CS 476
Project Report

uBooks

Mykyta Chernenky
chernenm@uregina.ca
and
Brea Monaghan
breamonaghan@gmail.com

05-03-2020

List of Figures

Contents

1. Problem definition and background

1.1 Types of problems that are being solved

University of Regina (U of R) students have many responsibilities. One example is the responsibility of purchasing a textbook or book for a specific class. However, acquiring a textbook has become a problem and/or annoyance for many students. For the reasons below, textbooks and/or books required for specific classes often become a problem and/or annoyance because:

- They become out of date really quickly
- They are expensive
- They are considered an extra unnecessary cost

In addition, there have been times when the U of R Bookstore is out of a specific book and students must turn to other places to acquire it. This leads to students needing to order a book to their house, purchase online in a PDF form or purchase at another physical store. Hence, this brings this report to the first very large problem that uBooks will be solving. uBooks will provide an easy route to search and purchase books required for University classes. A second example of a student responsibility is the selling of textbooks or books. For the reasons below, selling used books required for specific classes often becomes a problem and annoyance because:

- Selling books is considered a hassle
- Some students do not even know that the U of R Buy Back program exists
- The BuyBack program only occurs several days a year
- BuyBack program only offers a 50 percent offer on used or new prices
- BuyBack program is not always able to buy back your books
- U of R Book Store website has a very limited functionality
- U of R Book Store does not provide an appealing and user-friendly interface
- U of R Book Store does not provide an advanced search functionality
- Selling through Facebook Textbook Buy and Sell group is not user friendly

In addition, there is a lot of time that goes into selling or searching for specific books within the Facebook Textbook Buy and Sell groups. It requires hours of scrolling looking for potential sellers and/or buyers. Hence, this brings this report to the second very large problem that uBooks will be solving. uBooks will provide an easy route to sell books under title, author, edition, year and other classifications.

1.2 Motivations

As students ourselves, we wanted to come up with an application that solves our textbook buying and selling problems and helps thousands of others at the same university. We simply do not have this type of application being used or considered by students. The U of R Book Store website does not provide students with the ability to sell their own books. The Buy Back program that happens twice a year is only done at the physical store, which creates an inconvenience and a long line up. Moreover, the university does not facilitate students who want to sell their books to other students, and not the university.

In our new web application uBooks, we aim to provide the university students with the ability to buy and sell books anywhere at any time. Also, this will allow students to sell books for a reasonably higher price opposed to the fixed price that the university offers. As for students who want to buy textbooks, they will be able to buy the books for a cheaper price than at the university bookstore.

U of R Book Store website does not have an advanced search feature that allows students to search a book by title, genre, year, etc. With the large number of books for sale, it can get very frustrating for a student to find a book they want. To alleviate this problem, we have implemented an extensive search functionality that enables students to find the desired book with ease.

Judging by the poor and unresponsive interface and the limited functionality of the U of R Book Store website, we can conclude that it has been built using some old development technology. Poorly designed user interface creates a big inconvenience for the students and impacts the user experience with the website. In our new web application, we aim to provide exceptional user interface and a reasonably good user experience.

Mykyta's personal motivation is to build a much better web application than the U of R Book Store. The newly built application must provide an advanced search functionality and allow students to easily sell their own books. Moreover, the interface must be well designed to provide a pleasant user experience.

Brea's personal motivation is that she has over a dozen textbooks simply sitting at home because selling them is a hassle that she does not have time for. If we can help students like ourselves sell textbooks with ease and get some of their hard earned money back into their pockets, that is our motivation goal.

2. Feasibility Study

uBooks(University Books) is a pilot web application made strictly for the university students. The goal is to provide students with all necessary tools to sell, trade and buy the university books. uBooks will be a free-to-use platform, that will be funded by the advertisements. We are diving into a market that has high demand but not enough supply. As a textbook/book buying and selling web application, uBooks will be entering a Blue Ocean strategy in Saskatchewan and might enter a Red Ocean if we plan to expand further. By entering the Blue Ocean market at the University of Regina, we will be able to test and learn from our successes and mistakes before expanding. However, we also will be entering a Red Ocean immediately because we are aiming to succeed where the BuyBack Program and the Facebook Market Place/Groups could not.

In order to meet the growing competitive pressures, we aim to provide a free to use service with an exceptional customer experience. These pressures are coming from our main competitor, the University book store and other libraries with the buy back policy. Our application has a very high chance of success on the market as it allows students to buy and sell book from/to each other for a much better price. Using our service, the students are able to decide their own price for the book they want to sell. Currently the university is only willing to buy students' books for a fixed price that is unreasonably low. There is a distinct on a mark up once the university resells the books they bought from the students.

In conclusion, the students are better off buying and selling books among themselves instead of dealing with the U of R Book Store.

2.1 Prepare a Projected Income Statement

An income statement has been created for uBooks for the year 2021 when it is fully functional and available to the public, see Fig. 2.1. We estimated that in the first year, we will manage to get 3,500 users. Each user will on an average have at least 2 transactions within 2021, which would be equal to 7000 transactions. The average number that we calculated for a used textbook being sold by a student is $50. We will take 5% percent commission on each transaction and will receive an average of $2.50 per transaction. Therefore, our gross sales, net sales and gross profit is 7000 multiplied by 2.5, which equals $17,500. We will have no cost of goods sold as it is an application-based service. Every week for a year, the two of us founders will dedicate an hour of our time at a $15/hr rate of wage. Therefore our salaries/wages is 52 weeks multiplied by 15, which is $780. Multiply 780 by the 2 founders is $1,560 for salaries/wages. Insurance would be an average of $5,459 and legal fees would be an average of $2,300. We would budget to 100 dollars per month, for 12 months for advertising through Facebook and Instagram which would total up to $1,200. Our total expenses would be $10,519, leaving us founders with a net income of $6,981.

Income Statement 2021	
Revenues	
Gross Sales	$17,500
Less: Sales returns & rebates	-
Net Sales	$17,500
Costs of Goods Sold	
COGS	-
Gross Profit	$17,500
Expenses	
Operations	
Salaries & Wages	$1,560
Finance & Administration	
Insurance	$5,459
Legal fees	$2,300
Sales & Marketing	
Advertisting	$1,200
Total Expenses	$10,519
Net Income	$6,981

Figure 2.1: Income statement 2021

2.2 Conduct a Market Survey, or Perform Market Research

For a Market Survey, we have conducted an analysis on how desirable our product is and what we can do moving forward. Here are the 10 questions asked:

1. What is your age group? (ex.Under 18, 18-24, 25-34 etc). Shown in figure 2.2
2. Are you a University student? (Yes or No). Shown in figure 2.2
3. How would you rate the idea of uBooks? (Extremely desirable - Not at all desirable). Shown in figure 2.2
4. How likely are you to purchase any of our app if offered in the app store? (ex. 99 cents one time payment). (Extremely likely - Not likely at all). Shown in figure 2.2
5. How would you rate the value for money of the product (ex. 99 cents one time payment)? (Very Expensive - Very Cheap). Shown in figure 2.3
6. Would you rather the application be free with ads or purchasable with no ads? (Other and suggestions is an available selection). Shown in figure 2.3
7. Which of the following words would you use to describe uBooks? Select all that apply. (Reliable, Useful, High Quality, Unique, Good value for money, Overpriced, Impractical, Ineffective, Poor Quality and/or Unreliable). Shown in figure 2.3

8. How likely is it that you would suggest uBooks to a University student? (1 being Not likely at all - 10 being Most likely). Shown in figure 2.3

9. Do you have any comments, questions or concerns? Shown in figure 2.4

From our Market Survey, we had 14 responses and determined that our idea is a good idea and should be a Go.

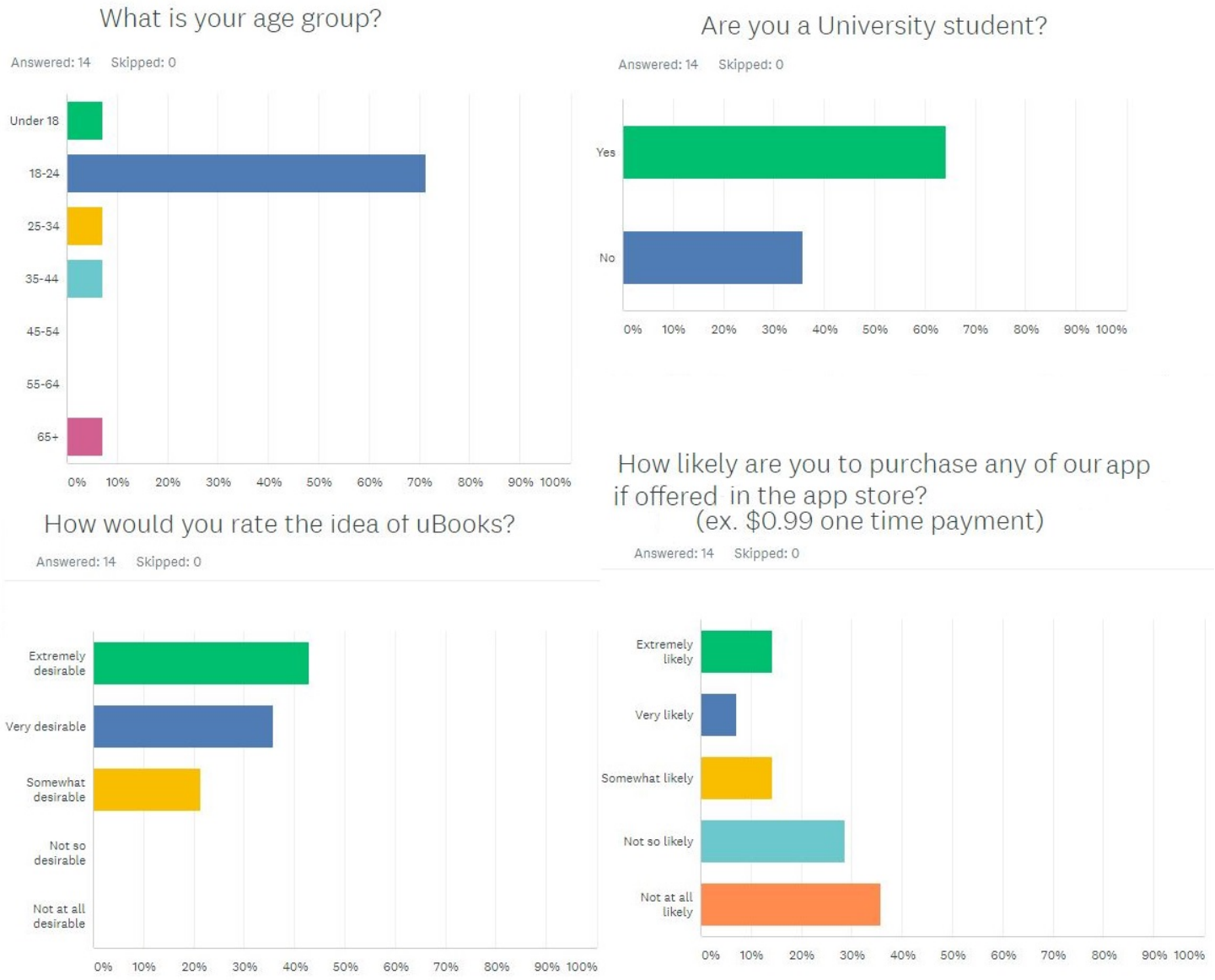


Figure 2.2: Survey Results - 1

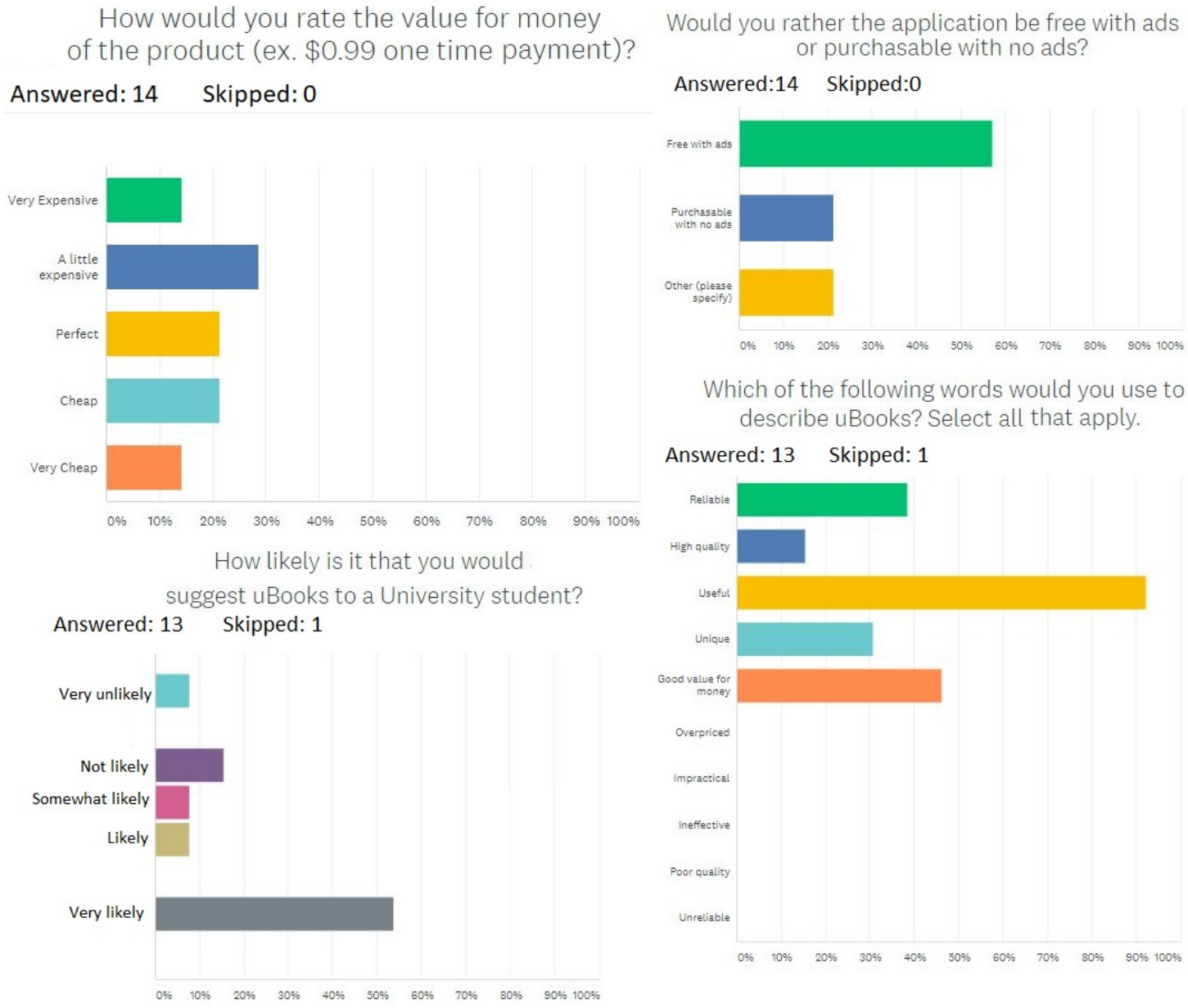


Figure 2.3: Survey Result - 2

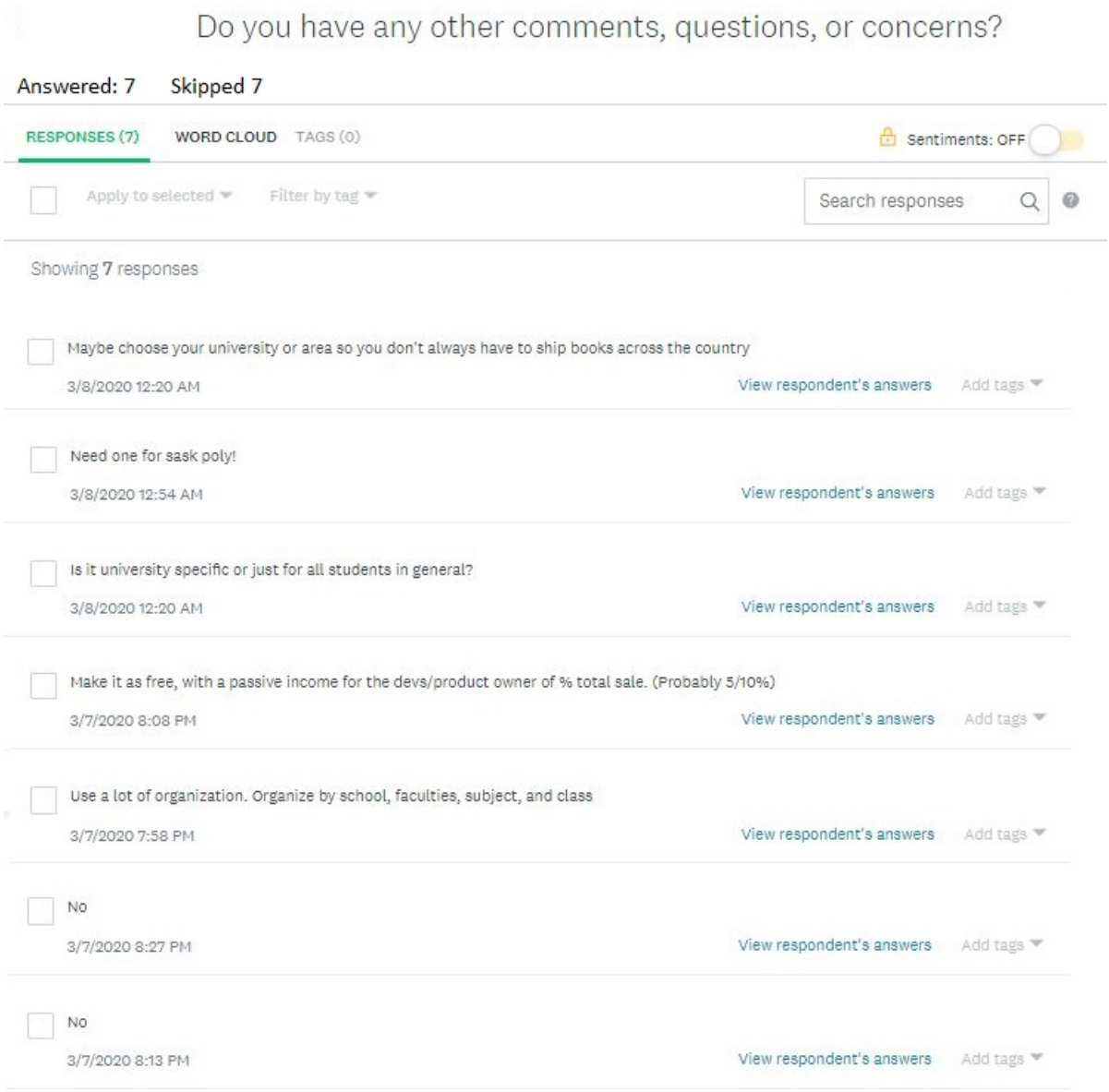


Figure 2.4: Survey Result - 3

2.3 Plan Business Organization and Operations

The main plan for business and organization will be labor controlled on our behalf. It will take time to create, test, officiate, promote and then administrate. Our start up costs consists of labor costs because creating and testing uBooks takes time. Keeping uBooks in operation will also consist of labor costs because it will take time for us to continue to test, promote and administrate.

2.4 Prepare an Opening Day Balance Sheet

Due to our business idea being app based and solely created by us, our Balance sheet will be extremely bland. Referencing Figure 2.5, our cash assets, current liabilities, total liabilities and shares will be zero. Our Software asset must be worth $6,981 to balance with our Net Income/Total Equity as shown in Figure 2.1. This leads to our Total Liabilities and Equity being $6,981.

Balance Sheet 2021	
Assets	
Cash	-
Software	$6,981
Total Current Assets	$6,981
Liabilities & Equity	
Current Liabilities	-
Total Liabilities	-
Equity	
Shares	-
Net Income	$6,981
Total Equity	$6,981
Total Liabilities & Equity	$6,981

Figure 2.5: Balance Sheet 2021

2.5 Review and Analyze All Data

After reviewing and analyzing all data, we have determined our goals are realistic in creating this web application. We do realize that this project will take a decent amount of our time, but are determined to come out of this with a good project and project report. As always, one must have a contingency plan. Our original plans were to make a little bit of income through ads and/or banner ads, but since specifically reviewing our survey, we determined that that was not the best of ideas. A suggestion was made and a contingency plan was created. Our contingency plan that we are now following through with is gaining income through a small percentage of commission from purchases made through the application.

2.6 Make a Go/Partial Go/No-Go Decision

All in all, we have decided that our idea is a Go and that our project is feasible. Our commitment to this project will be worth the time, effort and money at the end of it all. uBooks is motivational to us and will continue to fulfill our long term aspirations.

2.7 Project Development Costs

We have come up with a rough estimate of labour costs for the project development. In order to successfully deliver the project, it would require the following human resources: Web Developer, Database Administrator, System Analyst, Project Manager and Tester Analyst. We have calculated the costs using the average hourly rate taken from https://neuvoo.ca/. If we were to hire the developers, the project of such complexity would cost us $ 7,587 CAD plus taxes. However, we are talented and hard working students and thus can save the cost by developing the web application by ourselves.

Job Title	Hours allocated	Cost
Web Developer	120	$ 4,461
Database Administrator	15	$ 661.65
System Analyst	20	$ 777.6
Project Manager	20	$ 879
Software Tester	15	$ 807.8
Total:	190	$ 7,587.05

Figure 2.6: Project Costs

3. Software requirements elicitation and specification

3.1 Functional requirements list

3.1.1 General Functional Requirements

- **Optimized search:** The application provide the user with an extensive search functionality. The user is able to search books by genre, book title, author, etc.
- **Sign In:** The application requires the user to create an account and sign in to use the application.
- **Transaction History:** The application will keep records of the user's transactions.
- **Validation:** The application validates user input to avoid potential errors.
- **Administrator Dashboard:** The application will have the administrator dashboard that allows the website administrator to maintain if need be.
- **Real-time Database:** The application will use API to communicate with the database via HTTP requests.

3.1.2 Customer Functional Requirements

3.1.2.1 Buy Books

- **View all books:** The customer is able to browse through the book catalogue.
- **View book details:** The customer is able to view each book's description and details.
- **Filter by price:** The customer is able to filter books by price.
- **Filter by genres:** The customer is able to filter books by genres.
- **Search Books:** The customer is able to search books by title, genre, book authors, etc
- **Add a book to cart:** The user is able to add the desired book to their cart

3.1.2.2 Sell Books

- **Add a new book for sale:** The customer has the ability to add their own book for sale
- **Remove a book:** The customer is able to remove an existing book that he/she has posted for sale before
- **Edit book information:** The customer is able to edit the information of the book that he/she has posted for sale before

3.1.2.3 Cart

- **Edit Quantity:** The customer is able to edit the quantity of the books in the cart
- **Remove a book from the cart:** The customer is able to remove a book that he/she has previously added to the cart
- **Checkout:** The customer is able to check out once he/she is ready to place the order

3.1.2.4 Orders

- **View History of Orders:** The customer is able to view the history of all orders that he/she has made in the past
- **Check Status:** The customer is able to check the status of their order

3.1.2.5 Profile

- **Edit Profile:** The customer is able to edit their personal information in the profile settings
- **Change Password:** The customer is able to change their password in the profile settings

3.1.3 Admin Functional Requirements

3.1.3.1 Books

- **View all books:** The administrator is able to browse through the book catalogue.
- **Add a new book for sale:** The administrator is able to add a new book for sale
- **Remove a book:** The administrator is able to remove the book from the catalogue
- **Edit book information:** The administrator is able to edit the information of a book in the catalogue

3.1.3.2 Orders

- **View all orders:** The administrator is to view all orders that have been placed by the customers
- **Confirm order:** The administrator is able to confirm a pending order that has been placed by the customer
- **Reject order:** The administrator is able to reject a pending order that has been placed by the customer

3.1.3.3 Book Genres

- **Add a new genre:** The administrator is able to add a new book genre to the list of book genres
- **Remove a book genre:** The administrator is able remove an existing book genre from the list

3.1.3.4 Customers

- **Add a new customer:** The administrator is able create an account for a new customer
- **Edit customer information:** The administrator is able to edit the customer's profile information
- **Remove an existing customer:** The administrator is able to deactivate an existing customer's account if they failed to comply with the website's policies

3.1.3.5 Admins

- **Add a new administrator:** The administrator is able create an account for a new administrator
- **Remove administrator:** The main administrator is able to deactivate an administrator account if they are no longer employed with the uBooks

3.1.3.6 Countries

- **Add a new country:** The administrator is able to add a new country with its associated phone code
- **Remove a country:** The administrator is able to remove an existing country from the countries list
- **Edit country information:** The administrator is able to edit the country's phone code

3.2 Non-Functional requirements

- **User-friendly interface**: The application provides an intuitive and easy-to-use interface that creates a pleasing user experience.
- **Portability**: The application works desktops and mobile devices

3.3 Use Cases, Use Case Diagrams and Activity Diagrams

3.3.1 Analysis

After collecting the functional requirements, we can begin the analysis of the software requirements. In this stage, we will elaborate the requirements and model the use case diagrams. The main goal of the analysis is to provide a better understanding of the problem domain. This encompasses figuring out the details of the outline requirements and designing numerous system models that precisely outline the system's structure, behavior and interactions.

In order to provide a better description of our case study, we will be primarily using UML diagrams for system descriptions. The UML diagram depicts different types of users (actors) and the ways that they interact with each other and the overall system. These diagrams are often accompanied by the use case description and the activity diagram. The UML diagrams along with use descriptions must cover the functional requirements against the system.

