Lab Assignment 4: Project Report

Title of the Project: Online Bookstore Web Application

Group Number: 25

Team Members:
Sherry Ahuja - 251383252
Prashansa Arya - 251386077
Mathangi Chandrasekaran - 251387854

Date: 07/12/2023

1. Project Description:

1.1 Overview of the Project:

Bookstore application is an e-commerce website which allows users/customers to view a list of books, search for a book with its id and name, add books to wish list and order a wide range of books. Our application includes various components such as Homepage, book details, login, signup, cart and payment web pages to provide smooth user experience. The website uses modern web application framework for front-end and back-end providing user friendly experience to the application users.

1.2 Data Sources and Types:

The bookstore application is built using various data types and sources to manage unique information of users, books and the orders that were made through the web site. Below is the primary list of types and sources employed in our project:

1. Database – It is a structured collection of data that allows efficient storage, retrieval, manipulation, allowing application to interact with the stored information. List below is a collection of database tables used in our development.

    - **Users**


```
_id: ObjectId('656e59ce4a220c2700e2feb3')
username: "Mathangi C"
email: "cmathangi21@gmail.com"
password: "$2b$10$awFUQCZW5anSUdvat0NEjehJCBhjxss6Snwj7GEWrS.1xbdG7xXB."
isActive: true
role: "user"
createdAt: 2023-12-04T22:59:26.296+00:00
updatedAt: 2023-12-04T22:59:26.296+00:00
__v: 0
```


    - **Cart**

QUERY RESULTS: 1-2 OF 2


```
_id: ObjectId('656e02101c2adf823318a266')
userId: "test@gmail.com"
▸ books: Array (2)
createdAt: 2023-12-04T16:45:04.698+00:00
updatedAt: 2023-12-06T13:48:05.384+00:00
__v: 19
```


    - **Wishlist**

QUERY RESULTS: 1-10 OF 10


```
_id: ObjectId('656d44d6c831e9e4a7ac525b')
userId: "test@gmail.com"
name: "test collection 1"
description: "this is the test description for this collection"
▸ books: Array (2)
createdAt: 2023-12-04T03:17:42.866+00:00
updatedAt: 2023-12-06T14:42:46.337+00:00
__v: 0
isPrivate: true
```


- **Reviews**

QUERY RESULTS: 1-10 OF 10


```
_id: ObjectId('656d6fb84511bffbbcd2f3c5')
userId: "test@gmail.com"
booklistId: ObjectId('656d44d6c831e9e4a7ac525b')
review: "this is good"
createdAt: 2023-12-04T06:20:40.514+00:00
updatedAt: 2023-12-05T00:21:08.035+00:00
__v: 0
visibility: true
```


2. External APIs – It provides a standard way for web application to connect with external system/service to retrieve data and extend the functionality of their web application by integration additional functionality offered by the external system.

   Below is the list of external APIs, with which our application is connecting to provide added functionality.

   https://www.googleapis.com/books/v1/volumes?q=money&maxResults=25

Listed below are the API endpoints of our bookstore application.

