CITADEL

(Book Search Website)

Submitted by:

Yash Patel (200403570)

ynp062@gmail.com

Jaydeep Lakhani (200418547)

jli530@uregina.ca

PROJECT REPORT

submitted to the

Professor: Alain Maubert

Spring/Summer 2022 - CS 476

INDEX

1. Problem Definition

- Creating a system to post/view an author's written books online.
- Ability to view, search, edit, and delete books in the system
- In order to exchange information with other users on the network, individuals may also upload new books to their profiles (which must have a unique ISBN).
- *NOTE: This is a book explorer and doesn't have a viewable book feature in the system. This system is only for authors/users who want to see who has written what type of book and not the content inside the book.*

Application Domain: Book Search Website

- It's a library system that allows users to look for books by certain authors and read summaries of those works.

The Project's Motivations

- We discovered a website called **BookFinder.com** that discovers books that you want to buy and can also search for secondhand books, but we wanted to create something similar but solely a book explorer type of thing where users can just explore and learn about books and authors.
- In addition, from our research, we found that bookfinder.com has a poorly done user interface that made us think, let's make something similar with our own plans, which will make our project less daunting and more simplistic way to search books.
- There are numerous book websites on the internet, but none of them are book explorer websites, where users may conduct pre-reading research on books to increase their excitement for the material they will read.

2. Feasibility study

There are four feasibility studies we have included on our book search website.

1. **Economic Feasibility:** Since we are using the University of Regina's free domain, developing on it is the most advantageous choice financially. Our system is relatively simple to maintain because it was only created by two people, requires no outside assistance, and doesn't cost much money or time to run.

2. **Technical Feasibility:** As can be seen, it is a web-based application that only needs an internet connection to load when a job is being completed. It won't put a lot of strain on the system's physical components. It is also compatible with virtually any environment, including Windows, Mac, or any other sort, as well as any hardware for desktops or tiny screens.

3. **Operational Feasibility:** This viability is an indicator of how successfully it will carry out tasks when a user accesses our platform and a justification for why users ought to use our website. Our website is incredibly helpful for people looking to find books to read, get a brief summary before buying, learn more about authors and the books they've written, and share their content with others. Authors can also add book cover photos and summaries of books they've written to our system so that they can share their work and reach out to more readers.

4. **Personal Feasibility:** Two people may independently manage this website, which requires front and backend coding as well as website documentation. Our book search website benefits from not requiring any additional outside resources to complete any of these tasks.

- Our program has a more beautiful user experience in terms of aesthetics and is more engaging when compared to BookFinder.com.
- Consider that we examine the search pages for both websites. In any case, our online application is more appealing and makes it simple to obtain book summaries for pre-reading.
- Another option is themillions.com, which offers all the details of a book but not the synopsis and does not allow you to join and contribute your own books for sharing content.
- This website lets you search for books and compare the costs of new and used copies, but it is inferior to our website because it also lets you search for authors. For example, if you are looking for a specific book and you enjoy the writing of a particular author and want to learn more about their other books, you can do so in our application.

3. Software requirements elicitation and specification:

a) Functional requirements

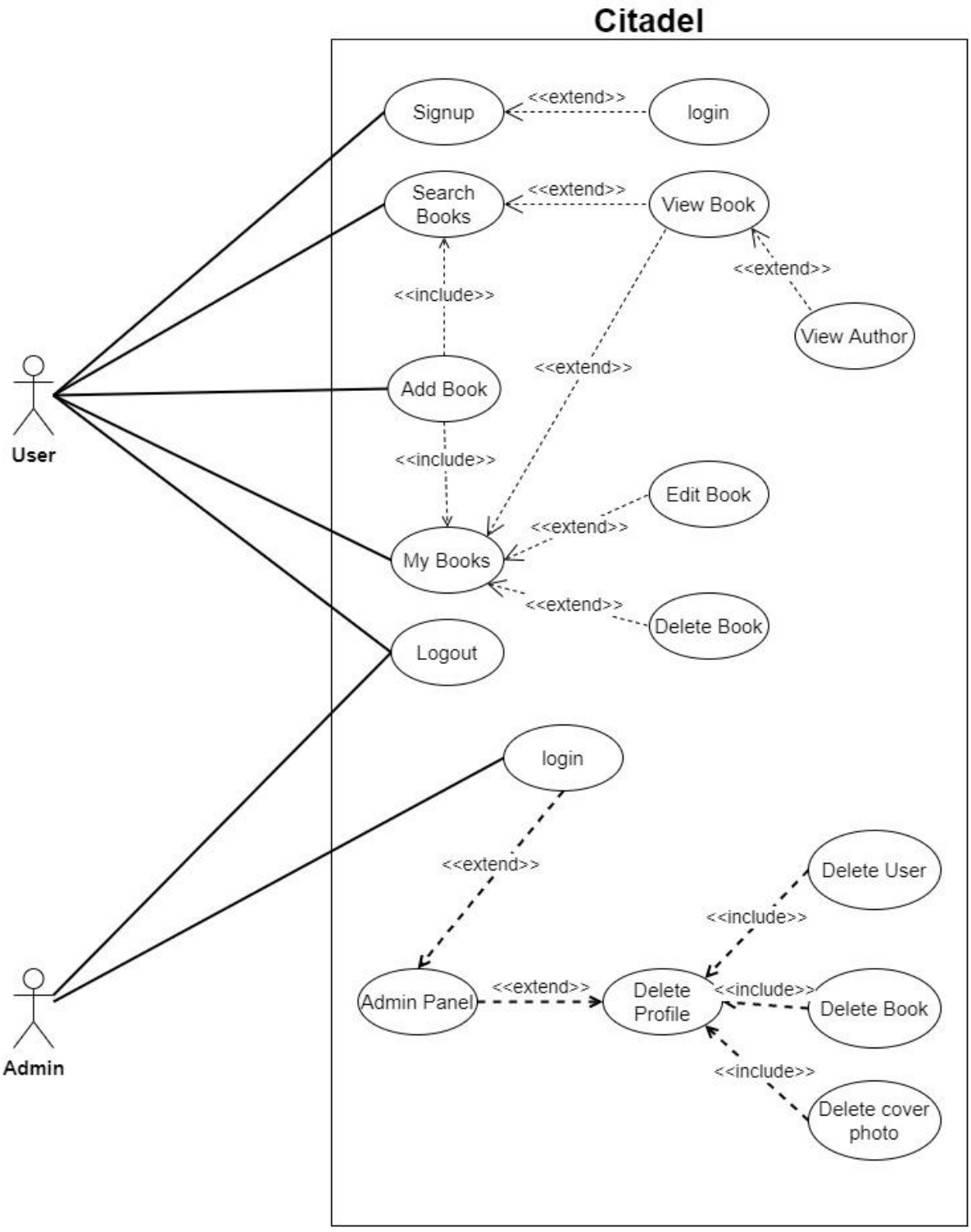
Customer:

- Login/logout: Every time a user has to login with their username and password to access the website and logout after use.
- Signup with an unused email/username to register in the system and then use it for login.
- Add a new book by giving a title, page count, published date and ISBN, summary, and uploading the book cover photo.
- Search books in the database by title, author name, ISBN, or published date.
- Edit book information like title, page count, published date, ISBN number and summary.
- Delete a book that belongs to the logged-in user in the database.
- View Book and its details.
- View another author's profile from within the book.

Admin:

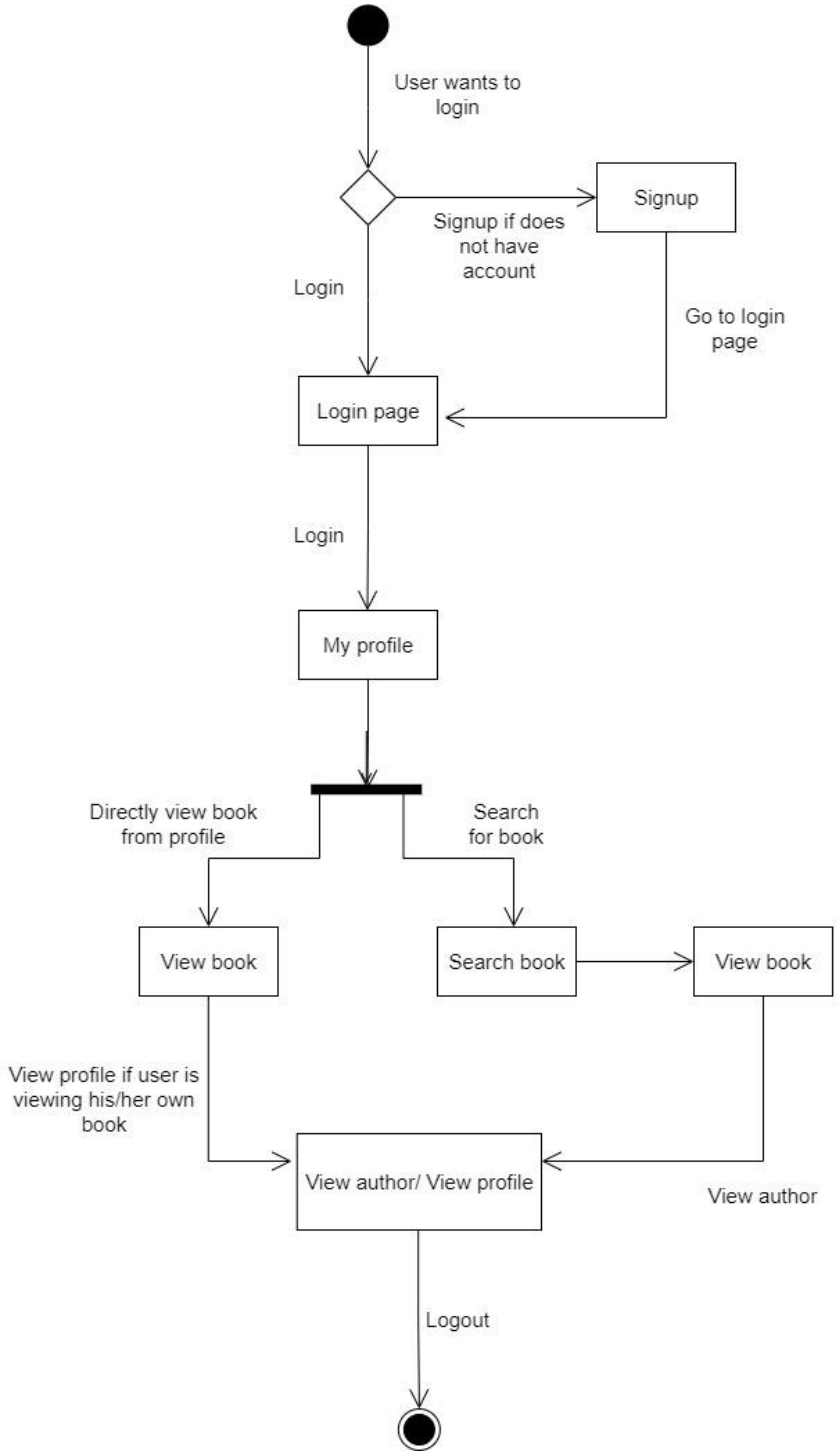
- Login into the admin portal using admin credentials
- Manage users by deleting them if need be. (Deletes the user, books that belong to them and the book cover for managing server storage space.)

b) Use Case Diagrams

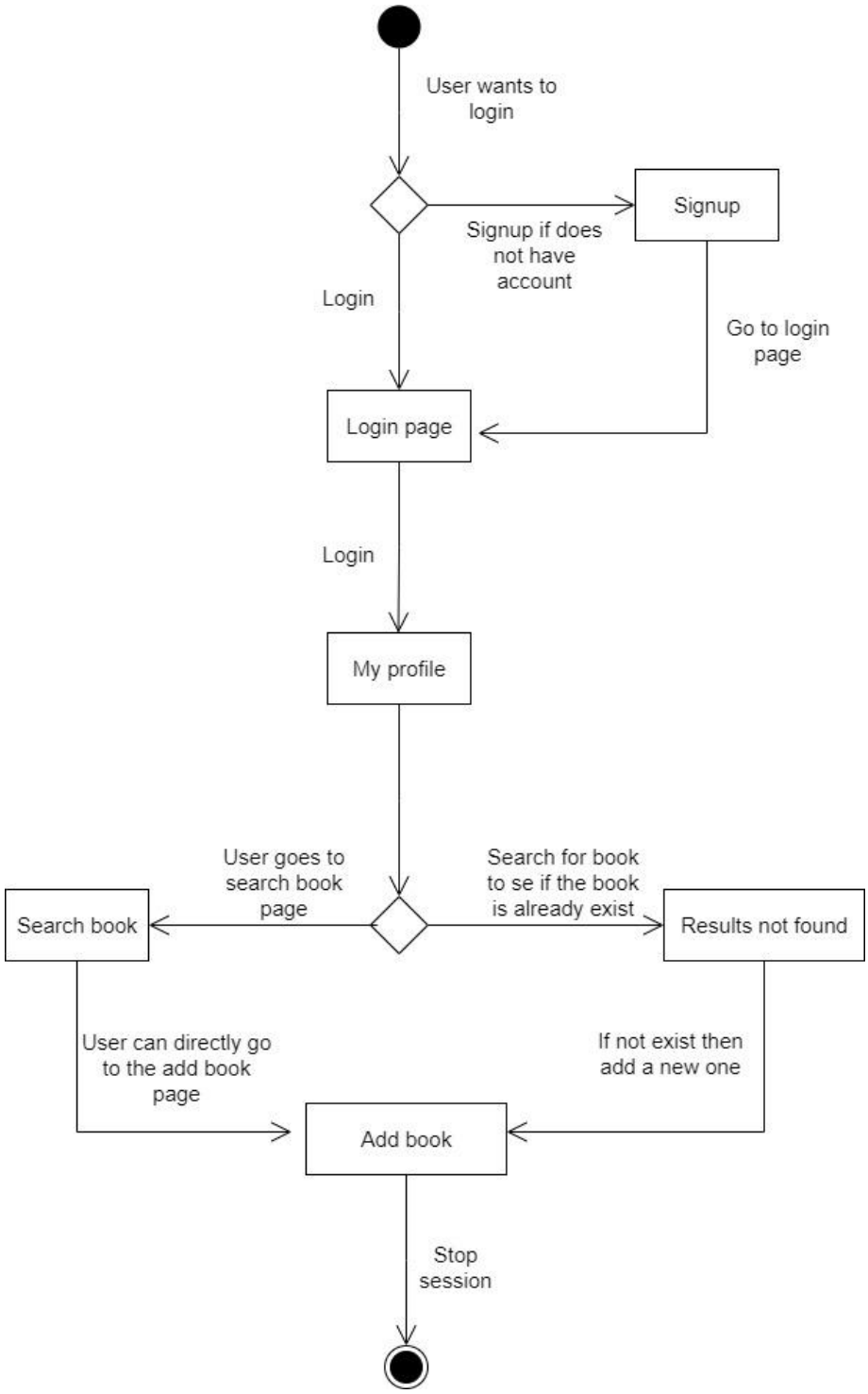


c) Describe two use cases Activity Diagram

CASE 1: User wants to view author of a book.