We have already identified three main actors: seller, buyer and the administrator. Additionally, we can establish relationships between the actors. There are two types of relationships: generalization and specialization. In our case, the most general actor is the user whose descendants are the seller and the buyer. In other words, the seller and buyer are the concrete actors of the user that is the abstract actor, see Figure 3.1

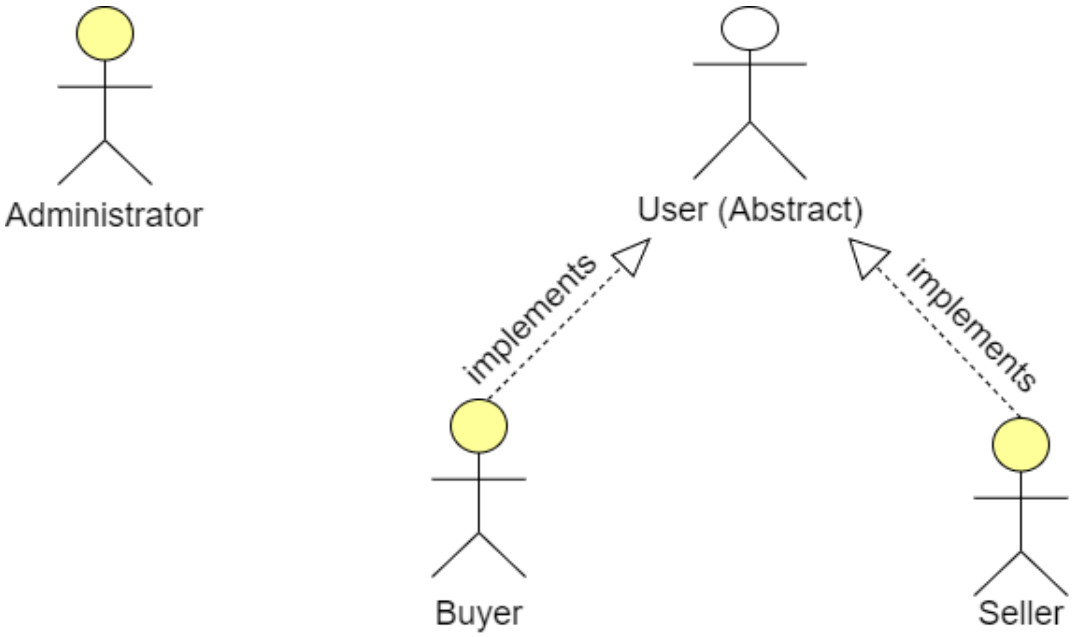


Figure 3.1: List of Actors

3.3.2 Overall Use Case Diagram

Figure 3.2 shows the overall use case diagram. This diagram displays the use cases of all actors together. It clearly defines how administrator, seller and buyer interact with each other and the system. As mentioned before, the seller and the buyer are the implementations of the abstract class User. It's evident that the use cases reflect the functional requirements. Throughout the diagram, I have connected the use cases using 'include' and 'extend' techniques. The include is used when another use cases must be executed before completing the current one. For example, the seller must be in the shopping cart to place an order. The extend is used when the another use case is optional. For example, when you add a new a book for sale, the default quantity is set to one item. However, you have the ability to increase the quantity to two or simply proceed to checkout.

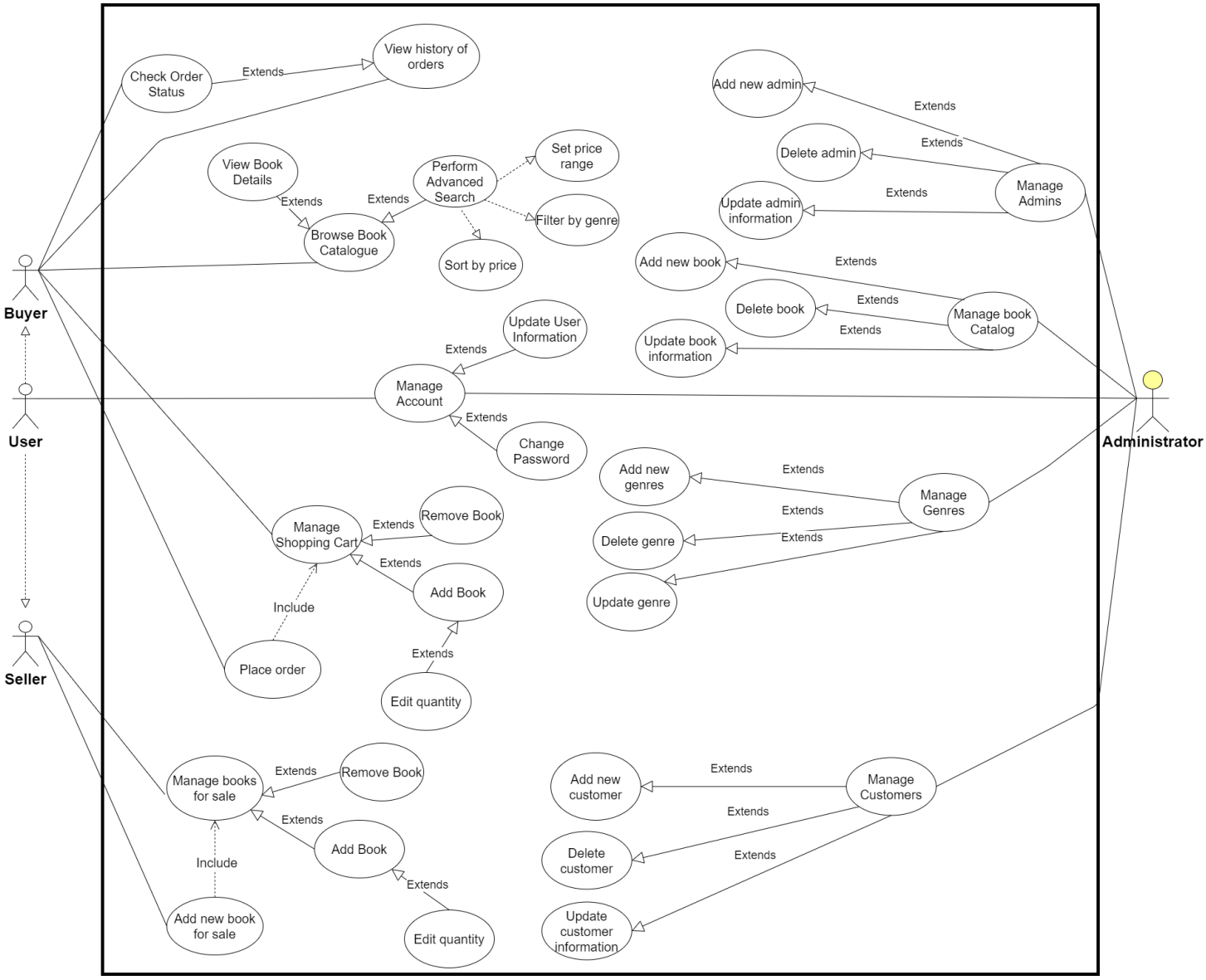


Figure 3.2: Overall Use Case Diagram

In order to provide a more detailed use case description, we have designed separate use case diagrams for each actor. We also described the most important use cases in detail.

3.3.3 Overall Activity Diagrams

Activity Diagram is a behavioral diagram as it describes what must happen in the system we are modelling. This diagram is often used top help people on the development and business sides of a company come together to grasp the same behavior and process.
Advantages of activity diagrams:

- It outlines the steps performed in the Use Case Diagrams
- Visualizes and explains the algorithm logic
- Explains the interaction and workflow between users and the system
- Makes it easy to understand complex use cases

We have designed two general activity diagrams for the customer and for the administrator. The Customer Activity diagram is shown in 3.3. The Administrator Activity Diagram is shown in 3.4. Both diagrams display the overall functionality of each actor. They provide a detailed description of all actions that can be performed by each actor. Moreover, the diagrams depict the flow of actions taken to perform each action. The diagrams are structured in a way so that every action can be performed one after another.

We have also provided smaller, use case specific activity diagrams for every important use case.

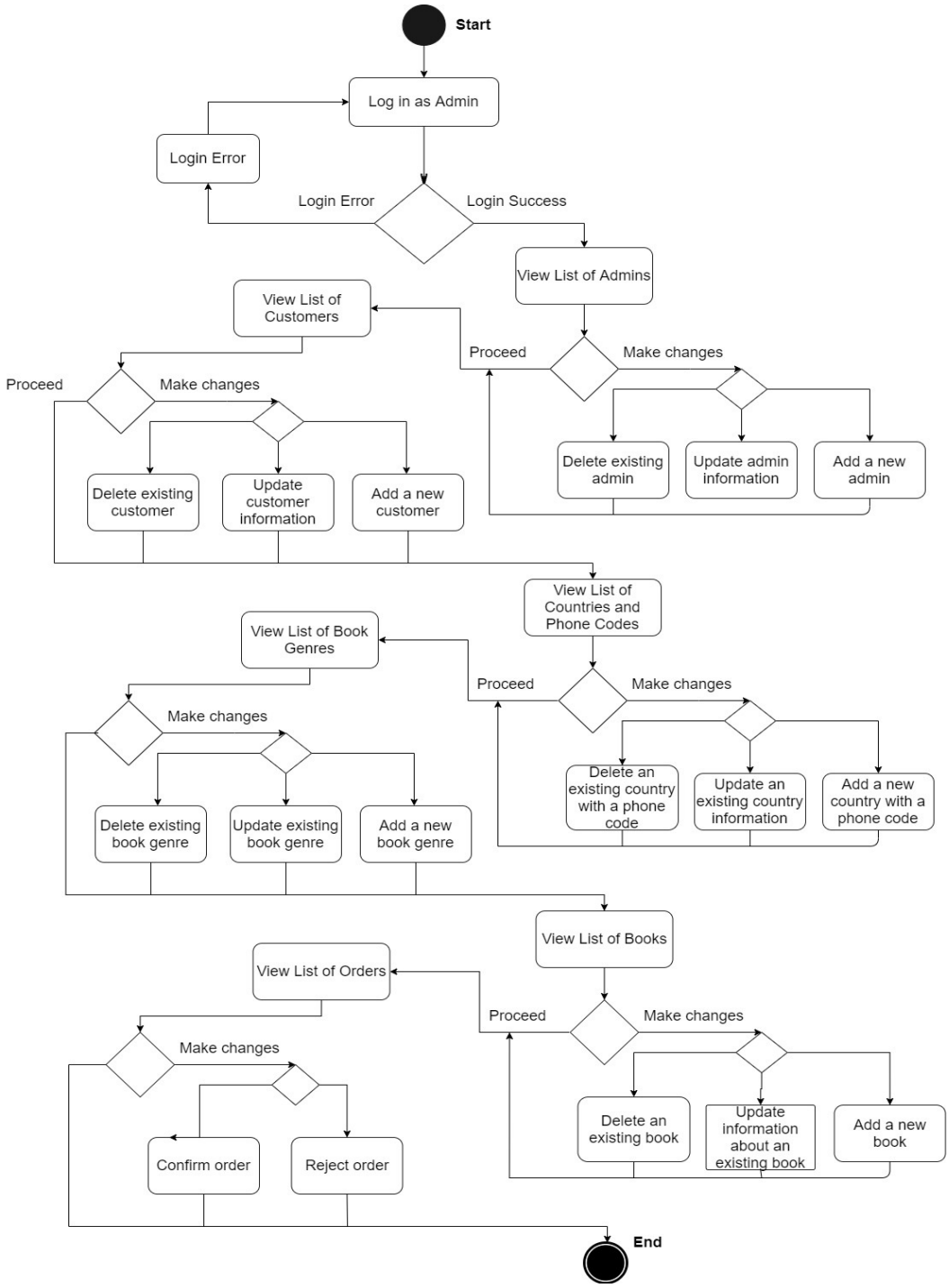


Figure 3.3: Administrator - Overall Activity Diagram

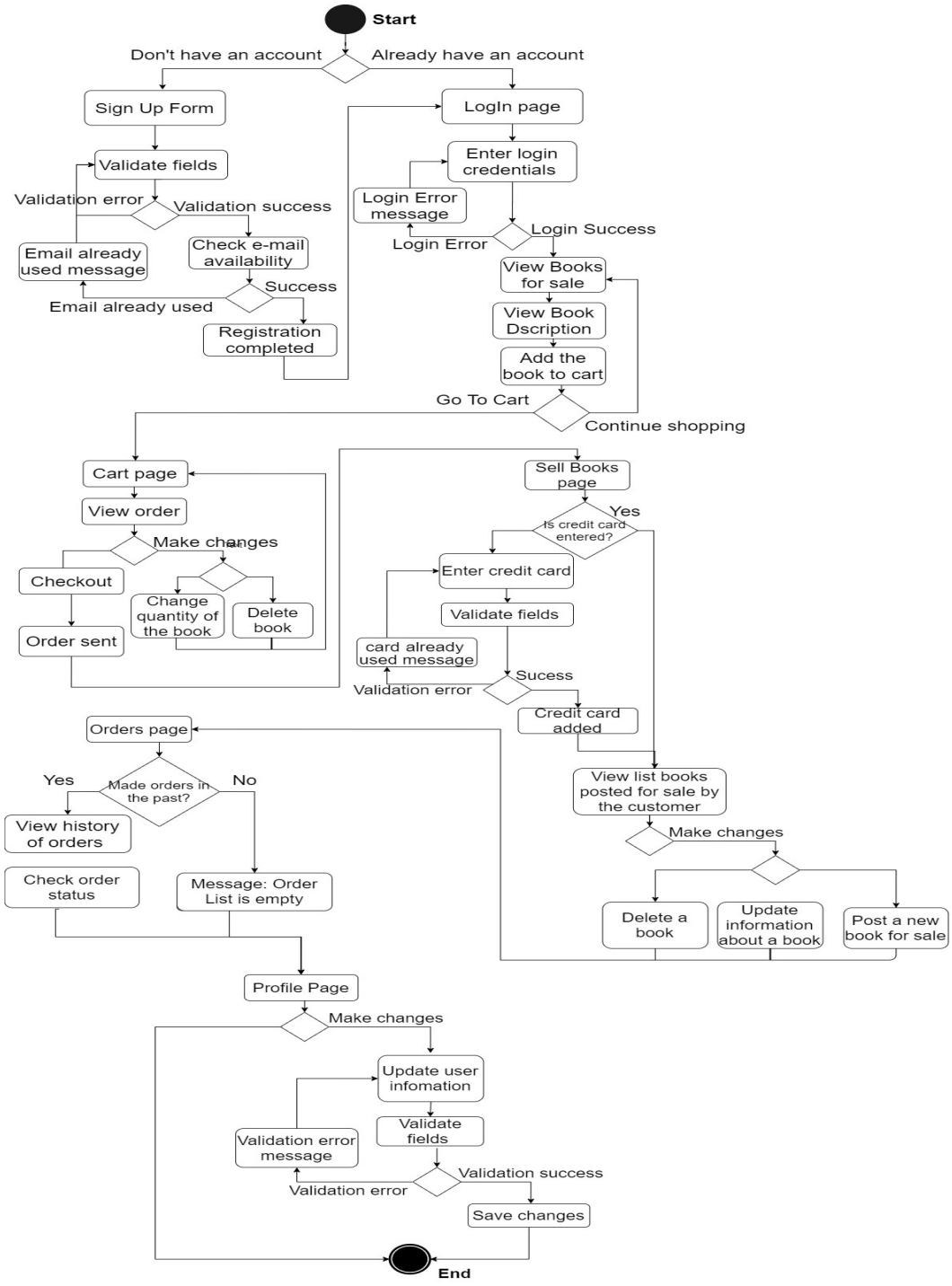


Figure 3.4: Customer - Overall Activity Diagram

3.3.4 User Use Case Description & Diagram

The user use case diagram, shown in Figure 3.5 is the simplest of all. As we previously mentioned, the abstract actor user is implemented by two concrete actors, the seller and the buyer. This is a very common architecture across many applications. Though in some applications the user can also be the abstract class of administrator. Due to the uniqueness and the complex nature of our application, the administrator is a separate actor that does not implement the user.

The reason we have included the abstract actor is because two actors share the same use case group that is Manage Account. Both seller and buyer are able to update their user information and change password.

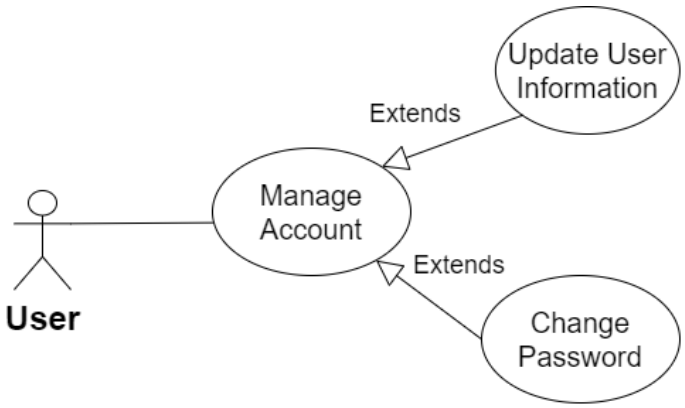


Figure 3.5: User Use Case Diagram

Use case name	Change Password
Actors	User (Seller or Buyer)
Description	The user wants to change their password
Precondition	1. User must be signed in
Postcondition	User has successfully changed their password
Normal Flow	3. Go to Profile Settings 4. Type in a new password
Alternative Flow	N/A
Exceptions	1. Password doesn't satisfy the validation criteria (at least 8 characters long, at least one upper case and at least one numeric character)
Includes	Manage Account

Figure 3.6: Change Password - Use Case Description

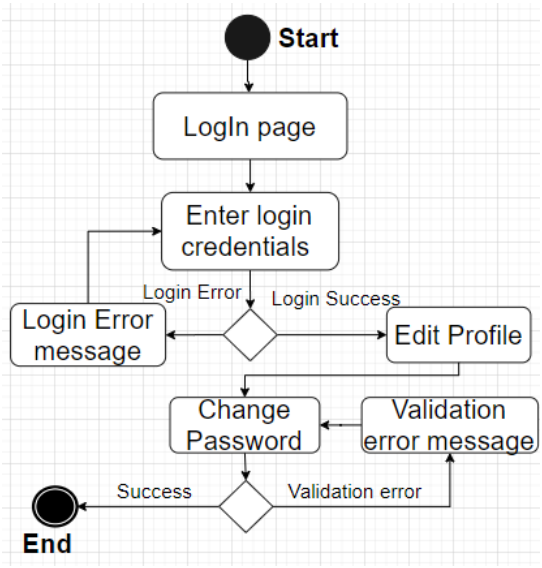


Figure 3.7: Change Password - Activity Diagram

3.3.5 Administrator Use Case Description & Diagram

Shown in Figure 3.8 is the Administrator use case diagram. Since the administrator is capable of doing everything, the diagram is fairly complex. As a result, the diagram has many use cases that describe what the administrator can do. All use cases have dependency with each other and thus are grouped accordingly. The administrator's use cases are divided in the following five components: Mange Admins, Manage Book Catalog, Manage Account, Manage Genres and Manage Customers.

The first use case group, **Manage Admins** extends the following three use cases: Add new admin, Delete admin and Update admin information.

The second use case group is **Manage Book Catalog** which has three extends: Add new book, Delete book and Update book information.

The third use case group is **Manage Account** which has two extends: Update User information and Change password.

The fourth group of use cases is **Manage Genres**. It has the following three extends: Add new genre, Delete genre and Update Genre.

The fifth and the last use case group is **Manage Customers**. This group has the following 3 extends: Add new customer, Delete Customer and Update customer information.

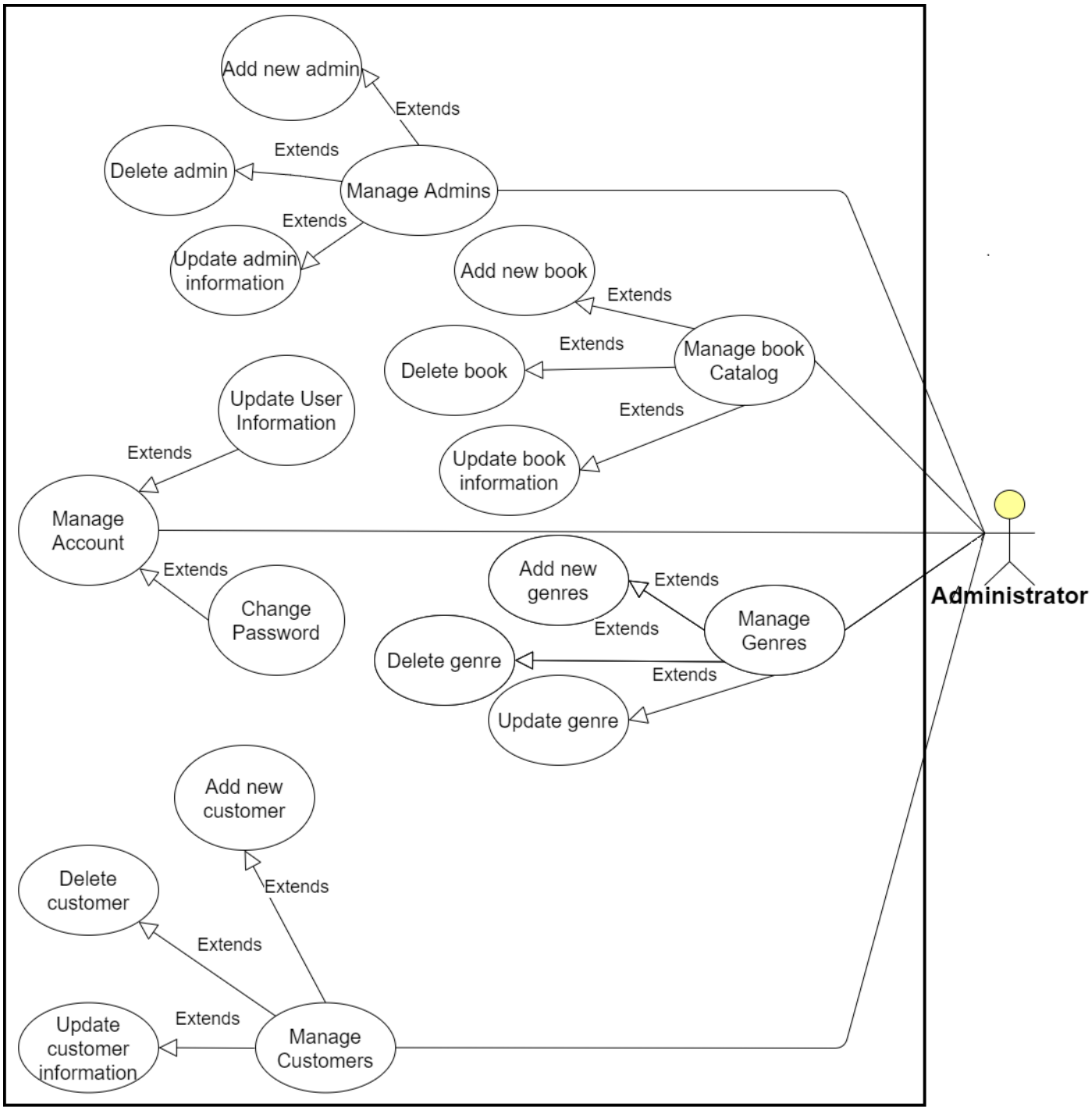


Figure 3.8: Administrator Use Case Diagram

Use case name	Admin Adds a New Book for Sale
Actors	Administrator
Description	The administrator adds a book for sale
Precondition	1. Admin must be signed in
Postcondition	A book has been added for sale
Normal Flow	1. Go to 'Books' page 2. Click the '+Add New' button 3. Fill in all fields with the book's information 4. Click the now available yellow 'Add' button below all the filled-in fields
Alternative Flow	N/A
Exceptions	N/A
Includes	1. Manage Books for Sale

Figure 3.9: Admin Adds a New Book For Sale - Use Case Description

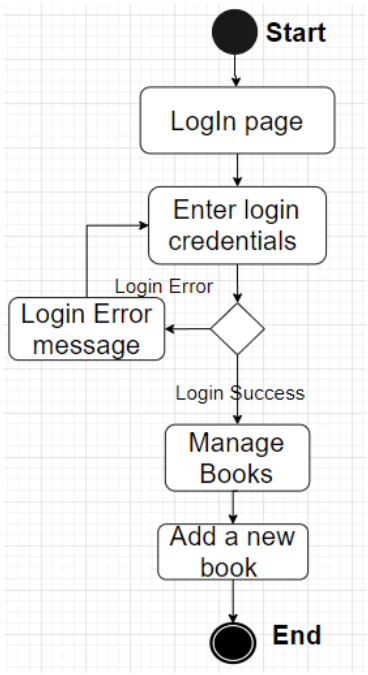


Figure 3.10: Admin Adds a New Book For Sale - Activity Diagram

Use case name	Delete an Existing Book
Actors	Administrator
Description	The administrator deletes an existing book that is for sale
Precondition	An Admin wants to remove a book 1. Admin must be signed in
Postcondition	The book has been removed
Normal Flow	1. Go to 'Books' page 2. Click trash icon on the bottom right corner of the book details' 3. Book will be deleted immediately
Alternative Flow	N/A
Exceptions	N/A
Includes	1. Manage Books for Sale

Figure 3.11: Admin Deletes an Existing Book - Use Case Description

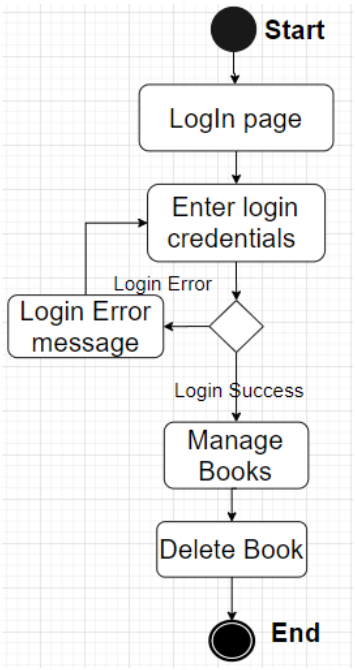


Figure 3.12: Admin Deletes an Existing Book - Activity Diagram

Use case name	Add a New Administrator
Actors	Administrator
Description	The administrator manually adds a new administrator
Precondition	1. The administrator must be signed in
Postcondition	A new administrator is added
Normal Flow	1. Go to 'Admins' page 2. Click the '+Add New' button 3. Enter the account information for a new administrator and click 'finish'
Alternative Flow	N/A
Exceptions	1. The account with this e-mail address has already been created
Includes	1. Manage Admins

Figure 3.13: Add a New Admin - Use Case Description

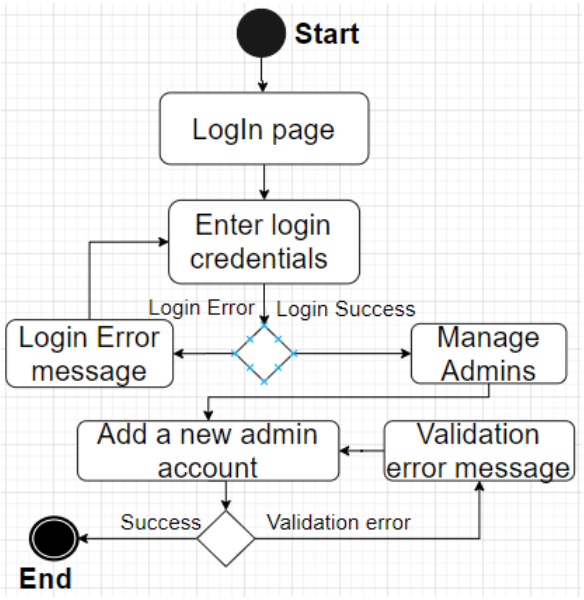


Figure 3.14: Add a New Admin - Activity Diagram

Use case name	Add a New Book Genre Use Case
Actors	Administrator
Description	The administrator adds a new Book Genre
Precondition	An Admin wants to remove a book 1. Admin must be signed in
Postcondition	The book has been removed
Normal Flow	1. Go to 'Book genres' page 2. Click the '+Add New' button 3. Enter the new genre and click the 'Add' button
Alternative Flow	N/A
Exceptions	Exception for Step 3: 1. If genre already exists, an error saying so will pop up and the application will prevent a duplicate from being added
Includes	1. Manage Book Genres

Figure 3.15: Add a New Book Genre - Use Case Description

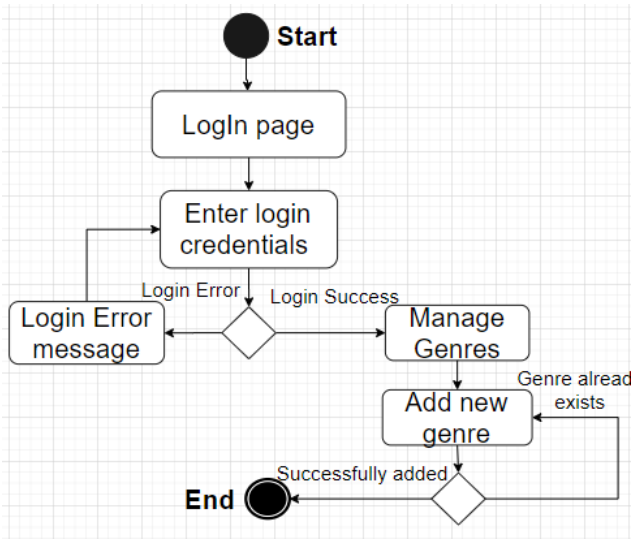


Figure 3.16: Add a New Book Genre - Activity Diagram

Use case name	Add a New Customer
Actors	Administrator
Description	The administrator manually adds a new customer
Precondition	1. The administrator must be signed in
Postcondition	A new customer is added
Normal Flow	1. The administrator goes to 'Manage Customers' page 2. The administrator clicks 'Add New Customer' 3. The administrator enters the account information for a new customer and clicks 'finish'
Alternative Flow	N/A
Exceptions	1. The account with this e-mail address has already been created
Includes	1. Manage Customers

Figure 3.17: Add a New Customer - Use Case Description

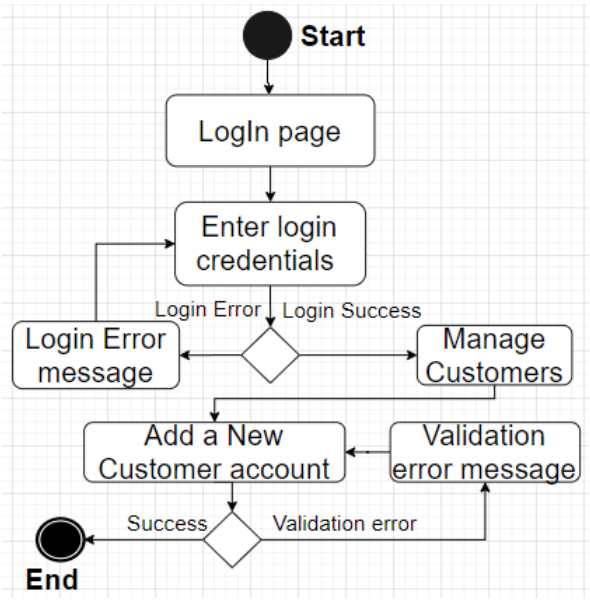


Figure 3.18: Add a New Customer - Activity Diagram

3.3.6 Buyer Use Case Description & Diagram

Shown in Figure 3.19 is the Buyer use case diagram. The diagram displays the actions that can be performed by the buyer. Note that in this example, the Buyer implements the abstract actor User. Most use cases have dependency with each other. Below is the explanation of all use cases and their dependency with each other.