http://localhost:3000/wishlist/getpublicbooklists

http://localhost:3000/wishlist/getuserbooklists/test@gmail.com

http://localhost:3000/wishlist/update/booklistId

http://localhost:3000/users/admin

http://localhost:3000/users/admin/:userid/toggle-active

http://localhost:3000/users/admin/:userid/toggle-role

http://localhost:3000/cart/getusercart/test@gmail.com

http://localhost:3000/cart/create

http://localhost:3000/cart/update

http://localhost:3000/cart/update

http://localhost:3000/wishlist/create

http://localhost:3000/wishlist/admin/togglevisibility/:reviewId

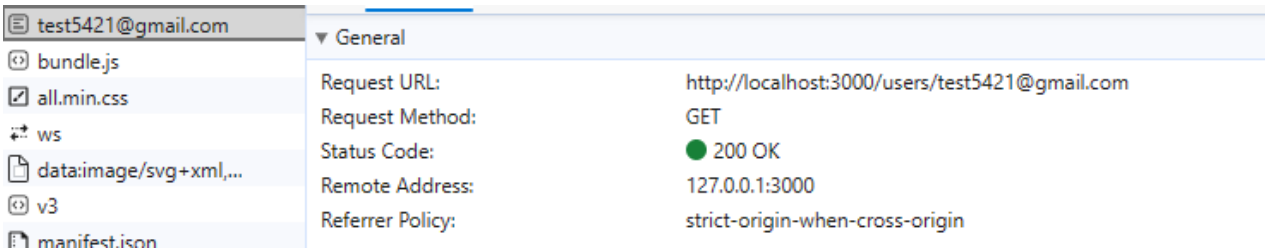
http://localhost:3000/wishlist/addreview

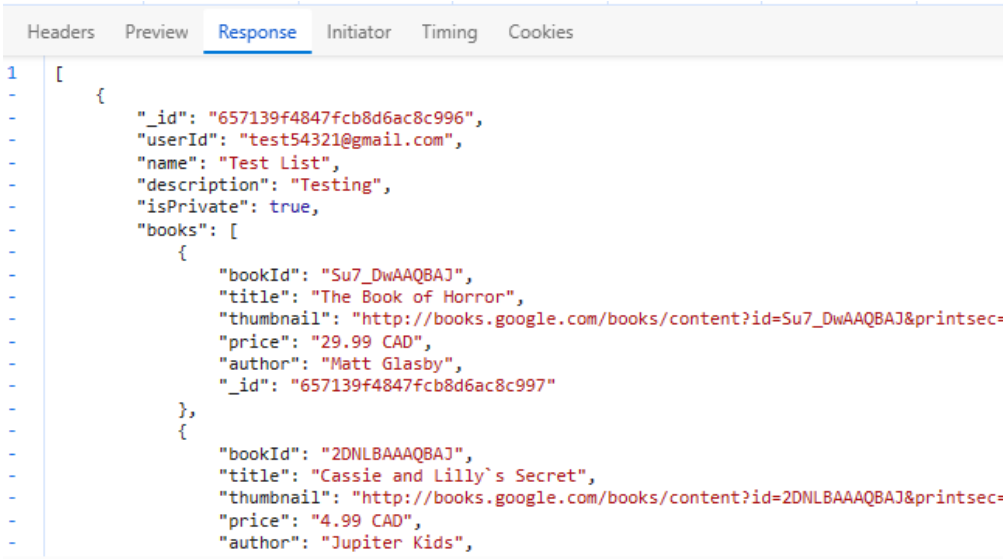
http://localhost:3000/auth/signup

http://localhost:3000/auth/login

http://localhost:3000/auth/google

3. Data Structure – It determines how the data is organized, stored, and accessed within applications. Below is a sample of JSON response from our application.

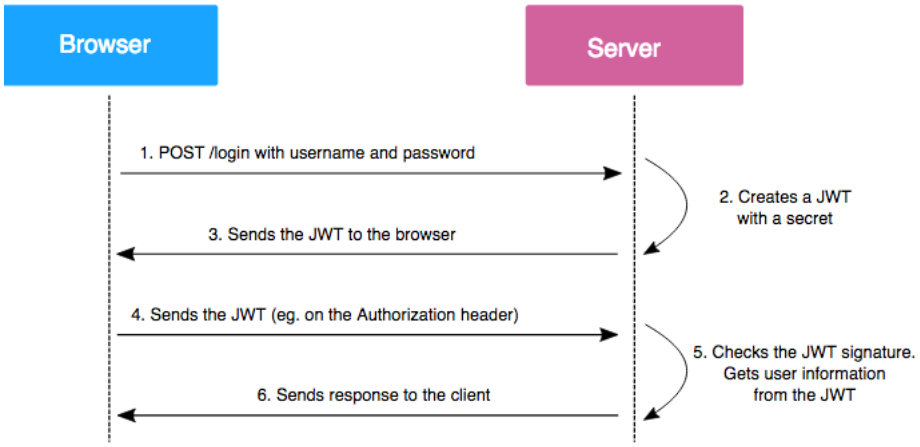


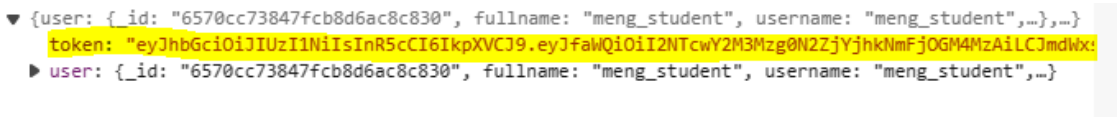


4. Static Files - It includes image files which are used to build visually appealing user interface design for the site users.