CASE 2: If a book hasn't previously been uploaded, the user wishes to first search for it before adding a new book.



d) Software Qualities:

Correctness

- The web application functions as intended.
- Data accuracy is ensured through login and registration.
- The functions to search, add, modify, and delete books should function as intended and carry out activities correctly.
- When a user searches for books, only those that match the search criteria should appear.
- Enters data correctly into the database without conflicting or misplacing it with other user's data.

Time-efficiency

- When multiple users access our web application, we need to provide timely responses with the correct data.
- Users need to get results quickly while searching a huge database of books that can improve their productivity.

Robustness

- The developed application must provide an appropriate response when the user enters the wrong data format.
- While another user is simultaneously using the program, the system should respond with accurate information.
- When a significant number of people utilize the system simultaneously, it shouldn't crash.

4. Top-level and low-level design:

a) MVC Architecture

As we can see, MVC stands for the Model, View, and Controller architecture, which is widely used to create online web applications. A prime example of MVC architecture in our project can be seen in many pages, but let's explain in brief some of them in pages such as signup.php and login.php. PHP code calls the backend MODEL file 'function.php' functions to operate operations for the backend and after completing the function it returns if it was successful at a given task, such as inserting a user or finding if a user exists in the database and then notifying the signup.php file.

➔ Model

The model in this instance is a low-level type that is used to interact with the database in accordance with commands from the controller. The model's primary function is to retrieve data from the database and provide it to the controller. As you can see, it is helpful for our application as we need to store user information in the database or get information from the database as needed using php files in the MySQL database. The files dbconnect.php, functions.php, and loggedinuserdata.php are the key files to interact with databases in our system to store and retrieve data from the database as needed.

➔ View

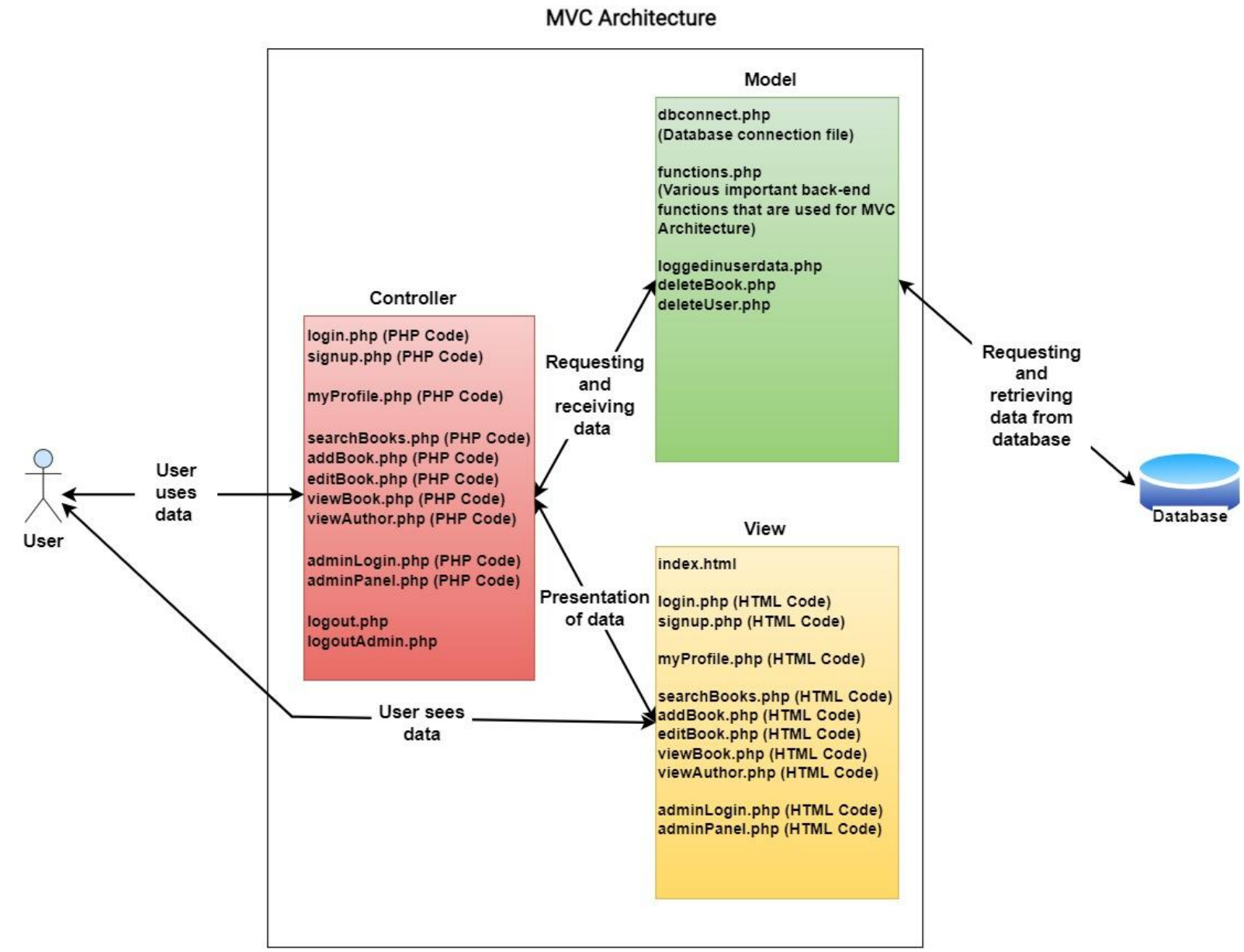
This model's sole duty is to present the website to the viewer in an appealing manner so that they have a better experience with the information, functions, and more. It doesn't interact with the model or database; it just communicates with the controller. The .html and .css and .php files which have html code in it that created the stunning interface of our web application are visible in our application.

➔ Controller

The major component of the MVC design that is actively working is the controller. This is the element that links the view and model together. It simply handles the data that the model receives and takes direct instructions from the user to obtain the data and show it to the user

through a view. It does not deal with data logic or other related issues. Javascript files are used in our application to manage input actively on what to input into the database in a valid format depending on given regex syntax. Php files such as addBook, editBook, myProfile, Login, and signup to insert and fetch data by asking the model object. Then the model object communicates with the controller to receive the data and the controller sends the data as output to the view.

Citadel MVC Architecture



b) Design Patterns

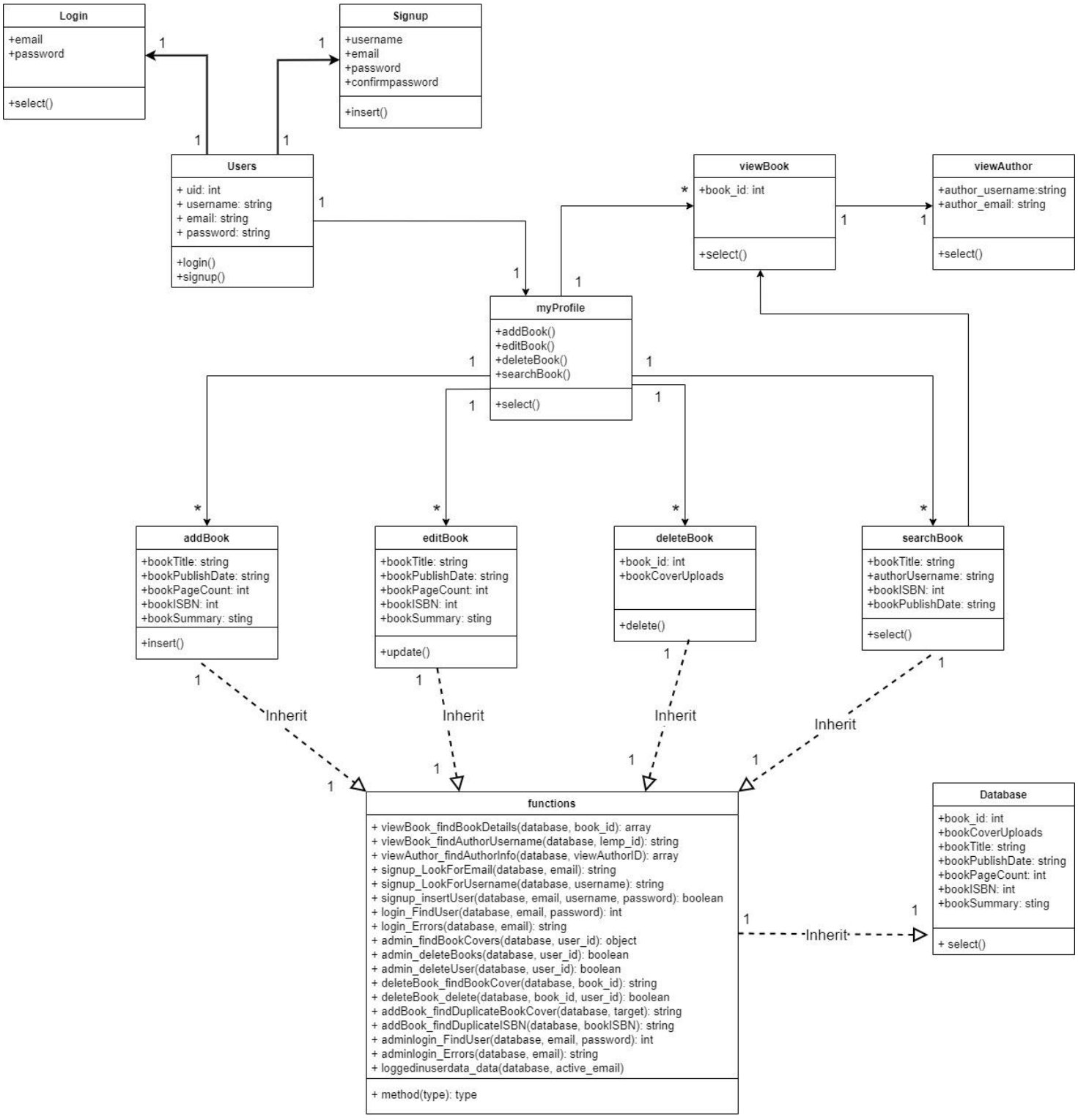
Observer Pattern

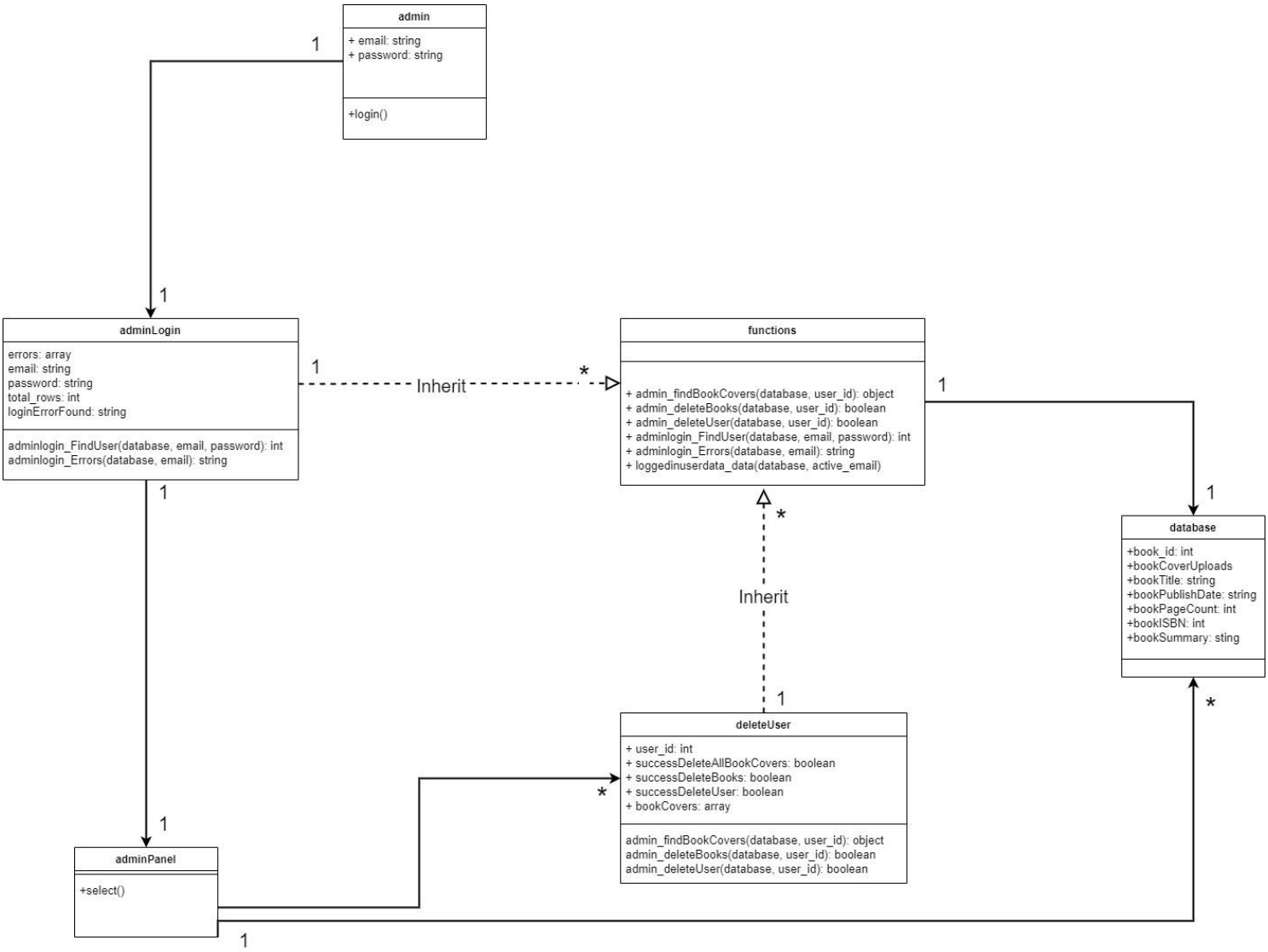
- Because observer patterns are behavioral patterns, they will alert other objects that are also watching a certain item if any changes are made to it. It can be done by the update() function in any class.
- The subject and the observers are the two important classes in this design, which explains how to build connections between classes. The subject may have as many observers as necessary.
- Here the observer pattern is very useful as it notifies other objects if any event changes to any of the objects.
- Let's just take an example for our system. adminPanel.php is a file that shows the list of users who are active in the database. Admin can delete the user by clicking on the button which in turn opens the deleteUser.php file for that specific user. deleteUser.php has imported a package for database connection (dbconnect.php) and database operations functions (functions.php). deleteUser.php makes a request one by one to delete items related to the user by first deleting book cover photos from the database. If the operations from the function calls are performed successfully then a success message is returned. Thereafter the same goes for deleting books related to the user and the actual user itself. All different delete functions are called and each returns whether the operation was successful. If everything was successfully deleted, the admin is notified that the user was deleted successfully. Here we see that the core blueprint of the observer design pattern has been implemented in our system.
- Many more important functions.php file have the Observer Design Pattern implemented in some sort of way.
- Examples of such files that have an Observer Pattern are listed below:
    - login.php, signup.php, addBook.php, deleteBook.php, viewBook.php
    - the core functions are implemented in functions.php