The first use case group is **Browse Book Catalogue** which has two extends: **Perform advanced search** and **vView book details**. The **Perform Advanced Search** is a cluster and consists of the following three use cases: **Set price range**, **Filter by genre** and **Sort by price (Min/Max)**. The next use case is **Place Order** which includes **Manage Shopping Cart**. The reason these two use cases have the 'include' relationship is because the buyer must first be in the Cart page to place an order. **Manage Shopping Cart** has 2 extends: **Remove a book from the cart** and **Add a book to the cart**. Once the book is added to the cart, the buyer is able to **edit quantity**. Finally, **Check Order Status** is extended from **View History of Orders**. This is because the buyer must be in the orders history page to view the status of his/her orders.

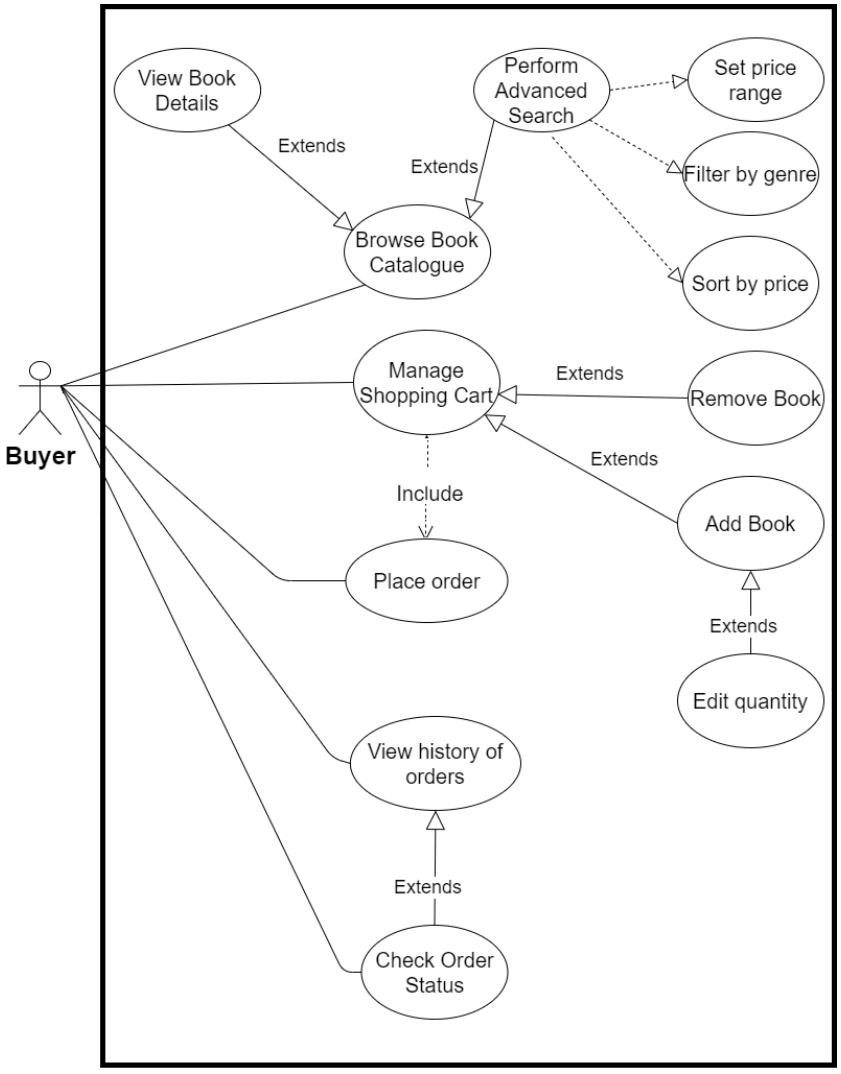


Figure 3.19: Buyer Use Case Diagram

3.3.7 Buyer Use Case Description

Use case name	Add Book To Cart
Actors	Buyer
Description	The User adds a new book to their cart
Precondition	The User wants to add a book to the cart: 1. User must be signed in 2. The desired is in stock
Postcondition	Book has been added to cart
Normal Flow	1. Search for a Book to buy 2. Click the add to cart button to add to cart
Alternative Flow	Alternative scenario 1 : 1. Search for a Book to buy with specific filters 2. View Book Description 3. Click the add to cart button to add to cart Alternative scenario 2: 1. View History of Orders 2. Click on the book you want to purchase again 3. Click the add to cart button to add to cart
Exceptions	Exception for Alternative scenario 1: 1. The book is not in stock and thus can't be purchased
Includes	N/A

Figure 3.20: Add a Book To The Cart - Use Case Description

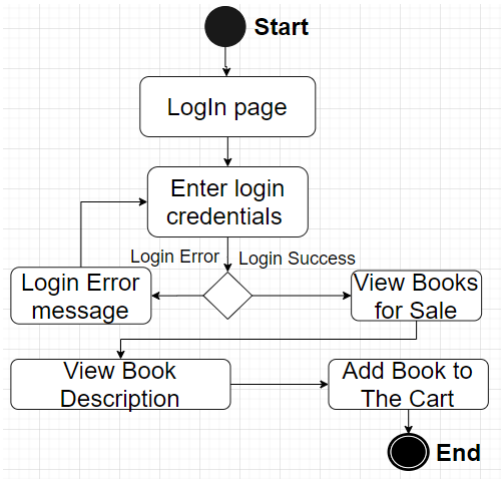


Figure 3.21: Add a Book To The Cart - Activity Diagram

Use case name	Place an Order
Actors	Buyer
Description	The buyer checks out and places an order
Precondition	1. User must be signed in 2. At least one book must be in the cart
Postcondition	Order is completed and pending approval
Normal Flow	1. Go to the 'Cart' page 2. Click the 'Checkout' button
Alternative Flow	N/A
Exceptions	Exception: 1. No books are in stock
Includes	1. Manage Shopping Cart

Figure 3.22: Place Order - Use Case Description

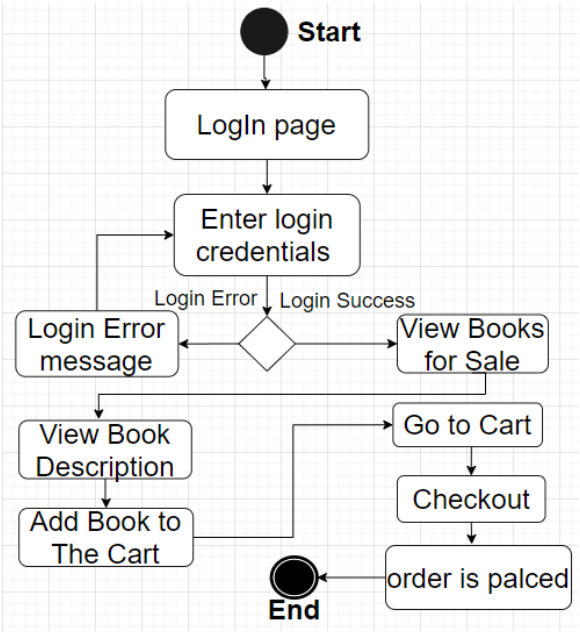


Figure 3.23: Place Order - Activity Diagram

3.3.8 Seller Use Case Description & Diagram

Shown in Figure 3.24 is the seller use case diagram. The diagram displays the use cases that are executed by the seller. As we have previously mentioned, The seller implements the abstract actor User. This is the simplest use case diagram that involves a concrete actor. However, it's main use case is by far the most complicated. This is because it involves two validation checks: one for log in and another for the credit card. Below is the use case description and the activity diagram.

The one and only use case group is **Manage books for sale** which has two extends: **Remove a book** and **Add a new book for sale**. Once seller has added a new book for sale, he/she can **edit quantity** of the added book.

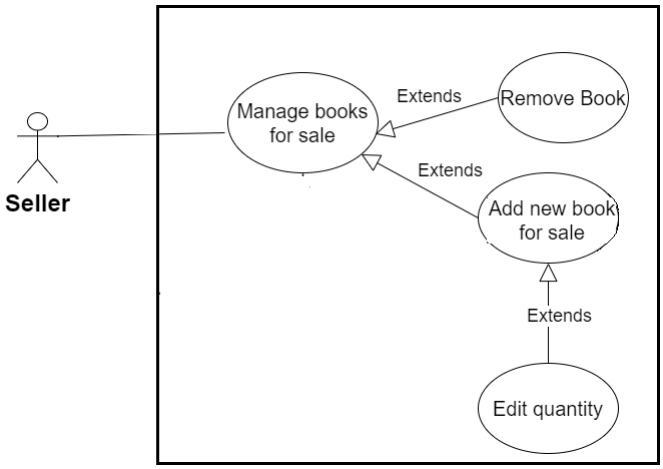


Figure 3.24: Seller Use Case Diagram

Use case name	Customer Adds a New Book For Sale
Actors	Seller
Description	The customer adds a new book for sale
Precondition	1. User must be signed in 2. User has entered their credit card information
Postcondition	A book is added to the list of books for sale
Normal Flow	1. Go to 'Sell Books' page 2. Click the '+Add New' button 3. Fill in all fields with the book's information 4. Click the now available yellow 'Add' button below all the fields
Alternative Flow	N/A
Exceptions	1. User did not previously enter their credit card information
Includes	Manage Books for Sale

Figure 3.25: Seller Adds a New Book For Sale - use Case Description

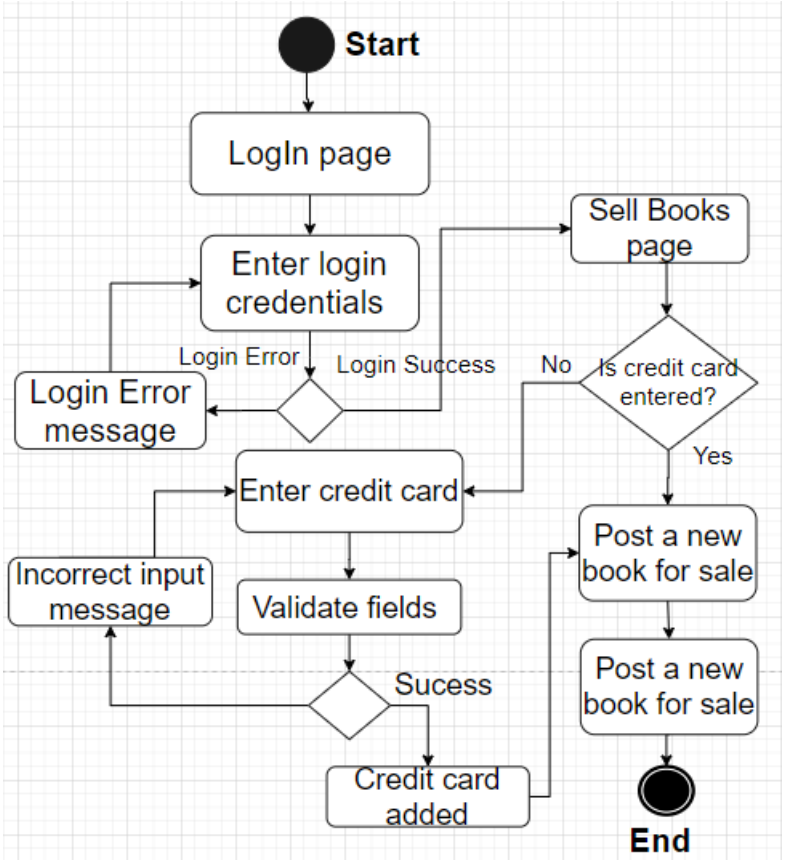


Figure 3.26: Seller Adds a New Book For Sale - Activity Diagram

3.4 Software qualities

We will be focusing on the software qualities correctness, time-efficiency, robustness and user friendliness. First, uBooks must have the correctness software quality because the user wants the service to work as expected. Second, uBooks must have the time-efficient quality because it is a very important factor of the application. It takes seconds to sign up and/or sign in. In addition, our search function rapidly produces results and our service is available 24/7, unlike our competitor the BuyBack Program. Third, uBooks must be Robust because High Performance is our most important software quality. If the Performance of uBooks is unstable and/or unresponsive, then correctness and user friendliness are negatively impacted. Last, uBooks must have user friendliness because of the need for simplicity, cleanliness and intuitive. For example, the uBooks Search function is simpler to use compared our competitors: Facebook groups and BuyBack program.

Here is the list of software qualities:

1. **Correctness:** The ability of software to perform their exact tasks, as defined by their specification.

   (a) **Usability:** Usability refers to the quality of a user's experience when interacting with software.

   - Users of all technological skill levels are able to utilize this web application
   - The Search function allows the user to search a book by genre, author, title and/or price range

- The design is Memorable to provide ease of navigation
- Only Users who have registered can access the system

(b) **Reliability:** An unreliable product is not user-friendly, since it will cause undue frustration for the user. uBooks is a user-friendly product is reliable and does not malfunction or crash. (TechTerms)

- Error messages and warnings must be concise, clear and unambiguous
- The final product will be well tested ensure the Reliability and Maintainability
- Every week, we will check on uBooks to update/maintain our web application, all the while monitoring it daily to ensure its reliability
- uBooks will work under any environment so long as the user has access to the internet
- If a new book is added with a typo, the user can update the book's information immediately
- Only administrators can create other administrators

2. **Time-Efficiency:** How well the software system handles capacity, throughput, and response time. (RequirementsQuest)

(a) **Availability:** The ratio of time a system or component is functional to the total time it is required or expected to function. (WhatIs.com)

- Our service/product is required to be available 24/7
- uBooks is more available than our competitor, specifically the BuyBack Program.
- Availability goes hand in hand with Reliability, because as Reliability increases, so does Availability
- If a user updates (for example) some profile information, the system shall ensure that other users shall automatically see the update within seconds

(b) **Throughput:** The ability to process a large volume of transactions within a certain time frame. (TechTarget)

- uBooks can contain and display the information of 1000+ books for sale
- uBooks can continue to operate properly with up to about 500 users/visitors at any given moment

3. **Robustness:** The degree to which uBooks continues to function in the presence of invalid inputs or stressful environmental conditions (Steve McConnell)

(a) **Reliability:**

- Error messages and warnings must be concise, clear and unambiguous
- uBooks is Extensible due to having a flexible architectural design to cater to future changes
- If a user enters incorrect values on the sign in page or an unauthorized user tries to access the system, the system will not crash
- A User will be unable to add a new book until all required informative fields are filled

(b) **High Performance:** How uBooks performs in terms of responsiveness and stability under a particular workload.

- The speed of the Search function on the Buy Books page is rapid
- High Performance goes hand in hand with robustness, correctness and user friendliness, because if High Performance decreases, so do the others
- As mentioned above, uBooks has a positive throughput

4. **User Friendliness:** User-friendly describes a hardware device or software interface that is easy to use

   (a) **Simplicity:** uBooks is a user-friendly interface is not overly complex. It is straightforward, providing quick access to common features or commands. (TechTerms)

   - Text is clear, visible and coloured accordingly
   - Specifically, compared to our competition, our search function provides the Simplicity that our competitors do not have for buying textbooks
   - Our customers are able to search for books depending on title, author, genre and price range
   - Signing up is a short task

   (b) **Cleanliness:** uBooks has a good UI is well-organized, making it easy/clean to locate different tools and options. (TechTerms)

   - Positive Customer Experience is present due to the visual flow of the UI
   - Fashionable, simplistic and uncluttered design of UI

   (c) **Intuitive:** In order for uBooks to be user-friendly, an interface must make sense to the average user and should require minimal explanation for how to use it. (TechTerms)

   - Customers of all technological skills can utilize our web application with ease
   - Buttons show that they are clickable by being coloured, labeled and/or universally recognizable
   - Anyone can sign up
   - All field names are meaningful

4. Top-Level and Low-Level Design

4.1 MVC Software Architecture

4.1.1 How MVC Works

To start, Model, View, and Controller separates uBooks into 3 components. To explain, I will be quoting from Teacher Tutorials: "

Model: Model represents shape of the data and business logic. It maintains the data of the application. Model objects retrieve and store model state in a database. Model is data and business logic

View: View: View is a user interface. View display data using model to the user and also enables them to modify the data. View is User Interface.

Controller: Controller: Controller handles the user request. Typically, user interact with View, which in-turn raises appropriate URL request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response. Controller is a request handler." (TeacherTutorials).

As you can see on Figure 4.1, the user will enter the URL into the browser, send the request to the controller. The controller will then render the view and the view will send a request to the controller. The view will display the model while the controller manipulates the model. A response is then sent back to browser. Note that AngularJS is an MVC based framework.

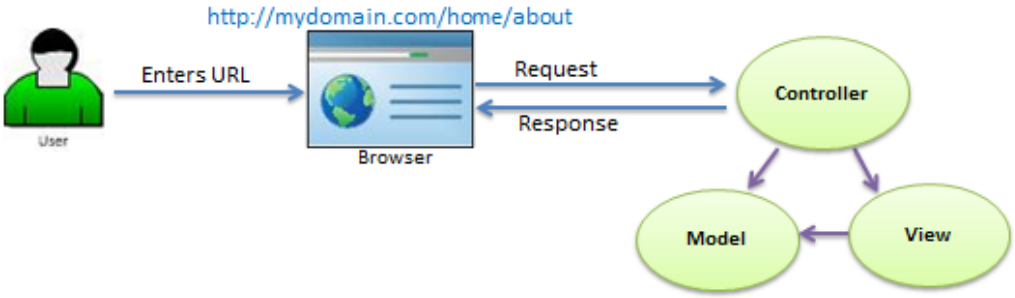


Figure 4.1: How MVC Works

Referencing Figure 4.2, "In general cases there is only one way of data binding in other framework but AngularJS provides two-way data binding it's continuous updates views and models data vice versa." (AmarInfotech)."The model is responsible for managing application data. It responds to the request from view and to the instructions from controller to update itself" (tutorialspoint). "A presentation/view of data in a particular format is triggered by the controller's decision to present the data" (tutorialspoint). "They are script-based template systems such as JSP, ASP, PHP and very easy to integrate with AJAX technology" (tutorialspoint). "The controller responds to user input and performs interactions on the

data model objects. The controller receives input, validates it, and then performs business operations that modify the state of the data model" (tutorialspoint).

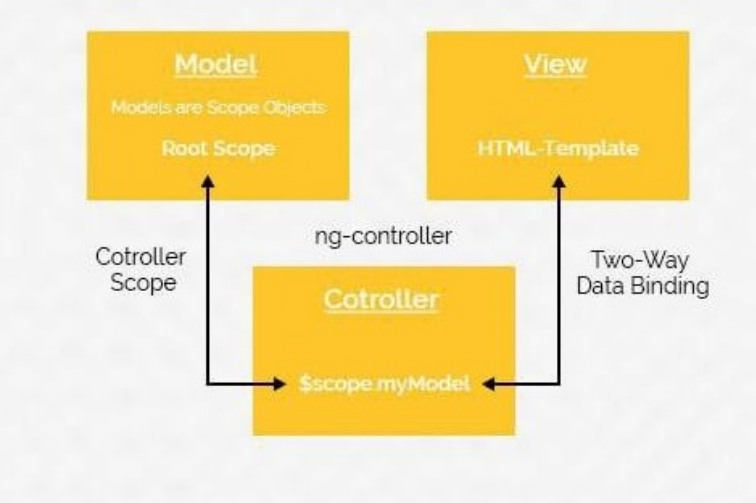


Figure 4.2: MVC Close Up

4.1.2 What uBooks Needs

- uBooks needs to hide the complexity of the system to our end users
- uBooks needs to be a system developed with high cohesion and loose coupling to support the software qualities correctness, robustness and reliability
- uBooks needs to immediately reflect the changes made by a user without having to refresh the page (ex. One user adds a new book and the other user must be able to see it immediately in Buy Books)

4.1.3 Why We Chose MVC

Due to the needs above, there are many reasons on why we used the Model, View and Controller (MVC) architecture. The MVC provides an architectural design that is distinctive in behaviour, appearance and state because:

- Of the scalability possibilities in large, medium and small projects
- There is less loading time: AngularJS reduces loading time compared to jQuery
- MVC has the ability to analyze the Document Object Model (DOM) and it binds based on the element attributes and JavaScript and jQuery use inner HTML for it (AmarInfoTech)
- File organization is made easy. "Routing is very easy and effective in AngularJS and file organization is done using routing so file organization is very easy in AngularJS" (AmarInfotech).
- The MVC promotes flexibility and reusability
- The complexity of the system is hidden to the end user since they only interact with the "View" Component of the MVC

- The MVC provides consistency in the system
- the MVC promoted decoupled code

4.2 Design Patterns

Observer Pattern

The first design pattern we would like to discuss is the behavioural Observer design pattern. "Observer design pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically " (tutorialspoint). As displayed in Figure 4.3, Subject represents the core abstraction while the Observer represents the variable abstraction. "The Subject prompts the Observer objects to do their thing. Each Observer can call back to the Subject as needed " (tutorialspoint).

Observer pattern is an optimal solution for a situation in which the subject must constantly keep the objects updated with the new information. This pattern is a huge improvement from doing singular updates from one dependency to the other. Moreover, the pattern facilitates loosely coupling design.

In our situation specifically, this design pattern helps us maintain one-to-many . Our Subject (Administrator) observes the Observer and the Observer Objects (Seller and Buyer). In addition, the Administrator implements the Subject. The main reason we used this pattern is because the MVC frameworks also use Observer pattern where Model is the Subject and Views are observers that can register to get notified of any change to the model.
A good example of how the Observer Pattern works in our application is the following: If the administrator(subject) adds a new book for sale, the customer(observer) will be able to see the book in the catalogue without having to refresh the page. This technique is also called reactive programming.

