CS5031 – Software Engineering Practice

190010714, 220031545, 220031985

University of St Andrews

30th March 2023

PRACTICAL 3 – A SHARED APPLICATION

1. INTRODUCTION :

The purpose of this technical report is to outline the development of a shared system application. The application is expected to have an admin interface, user accounts, and a feature for users to view, add and update information. The report outlines the project's requirements, architecture, design, and implementation details.

The system that the team has opted for the coursework's requirement was to create a generic online shopping system and to focus particularly on one product, we choose to build an **online bookstore application** that allows users to collaborate and share information.

THE SYSTEM MUST INCLUDE FEATURES AS PER THE REQUIREMENTS:

- An Admin interface.
- User accounts (which may require the admin interface to configure, you are
- for example not expected to be able to email users).
- The information which users can *view*, *add to*, and *update* (different users
- should have different permissions).

The team created the following requirements to effectively build the system after deciding on the above system and seeking **must-have features** on how they integrate with the decided system.

For development purposes, the team went ahead and decided to call the system **"Alpha".**

FUNCTIONAL REQUIREMENTS:

- **User Registration:** Alpha should allow users to register for an account, providing necessary information such as their name, email address, and password.
- **User Login:** Alpha should allow users to log in to their accounts using their usernames and passwords.
- **User Roles and Permissions:** Alpha should provide different levels of permission to users based on their roles and responsibilities.
- **Product Management:** Alpha should enable sellers to add, update, and delete products from the system, including product details.
- **Customer Management:** Alpha should enable users to manage customer information, including customer profiles and purchase history.
- **Search and Filter:** Alpha should provide search and filter options to allow users to find specific products or orders quickly.

NON – FUNCTIONAL REQUIREMENTS:

- **Performance:** Alpha should be designed to handle a large number of users and high traffic, ensuring fast response times and minimal downtime.
- **Security:** Alpha should provide secure access to the system, protecting user data and preventing unauthorized access or data breaches.
- **Scalability:** Alpha should be designed to accommodate future growth, enabling the system to expand as the business grows.
- **Resilience & Reliability:** The server should not crash even if it is fed improperly constructed or malicious input.

DESIGN

The shopping system has a **client-server architecture,** where the client-side consists of a *terminal* and a *GUI client*, and the server side provides a *RESTful API* to communicate with the clients. The server is implemented using Java and is designed to handle incorrectly formed or malicious input without crashing.

User Types:

There are three types of users in the system :

- **Admin**
- **Seller**
- **Customer.**

The admin is the superuser who oversees the system and has control over all users. The admin can **view** all users, **add or delete user accounts**, **change user types**, and **view requests from users to change their account types**. The admin has a separate menu with functions tailored to their specific functionalities.

The seller also has a separate menu with functions tailored to their specific functionalities. The seller is a user who can **add, update**, and **delete** products in the system. They can view their products and the orders of customers who have purchased from them. The seller can also view their account details and delete their account.

If a seller wishes to become a customer, they can submit a **request** to the admin, who will review the request before granting the request.

The customer is a user who can view products in the system and add them to their cart. They can **search for products by name, category, and author**, or **view all products at once**. Once they have added the desired products to the cart, they can proceed to checkout, where they can review their order and confirm the purchase.

The customer can view their account details, delete their account, and view their order history. The customer has a **separate menu** with functions tailored to their specific functionalities.

Functionality:

The system is designed to allow users to collaborate and share information by adding, updating, and deleting products. The system is designed to **ensure the privacy of users** and **prevent unauthorized access to user information**. The system is also designed to handle incorrectly formed or malicious input without crashing.

IMPLEMENTATION

BACKEND DEVELOPMENT:

The Alpha system is designed using the **Model-View-Controller (MVC) architecture**, with a service layer containing the business logic. This approach separates the system's concerns into three components: the Model, the View, and the Controller.

The Model is responsible for managing the system's data. This includes managing **user account information, product information, and order information**. The View is responsible for presenting the data to the user in an appropriate format, such as a **dashboard interface or a list of products**.