Simple Factory Pattern

- The factory pattern is in charge of constructing an object, but it lets its subclasses choose which class to start with.
- A direct object is more difficult to create than making a object in class. Subclasses have the ability to provide enhanced versions of objects thanks to factory methods.
- The factory pattern deals with the problem of manufacturing items without knowing precisely what kind of object will be made. This pattern is in charge of creating the object. The general and expected behavior of the subclasses is determined by a superclass.
- Let's take a practical example from our project on where Simple Factory is created. There are many files where it is implemented but one of the examples can be seen in login.php where we have created an object for functions.php where all the functions are defined for certain tasks to perform. Whenever every user submits a login button in our system we check if there are similar users in the system by functions.php. We create the object of that function and call whichever function that we like for this example .
- Examples of such files that have a Simple Factory Pattern are listed below:
    - deleteBook.php, signup.php, addBook.php, viewAuthor.php
    - the core functions are implemented in functions.php

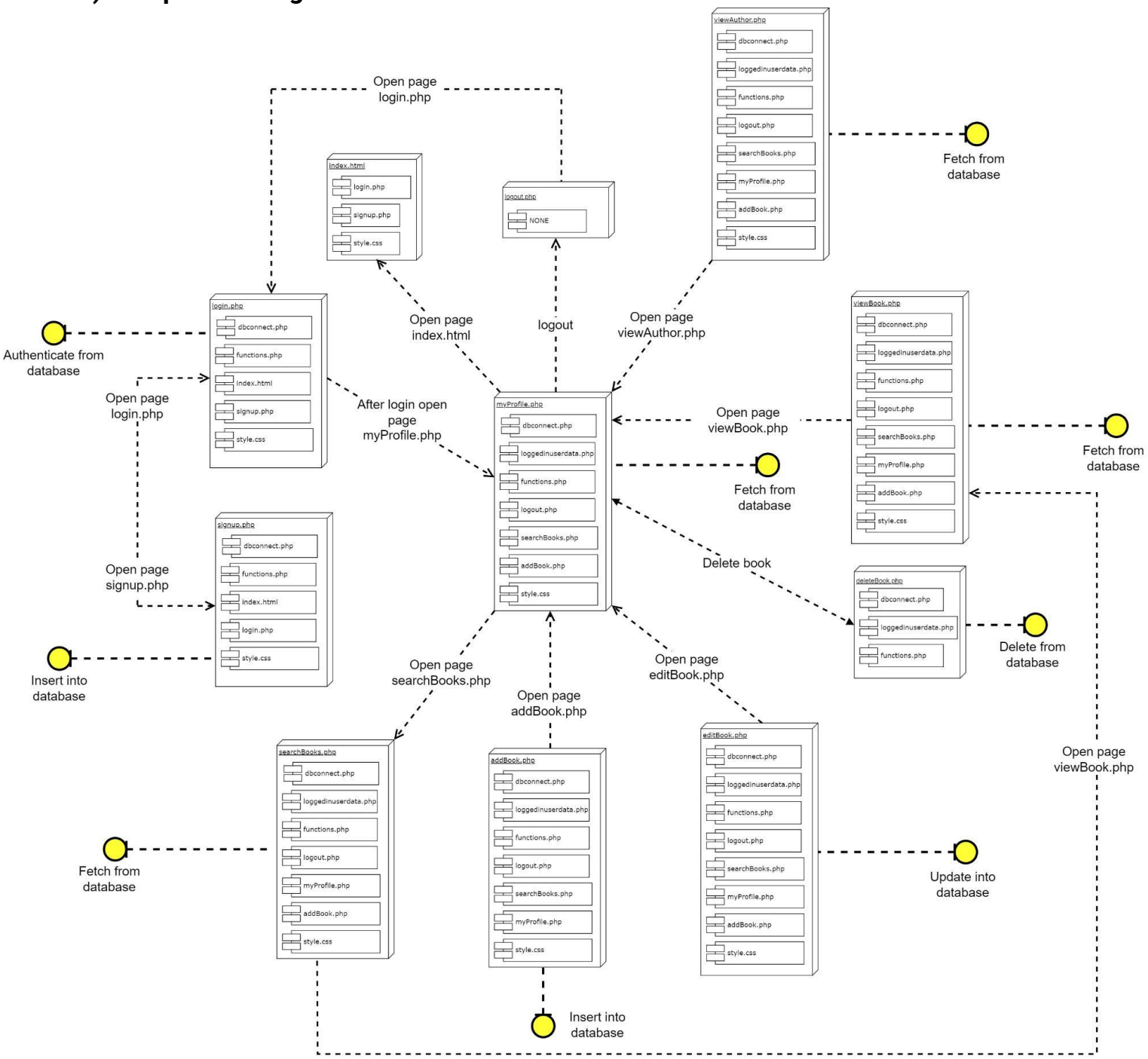
c) Class Diagram



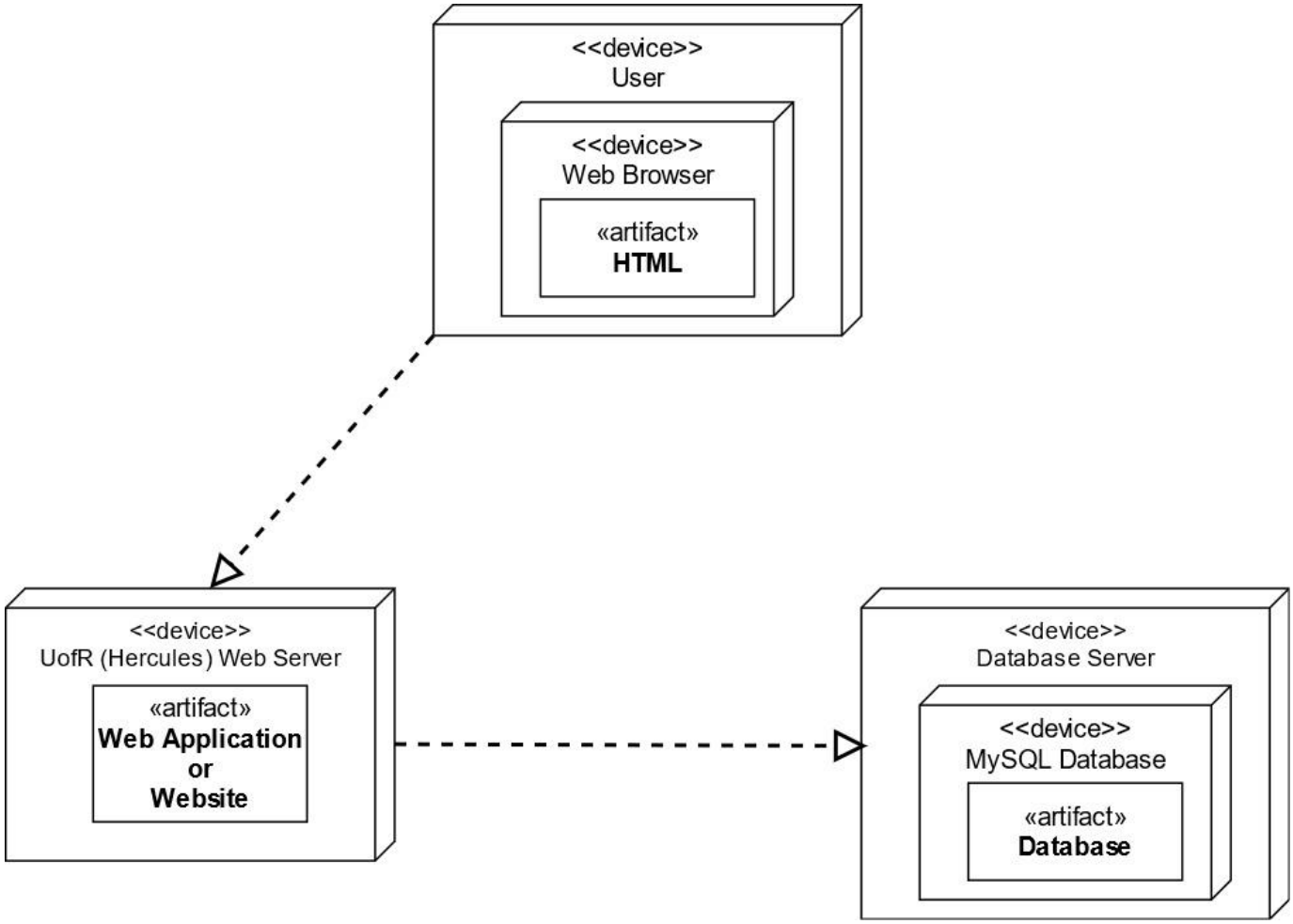

admin
+ email: string
+ password: string
+login()
1
1
adminLogin
errors: array
email: string
password: string
total_rows: int
loginErrorFound: string
adminlogin_FindUser(database, email, password): int
adminlogin_Errors(database, email): string
1
Inherit
*
functions
+ admin_findBookCovers(database, user_id): object
+ admin_deleteBooks(database, user_id): boolean
+ admin_deleteUser(database, user_id): boolean
+ adminlogin_FindUser(database, email, password): int
+ adminlogin_Errors(database, email): string
+ loggedinuserdata_data(database, active_email)
1
1
database
+book_id: int
+bookCoverUploads
+bookTitle: string
+bookPublishDate: string
+bookPageCount: int
+bookISBN: int
+bookSummary: sting
*
*
Inherit
1
deleteUser
+ user_id: int
+ successDeleteAllBookCovers: boolean
+ successDeleteBooks: boolean
+ successDeleteUser: boolean
+ bookCovers: array
admin_findBookCovers(database, user_id): object
admin_deleteBooks(database, user_id): boolean
admin_deleteUser(database, user_id): boolean
*
1
adminPanel
+select()
1

5. Programs:

a) Component Diagram



b) Deployment Diagram



c) System Data Tables Screenshots

```
mysql> SELECT * FROM Citadel_Admins;
+-----+----------+-----------------+-----------+
| aid | username | email           | password  |
+-----+----------+-----------------+-----------+
|   1 | Admin    | admin@gmail.com | Admin@123 |
+-----+----------+-----------------+-----------+
1 row in set (0.00 sec)
```

Citadel_Admins

```
mysql> describe Citadel_Admins;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| aid      | int(11)      | NO   | PRI | NULL    | auto_increment |
| username | varchar(255) | NO   |     | NULL    |                |
| email    | varchar(255) | NO   |     | NULL    |                |
| password | varchar(255) | NO   |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Citadel_Admins (Layout)

```
mysql> describe Citadel_Users;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| uid      | int(11)      | NO   | PRI | NULL    | auto_increment |
| username | varchar(255) | NO   |     | NULL    |                |
| email    | varchar(255) | NO   |     | NULL    |                |
| password | varchar(255) | NO   |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Citadel_Users (Layout)

```
mysql> describe Citadel_Books;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| bid         | int(11)      | NO   | PRI | NULL    | auto_increment |
| uid         | int(11)      | NO   | MUL | NULL    |                |
| title       | varchar(255) | NO   |     | NULL    |                |
| publishDate | date         | NO   |     | NULL    |                |
| pageCount   | int(11)      | NO   |     | NULL    |                |
| isbn        | bigint(20)   | NO   |     | NULL    |                |
| bookSummary | varchar(501) | NO   |     | NULL    |                |
| bookCover   | varchar(255) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

Citadel_Books (Layout)