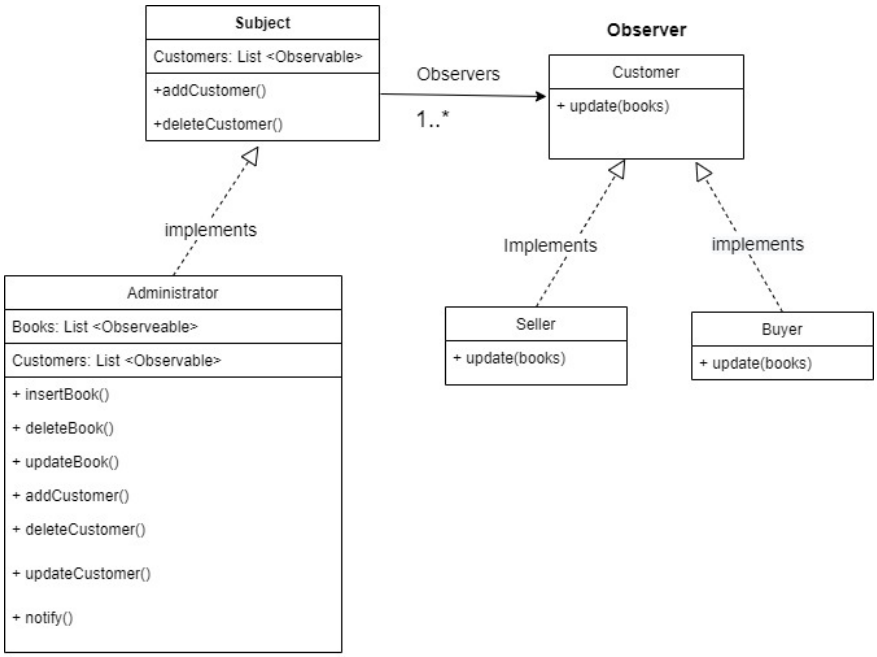


Figure 4.3: Observer Design Pattern

4.2.1 Observer Design Pattern - Subject

4.2.1.1 Subject.ts

```
/*
Mykyta Chernenky
200367631
CS 476 - Software Development
uBooks

[OBSERVER DESIGN PATTERN] Subject - TypeScript
 */

//Observer Design Pattern - SUBJECT

import { AdminsService } from '../services/admins.service';
import { CountriesService } from '../services/countries.service';
import { CustomersService } from '../services/customers.service';


export class Subject {
    customers: any;
    data: any;
    admins: any;
    authService: any;
    router: any;
    countries: any;
    constructor(
        private countriesService: CountriesService,
        private adminsService: AdminsService,
        private customersService: CustomersService
      ) {}

    addCustomer() {
         /* Checking if there are some users with the same email address. */

           let customerid = this.customers[this.customers.length - 1].UserID + 1;
           if (this.admins.find(admin => admin.UserID == customerid)) {
             customerid = this.admins[this.admins.length - 1].UserID + 1;
           }
           this.data.UserID = customerid;
           /* Using authService to add signed up user. */
           this.authService.setSignedUpUser(this.data);
           /* INSERT query to the customers' table of the database */
           this.customersService.insertCustomer(this.data).subscribe(() => {
             this.fetchData();
           });
         }
    deleteCustomer(id: number) {
        /* DELETE query to the customers' table of the database */
    this.customersService.deleteCustomer(id).subscribe(() => { this.fetchData(); });
    /* Delete table entry by customer's ID. */
    this.customers = this.customers.filter(customer => customer.UserID != id);
```

```
    }

    fetchData() {
        /* Load data to variable using countries service. */
        this.countriesService.getAllCountries().subscribe(
          country => {
            this.countries = country;
          },
          err => {
            console.log(err);
          }
        );

        /* Load data to variable using admins service. */
        this.adminsService.getAllAdmins().subscribe(
          user => {
            this.admins = user;
          },
          err => {
            console.log(err);
          }
        );

        /* Load data to variable using customers service. */
        this.customersService.getAllCustomers().subscribe(
          user => {
            this.customers = user;
          },
          err => {
            console.log(err);
          }
        );
      }

}
```

Listing 4.1: Subject implementation

Prototype Pattern

The second design pattern we would like to discuss is the creational Prototype pattern. "The Prototype pattern refers to creating a duplicate object while keeping performance in mind. This type of design pattern provides one of the best ways to create an object. It also involves implementing a prototype interface which tells to create a clone of the current object and is used when creation of object directly is costly " (tutorialspoint). As displayed in Figure 4.4, the Abstract Class User extends to the User Class Customer. The reasons we used the prototype pattern is because "Prototype allows us to hide the complexity of making new instances from the client. The concept is to copy an existing object rather than creating a new instance from scratch, something that may include costly operations. The existing object acts as a prototype and contains the state of the object. The newly copied object may change same properties only if required. This approach saves costly resources and time, especially when the object creation is a heavy process " (GeeksForGeeks).

Using this pattern we can easily create a new Customer object using the attributes of the user object. This technique makes our code reusable and more qualitative.

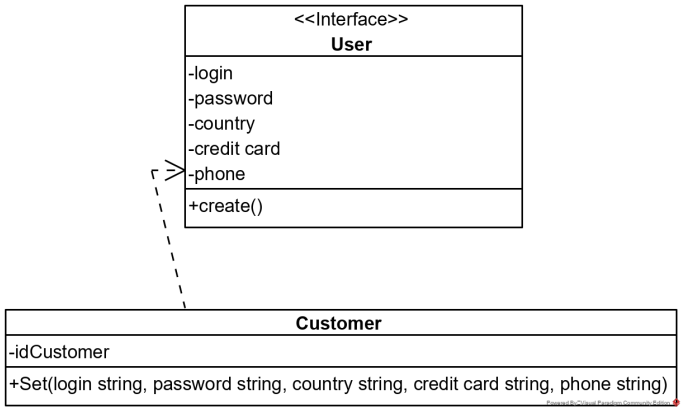


Figure 4.4: Prototype Design Pattern

4.2.2 Prototype Design Pattern - Part 1

4.2.2.1 user.ts

```
/*
Mykyta Chernenky
200367631
CS 476 - Software Development
uBooks

User class
*/

//PROTOTYPE Design Pattern Part 1

export class User { //User class that will be cloned
  constructor(
    private login: string,
    private password: string,
    private country: string,
    private creditCard: string,
```

```
    private phoneNumber: string
  ) {}

  Set( //function used to set values for the user
    login1: string,
    password1: string,
    country1: string,
    creditCard1: string,
    phoneNumber1: string
  ) {
    this.login = login1;
    this.password = password1;
    this.country = country1;
    this.creditCard = creditCard1;
    this.phoneNumber = phoneNumber1;
  }
}
```

Listing 4.2: Prototype Design Pattern - Part 1

4.2.3 Prototype Design Pattern - Part 2

4.2.3.1 customer.ts

```
/*
Mykyta Chernenky
200367631
CS 476 - Software Development
uBooks

Customer class
*/

//PROTOTYPE Design Patter - Part 2
import { User } from "./user"; //import user class

class Customer { //initiate customer class
  constructor(customerID: number, user: User) {} //2 parameters: customerID, and User
      object
  create(             //create Customer function
    customerID: number,
    login: string,
    password: string,
    country: string,
    creditCard: string,
    phoneNumber: string
  ) {
    var customer = new Customer(customerID, new User(login, password, country,
        creditCard, phoneNumber)); //created new Customer by cloning User
  }
}
```

Listing 4.3: Prototype Design Pattern - Part 2

Visitor Pattern

The final design pattern we would like to discuss is the behavioural Visitor design pattern. Referencing Figure 4.5, Visitor uses the Element AdminComponent which leads to all the Concrete Elements such as Admin-admins-component and admin-book-genres-component. The accept method within each of them calls the correct implementation based on the visitor passed, so it is the admin who tells if the countries component or the orders component is needed. Looking back at the Visitor, it also uses the Concrete Visitor called AdminComponentVistor and within it is all the concrete elements. The main reasons we used this pattern is for loose coupling and so future changes should not impact the system. To summarize, the Visitor pattern lets us define a new operation (such as the book genres component) in uBooks without changing the classes of the elements on which it operates (the admin component).

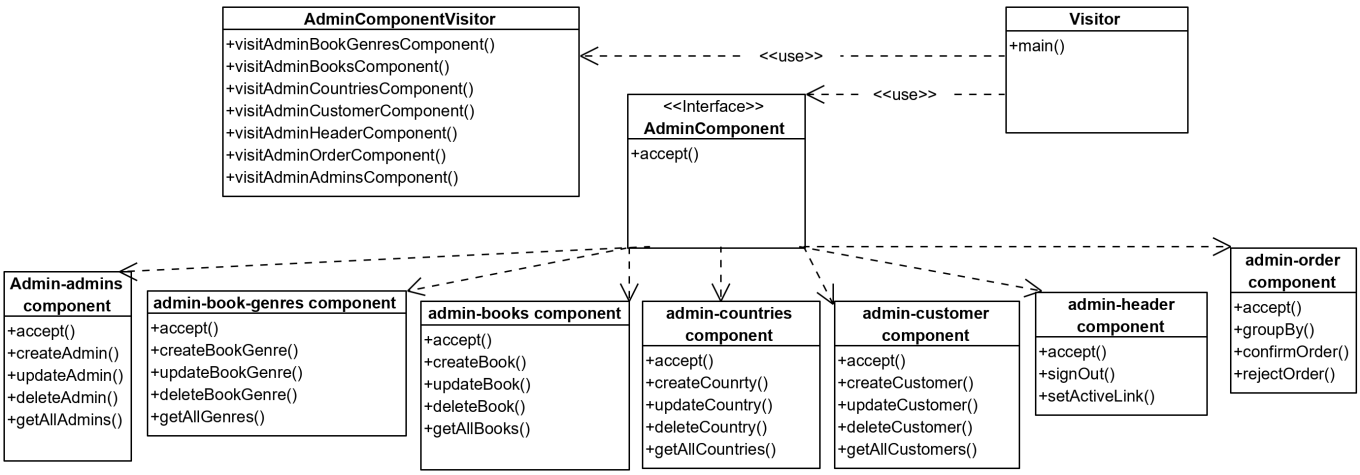


Figure 4.5: Visitor Design Pattern

4.2.4 Visitor Design Pattern - Part 2

4.2.4.1 admin-component-visitor.ts

```
/* Mykyta Chernenky
   200367631
   CS 476 - Software Development
   uBooks
Visitor Design Pattern - Part 2
*/
import { Visitor } from '@angular/compiler/src/i18n/i18n_ast';
import { AdminBookGenresComponent } from '../admin/admin-book-genres/admin-book-
    genres.component';
import { AdminBooksComponent } from '../admin/admin-books/admin-books.component';
import { AdminCustomersComponent } from '../admin/admin-customers/admin-customers.
    component';
import { AdminCountriesComponent } from '../admin/admin-countries/admin-countries.
    component';
import { AdminHeaderComponent } from '../admin/admin-header/admin-header.component';
import { AdminOrdersComponent } from '../admin/admin-orders/admin-orders.component';
import { AdminAdminsComponent } from '../admin/admin-admins/admin-admins.component';

export class AdminComponentVisitor {
```

```
    item: any;
   router: any;
    VisitPage(item: any) {
       if (this.item instanceof AdminBookGenresComponent) {
           this.router.navigate(["/admin/book-genres"]);
       }
       else if (this.item instanceof AdminBooksComponent) {
           this.router.navigate(["/admin/books"]);
       }
       else if (this.item instanceof AdminCountriesComponent) {
           this.router.navigate(["/admin/countries"]);
       }
       else if (this.item instanceof AdminCustomersComponent) {
           this.router.navigate(["/admin/customers"]);
       }
       else if (this.item instanceof AdminHeaderComponent) {
           this.router.navigate(["/admin/header"]);
       }
       else if (this.item instanceof AdminOrdersComponent) {
           this.router.navigate(["/admin/orders"]);
       }
       else if (this.item instanceof AdminAdminsComponent) {
           this.router.navigate(["/admin/admins"]);
       };
   }
}
```

Listing 4.4: Admin Component Visitor

4.2.5 Visitor Design Pattern - Part 1

4.2.5.1 visitor.ts

```
/*
Mykyta Chernenky
200367631
CS 476 - Software Development
uBooks

Visitor Design Pattern - Part 1
*/

import { AdminComponentVisitor } from './admin-component-visitor';

export interface Visitor {
    visit(item: AdminComponentVisitor);
}
```

Listing 4.5: visitor.ts

4.3 Class Diagrams

Due to high complexity of our project, the class diagram is very large. Also, since the application is done in Angular, the diagram consists of many services and components. I have successfully incorporated the design patterns with all other components of the project. Figure 4.6 shows the cropped out part with the design patters.

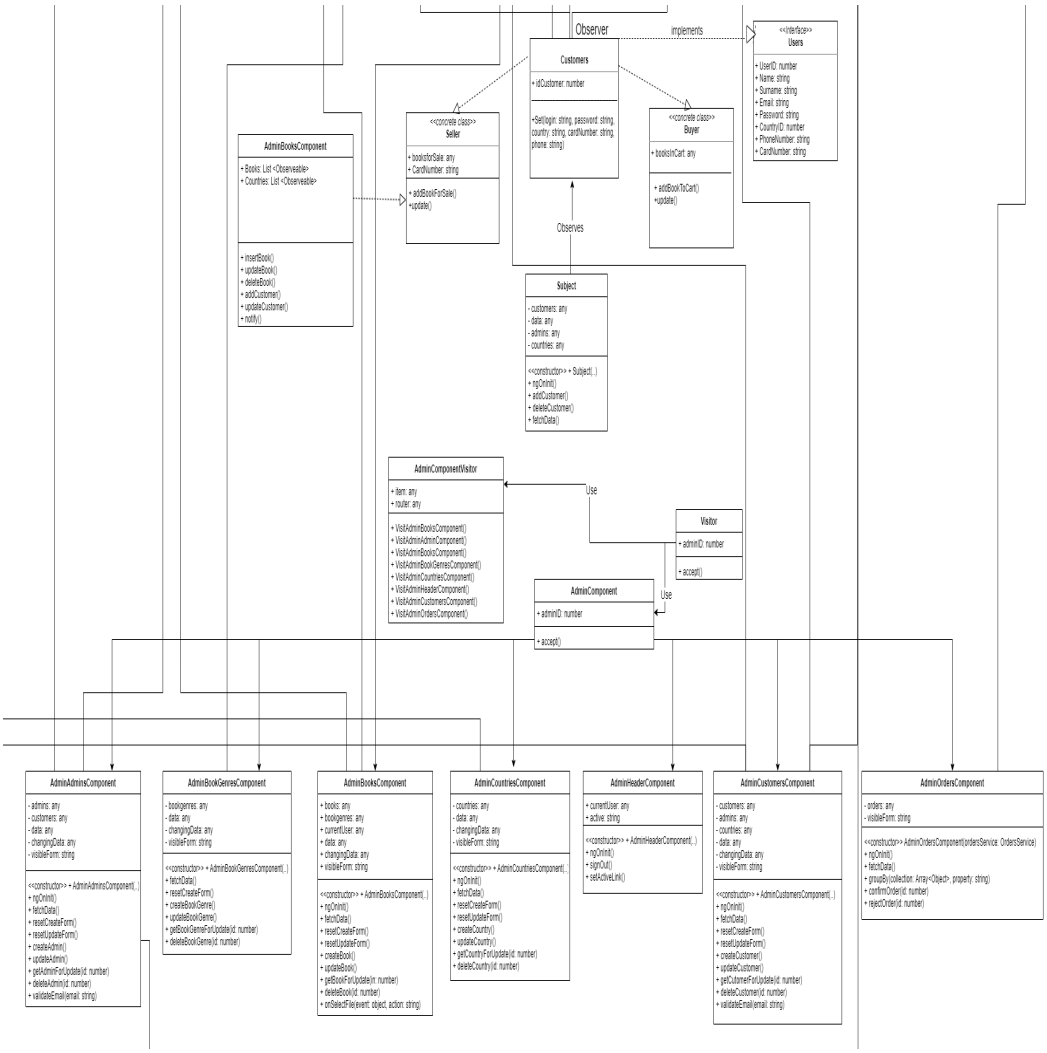


Figure 4.6: Design Patterns Within The Class Diagram

In the overall class diagram, I have highlighted the design patterns in different colours. This way I can easily see where they are located and how they interact with the rest of the project components.

1. Yellow Circle: Visitor Design Pattern 4.5
2. Blue Circle: Observable Design Pattern 4.3
3. Purple Circle: Prototype Design Pattern 4.4

4.3.1 Overall Class Diagram

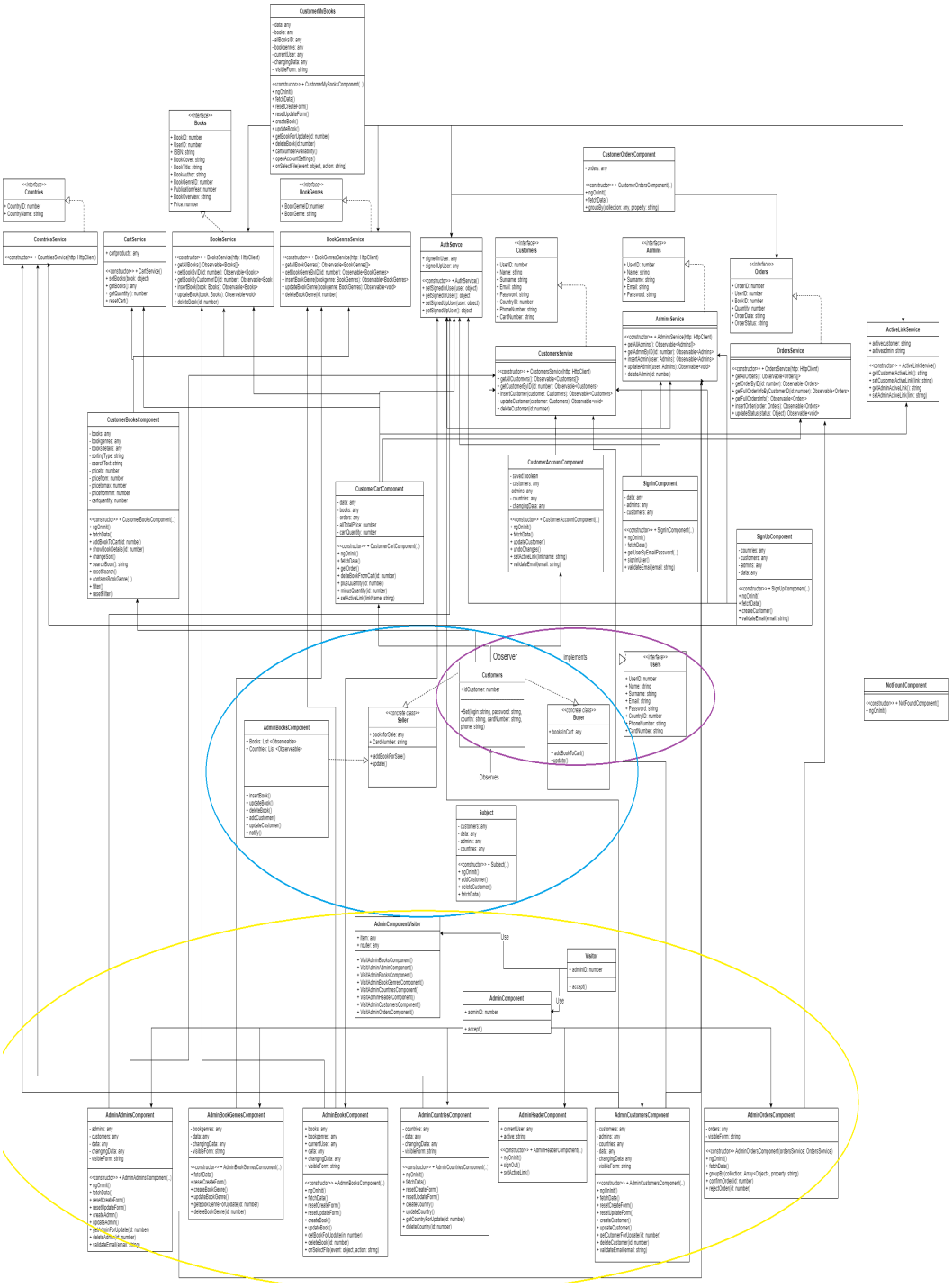


Figure 4.7: Overall Class diagram

5. Programs

5.1 Component Diagram

Component Server Diagrams

The first component diagram we will discuss is the Server Component Diagram, referencing Figure 5.1. This diagram depicts how our server part is set up. Server.js contains the logic to connect to real-time database ClearDB. As you can see on the diagram, it uses logic of express.js as an external package, which is also apart of our developer stack. The Server component also contains the routes. To add, there are different routes responsible for different components of the application. Each one of these routes connects to a table in the real time database and can modify it.

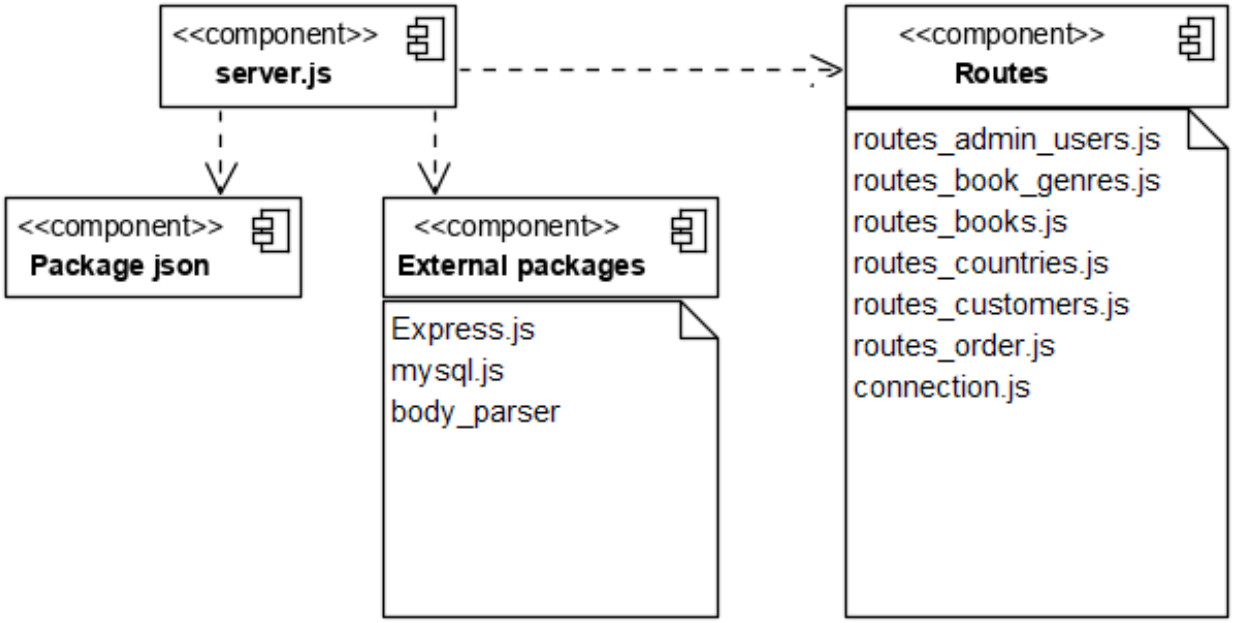


Figure 5.1: Component Server Diagram

Component Client Diagram

The other component diagram we will discuss is the Component Client Diagram, referencing Figure 5.2. This is the client side of our application. As we have mentioned before, there can be 2 types of users: Admin or Customer. There are admin classes that implement admin components. Admin components include the operations that can be done on the admin side. Such as, modify book catalogue, modify and manage orders, and so on. The customer component has the same functionality. With any angular application, we start off with index.html component that is then referred to router which decides where to go based on URL.

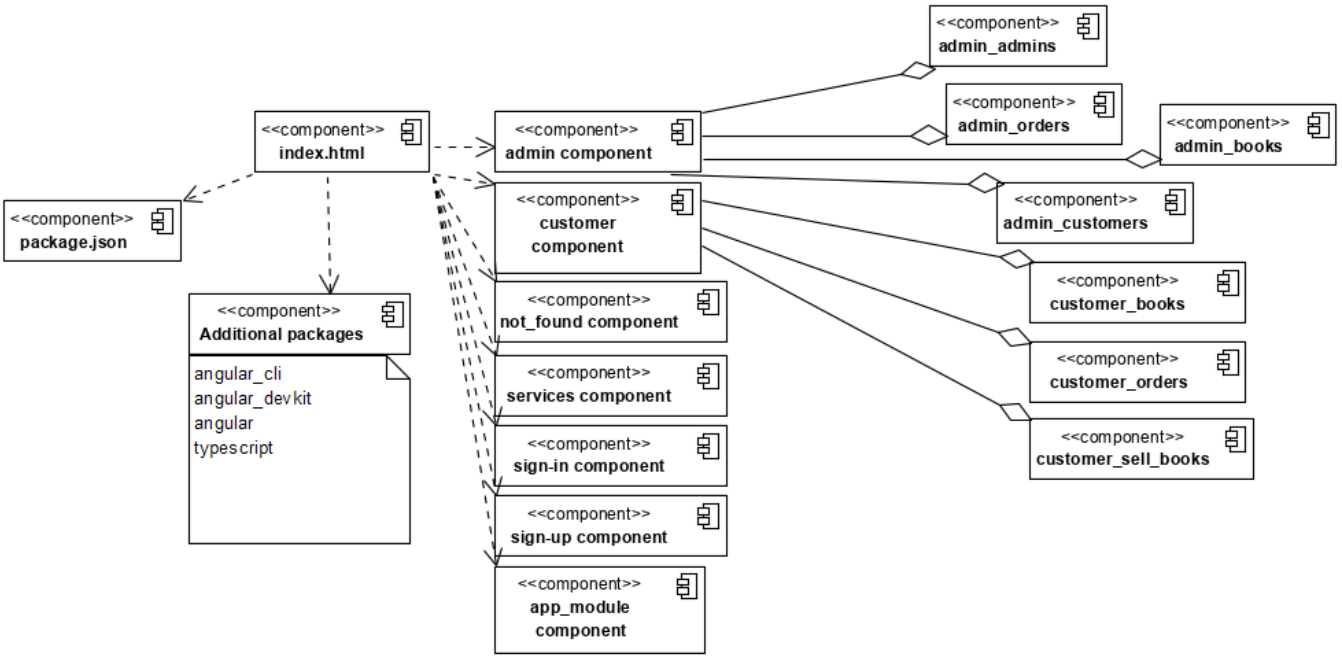


Figure 5.2: Component Client Diagram

5.2 Deployment Diagram

Here is our Deployment Diagram which relates to the MEAN stack architecture, referencing Figure 5.3. On the left side, you can see the client component stack. This layer controls what the user can see on the client side. The client layer connects to the business layer via HTTP/IP request. The business layer connects to the actual database layer which is of a real-time database ClearDB via HTTP/IP request. The business layer also connects to the Heroku Environment which is our host via HTTP/IP. The host is implemented by node.js and Heroku CLI. Note that any modern and up to date web browser can support our software.