Finally, the Controller acts as the intermediary between the Model and the View, managing user input and coordinating the system's response. In addition to the MVC architecture, the Alpha system also **includes a service layer that contains the system's business logic.**

This layer manages the processes and operations that are unique to the Alpha system, such as user authentication and product management. The service layer communicates with the Model to retrieve and update data as needed, and with the Controller to respond to user requests.

Overall, **the Alpha system is designed with a modular architecture** that separates concerns and enables **scalability**, **maintainability**, and **testability**. The use of the MVC architecture and service layer ensures that the system can be easily extended or modified in the future.

API DOCUMENTATION // JIANG

The API part is separated into five parts. They are userService API, requestService API, productService API, cartService API, and orderService API. Each part of API supplies services related to a different part of the system.

All implementations of API are based on the controller. Every API would call the method from the layer of the controller. For some "add" API, it receives a request body from the calling which could be in JSON format. It requires both variable name and variable value, such as " 'example': example ".

All the APIs are listed in the APPENDIX (below conclusion).

FRONT END DEVELOPMENT

TERMINAL

The terminal client uses the Web Client from Spring Boot to communicate with the server. There is an API call class to call every API from the server. To initialise this class, it requires passing a base URI, such as “http://localhost:8080”. All the data received from the user would be passed to the system to interact. If the user tries to get data from the server, the data structure would be Map<String, Map<String, String>> because there is no java object when communicating between the server and client.

UI (WEBSITE)

Features:

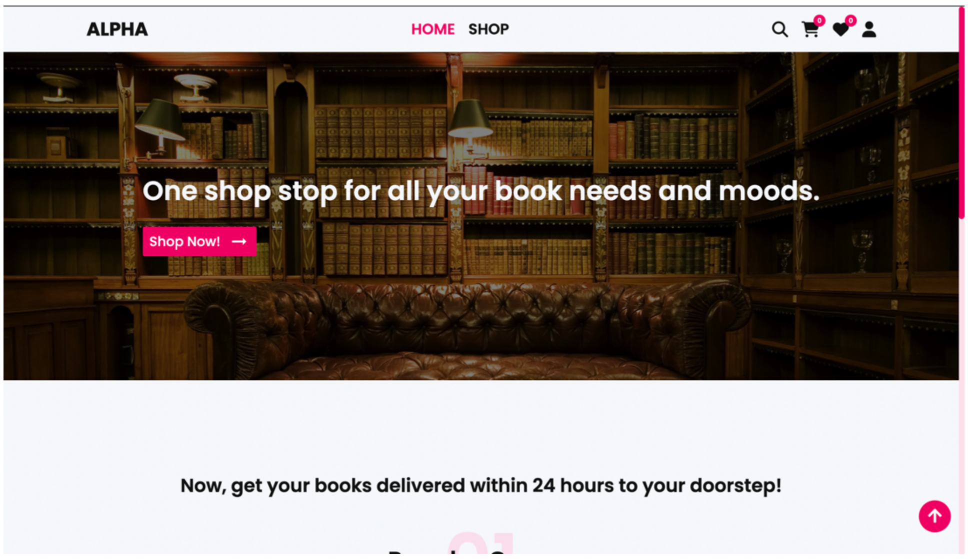
- List of categories
- Product listing page
- Single product page
- Filter, search, and sort products
- Users can click on a product to view detailed information on the product
- Wishlist Management
- Cart Management
- Address Management
- Orders Management