```
mysql> SELECT * FROM Citadel_Users;
+-----+----------+-----------------+-------------+
| uid | username | email           | password    |
+-----+----------+-----------------+-------------+
|   1 | Yash     | yash@gmail.com  | Yash@123    |
|   2 | Jaydeep  | jaydeep@gmail.com | Jaydeep@123 |
|  12 | Max      | max@gmail.com   | Max@123     |
+-----+----------+-----------------+-------------+
3 rows in set (0.00 sec)
```

Citadel_Users

d) Link of Web-based application

Main Website: http://www2.cs.uregina.ca/~ynp062/Citadel/index.html

Admin Path: http://www2.cs.uregina.ca/~ynp062/Citadel/adminLogin.php

6. Technical documentation:

a) List of programming languages

Front-Back End Programming Languages: PHP 8.0, Javascript

Database: MySQL 5.1.73 (OOP)

Web Server: UofR Titan/Hercules

b) List of reused algorithms and programs

https://regexr.com/3e48o

https://regexr.com/3bfsi

https://www.w3schools.com/howto/howto_css_menu_icon.asp

https://www.youtube.com/watch?v=gXkqy0b4M5g

https://www.w3schools.com/howto/howto_js_mobile_navbar.asp

https://stackoverflow.com/questions/15491894/regex-to-validate-date-formats-dd-mm-yyyy-dd-mm-yyyy-dd-mm-yyyy-dd-mmm-yyyy

c) List of software tools and environments

Front-End Programming:

- ➔ HTML5: This helped us to create basic inputs on our page for searchBook, add book and many others.
- ➔ CSS: This helped us to create the style of every single page to make it filled with animations, fluidity, minimalist and user-friendly websites.