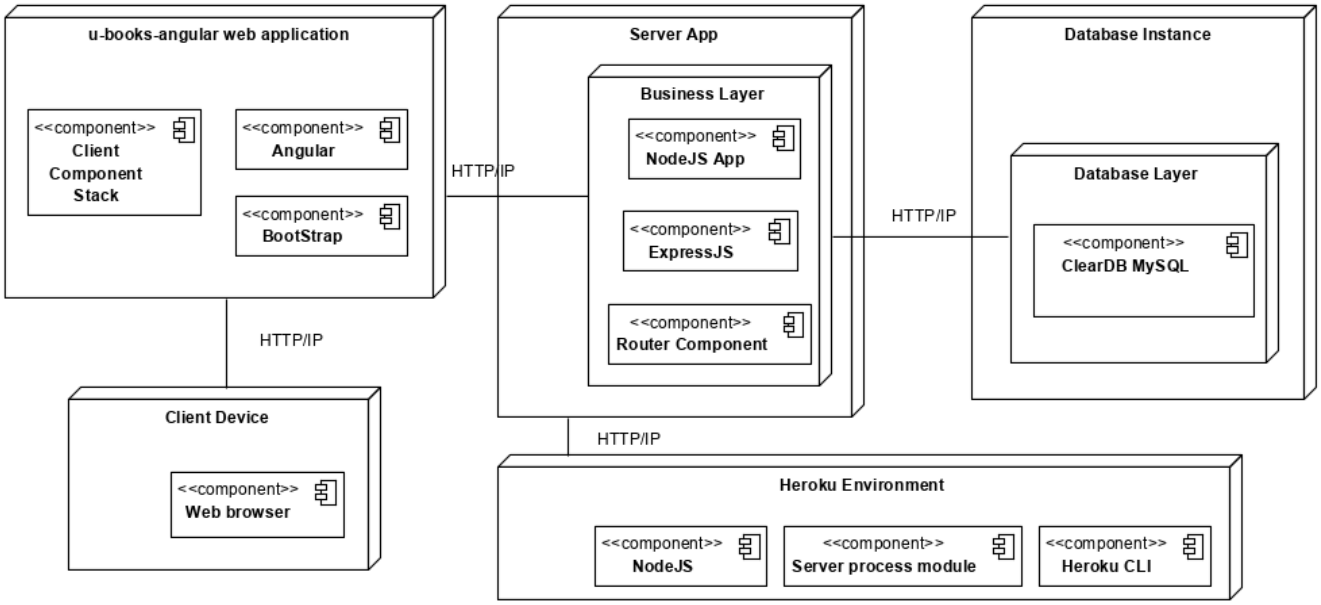


Figure 5.3: Deployment Diagram

5.3 Data Model

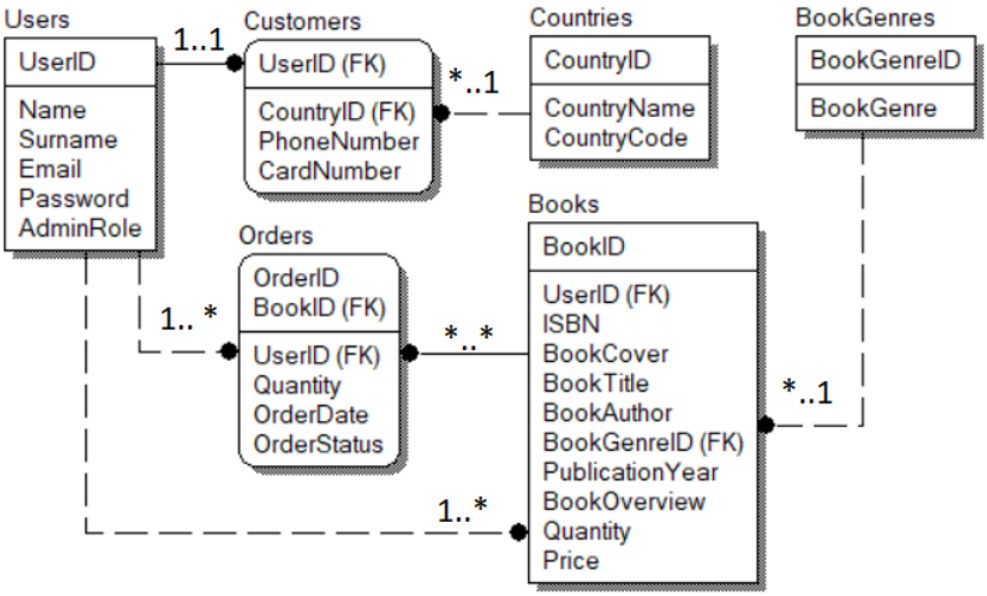


Figure 5.4: Data Model

5.4 Tables of Contents of the System Data

List of Tables:

1. Table Contents of Books: Reference Figure 5.5
2. Table Contents of Countries: Reference Figure 5.6
3. Table Contents of Customers: Reference Figure 5.7
4. Table Contents of Genres: Reference Figure 5.8
5. Table Contents of Orders: Reference Figure 5.9
6. Table Contents of Users: Reference Figure 5.10

5.5 Link to the web application

Here is the link to our web application https://u-books-angular.herokuapp.com/

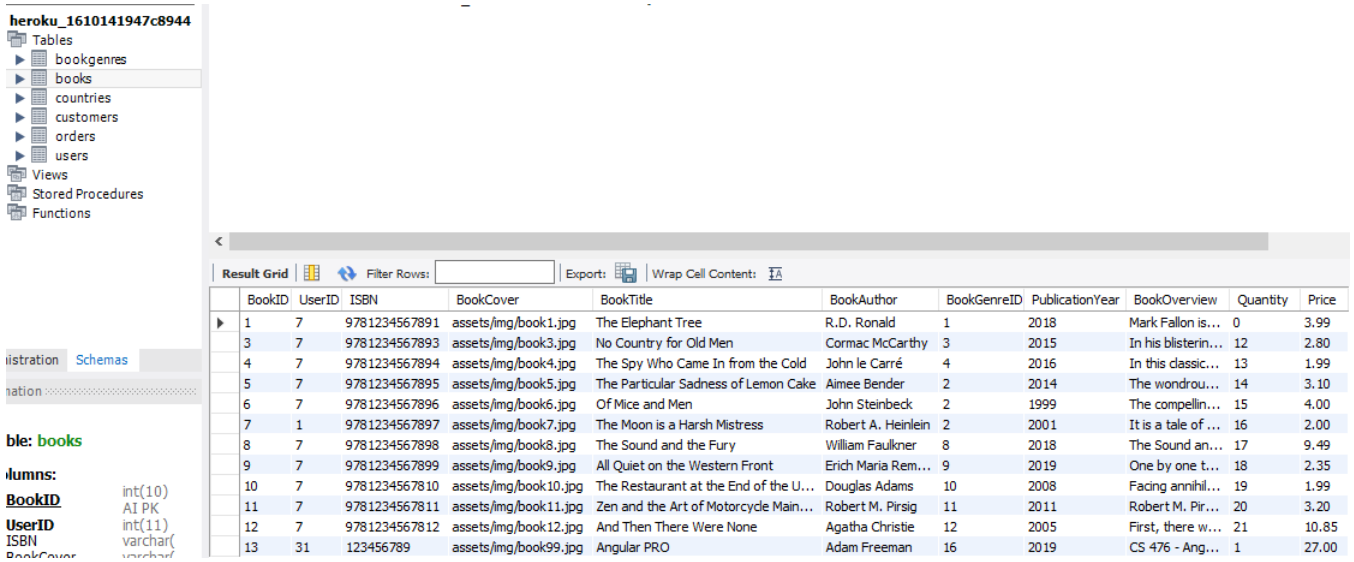


Figure 5.5: Table Contents of Books

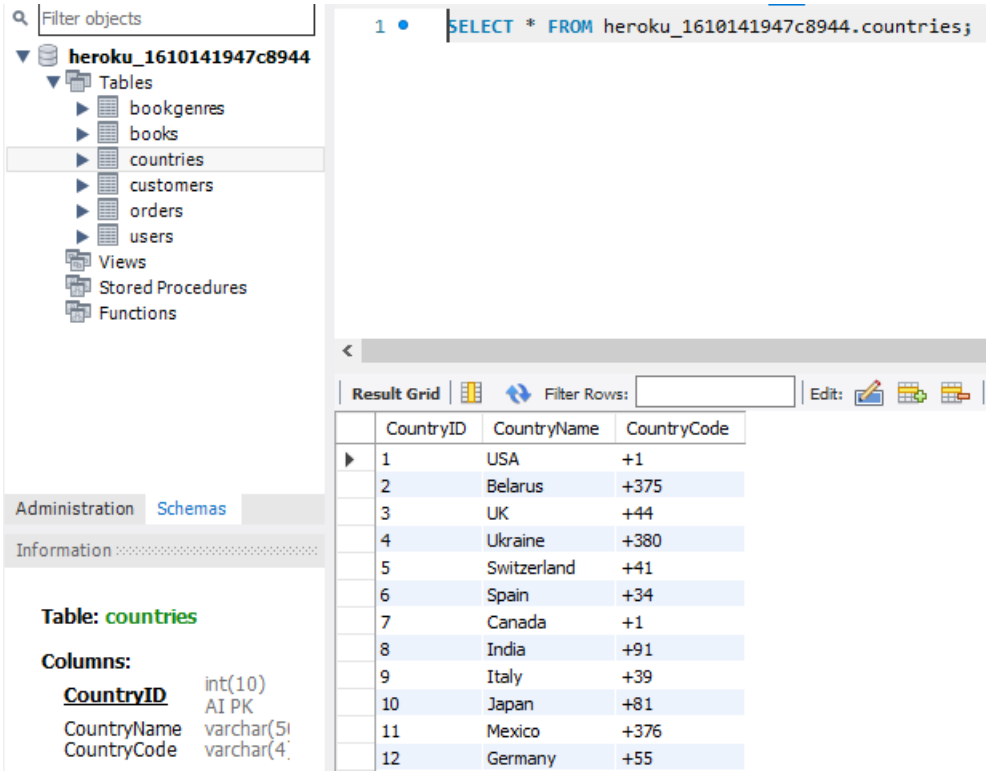


Figure 5.6: Table Contents of Countries

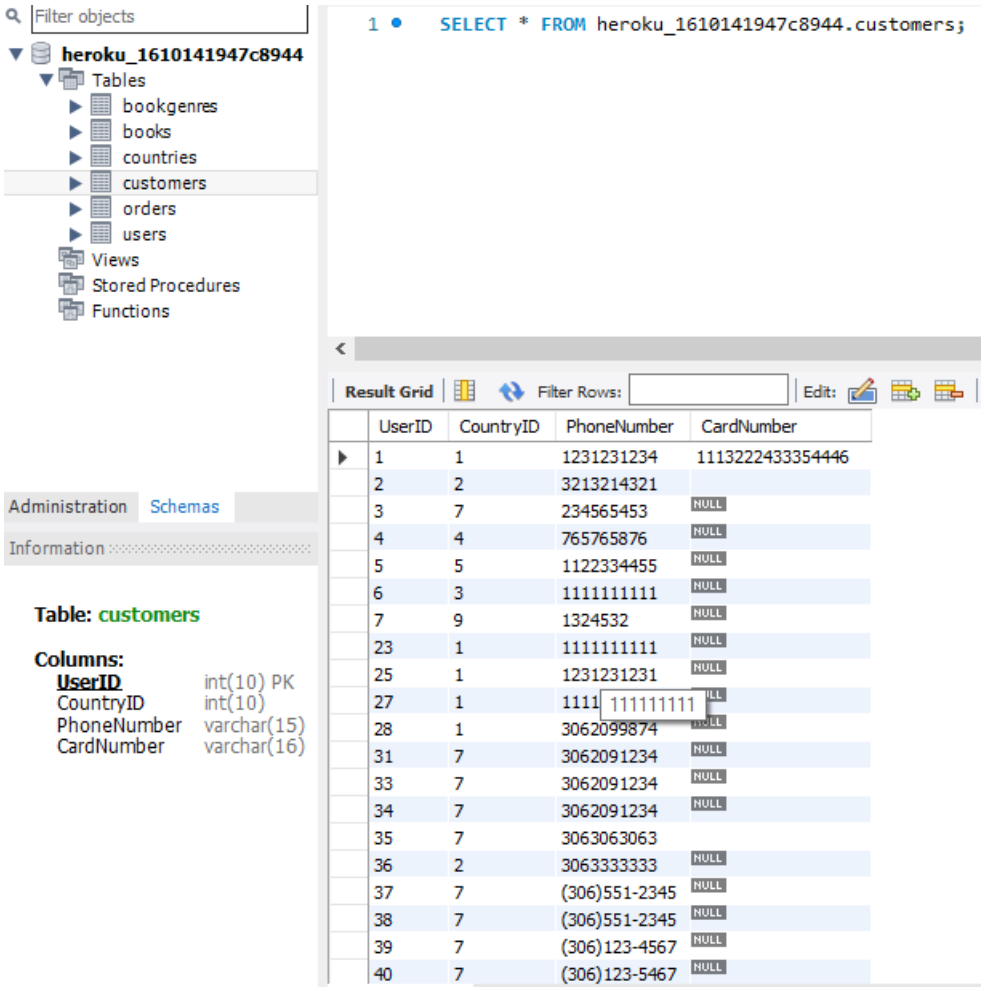


Figure 5.7: Table Contents of Customers

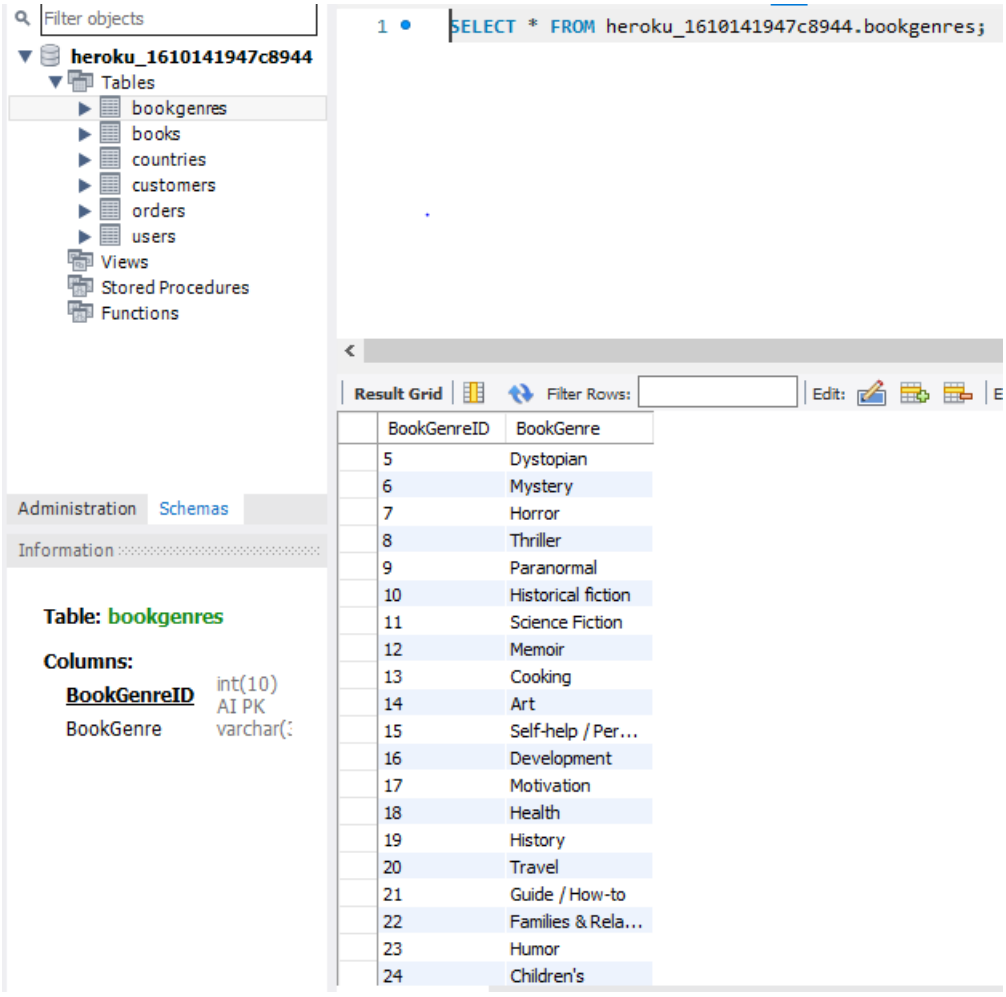
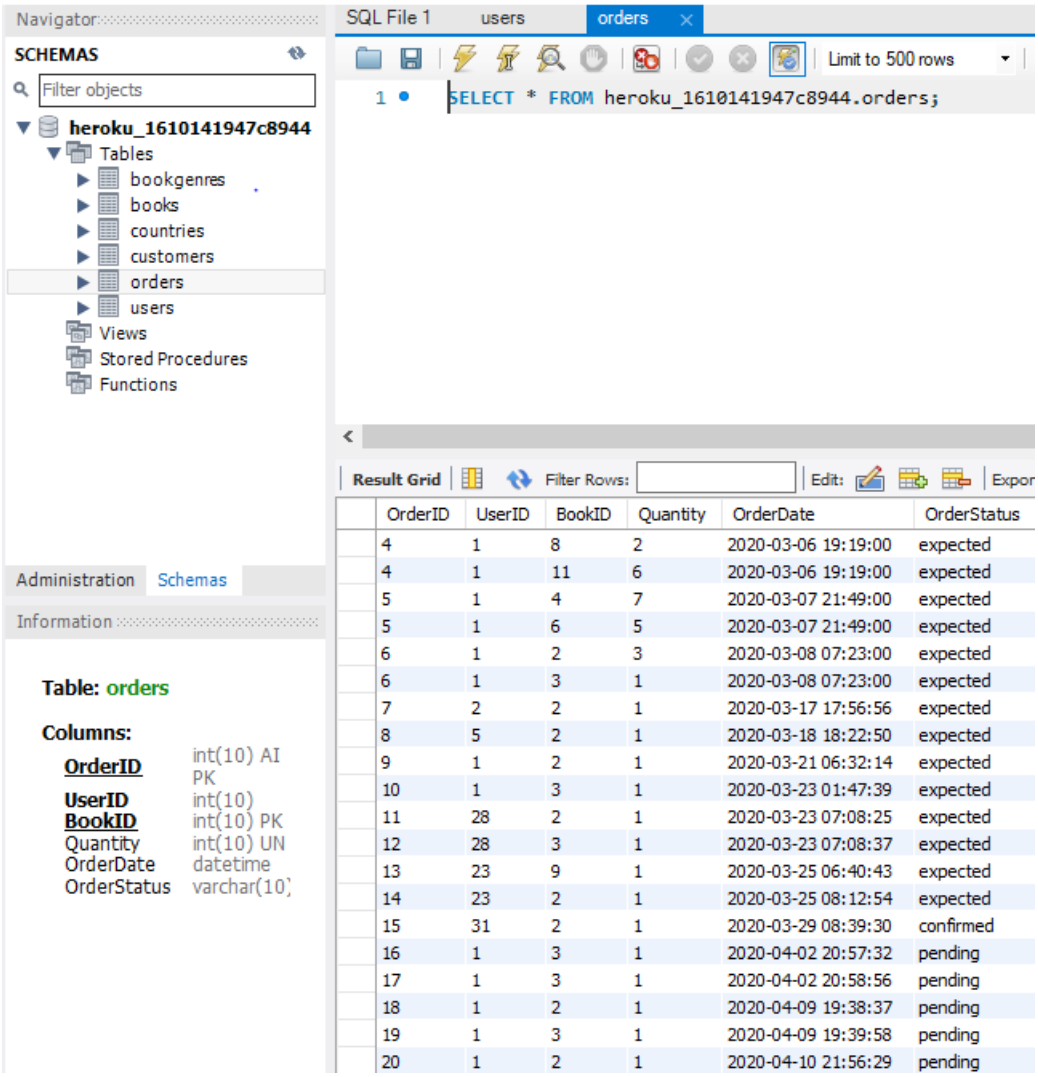


Figure 5.8: Table Contents of Genres



OrderID	UserID	BookID	Quantity	OrderDate	OrderStatus
4	1	8	2	2020-03-06 19:19:00	expected
4	1	11	6	2020-03-06 19:19:00	expected
5	1	4	7	2020-03-07 21:49:00	expected
5	1	6	5	2020-03-07 21:49:00	expected
6	1	2	3	2020-03-08 07:23:00	expected
6	1	3	1	2020-03-08 07:23:00	expected
7	2	2	1	2020-03-17 17:56:56	expected
8	5	2	1	2020-03-18 18:22:50	expected
9	1	2	1	2020-03-21 06:32:14	expected
10	1	3	1	2020-03-23 01:47:39	expected
11	28	2	1	2020-03-23 07:08:25	expected
12	28	3	1	2020-03-23 07:08:37	expected
13	23	9	1	2020-03-25 06:40:43	expected
14	23	2	1	2020-03-25 08:12:54	expected
15	31	2	1	2020-03-29 08:39:30	confirmed
16	1	3	1	2020-04-02 20:57:32	pending
17	1	3	1	2020-04-02 20:58:56	pending
18	1	2	1	2020-04-09 19:38:37	pending
19	1	3	1	2020-04-09 19:39:58	pending
20	1	2	1	2020-04-10 21:56:29	pending

Figure 5.9: Table Contents of Orders

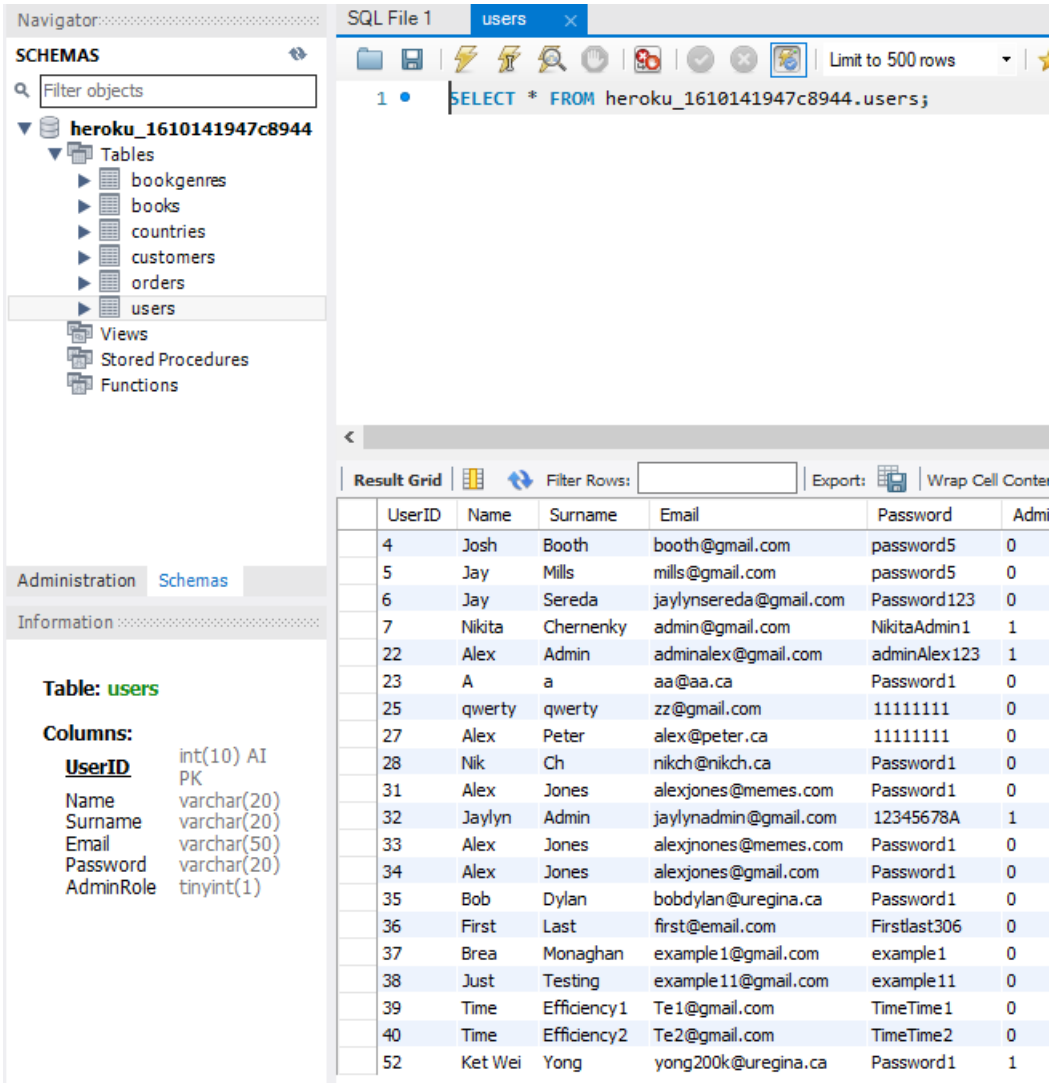


Figure 5.10: Table Contents of Users

6. Technical Documentation

6.1 List of Programming Languages

6.1.0.1 Programming Languages

- TypeScript
- Shell Script
- Git Bash

6.1.0.2 Structured Query Languages

- MySQL

6.1.0.3 Mark-up languages

- HTML
- CSS

6.2 List of reused algorithms and programs

- We have watched various YouTube tutorials for web application development, with the most useful being Angular - Node - MongoDB & Express (MEAN) Tutorial for Beginners - Getting Started [https://www.geeksforgeeks.org/prototype-design-pattern/]

  Using this resource, we were to teach ourselves how to build a three-tier web application that connects to the real-time database.

  One of the big challenges was getting our application to run on the actual hosting server. We have read numerous tutorial to figure out which hosting is the most suitable for our application. We have concluded that Heroku is the most suitable hosting for our application since it's free and supports the real-time database. To successfully deploy the web application on Heroku, we have followed the following tutorial: https://scotch.io/tutorials/how-to-deploy-a-node-js-app-to-heroku.

6.3 List of software tools and environments:

- IDE: Visual Studio Code
- Hosting: Heroku
- SQL Development: MySQL Workbench
- Repository Hosting GitHub
- Git Command Line: Git Bash/VSCode Terminal
- UML Diagrams: Draw.io/Visual Paradigm
- Development Package Managers: Cocoa Pods/NPM
- Testing: Google Chrome
- Cloud Database: ClearDB
- Server Runtime Environment: NodeJS

We have developed our web application using Angular framework. Angular is an open source TypeScript based web app development framework. Angular eases the process of the application development. It provides you with multiple tools that provide quick view and quick routing. You can control the view based on the back-end data easily. It is a handy resource for utilizing HTML templates and assembling data services for applications.

Angular is a great framework for web development. The benefits of using Angular include but are not limited to: cross platform development, speed, performance and/or productivity. Because we used Angular, we were able to implement the MVC Architecture and enhanced our design architecture. In addition, Angular includes TypeScript which fully complies to JavaScript, has good tooling, clean code and high scalability. TypeScript also helps to spot and eliminate common mistakes while coding.

In order to develop a three-tier application we needed to decide which developer stack must be used. After hours of researching, we have concluded that MEAN stack is the most suitable stack for our web application. MEAN developer stack consists of the following four components:

- M: MySQL ClearDB/ MongoDB (Real-Time Database)
- E: ExpressJS (Back-End Web Framework)
- A: Angular (Front-End Framework)
- N: NodeJS (Back-End Runtime Environment)

As shown in Figure 6.2, MEAN architecture consists of three tiers: Client Tier, Business Logic Tier and the Database Tier.

1) The client tier sends an HTTP request to the Business Logic Tier

2) The HTTP request is then handled by the NodeJS which then requests data from the database tier

3) The data request is then handled by the Database Server Clear DB which then sends the data back to ExpressJS (Business Logic Tier)

4) The ExpressJS will then send the data back to the client.