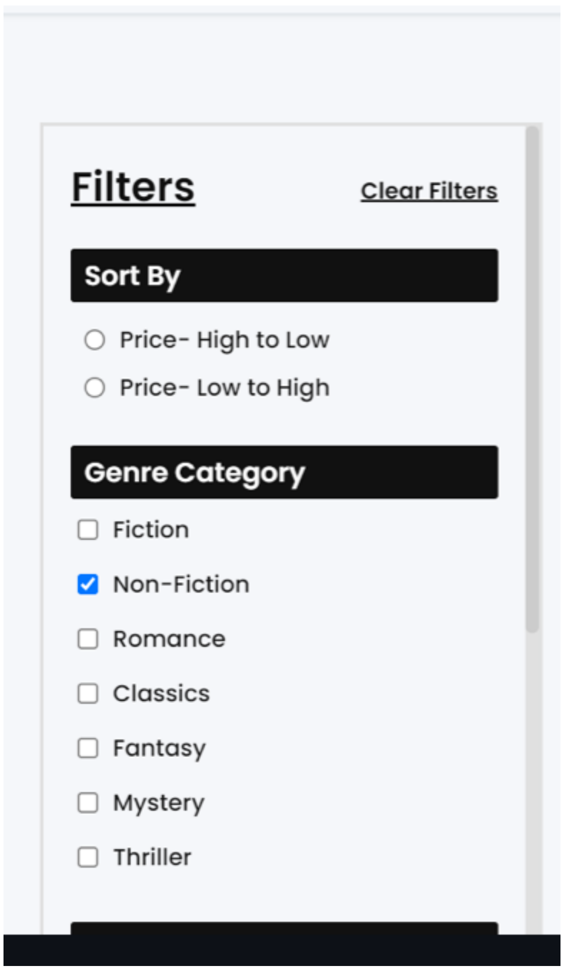
Technology/ Languages Used:

- CSS
- Elixir UI - Component Library
- ReactJS
- React Router v6

The Website was designed to be as intuitive and scalable as possible. Considering that we were asked to develop a complete application using the skills we have learned throughout the module, using agile work practices with a division of labour in a team, following a Scrum style of development, we decided to focus on creating a fully scalable React frontend. In order to achieve this, proper adherence to React practice was essential. The directory is structured into pages, routes, components, services, and react hooks.

Homepage:


ALPHA
HOME
SHOP
One shop stop for all your book needs and moods.
Shop Now!
Now, get your books delivered within 24 hours to your doorstep!



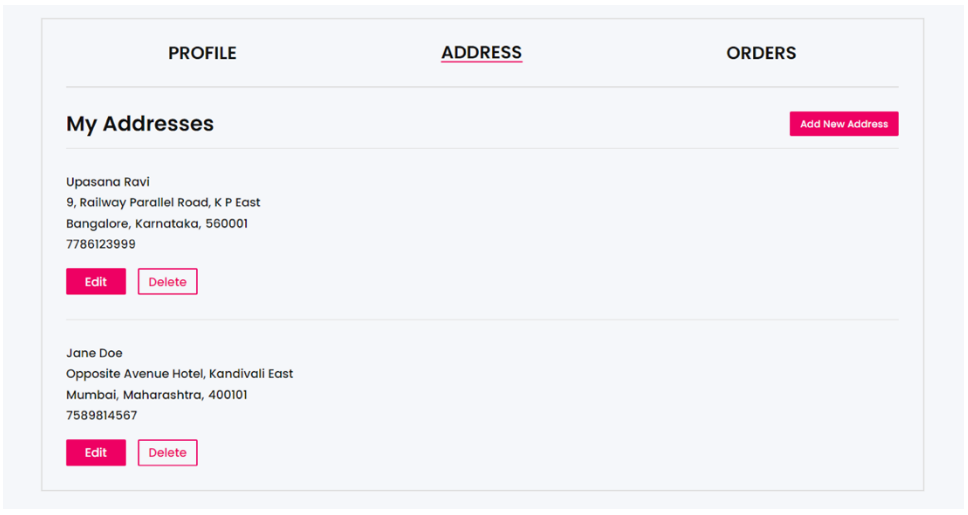
Filters Clear Filters

Sort By

- ○ Price- High to Low
- ○ Price- Low to High

Genre Category

- [ ] Fiction
- [x] Non-Fiction
- [ ] Romance
- [ ] Classics
- [ ] Fantasy
- [ ] Mystery
- [ ] Thriller