- ➔ Javascript: This helped us to prevent wrong input from users and prevent wrong data from getting into the back end.

Back-End Programming:

- ➔ PHP 8.0: This helped us to write back-end programming and do tasks efficiently.
- ➔ MySQL 5.1.73 (OOP): This helped us to create various database tables for our application for Users, Books, Admins.

Software Tools:

- ➔ Visual Studio Code: This helped us to write code in one of the best environments for web application building. It also helped us to connect to the titan server via port.
- ➔ Regex: This helped us to create rules for input insertion in a proper way.
- ➔ Google Pagespeed: For testing our pages performance.
- ➔ draw.io: For drawing all the diagrams for a project report.

Operating System:

- ➔ Windows 10: For our own efficiency and bug free environment we used Windows 10.

Web Browser:

- ➔ Google Chrome: Most used web browser that will help us develop and show the website in a proper format.

Web Server:

- ➔ UofR Titan/Hercules - University web server to host websites and use databases. (Free)

Color Pallet:

- ➔ Coolors (Color Pallet): Made our own user-friendly color pallet for having the user have a good time on our website. https://coolors.co/3af7f0-94faf0-ffffff-ff6347-3cb371 - Manually made base colors for ourselves but generate other colors afterwards

Fonts/Icons:

- ➔ https://ionic.io/ionicons