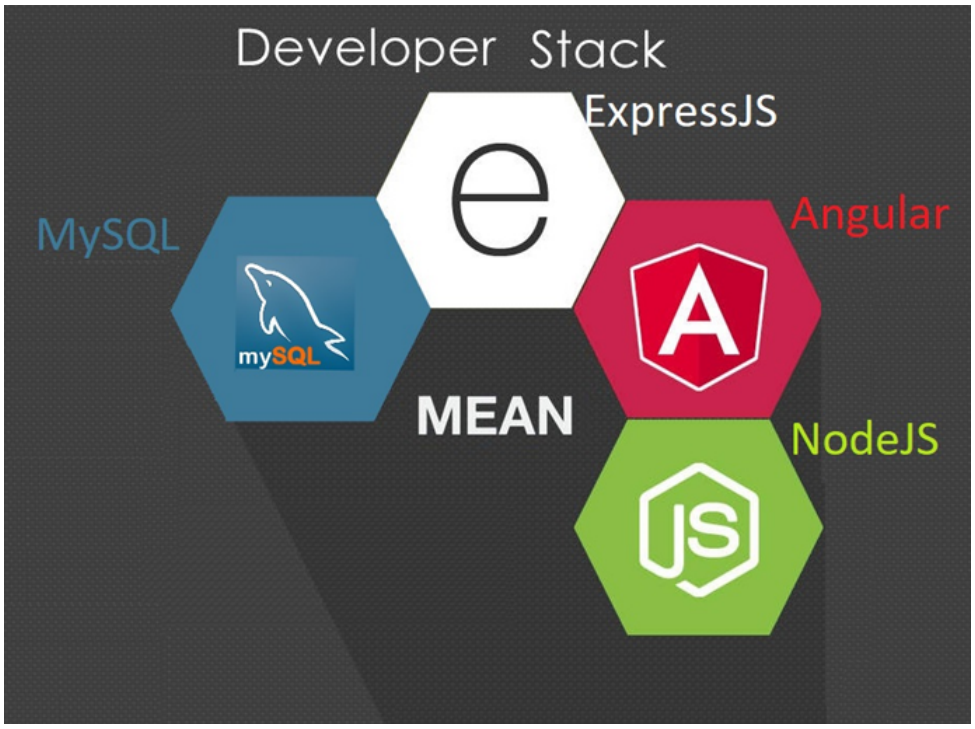


Figure 6.1: MEAN Developer Stack

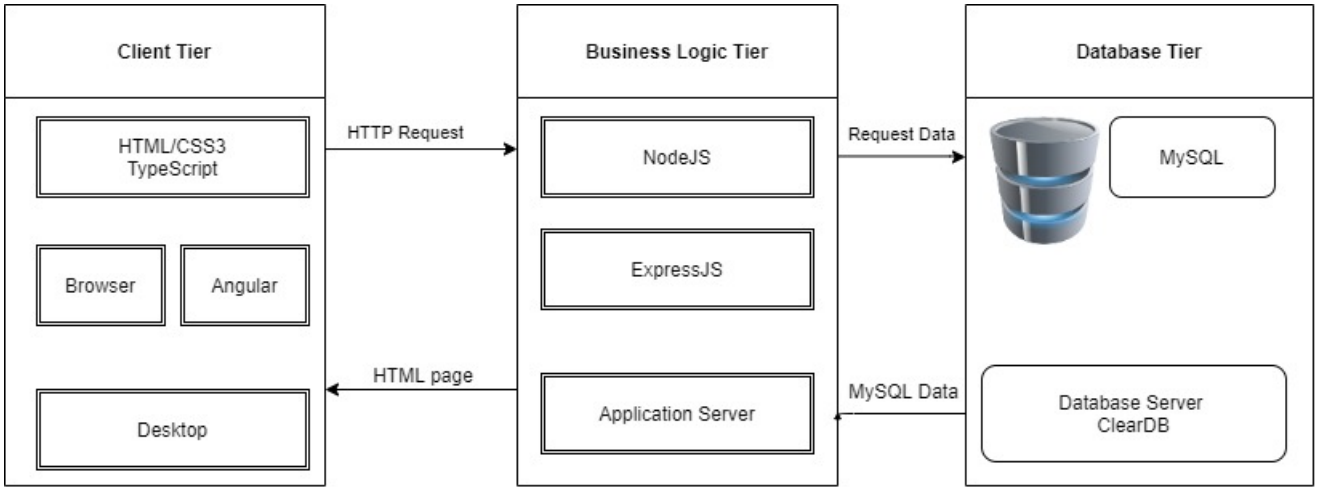


Figure 6.2: MEAN Stack Architecture

7. Acceptance Testing

Functional testing is also known as the black box test by the fact of treating software as a box whose content is unknown and which we can only see the outside. For example, with the supplied input data, the responses are produced as output. Functional testing involves two steps: identify the functions that the software must perform and then create test cases that are able to check if these functions are performed by the software. In this there are 5 functional test cases, one input and one output screenshot each.

7.1 Functional Testing

Case 1 - User: Sign Up

Table 7.1: Functional Testing Table 1

Input	User's name, surname, country, phone number, email address and password
Output	Successful Sign Up
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	1. Go to https://u-books-angular.herokuapp.com/ 2. Click Create an Account (Go to https://u-books-angular.herokuapp.com/#/sign-up) 3. Enter valid sign up credentials 4. Click the 'Sign Up' button 5. The user account has been successfully created and the user is redirected to the 'sign in' page
Screenshots	Figures 7.1 and 7.2

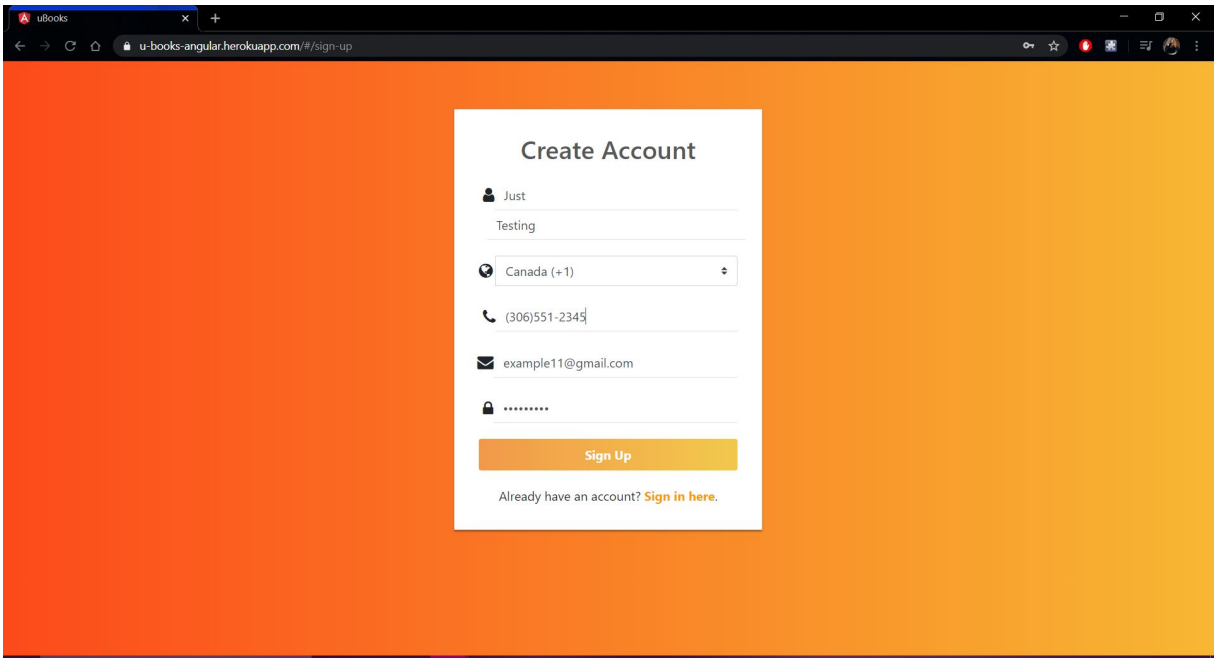


Figure 7.1: Functional Case 1 Input

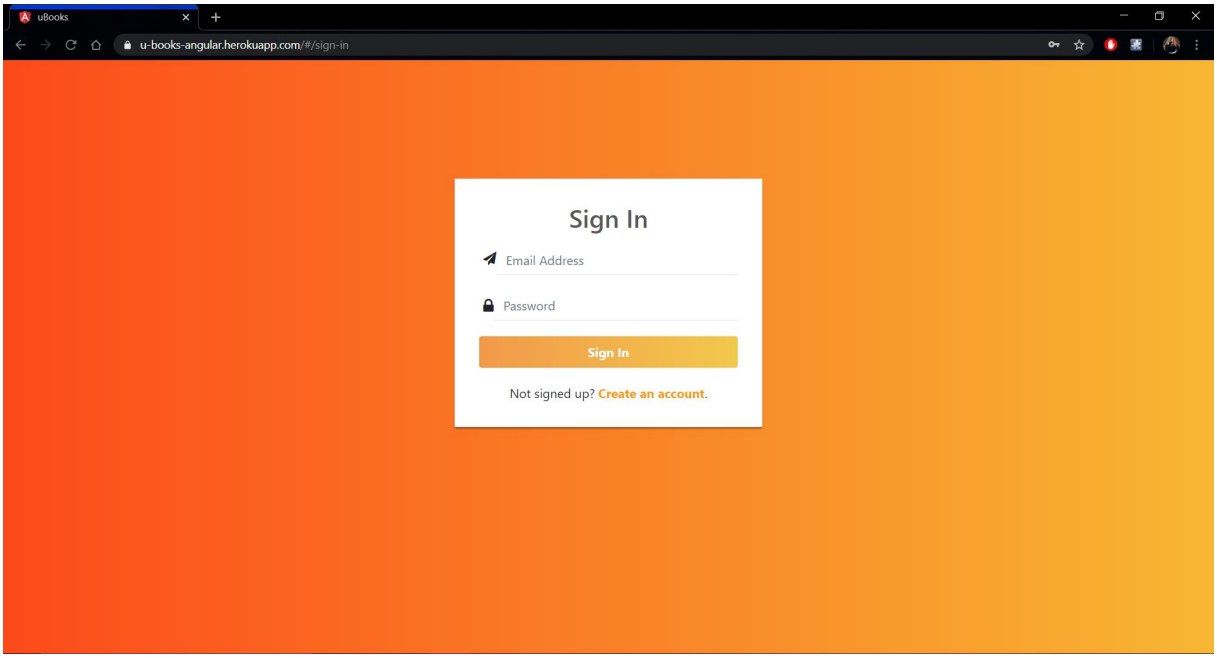


Figure 7.2: Functional Case 1 Output

Case 2 - User: Sign In

Table 7.2: Functional Testing Table 2

Input	Valid email address and password of an existing user account
Output	Successful sign in
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	1. https://u-books-angular.herokuapp.com/ 2. Enter valid sign in credentials 3. Click the 'Sign In' button 4. The user account has been successfully signed in and the user is redirected to the 'Buy books' page
Screenshots	Figures 7.3 and 7.4

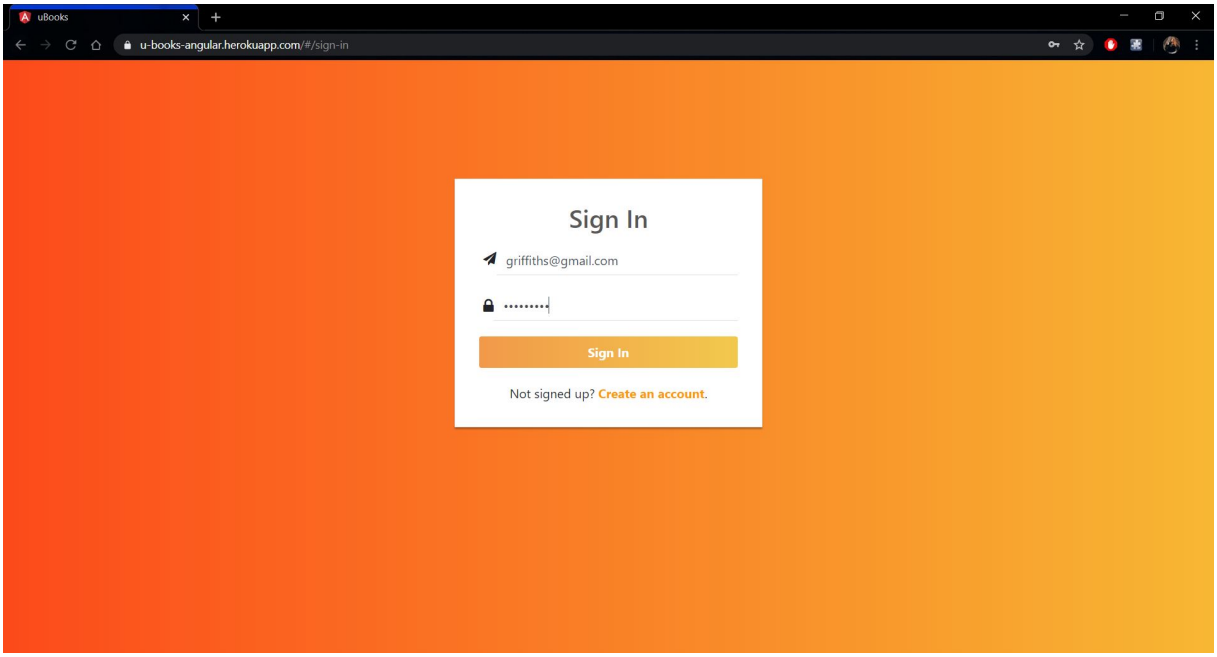


Figure 7.3: Functional Case 2 Input

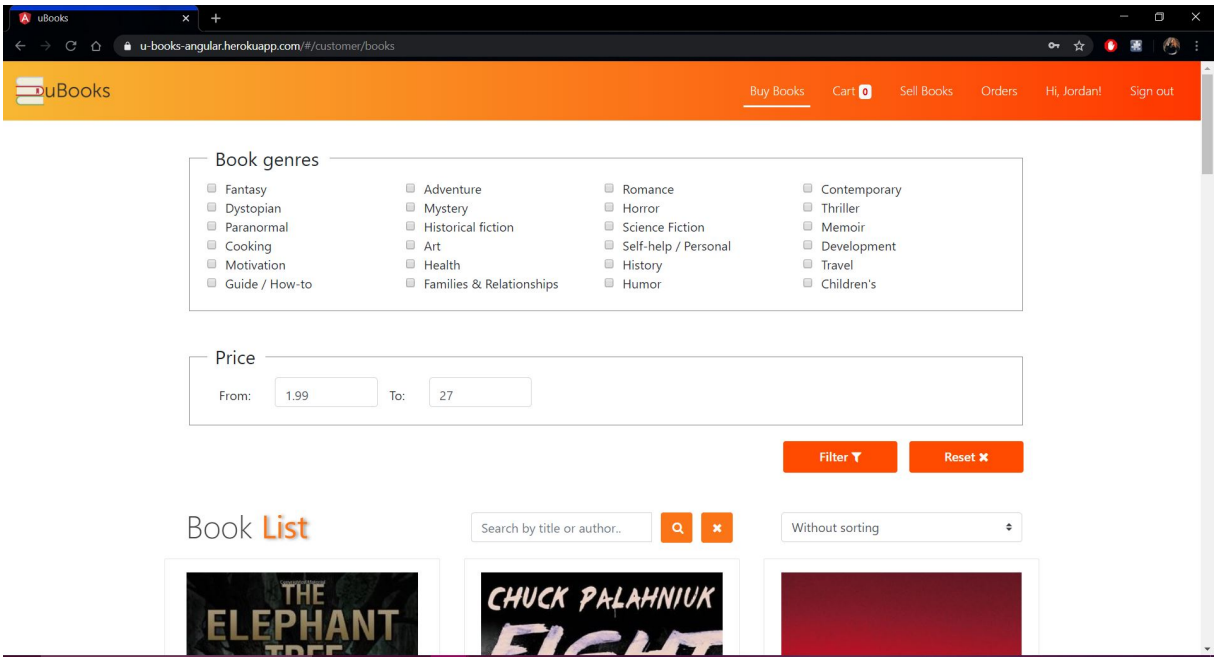


Figure 7.4: Functional Case 2 Output

Case 3 - User: Add Book to Cart

Table 7.3: Functional Testing Table 3

Input	Highlighted 'Add to Cart' button is clicked
Output	Book has successfully been added to cart as shown on 'Cart' page
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	Assuming User is Signed In: 1. Go to https://u-books-angular.herokuapp.com/#/customer/books 2. Search for a book 3. Click the highlighted add to cart button 4. Go to 'Cart' page (Go to https://u-books-angular.herokuapp.com/#/customer/cart) to see item in Cart
Screenshots	Figures 7.5 and 7.6

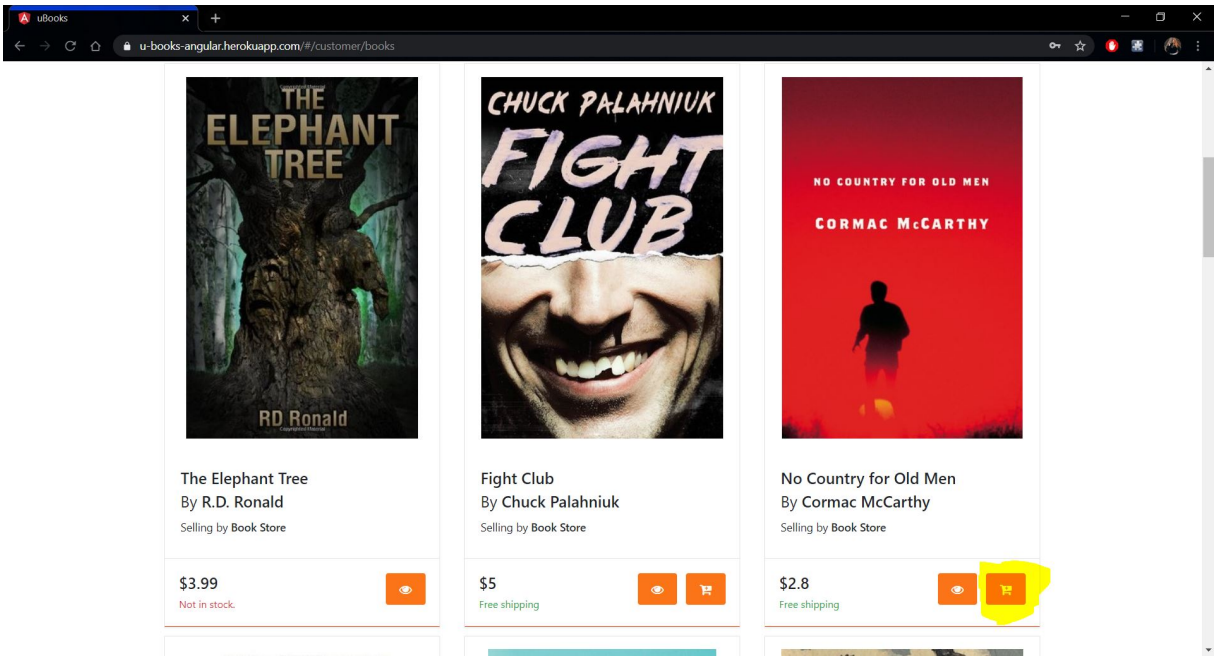


Figure 7.5: Functional Case 3 Input

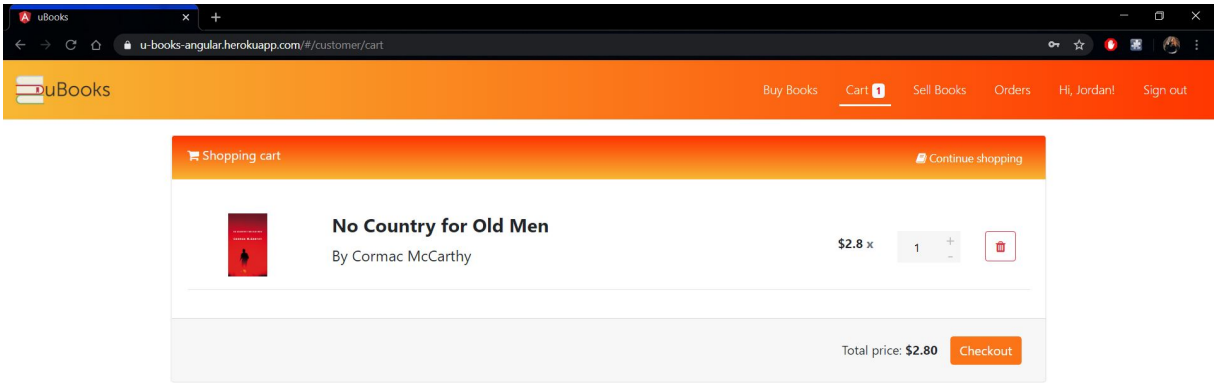


Figure 7.6: Functional Case 3 Output

Case 4 - User: Checkout

Table 7.4: Functional Testing Table 4

Input	Highlighted 'Checkout' button is clicked
Output	Checkout successful
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	Assuming User is Signed In: 1. Go to 'Cart' page (Go to https://u-books-angular.herokuapp.com/#/customer/cart) 2. Ensure order is correct 3. Click 'Checkout' button 4. Popup indicates that order is completed and pending. Also, the cart is now empty.
Screenshots	Figures 7.7 and 7.8

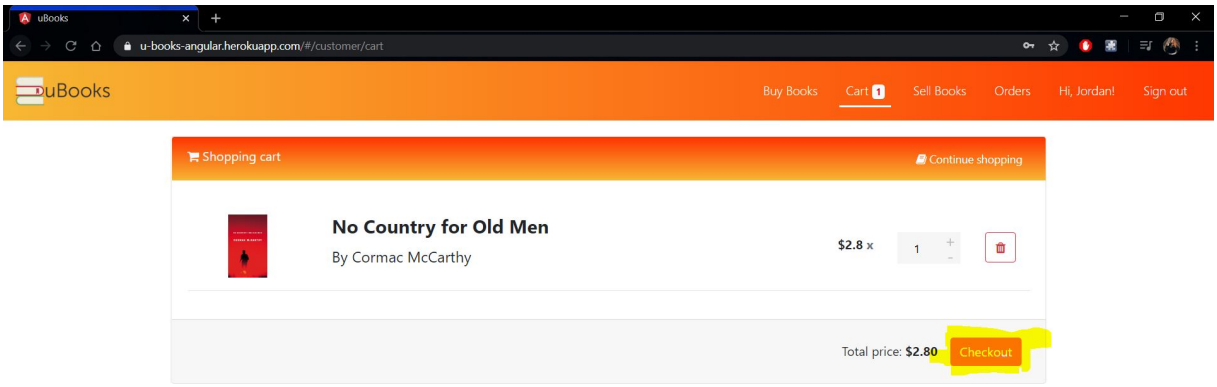


Figure 7.7: Functional Case 4 Input

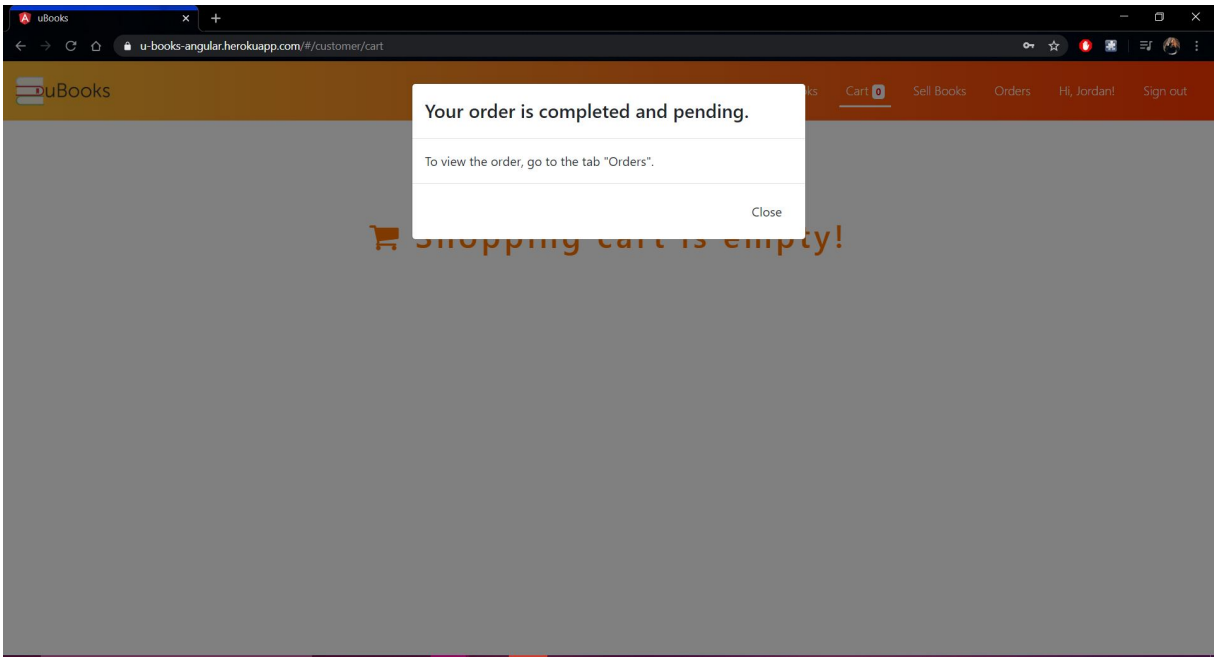


Figure 7.8: Functional Case 4 Output

Case 5 - User: Delete Book from Cart

Table 7.5: Functional Testing Table 5

Input	Highlighted 'Delete' button is clicked
Output	Cart is now empty
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	Assuming User is Signed In and a book has been added to cart: 1. Go to 'Cart' page (Go to https://u-books-angular.herokuapp.com/#/customer/cart) 2. Click 'Delete' button on unwanted book 3. 'Cart' page displays 'Shopping cart is empty!'
Screenshots	Figures 7.9 and 7.10

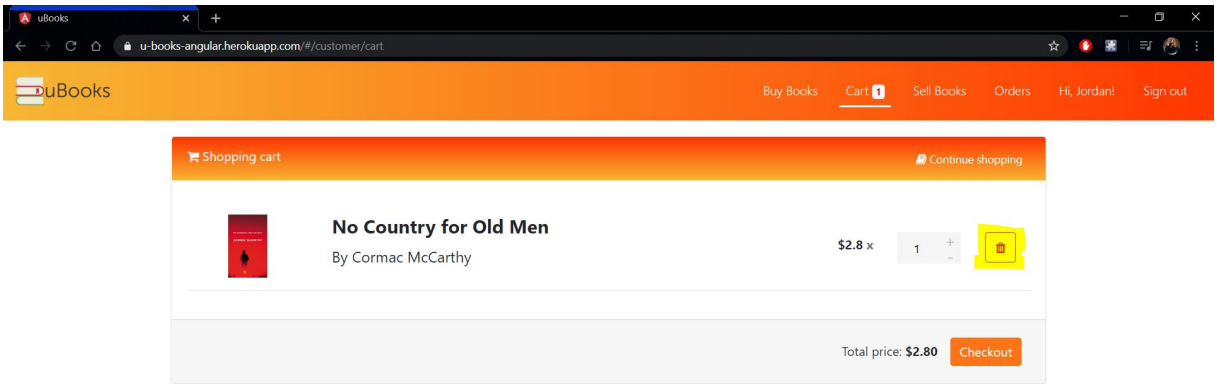


Figure 7.9: Functional Case 5 Input

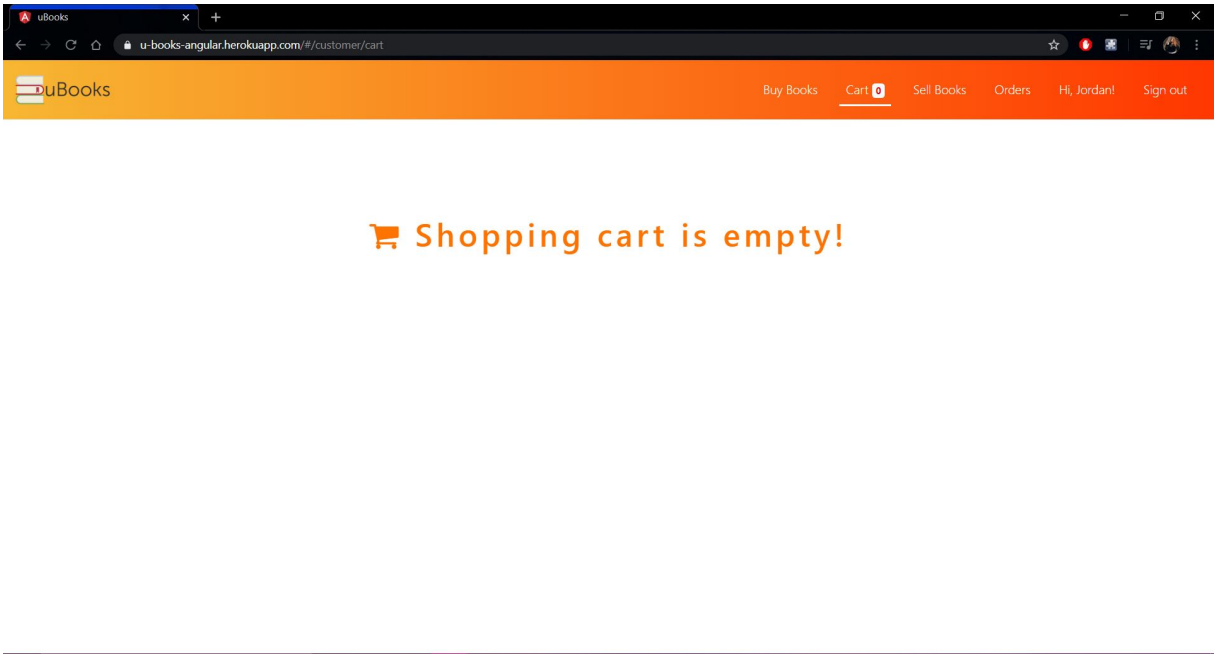


Figure 7.10: Functional Case 5 Output

7.2 Robustness Testing

Robustness testing is a method focused on testing the robustness/correctness of a software. Robustness tests determine the system's fault tolerance. In this section, there will be inputs with invalid credentials or fault tolerance events that will be tested by the robustness of their outputs. Again, there will be 5 robustness test case, one input and output screenshot each.