PROFILE ADDRESS ORDERS

My Addresses

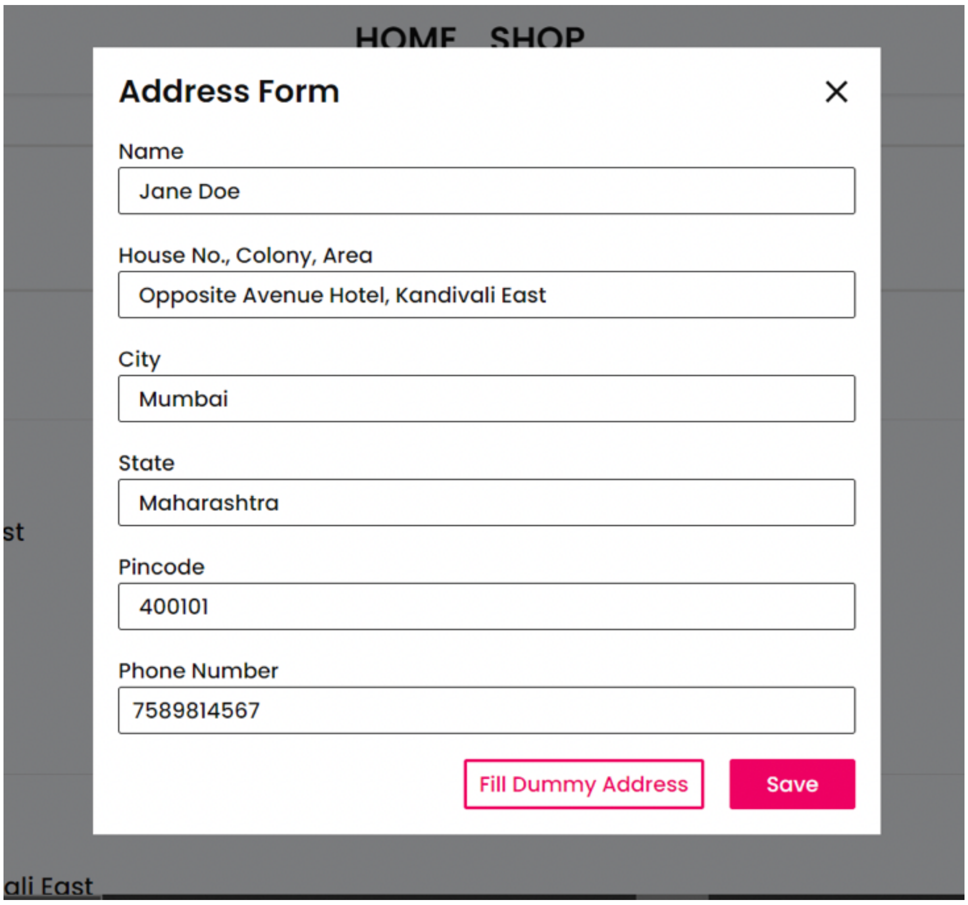
Add New Address

Upasana Ravi
9, Railway Parallel Road, K P East
Bangalore, Karnataka, 560001
7786123999