- ➔ https://fonts.google.com/

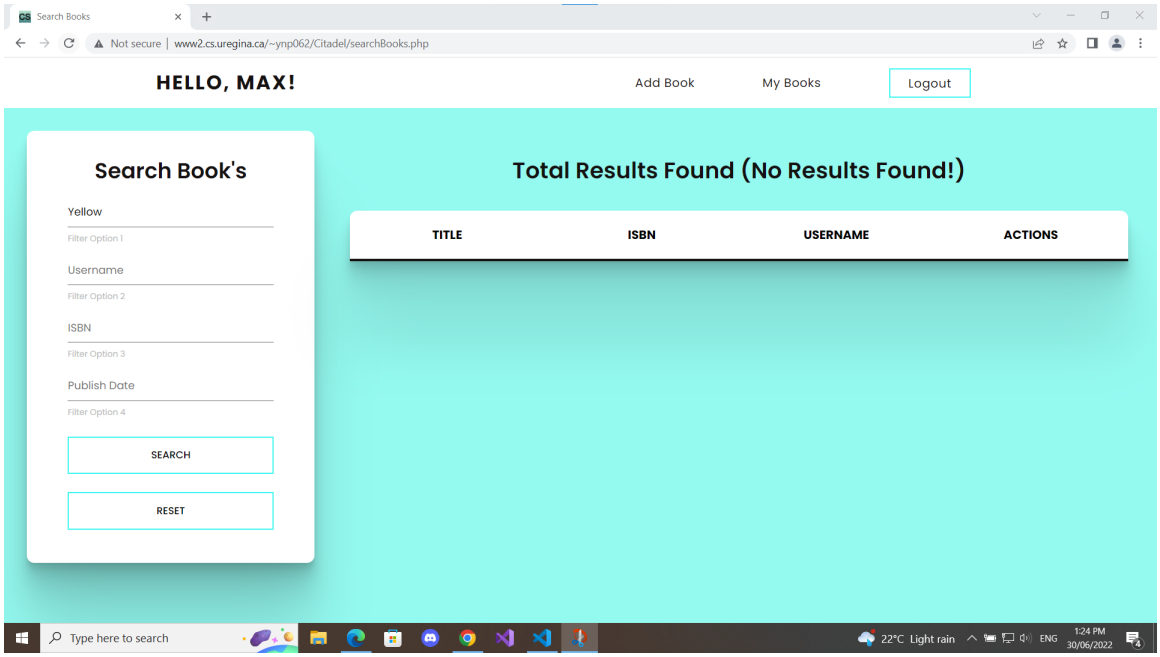
Diagrams/Documentation:

- ➔ Draw.io: For diagrams we used free and easy to use website.
- ➔ Google Docs: For documentation we used free to use document creator. (Easy to use)

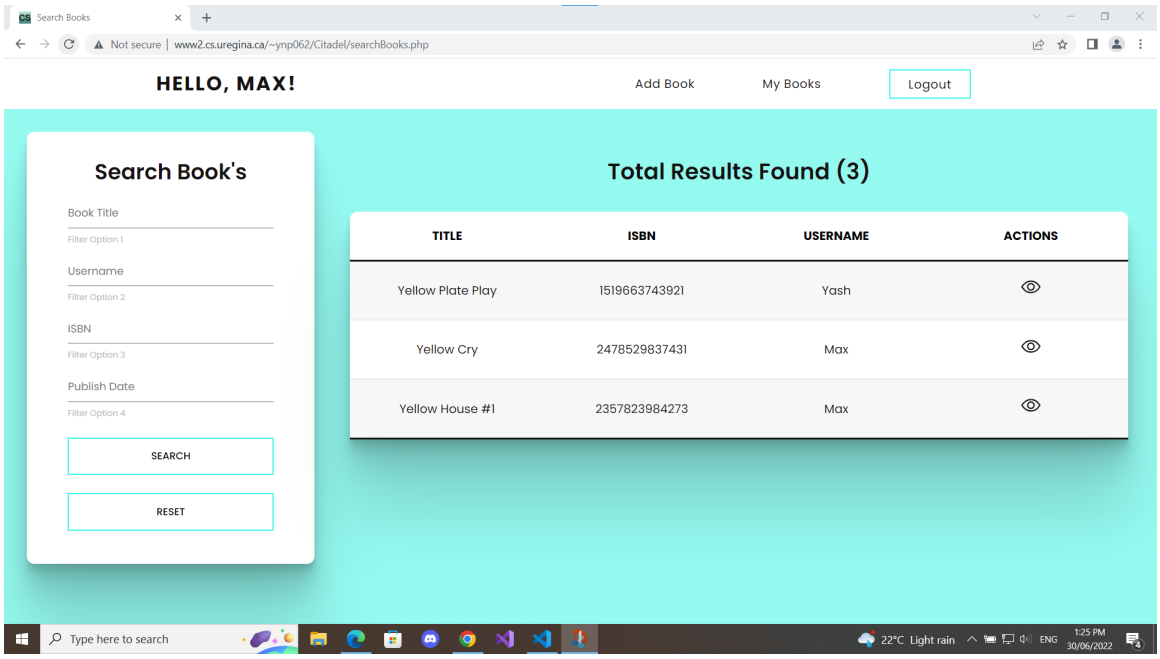
7. Acceptance testing:

a) Functional testing

CASE 1: Test functionality for book search. Users input either of the filters and based on that, the search is done.

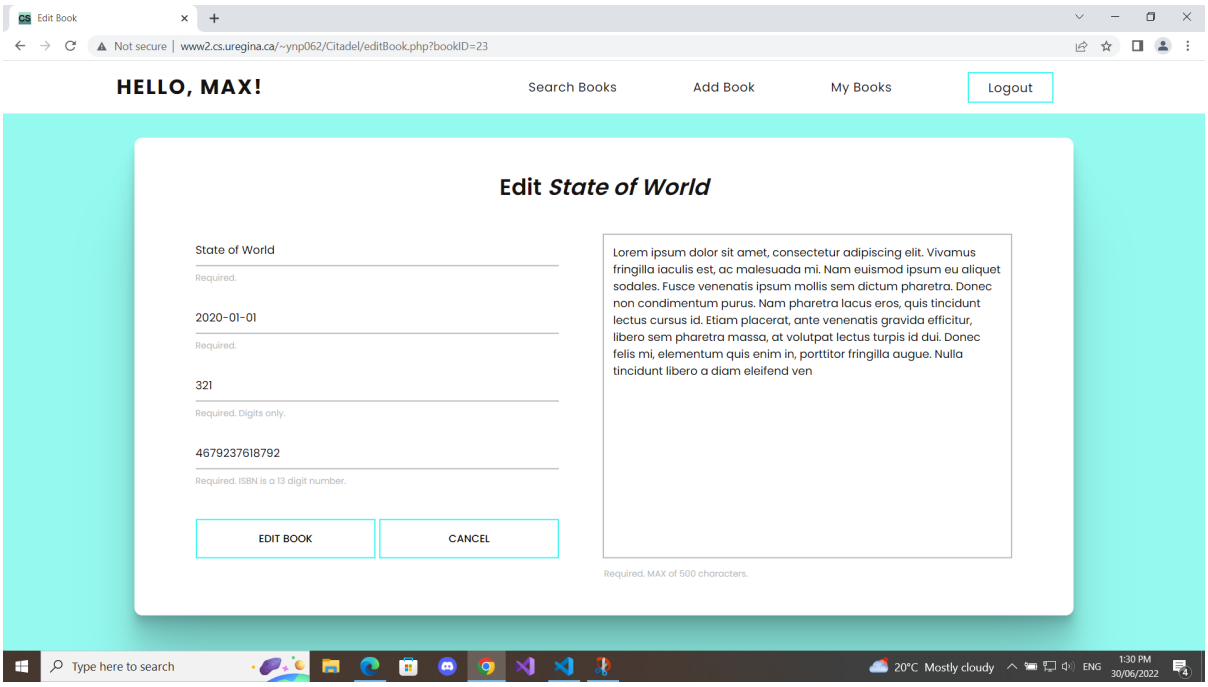


Input is a Book Title that has the word "Yellow".

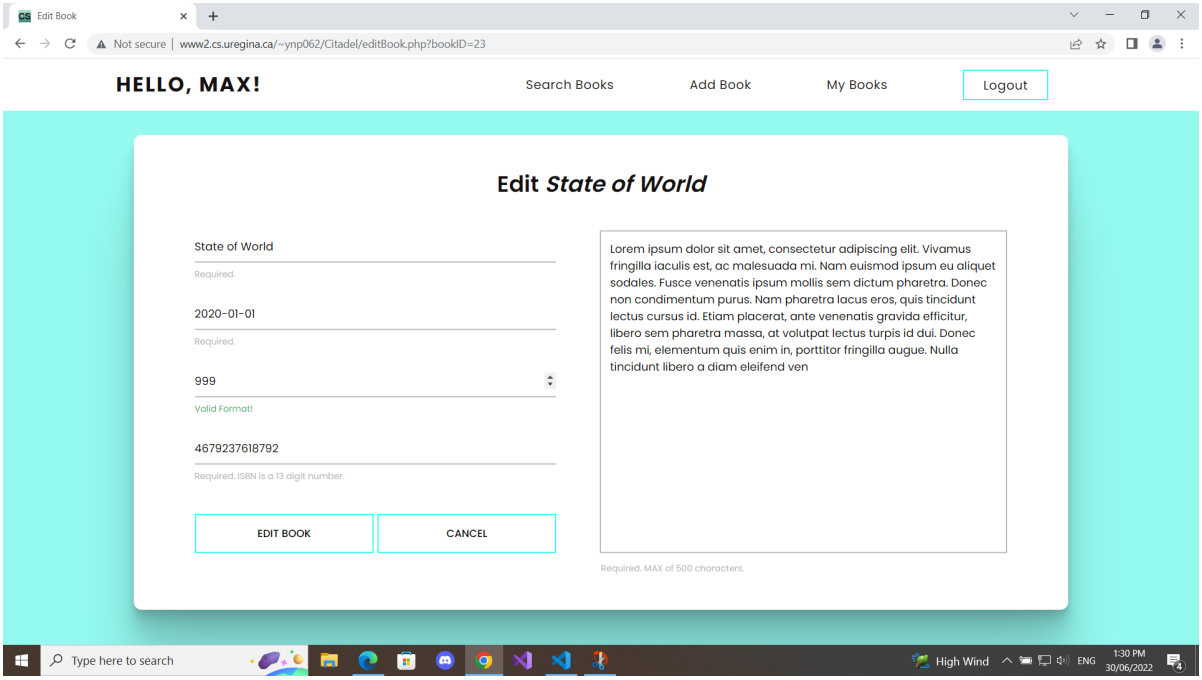


After the user clicks on the search button, search is done and the output is as follows.

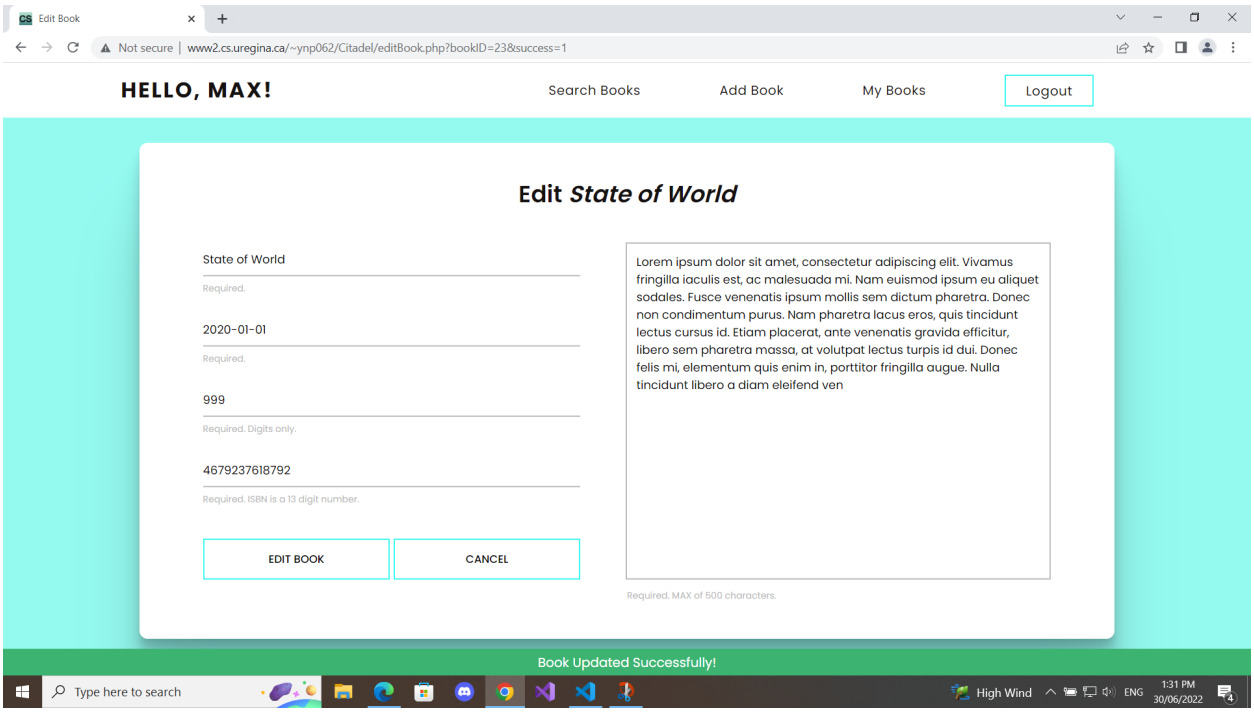
CASE 2: Test functionality for book edit. Users can edit books which belong to them.



Before editing the book Page Count is "321".

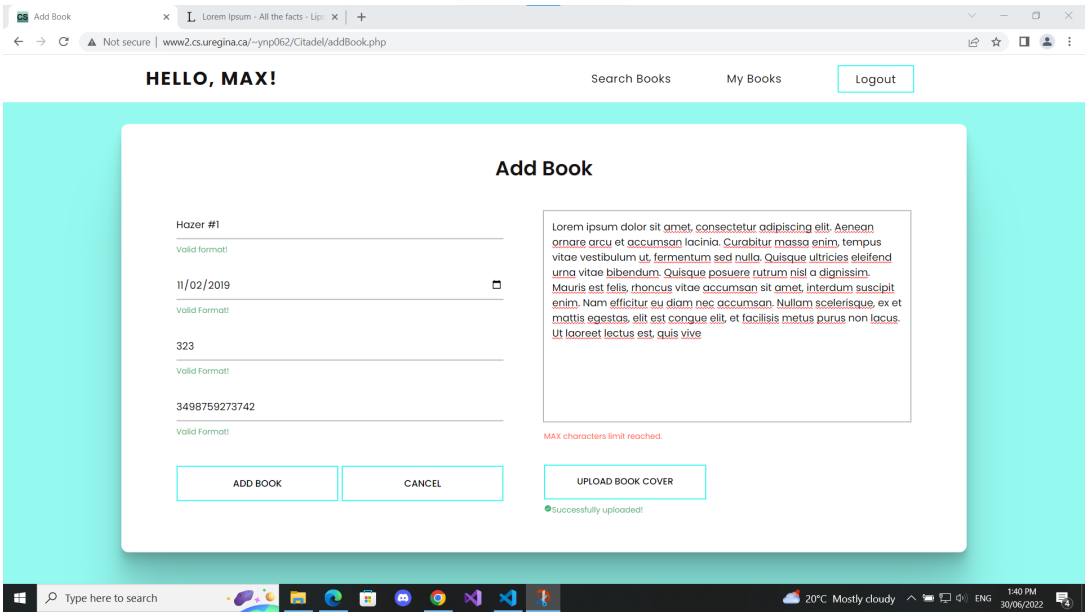


Input for Page Count for editing book is "999".

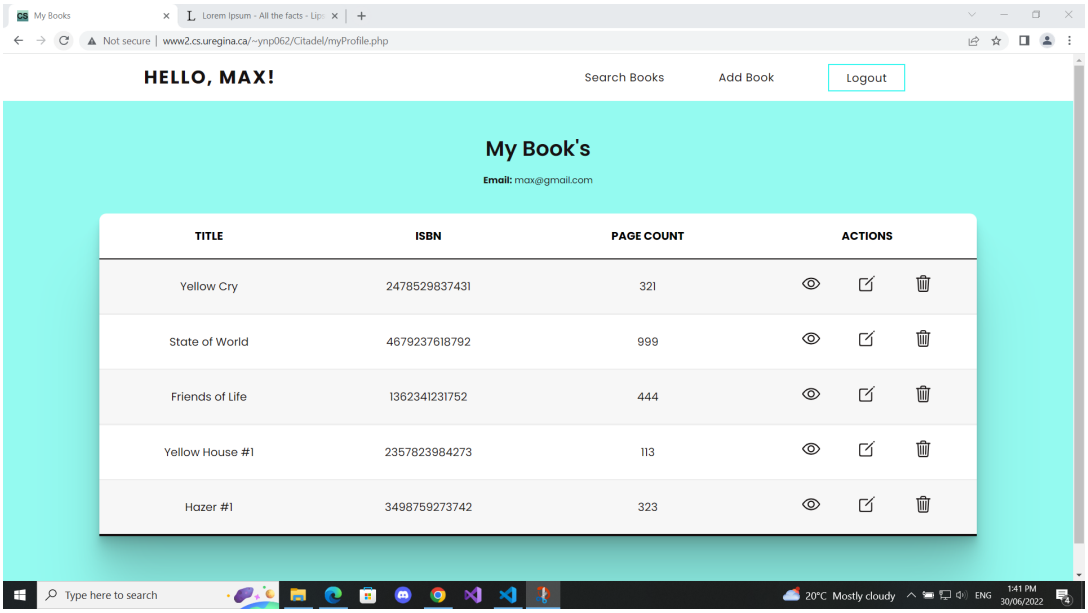


After the user clicks on the Edit Book button, the user is given feedback that his book has been edited successfully (Best Case Scenario).

CASE 3: Test functionality for Adding a Book. User can add books anytime (Must have unique ISBN).

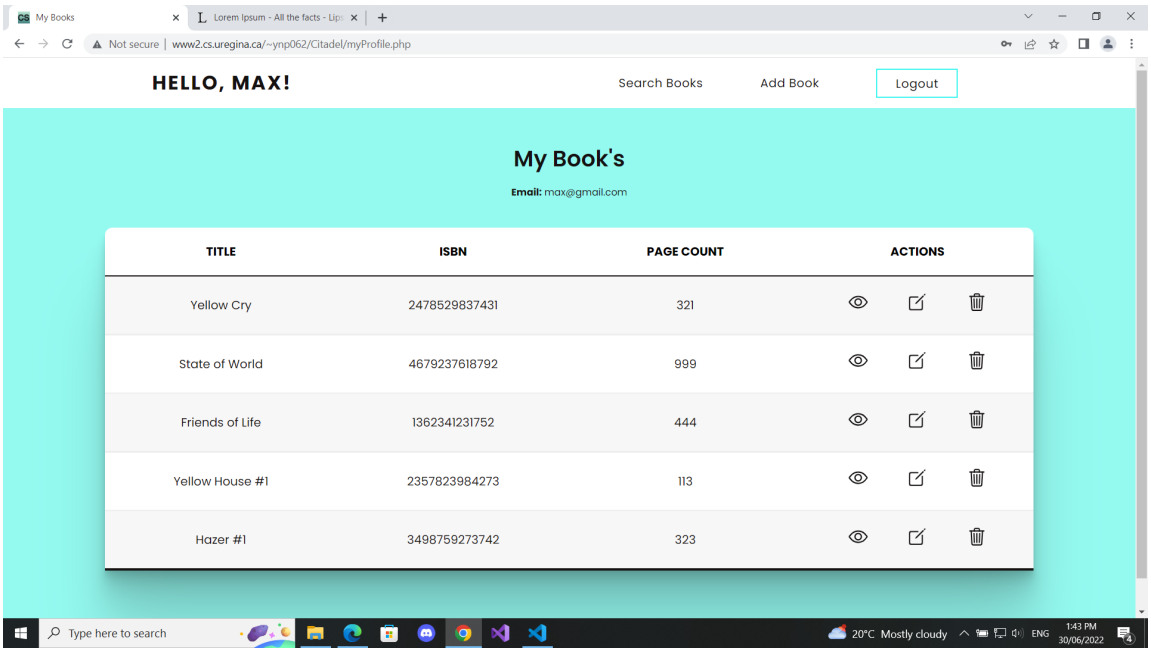


Input is book details and a book cover.

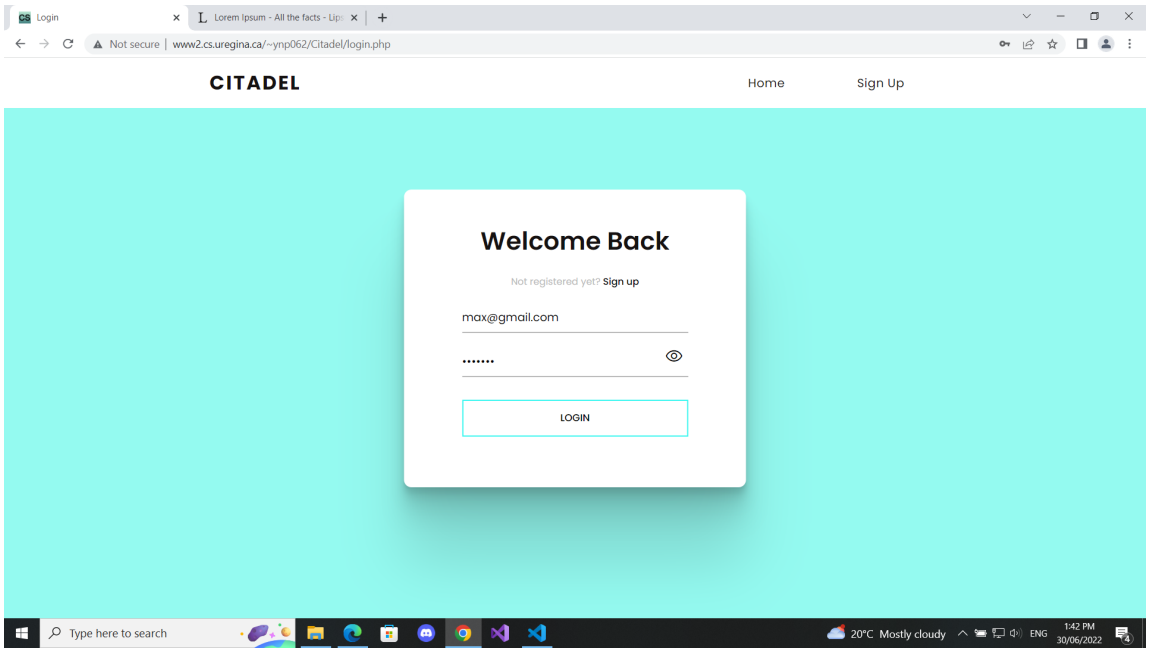


Output is when the user gets back to his profile when the book is successfully added where it shows the user's profile.