Case 1 - Invalid Email

Table 7.6: Robustness Testing Table 1

Input	Non-existing email
Output	Popup: That email does not exist with any user. Failed Sign In attempt
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	1. Go to https://u-books-angular.herokuapp.com/ 2. Enter an email that does not exist 3. Popup indicates that that email does not exist with any user 4. User must click 'OK' on popup and re-enter a valid email
Screenshots	Figures 7.11 and 7.12

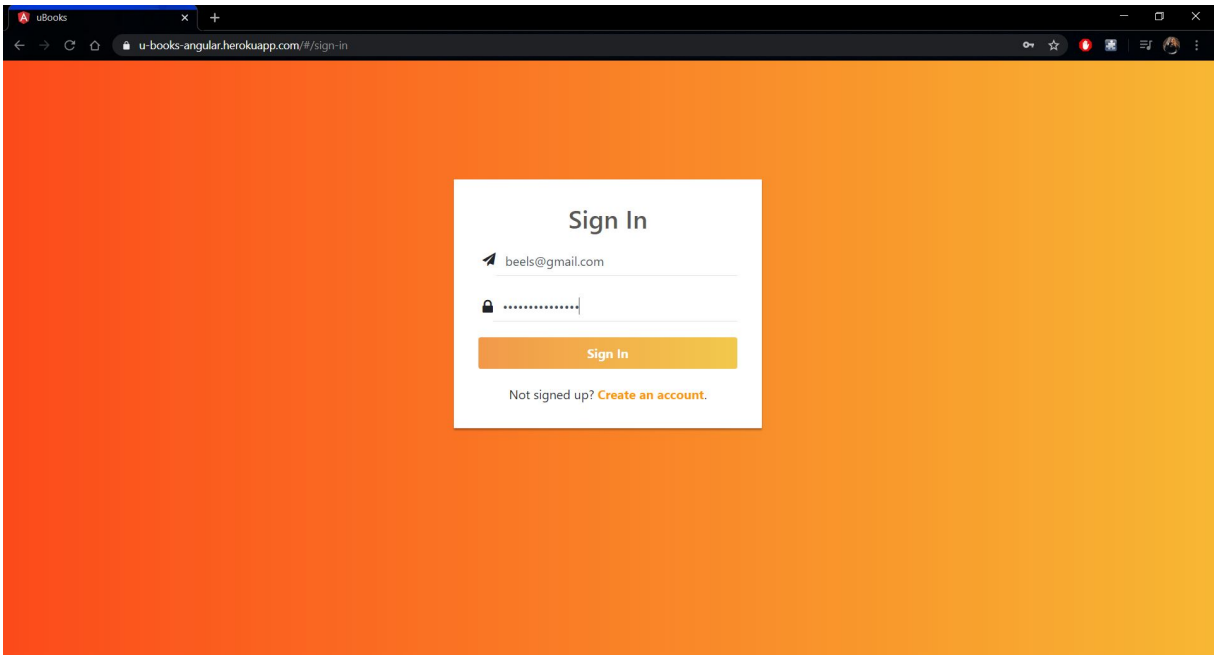


Figure 7.11: Robustness Case 1 Input

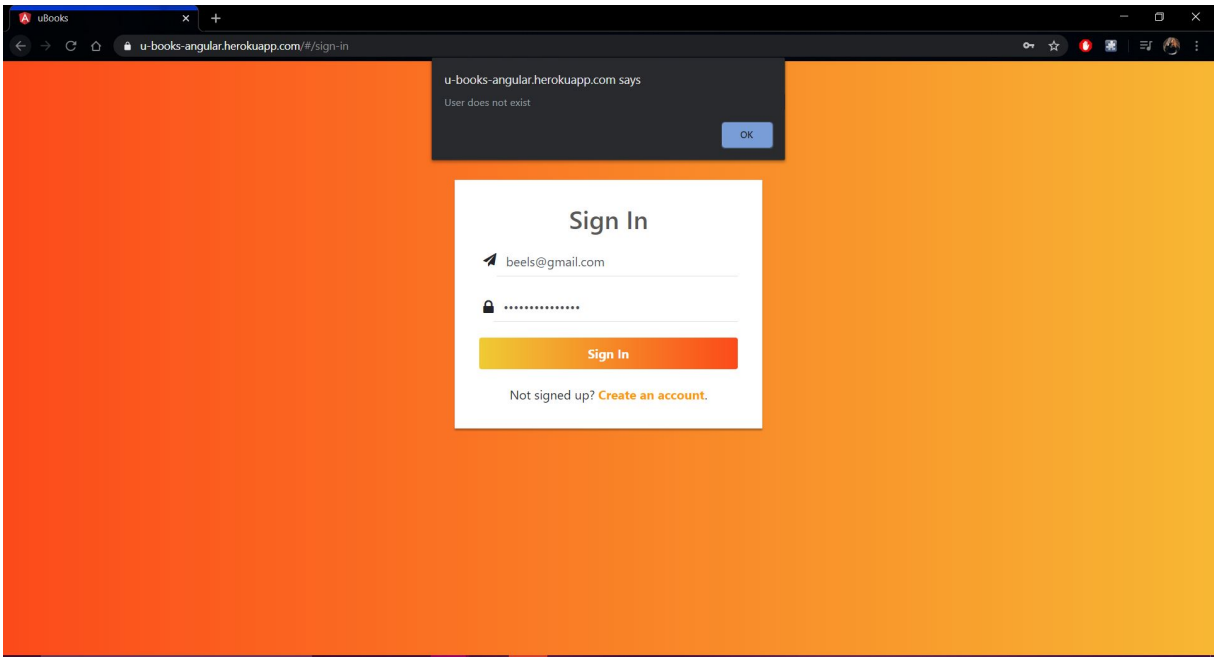


Figure 7.12: Robustness Case 1 Output

Case 2 - Sign Up with Invalid Credentials

Table 7.7: Robustness Testing Table 2

Input	An invalid telephone number, email and password
Output	Failed Sign Up attempt and the reasons why are displayed
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	1. Go to https://u-books-angular.herokuapp.com/ 2. Click Create an Account (Go to https://u-books-angular.herokuapp.com/#/sign-up) 3. Enter invalid sign up credentials 4. Click the 'Sign Up' button 5. The user account creation has failed and reasons why are displayed. 6. Note that: • Telephone numbers must be in the format: (306)123-4567 • Emails must be in the format: something@something.something • Passwords must contain: At least 1 upper case letter, at least 1 number and at least 8 characters long
Screenshots	Figures 7.13 and 7.14

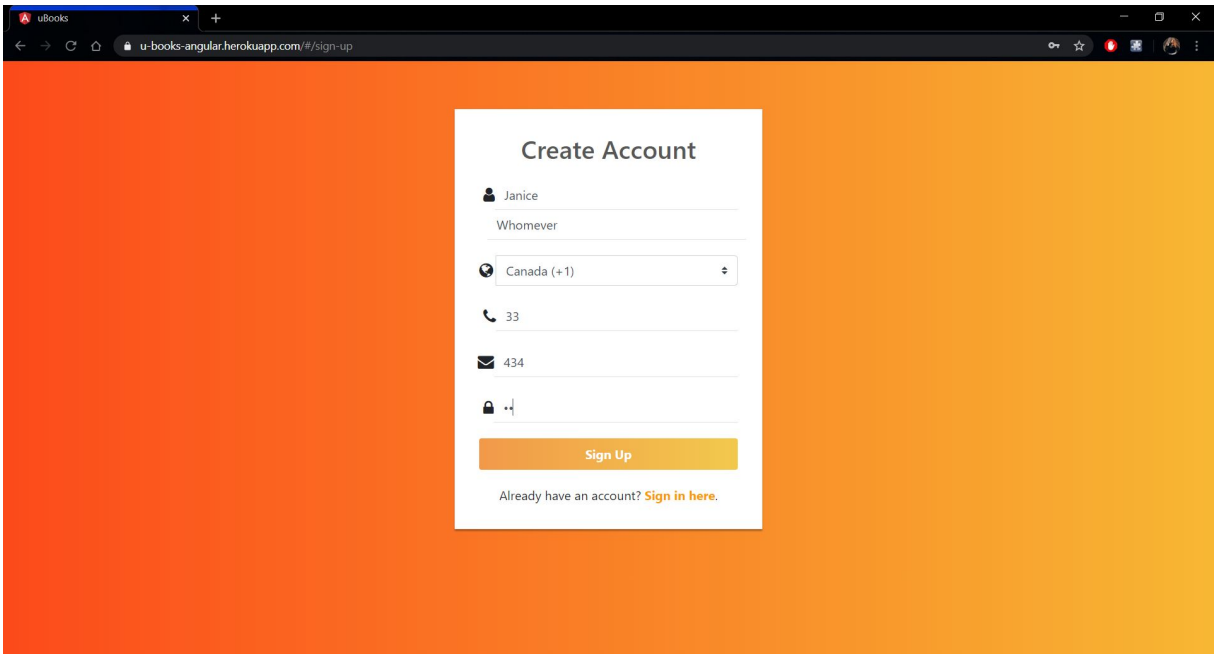


Figure 7.13: Robustness Case 2 Input

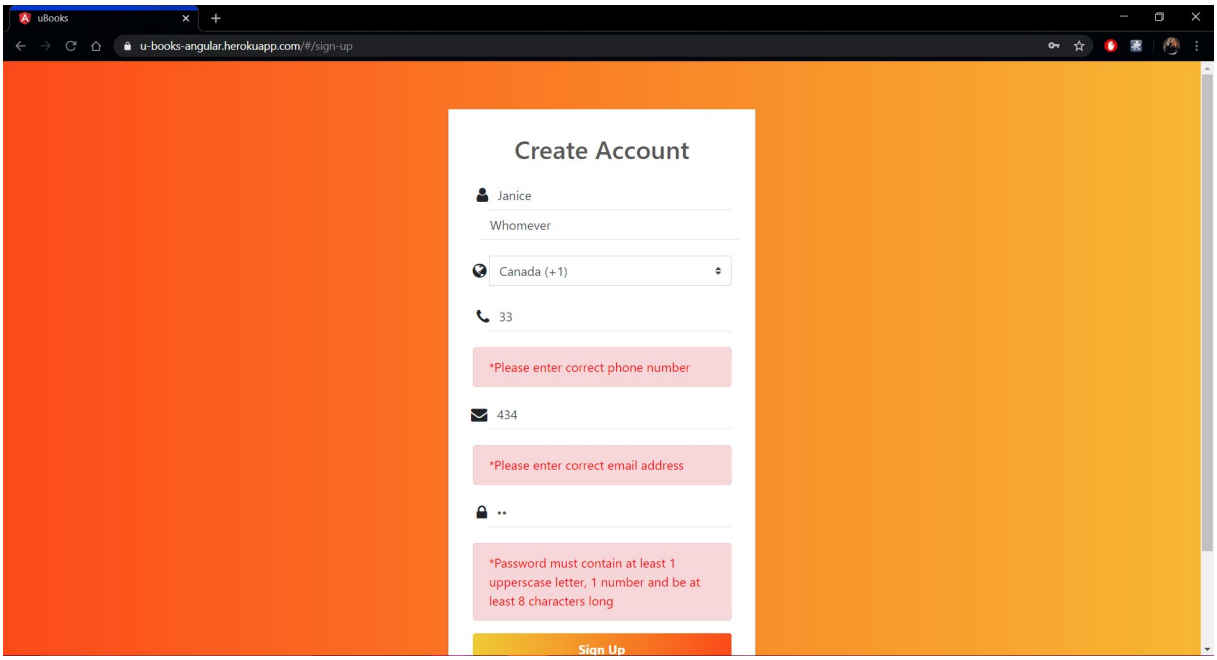


Figure 7.14: Robustness Case 2 Output

Case 3 - User: Non-existing Author Searched For

Table 7.8: Robustness Testing Table 3

Input	Non-existing author name
Output	No book is displayed
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	Assuming the User is Signed In: 1. Go to 'Buy Books' page (Go to https://u-books-angular.herokuapp.com/#/customer/books) 2. Search by an author name that does not exist (ex. Banks) 3. Click the 'Search' button 4. No book is displayed
Screenshots	Figures 7.15 and 7.16

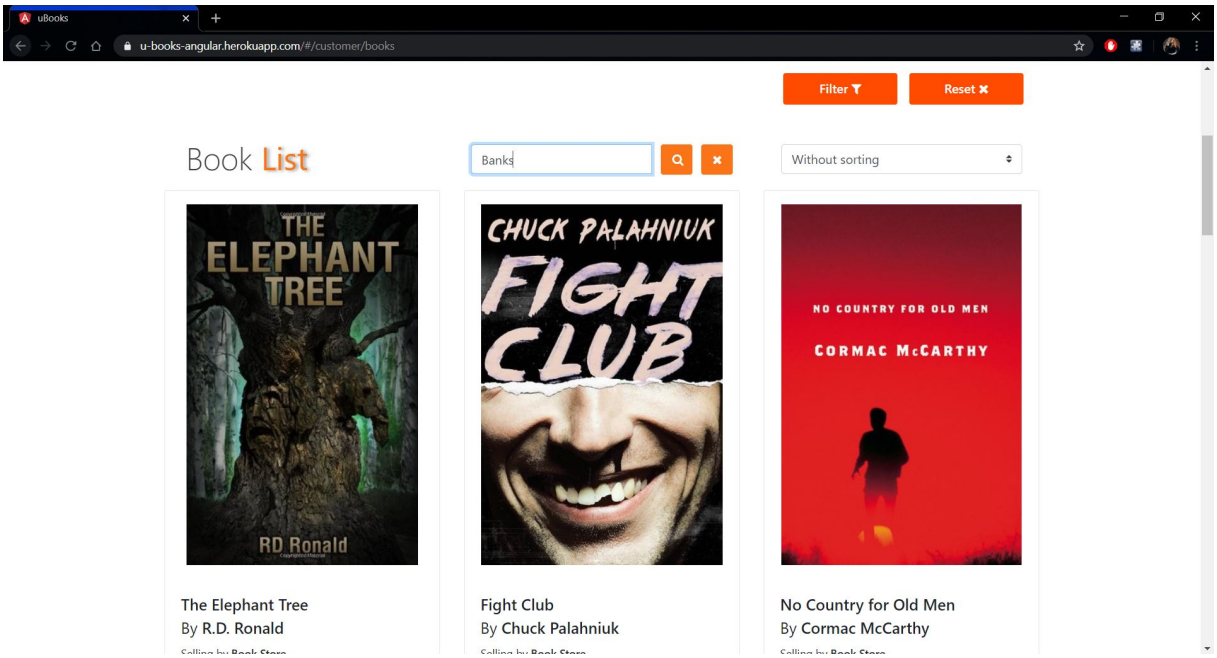


Figure 7.15: Robustness Case 3 Input

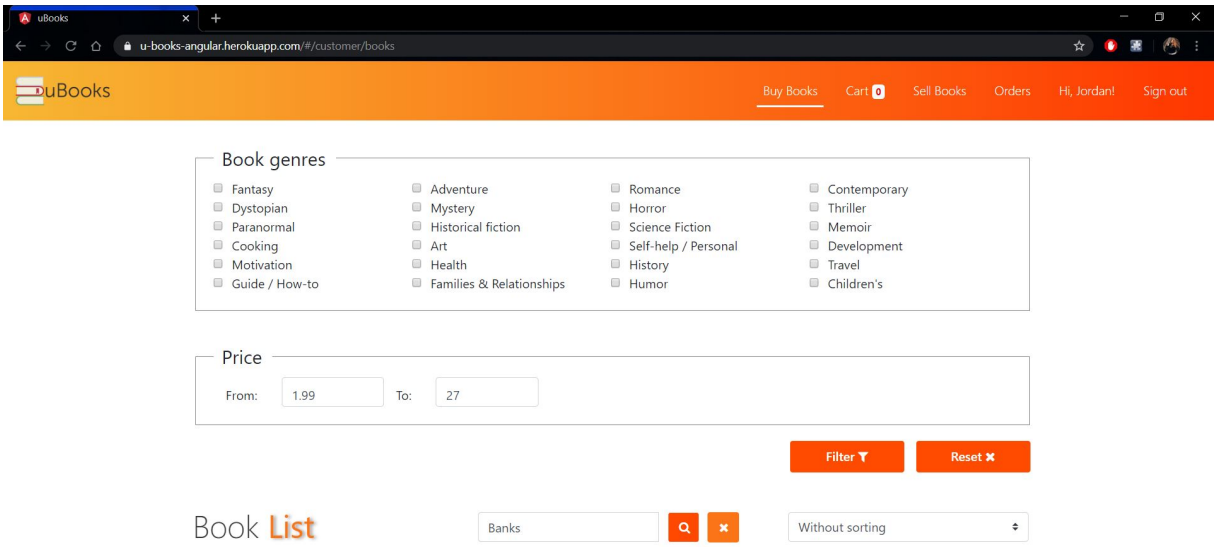


Figure 7.16: Robustness Case 3 Output

Case 4 - Admin: Adding a New Admin with Invalid Credentials

Table 7.9: Robustness Testing Table 4

Input	Invalid email and password
Output	Pop up: There is an invalid email entered or there are other invalid fields
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	Assuming the Admin is Signed In: 1. Go to 'Admins' page (Go to https://u-books-angular.herokuapp.com/#/admin/admins) 2. Click '+Add New' button 3. Enter name, surname, invalid email and an invalid password (ex. Queen, Bee, buzz, b. Respectively) 4. Pop up displays that not all fields are filled in or there are invalid fields. 5. Note that: • Emails must be in the format: something@something.something • Passwords must contain: At least 1 upper case letter, at least 1 number and at least 8 characters long
Screenshots	Figures 7.17 and 7.18

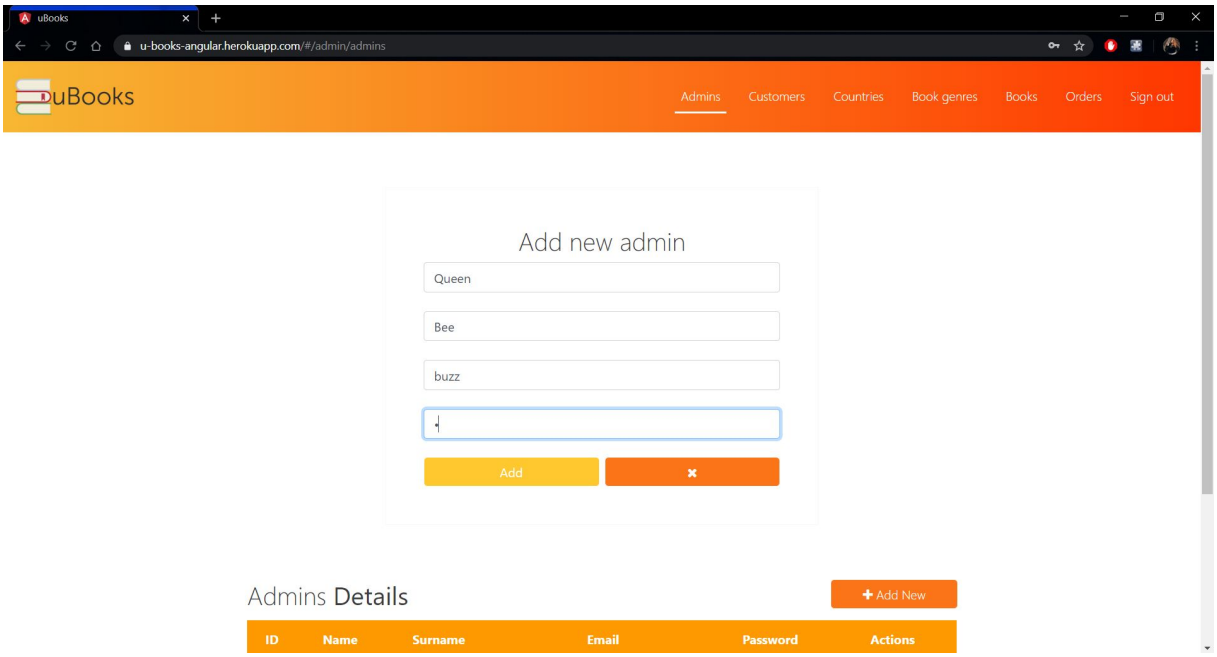


Figure 7.17: Robustness Case 4 Input

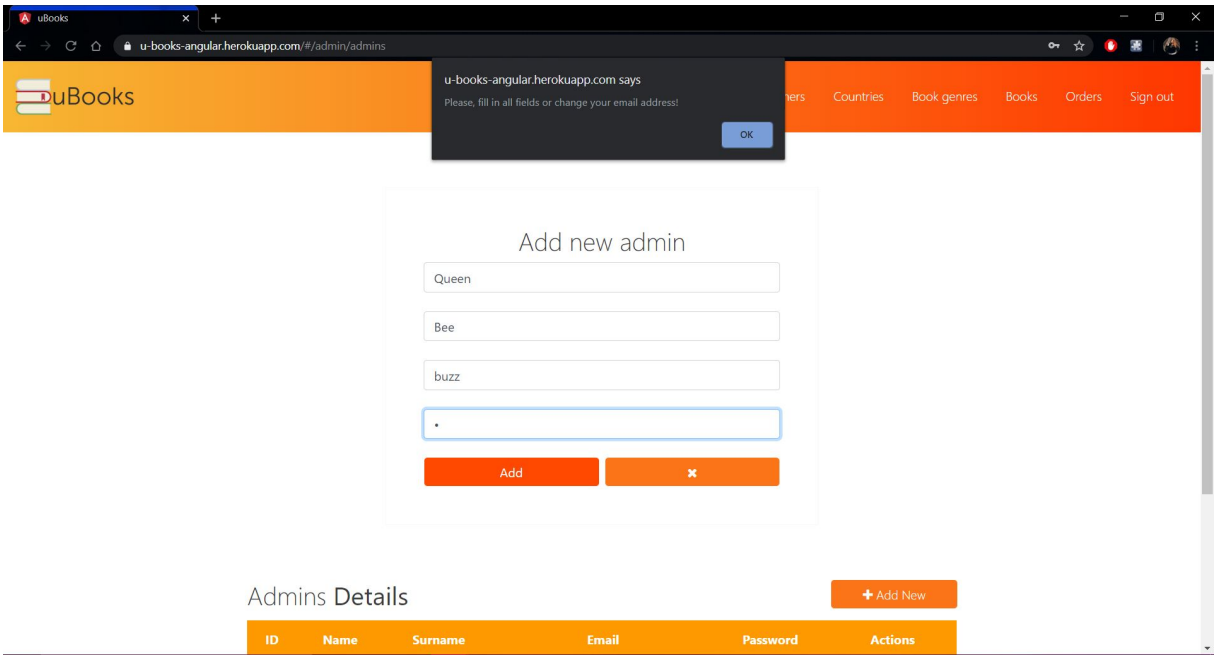


Figure 7.18: Robustness Case 4 Output

Case 5 - Admin: Attempt to Add New Book while Missing one Field

Table 7.10: Robustness Testing Table 5

Input	All fields are filled, except the 'Book title' field
Output	Admin is unable to add a new books
Software	uBooks (https://u-books-angular.herokuapp.com/)
Flow of events	Assuming the Admin is Signed In: 1. Go to 'Books' page (Go to https://u-books-angular.herokuapp.com/#/admin/books) 2. Fill in all fields except 'Book title' 3. Admin is unable to add a new book without all fields filled
Screenshots	Figures 7.19 and 7.20