Edit Delete

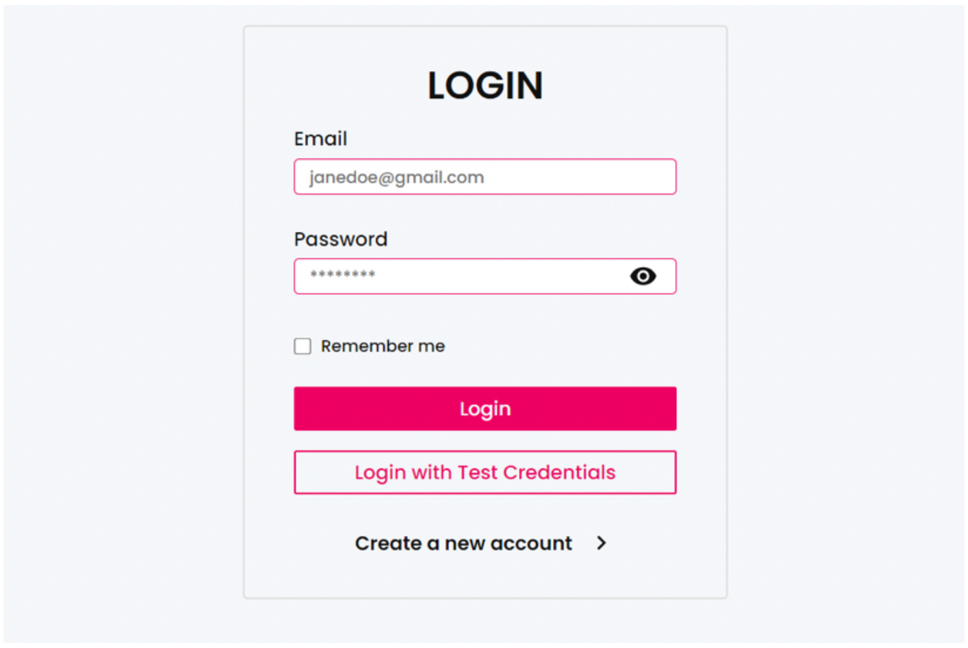
Jane Doe
Opposite Avenue Hotel, Kandivali East
Mumbai, Maharashtra, 400101
7589814567

Edit Delete



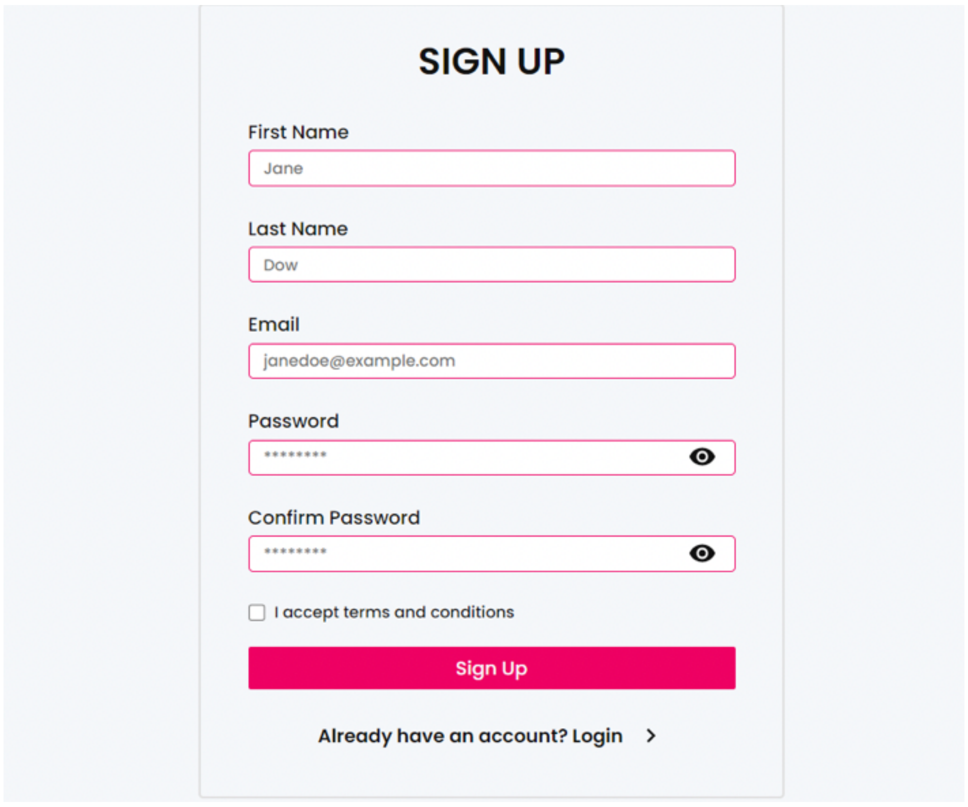
Address Form

Name	Jane Doe
House No., Colony, Area	Opposite Avenue Hotel, Kandivali East
City	Mumbai
State	Maharashtra
Pincode	400101
Phone Number	7589814567