CASE 4: Test login functionality.



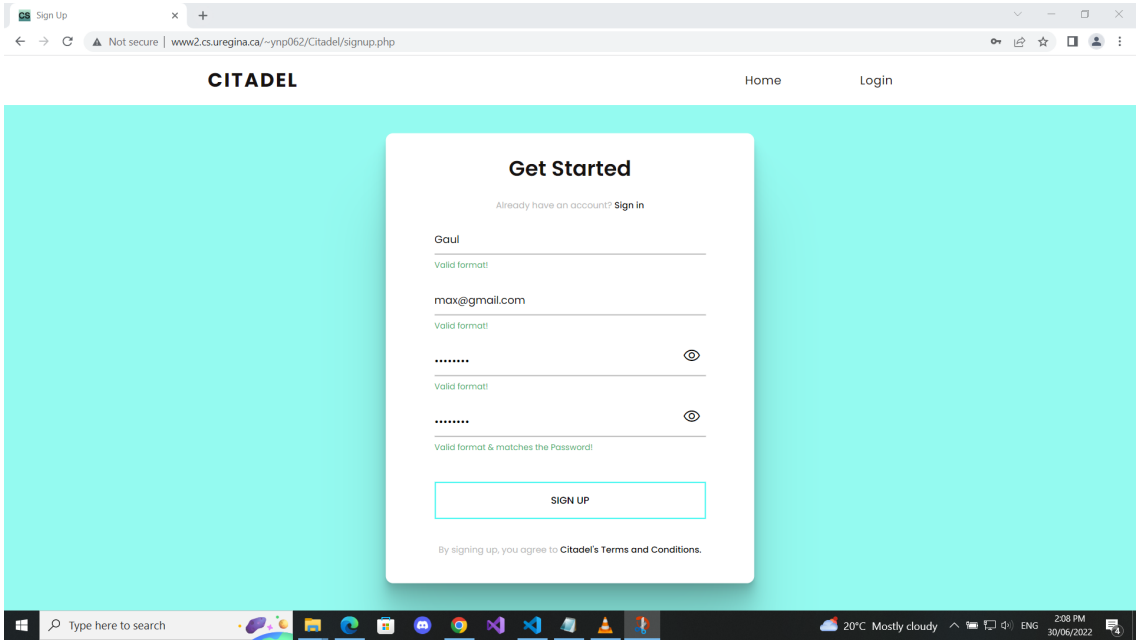
Input is the user's credentials (Email, Password).



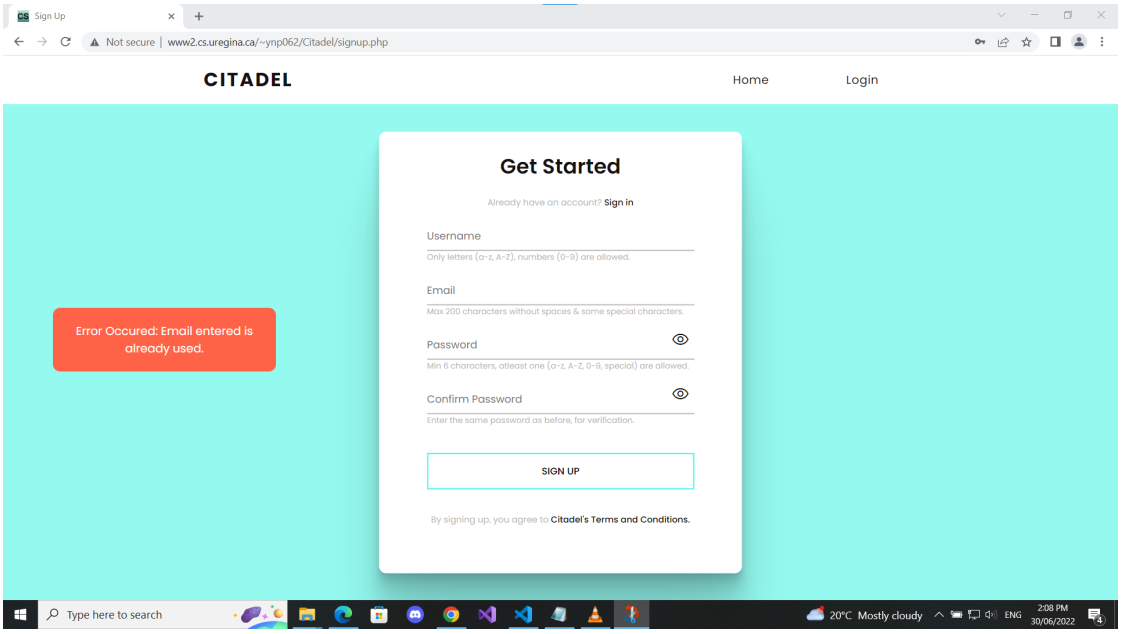
After success, the user gets taken to his profile.

b) Robustness testing

CASE 1: Test signup robustness

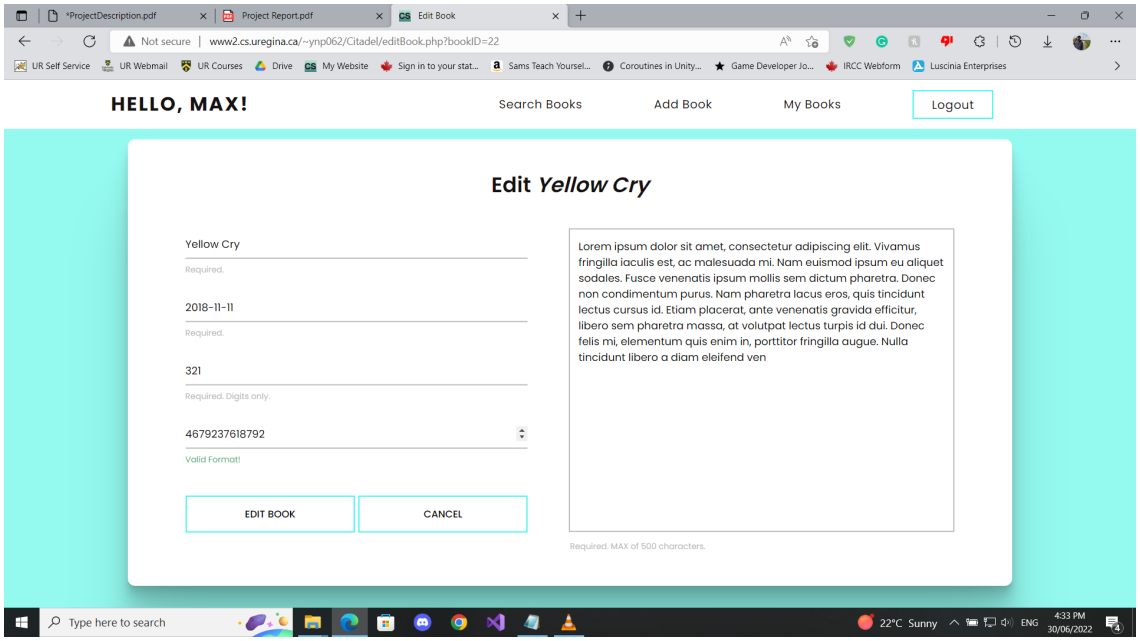


Input is user signup information.

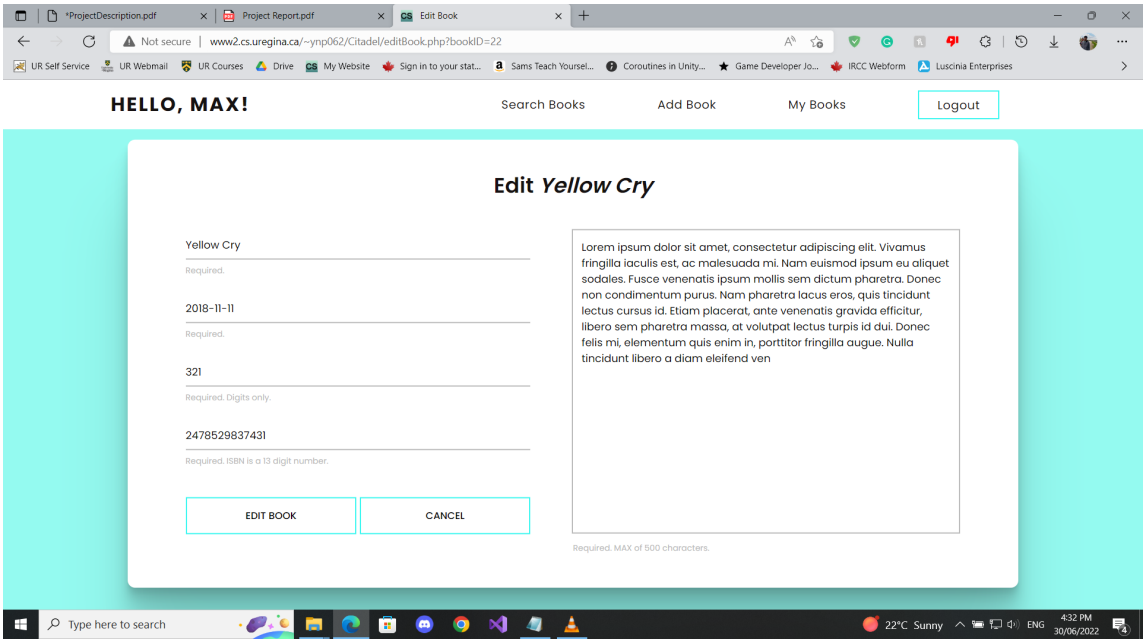


If anything is missing/empty then an error will be shown which is empty. If email/username exists in the database then an error will show that the user already exists.

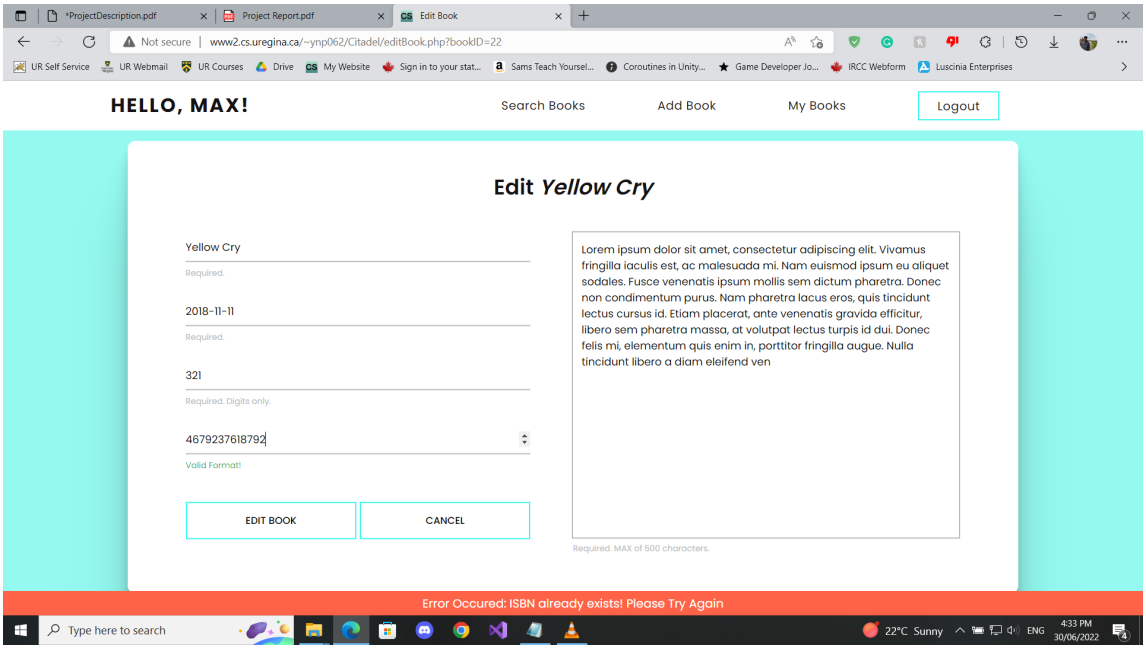
CASE 2: Edit book robustness.



Before editing a book the information related to the book is shown in the textboxes.

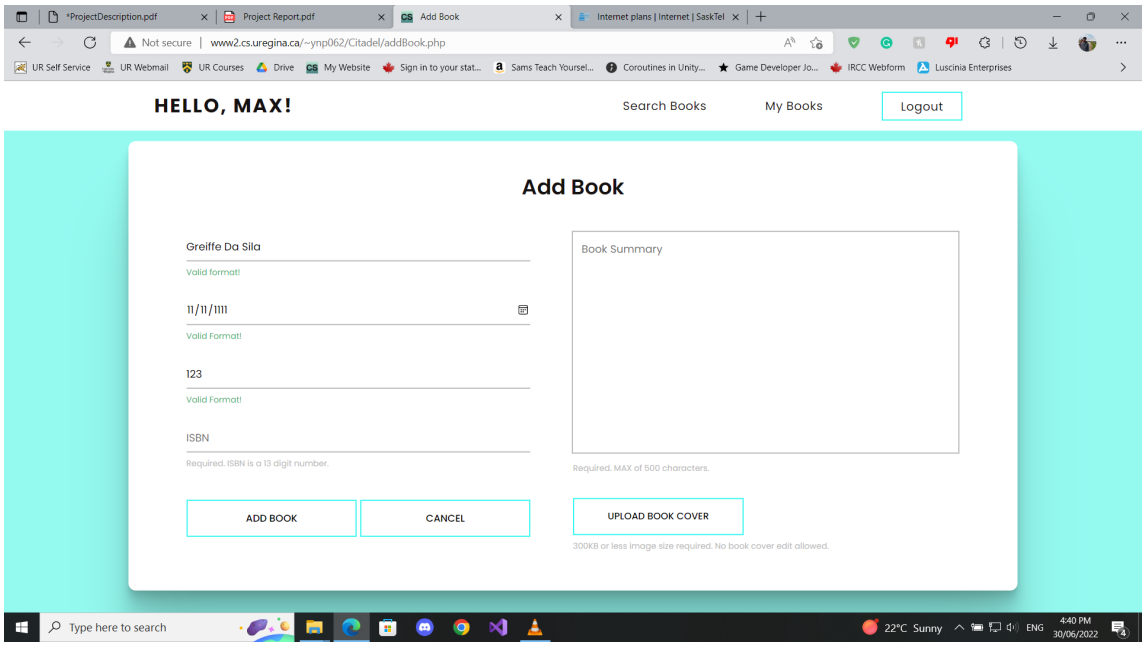


Input for ISBN change. (the given ISBN: 4679... is already in the system)

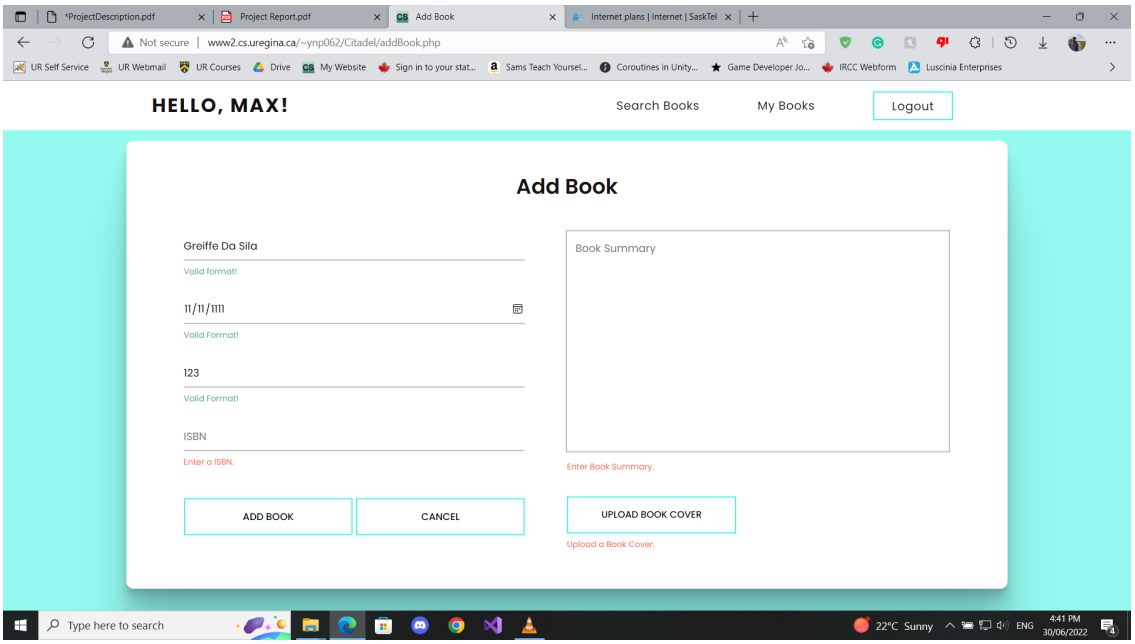


After use, clicks on edit book button system checks first wheather ISBN exists in the database or not and if there exists then error is shown. If anything is empty, errors are shown respectively.

CASE 3: Test add book robustness

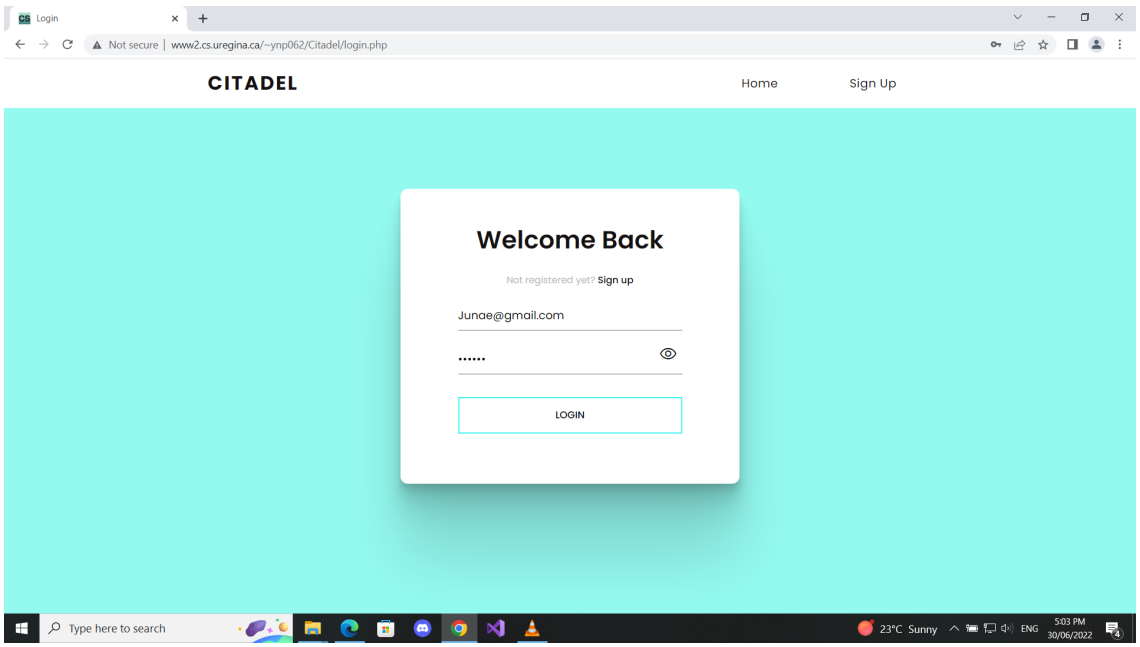


Input is book details and a book cover.

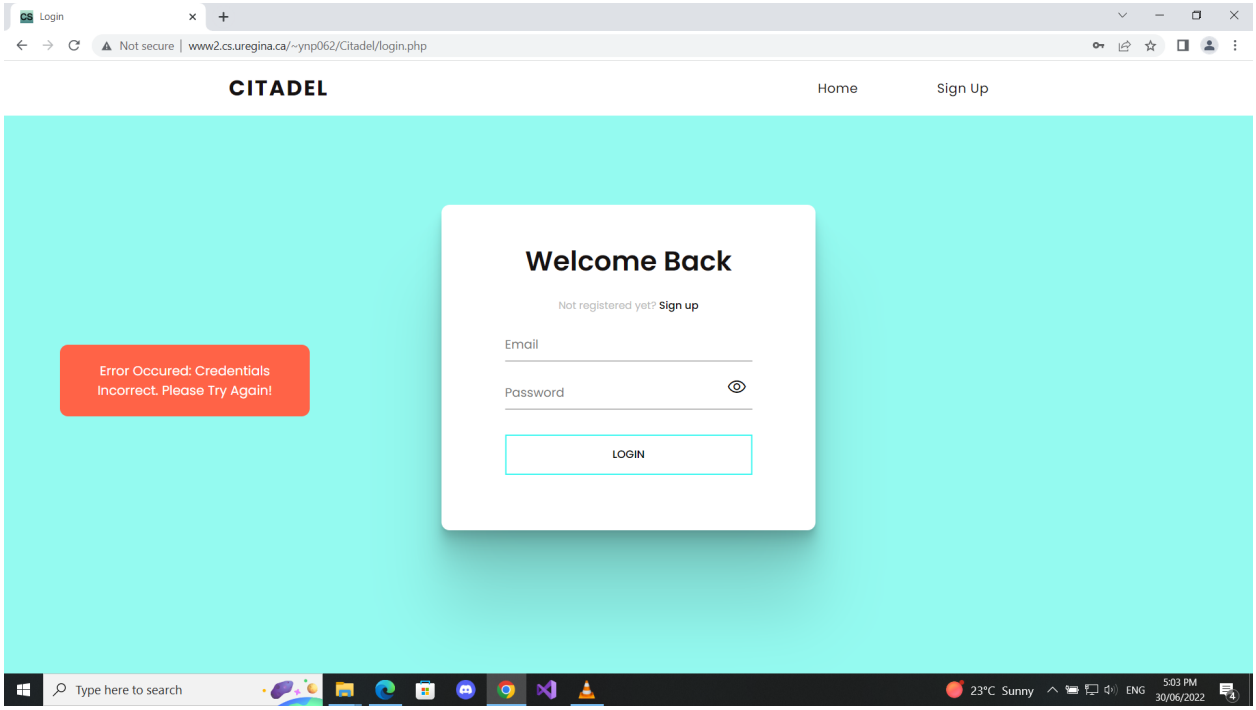


If the book cover is oversize then an error will be shown that the book is more than 300kb in size. If ISBN exists then an error will be shown because ISBN must be unique in the database.