5. Authentication Tokens – It is used to provide user authentical for secure communication, one such tool used in our web application is JSON Web

Tokens (JWT). Below highlighted screenshot is an example of JWT token generated for a test user for a one particular session.





6. Version Control System – It includes the usage of Git repository for building web applications to track changes in code and configuration. Listed below are branches used for collaborating our application.

    - Frontend-init
    - Backend-init
    - Components
    - Screens
    - Routes
    - Middleware
    - Controller

7. Testing – It ensures that each component in the web application works as intended including its behavior within the application. Below listed are the testing credentials which are a combination of email id and password used to test the functionality of our application.

    - **Admin:**

      Email id: admin@gmail.com

      Password: admin123

    - **Authenticated user:**

Email id: test54321@gmail.com

Password: qwerty

1.3 Frameworks and Libraries:

Framework and Libraries are essential tools that offers prewritten code and reusable components, which significantly increase the development process and serve as a foundation for building web application with smooth user experience.

Various libraries and frameworks that were implemented for building the bookstore application are listed below.

Frontend Framework and Libraries - They provide tools to streamline the process of building user interfaces and enhance the functionality of web applications by providing reusable pre-built components, templates, and utilities us to create responsive and interactive user interfaces.

- React JS
- Bootstrap
- React Router
- Axios

Backend Framework and Libraries - They are server-side framework that provides tools and libraries that can automate certain aspects of development thus reducing the time spent on basic tasks. They also help in providing similar performance on all devices regardless of the size and network speed, resulting in efficient scaling of the web application based on the future traffic. In addition, it also provides numerous components and features to ensure the security of the web application.

- ***Express.js:*** A web application framework for Node.js.
- ***Cors***: A middleware for handling Cross-Origin Resource Sharing.
- ***Passport***.js: An authentication middleware for Node.js.
- ***Mongoose***: An ODM (Object Data Modeling) library for MongoDB and Node.js.
- ***Express-session***: A middleware for session management in Express.
- ***Body-parser***: A middleware to parse incoming request bodies.
- ***Nodemailer***: A module for sending emails from Node.js applications.
- ***JWT (JSON Web Token):*** A library for generating and verifying JSON web tokens.
- ***Stripe***: A payment processing library for handling online payments.

- ***Bcrypt***: A library for hashing passwords.
- ***dotenv***: A zero-dependency module for loading environment variables.
- ***Express Router:*** A module for creating modular, mountable route handlers.
- ***bcrypt***: A library for hashing passwords.
- ***jsonwebtoken***: A library for creating and verifying JSON web tokens.

Testing framework and libraries: The below listed are the testing framework and library used for writing test cases for our web application. To execute the tests, we ran the Mocha test runner which provided us the results whether the test case passed or failed. For each test case, assertions are performed using Chai library to test various aspects of our web application such as the API endpoints, user interface and business logic.

- Mocha
- Chai

1.4 Roles of Group Members:

Below table lists the roles and responsibility of each team member in our project

Student Name	Email Address	Framework/Library Exp.	Third-party tools or services Exp.	Responsibilities in this project
Prashanna Arya - 251386077	parya6@uwo.ca	React, HTML, CSS, JS, Bootstrap	Postman, GITHUB, Putty, AWS	-Website Design and Layout using wireframes -Front-End Development -Testing and validation -Cross device compatibility -Git and Deployment - RESTful Web API Development
Mathangi Chandrasekaran - 251387854	mchand53@uwo.ca	React, HTML, CSS JS, Bootstrap	GITHUB, Putty, AWS	- Website Design and Layout using wireframes - Front-End Development - Integration with backend apis - Database management using MongoDB - Git and Deployment - Front end Validation
Sherry Ahuja - 251383252	sahuja49@uwo.ca	Node.js, Express js, po	MongoDB, JWT, Google API, Firebase, POSTMAN, GITHUB, Putty, AWS	- Website Design and Layout using wireframes - Database design - RESTful Web API Development and Testing - User Authentication Implementation - Database management using MongoDB - Git and Deployment - Back-end Validation - Integration with external APIS - Checkout and payment