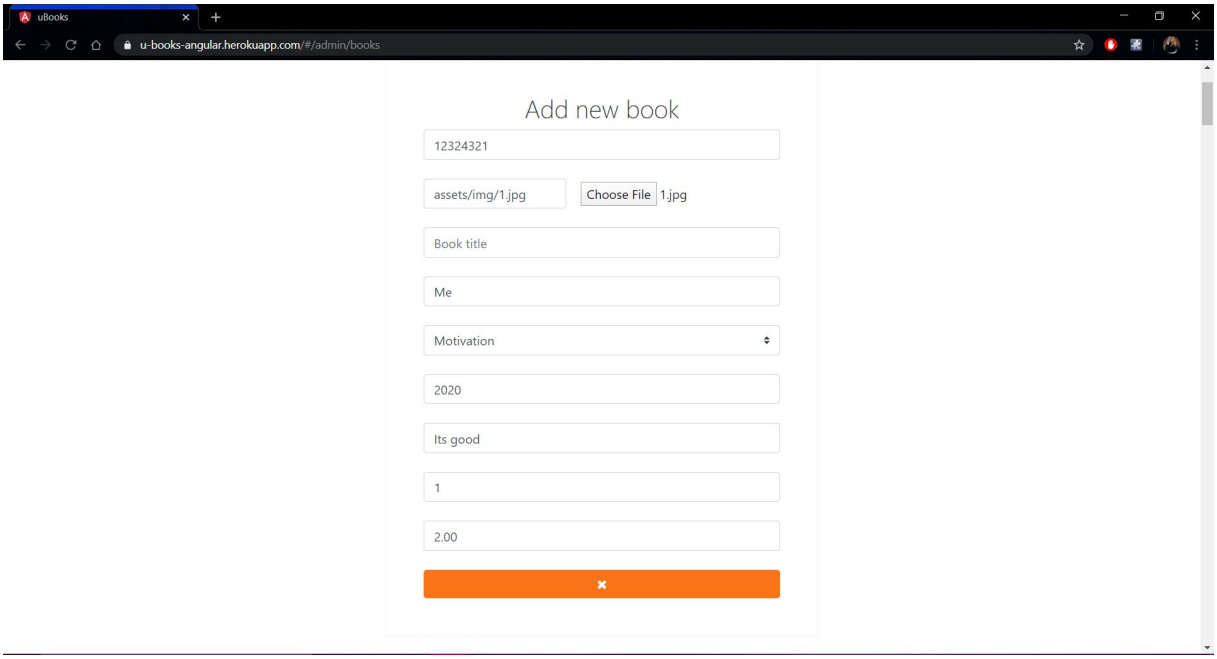


Figure 7.19: Robustness Case 5 Input

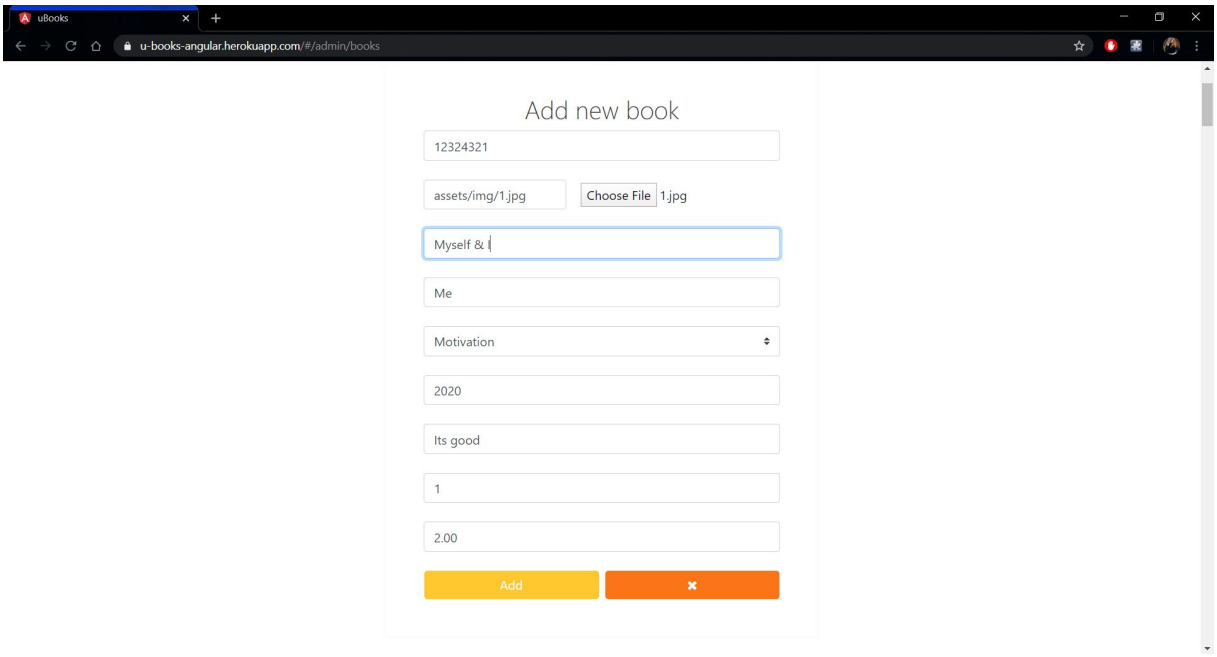


Figure 7.20: Robustness Case 5 Output

7.3 Time-Efficiency testing

Time-Efficiency/Performance was first determined by the WebPageTest.org. The Figures 7.21 and 7.22 show the results of uBooks in this tool. Our software could be refined to be faster in the future, but for now the Time-Efficiency is OK because the slow load time is the result of the free domain we are using and the real-time data base that has a three way connection via http request.

Tester: Toronto-69.87.222.251
Re-run the test

View JSON result
Raw page data - Raw object data
Export HTTP Archive (.har)
View Test Log

								Document Complete			Fully Loaded			
	Load Time	First Byte	Start Render	First Contentful Paint	Speed Index	Last Painted Hero	First CPU Idle	Time	Requests	Bytes In	Time	Requests	Bytes In	Cost
First View	14.722s	0.310s	14.600s	14.536s	14.602s	14.600s	> 14.600s	14.722s	10	6,134 KB	14.817s	11	6,139 KB	$$$$$
Repeat View	4.402s	0.312s	4.400s	4.340s	4.401s	4.400s	> 4.400s	4.402s	10	6,134 KB	4.402s	10	6,134 KB	

Figure 7.21: Loading Times

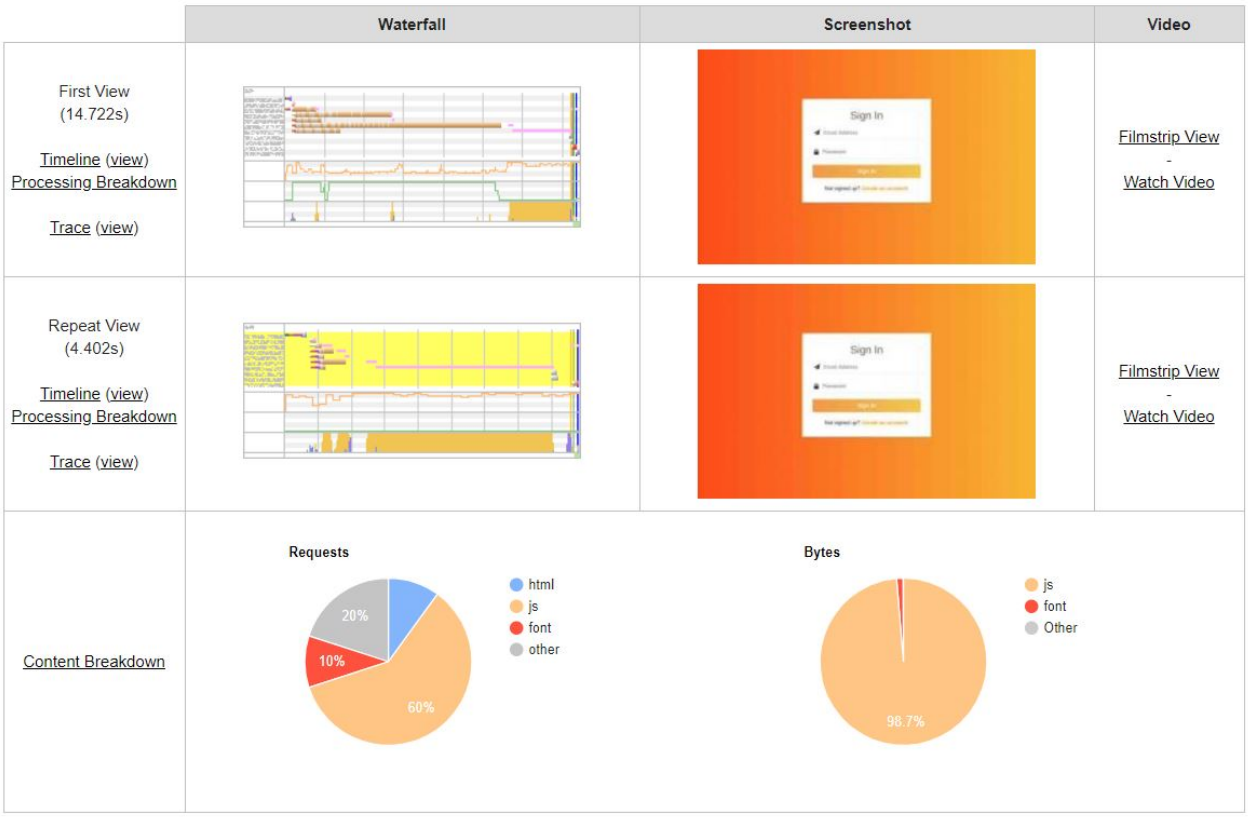


Figure 7.22: Loading Times Details

"Efficiency testing tests the amount of resources required by a program to perform a specific function. In software companies, this term is used to show the effort put in to develop the application and to quantify its user-satisfaction" (TheEconomicTimes). In the rest of this section, there will be 3 time tests for each of the 5 events. Page names will be included. In addition, a BenchMark of average time and maximum wait time in ms will be calculated from the 3 times. There will be one screenshot for each event to display the first time test result.

Case 1 - Sign In

Page	Event	Time	BenchMark
Sign In and Buy Books	Login with the correct credentials and load Buy Books page	1. 5100ms 2. 5637ms 3. 5046ms	Average: 5261ms, Max. wait time <= 5637ms

Figure 7.23: Time-Efficiency Case 1 Table

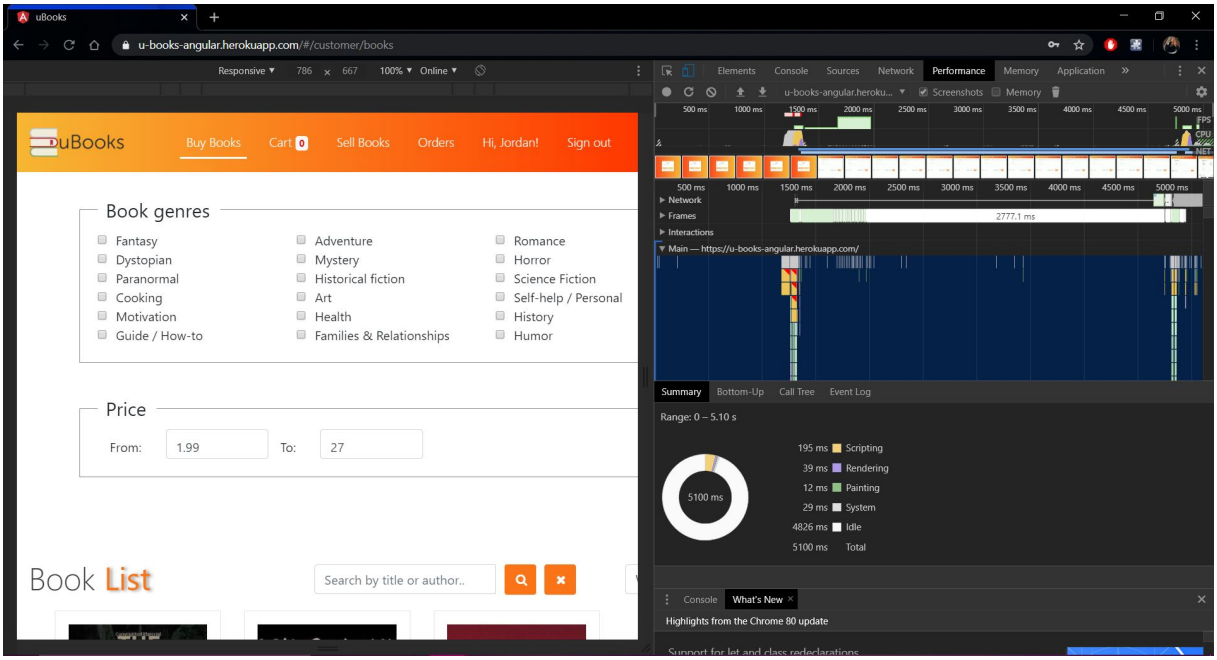


Figure 7.24: Time-Efficiency Case 1

Case 2 - Sign Up

Page	Event	Time	BenchMark
Sign Up and Sign In	Sign Up with the correct credentials and load Sign In page	1. 5192ms 2. 2852ms 3. 2842ms	Average: 3628ms, Max. wait time <= 5192ms

Figure 7.25: Time-Efficiency Case 2 Table

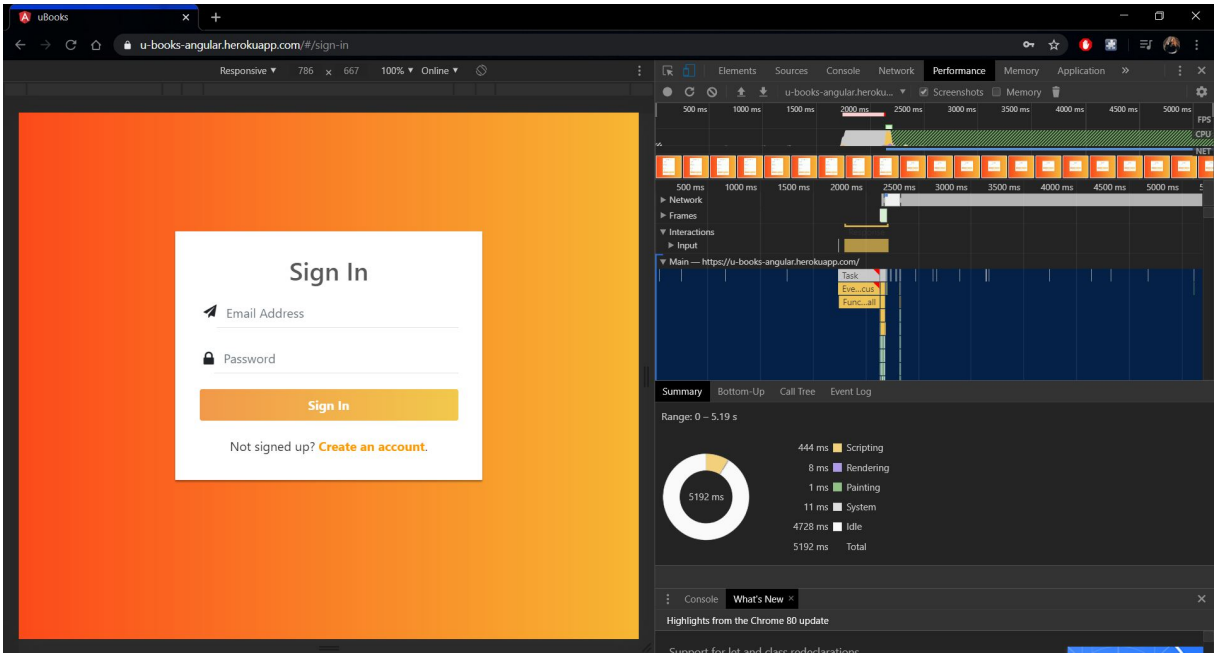


Figure 7.26: Time-Efficiency Case 2

Case 3 - Search for a Specific Book by Title

Page	Event	Time	BenchMark
Buy Books	Search for a specific book by title	1. 357ms 2. 354ms 3. 409ms	Average: 266ms, Max. wait time <= 408ms

Figure 7.27: Time-Efficiency Case 3 Table

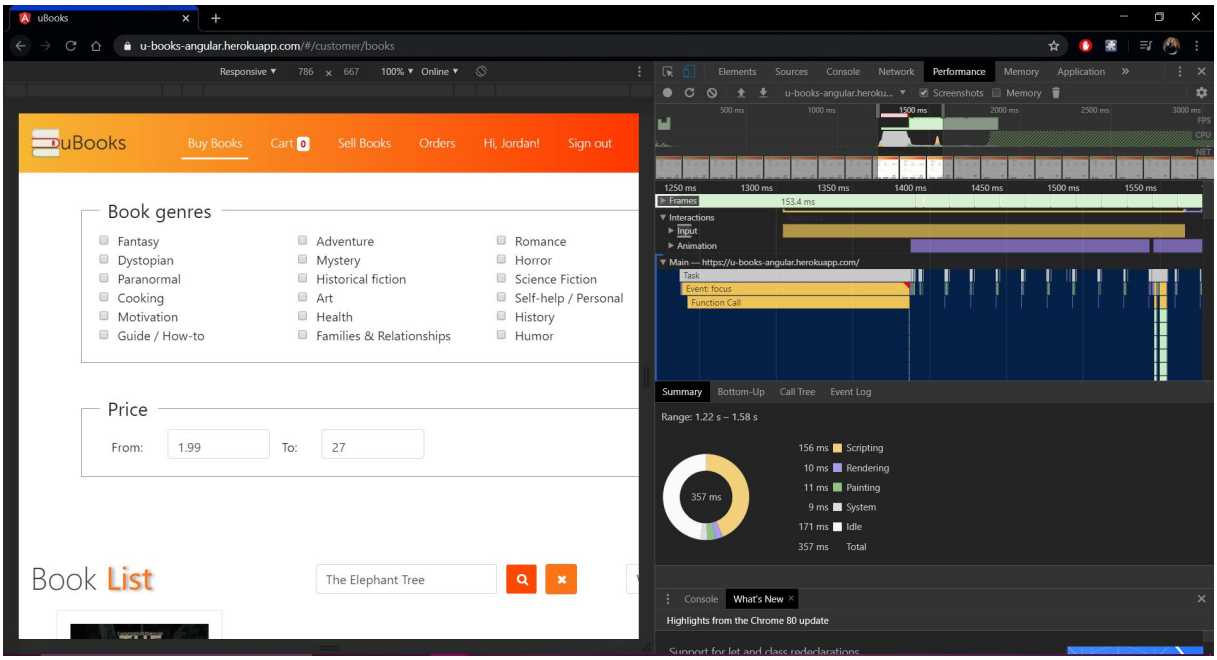


Figure 7.28: Time-Efficiency Case 3

Case 4 - Add Book to Cart and Checkout

Page	Event	Time	BenchMark
Buy Books and Cart	Add book to cart and go checkout	1. 10283ms 2. 12884ms 3. 11244ms	Average: 11470ms, Max. wait time <= 12884ms

Figure 7.29: Time-Efficiency Case 4 Table

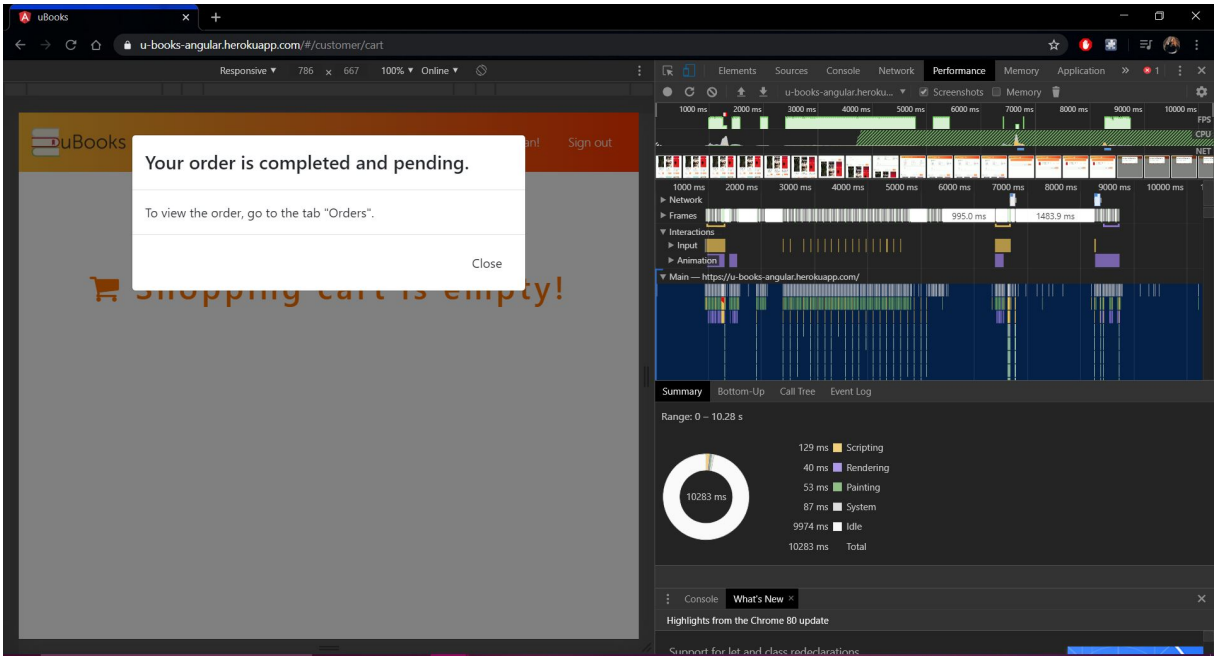


Figure 7.30: Time-Efficiency Case 4

Case 5 - Sign Out

Page	Event	Time	BenchMark
Buy Books and Sign Out	Clicking sign out loads sign in page	1. 1629ms 2. 1102ms 3. 1468ms	Average: 1400ms, Max. wait time <= 1629ms

Figure 7.31: Time-Efficiency Case 5 Table

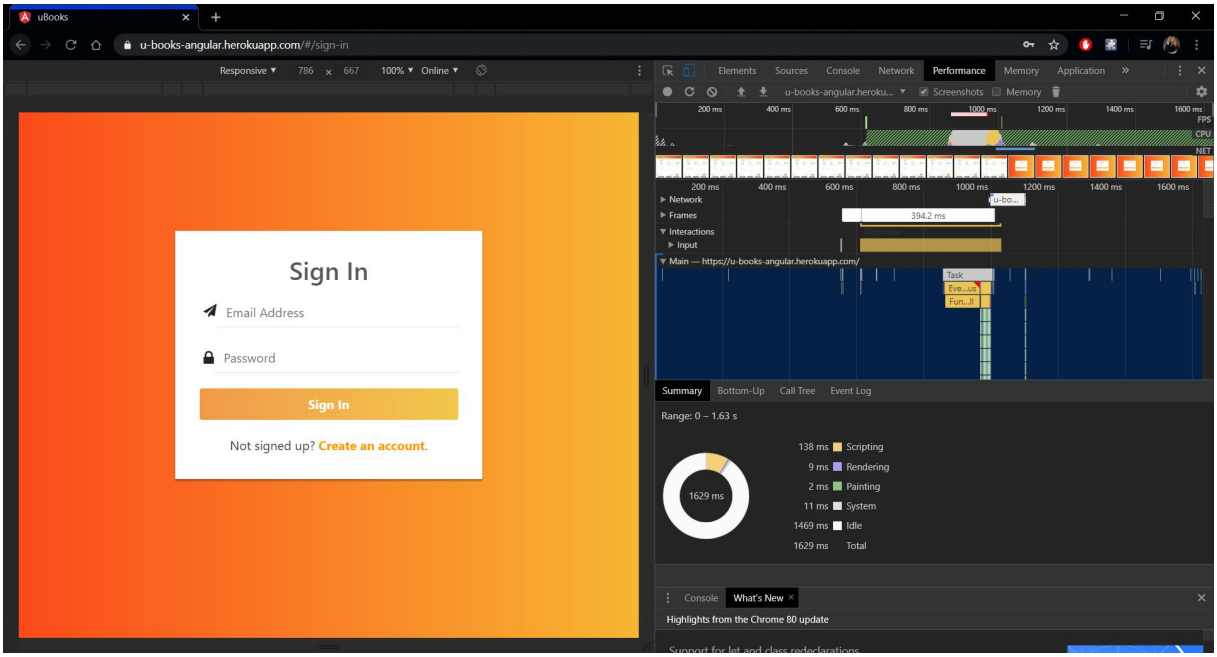


Figure 7.32: Time-Efficiency Case 5

Bibliography

[1] Academind - *Angular - Node - MongoDB & Express (MEAN) Tutorial for Beginners - Getting Started* (June 8, 2018). Retrieved 12th of April 2020, from Youtube https://www.youtube.com/watch?v=1tRLveSyNz8

[2] AmarInfotech - *Why AngularJs Is Best Javascript Framework For Front End Development?*(n.d.). Retrieved 24th of April 2020, from https://www.amarinfotech.com/why-angularjs-best-javascript-framework-for-front-end-development.html

[3] GeeksForGeeks - *Prototype Design Pattern* (n.d.). Retrieved 4th of April 2020, from https://www.geeksforgeeks.org/prototype-design-pattern/

[4] Lucid Software, *UML Activity Diagram Tutorial* Retrieved 3 April 2020, from https://www.lucidchart.com/pages/uml-activity-diagram

[5] Requirements Quest - *Non-functional Requirement Examples*(n.d.). Retrieved 12th of April 2020, from https://requirementsquest.com/nonfunctional-requirement-examples/

[6] Scotch - *Deploy a Node App to Heroku* (n.d.). Retrieved 12th of April 2020, from https://scotch.io/tutorials/how-to-deploy-a-node-js-app-to-heroku

[7] Steve McConnel, Microsoft Press - *Code Complete, 2nd Edition* (2004). Retrieved 12th of April 2020.

[8] TechTerms - *User-Friendly* (January 29, 2014). Retrieved 12th of April 2020, from https://techterms.com/definition/user-friendly

[9] TheEconomicTimes - *Definition of 'Efficiency Testing'* (n.d.). Retrieved 11th of March 2020, from https://economictimes.indiatimes.com/definition/efficiency-testing

[10] tutorialspoint- *AngularJS - MVC Architecture* (n.d.). Retrieved 25th of March 2020, from https://www.tutorialspoint.com/angularjs/angularjs_mvc_architecture.htm

[11] tutorialspoint- *Observer Pattern* (n.d.). Retrieved 25th of March 2020, from https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

[12] tutorialspoint- *Prototype Pattern* (n.d.). Retrieved 25th of March 2020, from https://www.tutorialspoint.com/design_pattern/prototype_pattern.htm

[13] TutorialsTeacher - *MVC Architecture* (n.d.). Retrieved 20th of March 2020, from https://www.tutorialsteacher.com/mvc/mvc-architecture

[14] WhatIs.com - *Reliability, Availability and Serviceability (RAS)* (Mar. 2011). Retrieved 12th of April 2020, from https://whatis.techtarget.com/definition/Reliability-Availability-and-Serviceability-RAS