CASE 4: Test login robustness



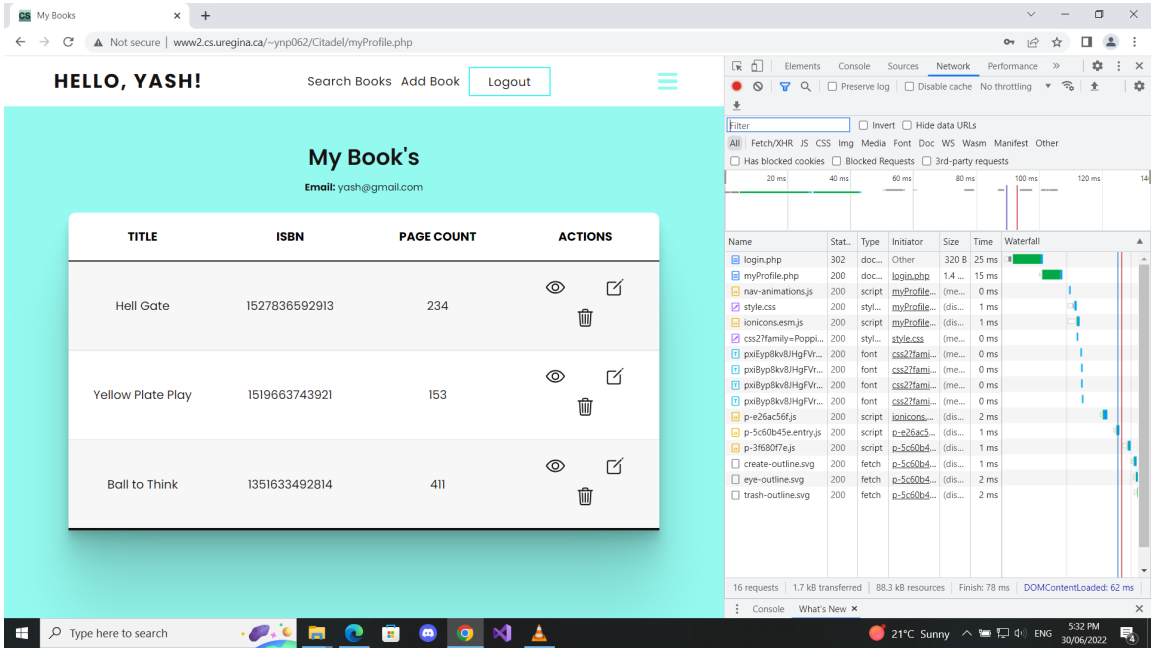
Input is user credentials (Email, Password).



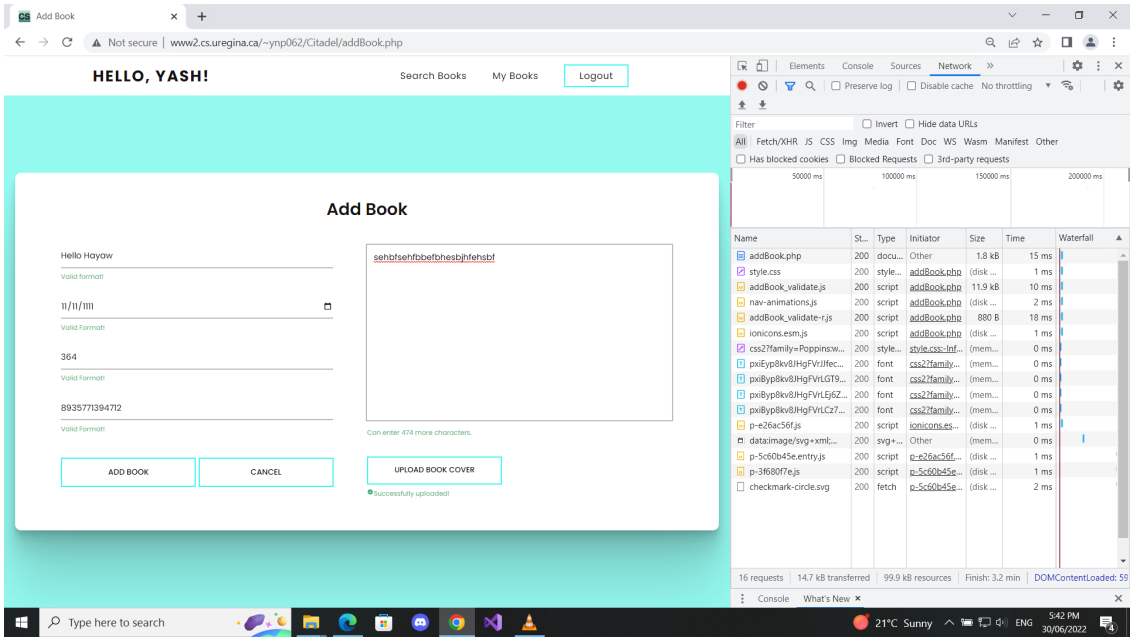
If email exists but not the password then error is shown regarding "password". If email doesn't exist then an error is shown regarding "credentials" being incorrect.

c) Time-efficiency testing

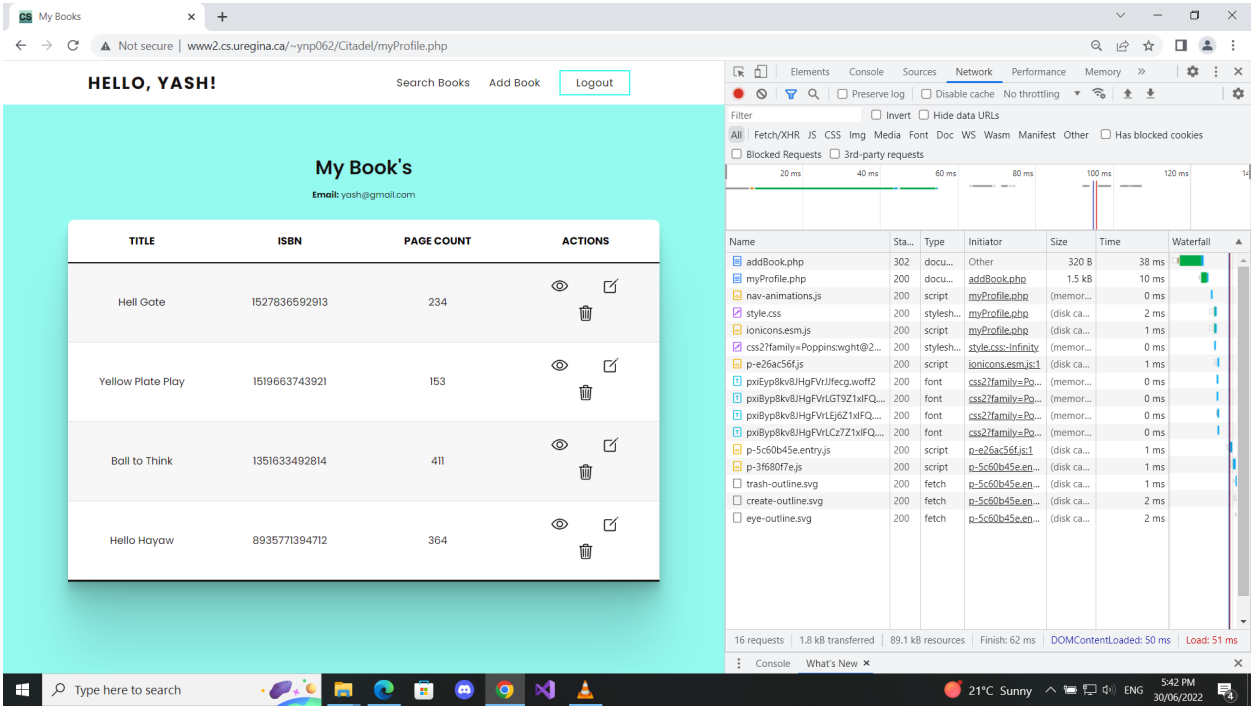
Manual Testing: (Using Google Chrome Network)



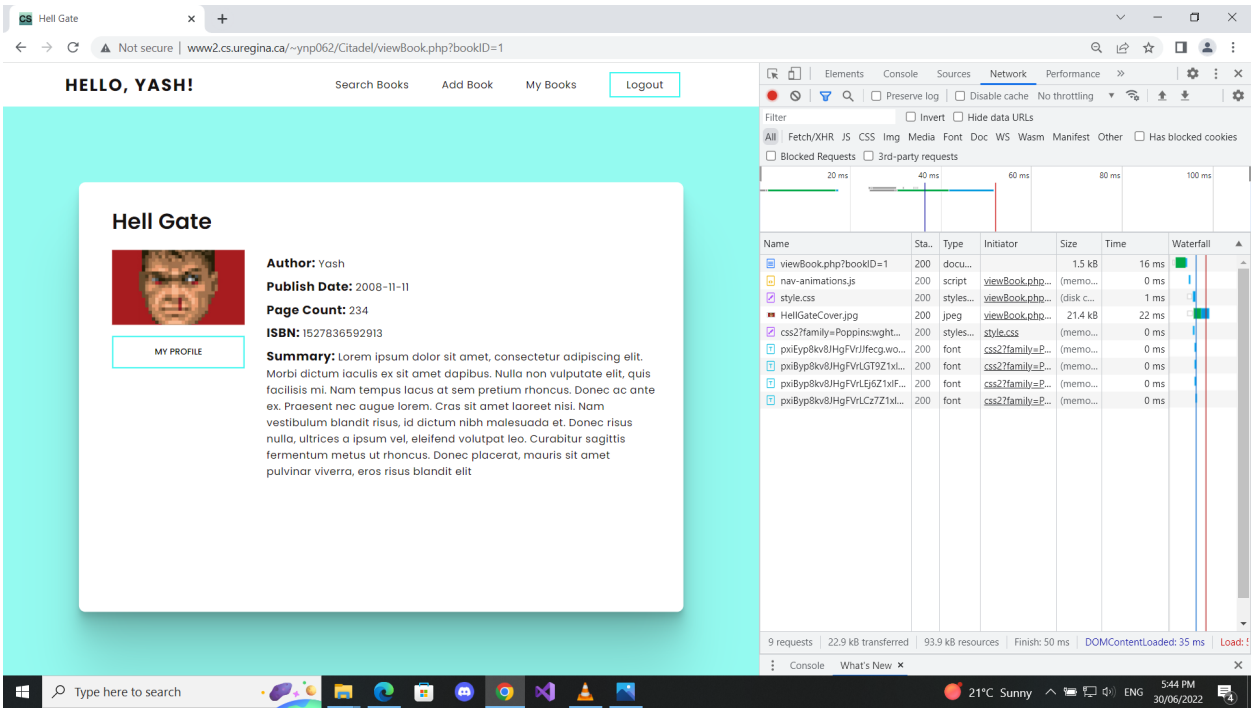
myProfile.php (After logging in from login.php)



addBook.php (From myProfile.php)

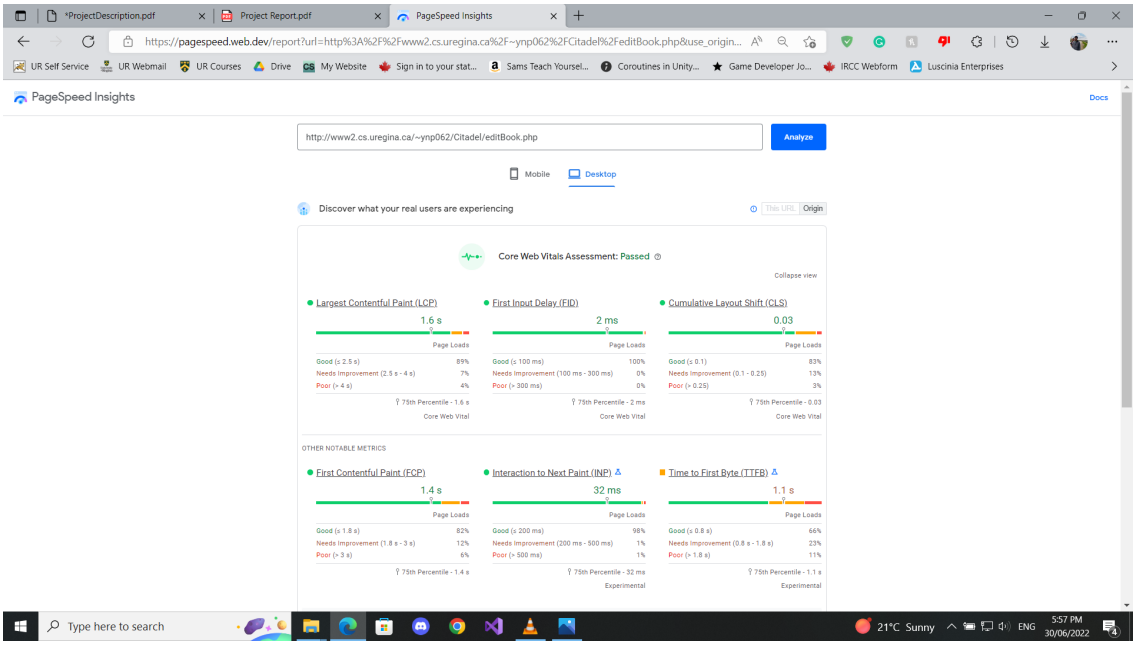


myProfile.php (Back after adding book)

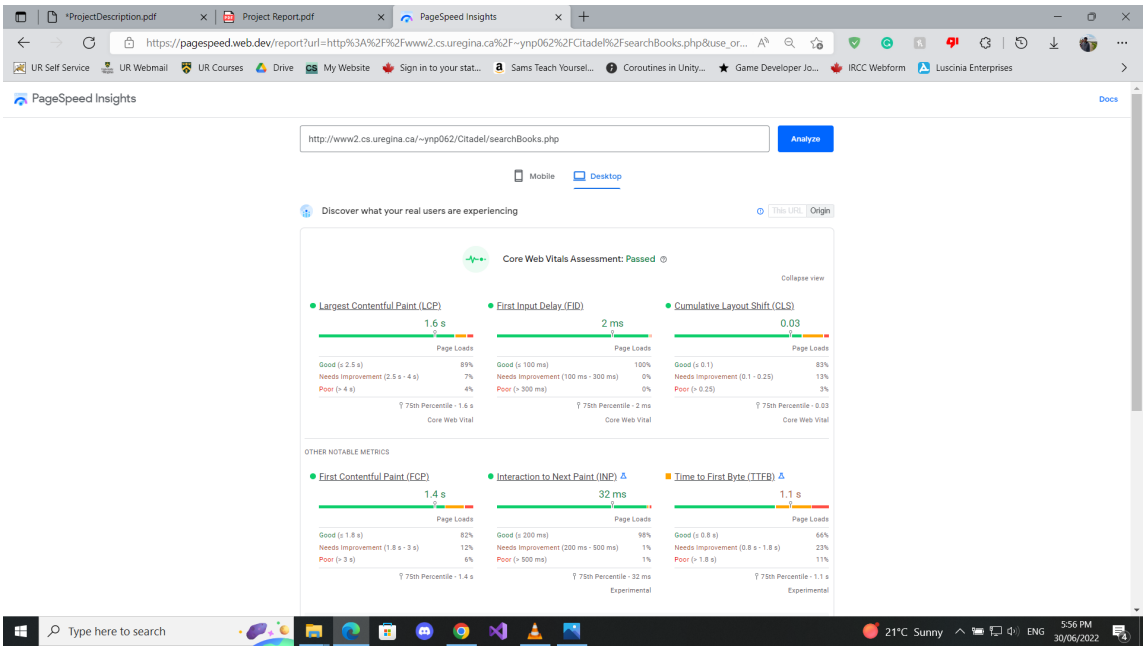


viewBook.php (From myProfile.php)

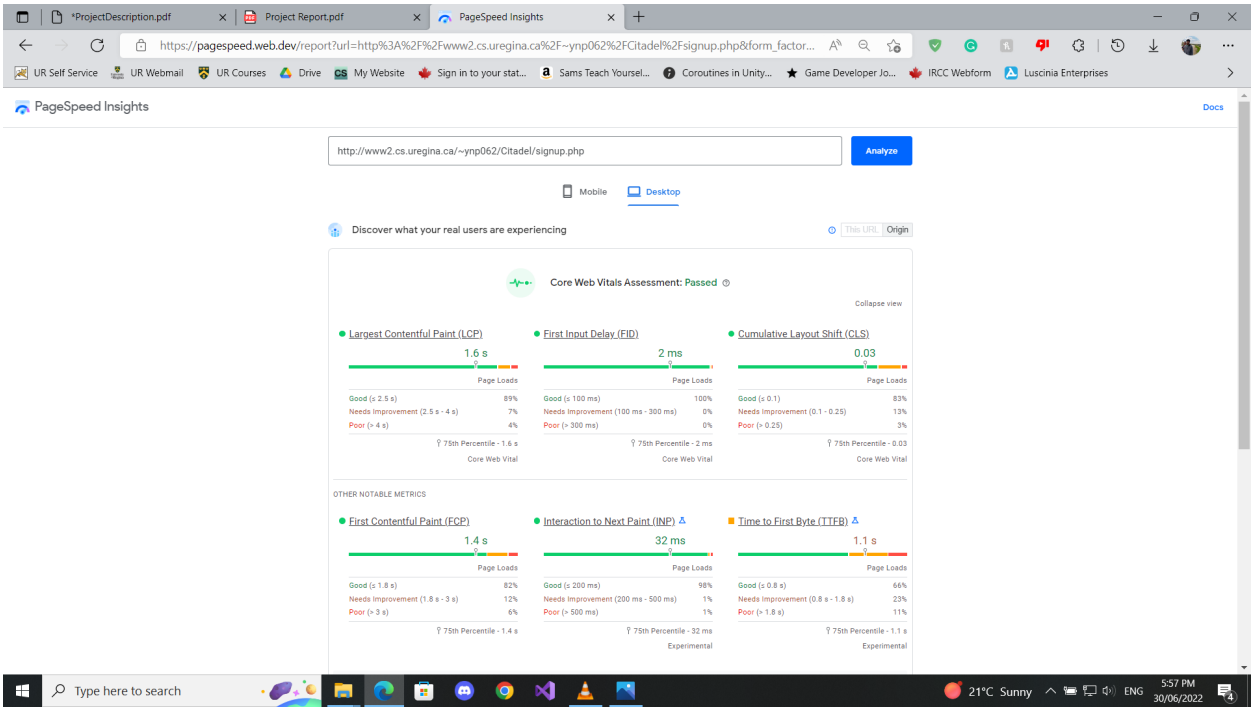
Automatic Testing: (Using Google Pagespeed)



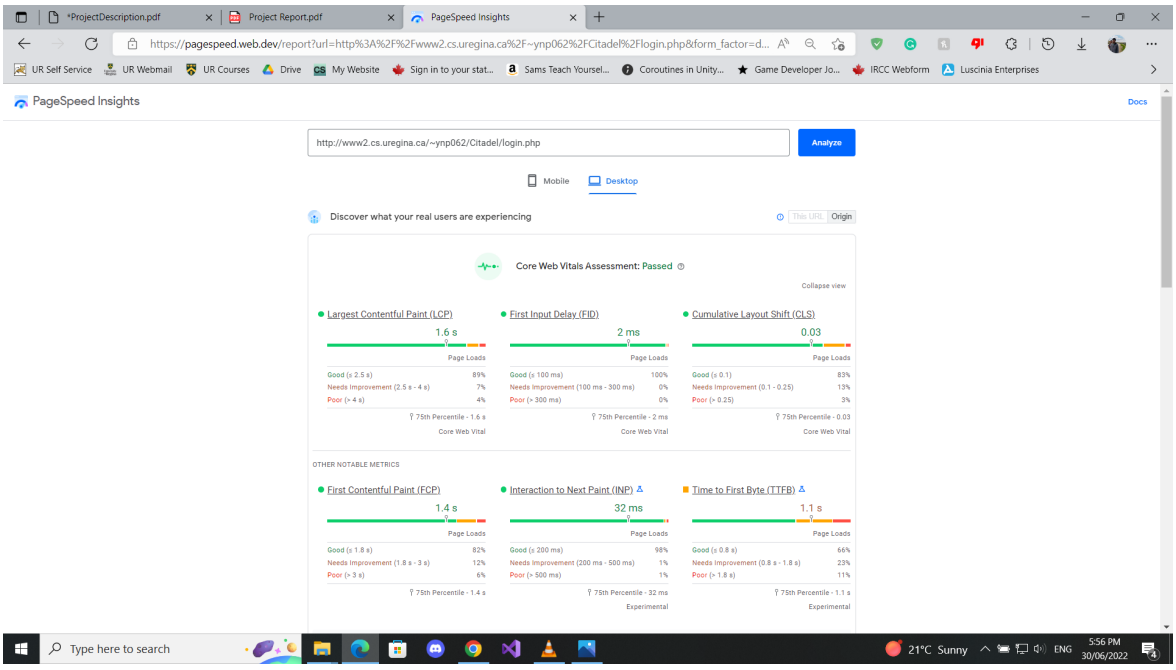
editBook.php



searchBooks.php



signup.php



login.php

References

Online: https://www.youtube.com/watch?v=mtZdybMV4Bw

UML specification Document: https://www.omg.org/spec/UML/2.5.1/PDF

Online Website:
https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1#:~:text=%2DMVC%20is%20an%20architectural%20pattern,the%20view%20whenever%20data%20changes.

Online Comparison Website: https://www.bookfinder.com/

Online Comparison Website: https://themillions.com/find-books

UR Courses PDF: Design Patterns Elements of Reusable Object-Oriented Software