Listing System feature (FR-PLS)[7].

Feature ID	Feature Name	Description
FR-PLS13	Set Price	Set the price for the listing. Cannot be free.

The functional requirements change for User Review is:

Table A.3.: User Review System feature (FR-UR)[7].

Feature ID	Feature Name	Description
FR-UR3	Reply to Review	The application shall allow a seller to reply to a review on their profile. (One per review)

The functional requirements change for the Search and Sorting System is:

Table A.4.: Search, Sort and Trending System features (FR-SST)[7].

Feature ID	Feature Name	Description
FR-SST7	Filter	The system shall allow users to filter search results or the store to easily find postings of interest.

The functional requirements change for the Administrative Profile is:

Table A.5.: Administrative Profile System feature (FR-AP)[7].

Feature ID	Feature Name	Description
FR-AP8	Newcomer Fee	The application shall charge a 3% transaction fee instead of 8% for the first 10 items sold by a new seller.

Note: Generate site activity reports (example: number of items sold, listed by top sellers, etc.) already implied by FR-AP1 in the SRS document.

A.0.3. Non-Functional Requirements Changes

The non-functional requirements changes are:

- Sellers no longer offer refunds.
- Include a 'Privacy Policy' page.
- Include an 'About Us' page.
- When a product's quantity goes to zero, don't delete the product permanently from the site.
- The review system is tied to the seller and buyer, not the product.
- Mobile-version is not to be implemented, instead do website-external study paper on how long it would take and what technology it would use.

Alphabetical Index

Glossary

application programming interface A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service[6].

back-end The part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data[6].

cache An auxiliary memory from which high-speed retrieval is possible [6].

cookie A packet of data sent by an Internet server to a browser, which is returned by the browser each time it subsequently accesses the same server, used to identify the user or track their access to the server [6].

database A structured set of data held in a computer, especially one that is accessible in various ways[6].

front-end Relating to or denoting the part of a computer system or application with which the user interacts directly[6].

interface is a mark up language specially suited for scientific documents[6].

query is a request for information from a database [10].

schema A schema is a collection of database objects (currently, tables) associated with one particular database username [9].

server A computer or computer program which manages access to a centralized resource or service in a network[6].

session The session of activity that a user with a unique IP address spends on a Web site during a specified period of time [10].

stand-alone of computer hardware or software able to operate independently of other hardware or software[6].

token A sequence of bits passed continuously between nodes in a fixed order and enabling a node to transmit information [6].

Acronyms

API Application Programming Interface.

ERD Entity-Relationship Diagram.

JSON JavaScript Object Notation.

SRS Software Requirements Specification.

UML Unified Modeling Language.

Bibliography

[1] Douglas K Barry. *Representational State Transfer (REST)*. Nov. 2019. URL: https://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html.

[2] Matt Brian. *Google's new 'Material Design' UI coming to Android, Chrome OS and the web*. June 2014. URL: https://www.engadget.com/2014/06/25/googles-new-design-language-is-called-material-design/.

[3] *Flask*. URL: https://palletsprojects.com/p/flask/.

[4] *JSON*. URL: http://www.json.org.

[5] Olga Ormandjieva. *Software Design Document (SDD) Template*. 2003. URL: https://sovannarith.files.wordpress.com/2012/07/sdd_template.pdf.

[6] Oxford University Press. *Lexico dictionary*. Nov. 2019. URL: https://www.lexico.com.

[7] *Requirement Changes Interview with Dr. Hakim Mellah and Dr. Aiman Hanna*. personal communication. In-Person Interview. Oct. 2019.

[8] *SQLAlchemy*. URL: https://www.sqlalchemy.org.

[9] Stephens, Plew, and Jones. *Sams Teach Yourself SQL in 24 Hours, 4th Edition*. Sams, May 2008.

[10] Webopedia. *Webopedia: Online Tech Dictionary for IT Professionals*. Nov. 2019. URL: https://www.webopedia.com/.