1.5 Challenges Faced and Solutions:

Adding External Authentication: Implementing authentication through third party services such as Google sign in was quite challenging to us, we were able to solve this by using OAuth 2.0 authentication protocol to integrate with external providers and enhance the user authentication.

Styling: We faced challenges when styling our web application to improve user interface to provide visually appealing interface to the user. To solve this, we used the CSS framework Bootstrap

Integration: Integration of front end and back end posed a challenge to us, we were able to overcome this using RESTful APIs using HTTP protocol such as GET, PUT, POST, DELETE.

Device Compatibility: Enabling cross device compatibility was a significant challenge in our web application, since we had to adapt to various screen sizes and resolution like mobile phones, tablets and large desktop monitors. To address this challenge, we had to modify the CSS and adjust the layout using Bootstrap. Additionally, we had to test our application on multiple devices to ensure it looks good and functions properly.

Browser Compatibility: We had to ensure that our website is appearing consistently across different browsers, to ensure this we had to conduct cross browser testing, by checking if all pages are same across different browsers like Google, Microsoft Edge.

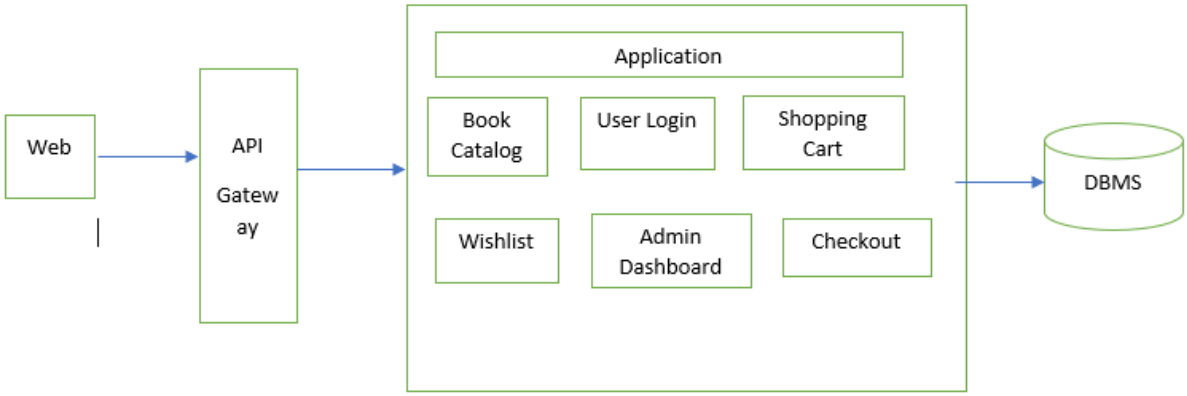
2. Planning and Design:

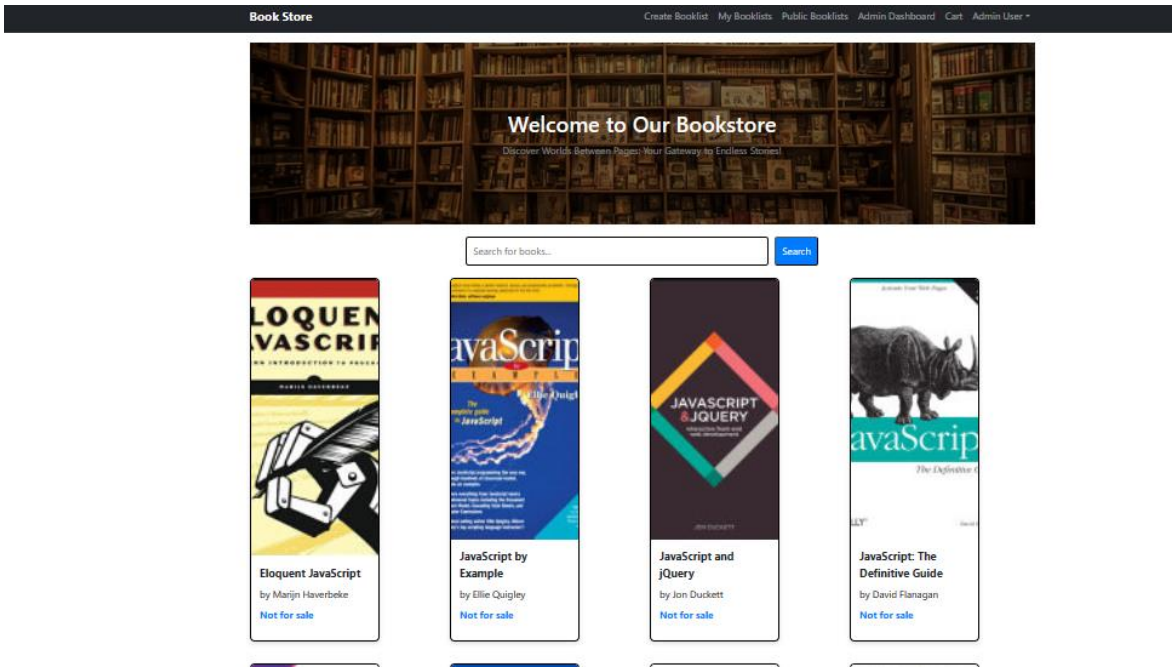
2.1 Project Scope and Requirements:

The bookstore web application aims to provide user friendly and responsive ecommerce platform for selling books to its customers. It offers seamless shopping experience by providing them options to browse from the book catalog, search a wide range of books through keywords, add books of their choice to wish list and cart. The application has limited functionality for various users such as authenticated, unauthenticated and admin.

2.2 Wireframes and Basic Design:

We have designed below monolithic architecture for designing our book store application.





2.3 Development Tools and Environments:

We preferred to use the tools below and environment based on our group discussion and stack used for building our web application.

Integrated Development Environment (IDE): We have used Visual Studio Code (VS Code) code editor to develop our project both front and back end

Version Control System: We have used Git in our project for tracking changes and collaborating within our team.

Figma: We have utilized Figma to create the basic design of our web application, it allows to create prototype from the wireframe design. We were able to create wireframe design for different screen sizes using this tool for our web application

Postman: We used Postman for interacting with the APIs, i.e., sending HTTPS requests to the web server and receiving response back as part of testing our backend API endpoints. We were able to create and send various HTTP request such as GET, PUT, POST, DELETE.

MongoDB Compass: It is a Graphical user interface to interact with MongoDB databases. MongoDB is NoSQL database which stores data in JSON like format, since our data had JSON response, we had to choose MongoDB as database for our web application.

Deployment platform: Deploying the application is essential for it to be accessible to users and scalable to ensure, the application's infrastructure is maintained regularly. We have deployed our web application on Amazon web service.

3. Implementation:

3.1 Key Features:

The online bookstore web application implements several features to enhance the user experience and range of functionality to build an interactive and responsive website.

Book Discovery: It allows users to explore different genre books listed in the website and search for a specific book their choice based on the book id and book name. Users can interact with our user-friendly interface to browse and search for the books they would like to buy.

New User registration and Login: Users new to the website can register using our sign-up page and log in to access their personalized profile, to purchase books from the website. Additionally, we have added functionality to sign in to our website using third party services such ss Google sign in to ensure smooth user experience.

Shopping Cart: Users are allowed to add books of their choice from the book catalog to the cart, choose the number of quantities, remove the items. Based on the user's selection, the application calculates the total cost of the purchase as the product of quantity and the price of each book. The application allows user to review and modify the cart contents before proceeding to the payment section. Further, we have added functionality such that only authenticated user has access to add items to the cart.

Checkout and Payment: The application helps users to complete secure checkout process, providing clear order summary to review their purchase details. Users can make payment through our integrated payment gateway (i.e., Stripe), upon successful completion of payment the users receive a confirmation message.