LOGIN

Email	janedoe@gmail.com
Password	********


SIGN UP

First Name

Jane

Last Name

Dow

Email

janedoe@example.com

Password

****

Confirm Password

****

I accept terms and conditions

TEST-DRIVEN DEVELOPMENT (TDD)

The development team has decided to adopt a process that involves building the front end of the command line interface first and then developing the back end to support the system's logic.

To ensure TDD was strictly adhered to, **220031545** was tasked with writing the tests after working on the view, every time there was a need to call the controller, it meant writing back-end logic, and as per the TDD approach, it meant writing failing tests and then writing code for the tests. **220031985** role was to write the logic to the failing tests and ensure that it passed the testing.

This was done to ensure that the code met the necessary criteria, handled edge case scenarios, and allowed the development team to discover flaws earlier in the development process. JUnit 5 was used for testing, Mockito was used for API testing,

Gitlab was used for version control, and Spring MVC was utilised for server code.
To manage the development process**, Git branches** have been created for each team member.

Merge requests are utilized for code reviews and approvals by other team members. **Issues are also created** to track bugs, testing, or new feature additions.

The design of the system was based on the functional requirements that the team had drew from the assignment requirements and mentioned in the report, based on this data, tests were created for the model layer, service layer and API. For example:

Requirement: A user should be able to create an account.
Test case: When a user clicks on the "Create an Account" button, they should be redirected to the account creation page.
Test result: The test fails, as the button does not lead to the correct page.

The API tests were then constructed using the Mock-MVC framework, and the API code was also modified. The group recognised that this had some drawbacks and restrictions due to the extra time required to change the tests and code, but it allowed us to make progress rather than being prevented from starting. We managed this by having constant contact via the scrum meeting, outside meeting conversations, and shared API learning. This meant we could properly organise our time and finish the assignment.

Overall, the development team's use of the TDD approach and other development approaches contributed to the system's quality, addressing edge case scenarios, and allowing the team to identify defects earlier in the development process. We were able to efficiently coordinate our time and accomplish the job because of the team's consistent interactions and shared learning.

SCRUM + AGILE DEVELOPMENT

Meeting 1: Understanding the specs and deciding on the system.

SCRUM LEADER : 220031985

After discussing the specifications, we decided to build a shared system that allows users to collaborate and share information related to online shopping. We named our system "Alpha" for development purposes.
During the meeting, we also decided on the core features that Alpha should include.

These features are:

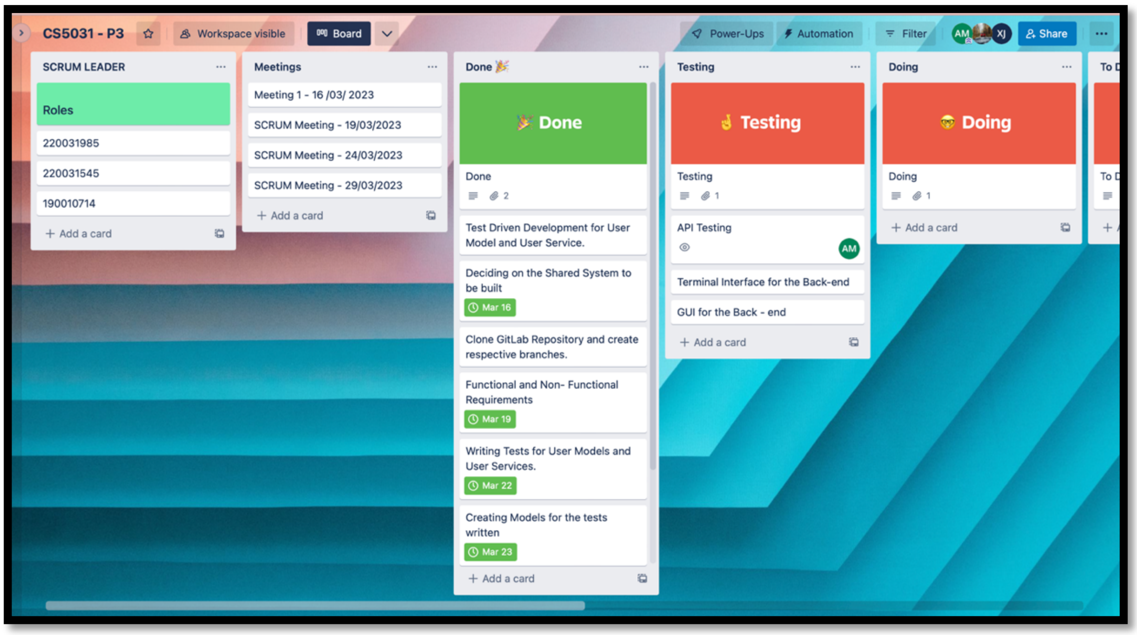
- Admin Interface: An interface that allows the administrator to configure user accounts and permissions, manage the content of the system, and monitor user activity.
- User Accounts: Alpha should have user accounts (In this system there are two types of users: seller and customer) to enable users to log in, access the system, and collaborate with others. The admin interface should be used to set up and manage these accounts and Users can create individually but cannot be an admin.
- User Permissions: Alpha should provide different levels of permission to users based on their roles and responsibilities. For instance, a seller who is responsible for managing products should have permission to add and update products, while a customer should be able to view and buy products from the seller.
- Information Sharing: Alpha should allow users to share information related to online shopping, such as product information, orders

Meeting 2: Designing the System

SCRUM LEADER: 190010714

During this meeting, we discussed the system design and came up with a plan to implement the core features we had decided on earlier.

- We decided that the system should have a dashboard / Menu (Command Line Interface) where users can access all the system's features, including managing their account settings and collaborating with others. Depending on the account type, an appropriate response should be appropriate be triggered.
- We also discussed the data storage design and how the different components of the system would be integrated. Given the time constraint, we opted for writing and reading to file (.ser).
- Using **Trello,** we assigned tasks to team members based on their strengths and areas of expertise, ensuring adherence to SCRUM-based practices, and maintaining regular updates on project progress.
- We also discussed the timeline for the project and set deadlines for each task.



CS5031 - P3 | Workspace visible | Board | Power-Ups | Automation | Filter | Share

SCRUM LEADER
- Roles
- 220031985
- 220031545
- 190010714
- + Add a card

Meetings
- Meeting 1 - 16 /03/ 2023
- SCRUM Meeting - 19/03/2023
- SCRUM Meeting - 24/03/2023
- SCRUM Meeting - 29/03/2023
- + Add a card

Done
- Done
- Test Driven Development for User Model and User Service.
- Deciding on the Shared System to be built (Mar 16)
- Clone GitLab Repository and create respective branches.
- Functional and Non- Functional Requirements (Mar 19)
- Writing Tests for User Models and User Services. (Mar 22)
- Creating Models for the tests written (Mar 23)
- + Add a card

Testing
- Testing
- API Testing
- Terminal Interface for the Back-end
- GUI for the Back - end
- + Add a card

Doing
- Doing
- + Add a card

To D

Figure 1: Trello Board Usage for SCRUM Implementation.

Meeting 3 – discussion on integrating the backend to the front end

SCRUM LEADER: 220031545

During our third meeting, the team progressed towards fulfilling the requirements by shifting focus to front-end development. Specifically, two dedicated views were required - the Terminal View and the GUI View - which relied on the API calls to the backend.

To begin with, the team had already written test cases for the backend and had successfully implemented the respective codes for API calls. For the GUI View, a website interface was developed by team member **190010714.** The interface was designed to be fast, reactive, and well-integrated with the backend via API calls.