User Authentication: The application ensures secure authentication to differentiate authenticated and unauthenticated users to ensure that only

authorized users area lowed to access certain functionality, such as only authenticated users are allowed to purchase the product by proceeding to the checkout page, add a review for the books and view up to 20 books and their entire information, whereas the unauthenticated users can view only 10 books. Additionally, the unauthenticated are allowed to view only fewer details when searching under list.

Search Functionality: This allows users to select books of the required genre; the search functionality allows users to search for a book based on the book id and book name.

Cross-Device Compatibility: Our web application is designed to work effectively on various devices including mobile, desktop, laptop and tablets, thus allowing user interface to adjust to different screens size and resolution providing user friendly experience to the site users.

Integration of third-party services: The web application makes use of a Google external API to retrieve book information, it ensures that the bookstore application has access to a wide range of book database with manually creating and maintaining the book catalog to display the list of available books.

Administrator user: An admin is a special user with exclusive privileges and access within the site. The administrator user can assign site privileges to one or more existing user. They also have an option to toggle between hidden and unhidden visibility function to display the review for a particular book by a user. We have also added functionality to the administrator to mark user account as "deactivated" when the user is not active and remark them as "active" again, this allows to manage and track user's participation in the website

3.2 Code Snippets:

Listed below is our snippet of code showing different aspect of our bookstore application.

Figure 1 Snippet of Cart functionality

```
<Col xs={12} md={6}>
  <div>
    <h1 className="font-weight-bold mb-4">{book.title}</h1>
    <p className="text-left">
      <strong>Authors:</strong> {book.authors && book.authors.join(', ')}
    </p>
    <div className="mb-3 text-left">
      <strong>Description:</strong>
      <p dangerouslySetInnerHTML={{ __html: book.description }} />
    </div>
    <p className="text-left">
      <strong>Publisher:</strong> {book.publisher}
    </p>
    <p className="text-left">
      <strong>Published Date:</strong> {book.publishedDate}
    </p>
    <p className="text-left">
      <strong>Categories:</strong> {book.categories}
    </p>
    {/* Display other details as needed */}
  </div>
</Col>
</Row>
```

Figure 2 Snipet of view Book Details

```
<div style={styles.container}>
  <h2 style={styles.heading}>Admin Dashboard</h2>
  <table style={styles.table}>
    <thead>
      <tr>
        <th>Email</th>
        <th>Role</th>
        <th>Active</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      {users.map((user) => (
        <tr key={user._id} style={styles.row}>
          <td>{user.email}</td>
          <td>{user.role}</td>
          <td>{user.isActive ? 'Yes' : 'No'}</td>
          <td>
            <button
              style={styles.activeButton}
              onClick={() => handleToggleActive(user._id)}
            >
              Toggle Active
            </button>
            <button
              style={styles.roleButton}
              onClick={() => handleToggleRole(user._id)}
            >
              Toggle Role
            </button>
          </td>
        </tr>
      ))}
```

Figure 3 Snippet of Admin Dashboard

```
    const formattedBooks = result.data.items.map((item) => {
      const hasPrice = item.saleInfo && item.saleInfo.retailPrice;
      return {
        bookId: item.id,
        title: item.volumeInfo.title,
        author: item.volumeInfo.authors
          ? item.volumeInfo.authors[0]
          : 'Unknown',
        price: hasPrice
          ? `${item.saleInfo.retailPrice.amount} ${item.saleInfo.retailPrice.currencyCode}`
          : 'Not for sale',
        thumbnail: item.volumeInfo.imageLinks
          ? item.volumeInfo.imageLinks.thumbnail
          : 'https://via.placeholder.com/150',
      };
    });

    dispatch({ type: 'FETCH_SUCCESS', payload: formattedBooks });
  } catch (err) {
    dispatch({ type: 'FETCH_FAIL', payload: err.message });
  }
};

const addToCartHandler = async (bookId, bookTitle, bookImage, price) => {
  console.log('Adding to cart with price:', price);
  try {
    let userId;

    if (isAuthenticated) {
      userId = userInfo.user.email;
    } else {
```

Figure 4 Snipet of Home page

3.3 Testing:

We have implemented both Unit testing and Functional testing for our web application. In unit testing we have tested the individual components works as designed. As a result of this testing, we were able to identify bugs earlier during our development process and refactored them to make each unit work as expected.

In Functional testing, we tested the functionality of the web application as per the requirement provided. In includes end to end testing of the application which includes the behavior of the application from the user's perspective. Bothe the unit and functional testing can be automated, but we choose to test them manually as automation process can be computationally expensive and would be time consuming which is more often used in industry when dealing with larger application. Testing is done to provide user a seamless experience by identifying and addressing the issues at an earlier stage in the development process.

4. Results and Outcomes:

4.1 Achievements:

We were able to achieve the business goals by creating a real time e-commerce web application which was deployed in amazon web service (Link). Our live website aims the users to purchase a wide range of book from our collection. We have followed the Software development life cycle during our development process which includes the planning, designing the layout of our web application, implementing the real time website, testing each component and deploying them in cloud. By following the SDLC approach, we were able to successfully transform the business logic into technical implementation.

4.2 Lessons Learned:

We have learnt to follow the design principles to provide responsive and visually appealing user experience for our site users. During our planning phase, we had to

rework on designs to ensure the requirements are satisfied clearly. Further, we had chance to learn the basics of cloud when using it for our deployment, using EC2 instances.

Additionally, we had also worked on the test cases using frameworks like Mocha and Chai to ensure that our web application works as per the given functional requirements. Integrating our front end and backend was challenging, but we were able to resolve it using RESTful APIs and MongoDB for database. Choosing the right technology stack, which in our case was MERN stack i.e., MongoDB, ExpressJS, ReactJS and NodeJS helped us to develop robust web application.

5. Conclusion:

5.1 Project Summary:

This project involved the creation of a comprehensive web application for a book store, focusing on enhancing user experience and functionality. It successfully integrates various features like user authentication, book catalog, shopping cart, and checkout, offering a seamless shopping experience. The bookstore application utilizes frameworks and tools for both front and backend. During the development phase, we addressed various challenges like layout, integration and testing to develop a robust bookstore application.

5.2 Future Enhancements:

We had few improvements that can be implemented to our web application, but due to limited time constraint we could not add those functionalities. Listed below are few suggestions that can be implemented to enhance the functionality of the web application in future.

Recommendation System: We can implement a recommendation system to that suggests users list of books based on their personal preferences from browsing history and purchase history, which would increase the user experience and boost the sales in our web application.

Integration of social media: Currently, our application allows users to sign in through third party services like Google, further we would allow users to connect with their social media account like Facebook and share about the recent purchase and reviews in the feed with their user's connection.

Architecture shift: The bookstore application can be transitioned from monolithic to microservice architecture to improve scalability, flexibility and fault tolerance. This change will allow independent scaling and deployment of service with fault tolerance such that if one of the service fails, it des not affect the entire application as other service can function independently.

Search functionality: Our web application focus on searching books based on keywords and genre at present, which we would enhance this by adding advance filter and sort option to filter books by author, publication date, users rating and sort book catalog with popularity, price and clearance to improve their overall search experience.

Scalability: Currently the traffic in our web site goes through a single server, we can implement load balancer to distribute site traffic across multiple servers, thus enabling horizontal scaling to improve performance and responsiveness of the application during high traffic.

Save and payment functionality: The bookstore application does not store any user information like address or credit card information in the user's profile, we would add this functionality in the future to improve the user experience. Further, we a have integrated only stripe as the payment gateway, this can be improved by integrating more payment gateways like PayPal, Amazon Pay.

6. References:

- https://getbootstrap.com/docs/5.3/getting-started/introduction/
- https://medium.com/@Logicssphere/what-is-the-purpose-of-frameworks-and-libraries-in-web-development-and-provide-examples-of-3fe1f5233854
- http://www.passportjs.org/
- https://www.npmjs.com/package/email-verification
- https://crackstation.net/hashing-security.htm
- https://devdocs.io/react/