During the meeting, a demo was presented by the assigned team members showcasing the functionality of the Terminal and GUI views. The team also conducted testing and provided feedback and suggestions for improving the functionality of both views.

With this, the team concluded the third meeting, having made significant progress towards meeting the project requirements.

Meeting 4: Final Review and Product Demo

The team was in the process of conducting a final quality assurance check before submission. This involves ensuring that the back-end development is properly integrated with the front-end and that the system as a whole is functioning correctly.

To achieve this, each team member has set up private stations to test the system in a controlled environment. Feedback has been collected from each team member, and the system has been refactored based on this feedback to improve its overall quality.

This approach of testing the system under various conditions and taking feedback from team members was an effective way to identify and address any issues that may have been missed during development. By taking this final step to ensure that the system is of high quality, the team can submit the system with confidence.

CONCLUSION

The shopping system developed by the team meets the assignment's requirements. The system is a collaborative and information-sharing online bookshop application. The system is meant to have three categories of users: admin, seller, and customer, each with its own set of functions and rights.

The system is built to handle improperly constructed or malicious input without crashing, protecting user privacy and preventing unauthorised access to user data. Overall, the team successfully designed the shopping system employing agile work processes with team division of labour and a Scrum development methodology.

TOTAL WORDS : 2470

APPENDIX

Method	API	Description
POST	/userService/add With requestBody {“firstName”:example” lastName”:example” username”:example”	create a new user

	email":example "password":example "accountType":example}	
GET	/userService/adminLogin?user=example&password=example	for admin to log in
GET	/userService/viewAll	get all the users
GET	/userService/Login?user=example&password=example	for user to log in
POST	/userService/adminGenerate	generate default admin
POST	/userService/deleteViaAdmin/{username}	delete a user through admin
POST	/userService/delete	delete self-user account
POST	/requestService/updatePermission/{username}/{requestType}	update permission of user
POST	/requestService/storeRequest/{username}/{requestType}	store the request
GET	/requestService/viewRequests	view all the requests
POST	/productService/add With requestBody {"prodcutID":example, "productName":example, "productDescription":example, "author":example "productPrice":example "productQuantity":example "productCategory":example "sellerUsername":example}	add a new product

GET	/productService/searchName/{productName}	search product by name
GET	/productService/searchCategory/{productCategory}	search product by category
GET	/productService/searchAuthor/{productAuthor}	search product by author
GET	/productService/viewAll	view all products
POST	/productService/delete?id=example&seller=example	delete the product
GET	/productService/view/{sellerUsername}	view products given seller
GET	/productService/check?seller=example&id=example	check whether user have updating permission
POST	/productService/updateName?id=example&name=example	update the product name
POST	/productService/updateDescription?id=example&description=example	update the product description
POST	/productService/updatePrice?id=example&price=example	update the product price
POST	/productService/updateQuantity?id=example&quantity=example	update the product quantity
POST	/productService/updateCategory?id=example&category=exampl	update the product category
POST	/productService/updateAuthor?id=example&author=example	update the product author
GET	/cartService/view/{username}	view the customer cart
POST	/cartService/add	add an item to the cart

	With requestBody {“cartID”:example, “productID”:example, “customerName”:example, “productName”:example, “productPrice”:example, “productQuantity”:example, “productTotal”:example, “sellerName”:example}	
POST	/cartService/delete/{cartID}	remove an item from the cart
POST	/orderService/add With requestBody {“orderID”:example, “productID”:example, “customerName”:example, “productName”:example, “productPrice”:example, “productQuantity”:example, “productTotal”:example, “sellerName”:example}	add an order to the system
GET	/orderService/searchName/{user name}	get orders by customer username
GET	/orderService/searchSeller/{selle rUsername}	get orders by seller username
POST	/orderService/update/{productI D}/{productQuantity}	update the product quantity after made an order

POST	/save	save the system data
POST	/load	load the system data