SpareB&B Student Accommodation

Student name ; Sawsan mounes

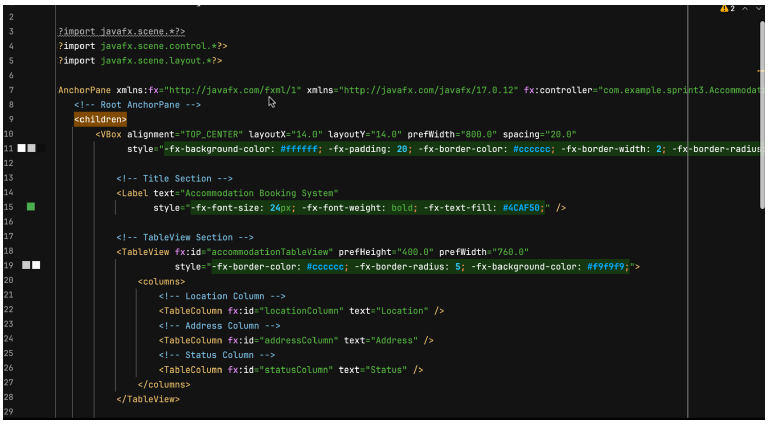
Student ID : MOU23620003

Schematic Wire-Frame

The structure and arrangement of the GUI for the Student Accommodation Booking System are shown graphically in the schematic wireframe. It shows how the main elements such as text fields, buttons, tables, and input fields are arranged within the interface. This wireframe gives users a clear visual representation of how the system will look and work with them including the sign-up page and the log in page, allowing them to effectively student accommodation bookings.

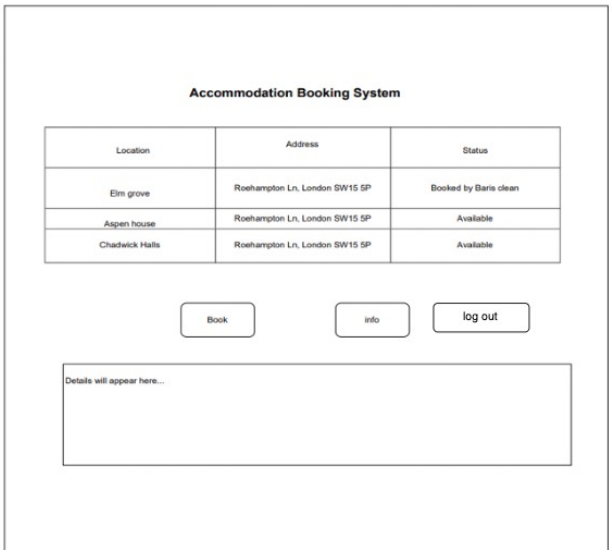
FXML

The FXML file is linked to the AccommodationController, which is responsible for managing and displaying all the information about accommodations.



```
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/17.0.12" fx:controller="com.example.sprint3.Accommodat
    <!-- Root AnchorPane -->
    <children>
        <VBox alignment="TOP_CENTER" layoutX="14.0" layoutY="14.0" prefWidth="800.0" spacing="20.0"
              style="-fx-background-color: #ffffff; -fx-padding: 20; -fx-border-color: #cccccc; -fx-border-width: 2; -fx-border-radius

            <!-- Title Section -->
            <Label text="Accommodation Booking System"
                   style="-fx-font-size: 24px; -fx-font-weight: bold; -fx-text-fill: #4CAF50;" />

            <!-- TableView Section -->
            <TableView fx:id="accommodationTableView" prefHeight="400.0" prefWidth="760.0"
                       style="-fx-border-color: #cccccc; -fx-border-radius: 5; -fx-background-color: #f9f9f9;">
                <columns>
                    <!-- Location Column -->
                    <TableColumn fx:id="locationColumn" text="Location" />
                    <!-- Address Column -->
                    <TableColumn fx:id="addressColumn" text="Address" />
                    <!-- Status Column -->
                    <TableColumn fx:id="statusColumn" text="Status" />
                </columns>
            </TableView>
```

Wire-frame



The Table View has three columns:

- **Location:** Accommodation name.
- **Address:** Shared postcode for all accommodations.
- **Status:** Shows availability or "Booked by [User Name]" if reserved, managed by the BookingSingleton class.

Buttons:

- **List Accommodations:** Displays all available options.
- **Book:** Allows reservation of an accommodation.
- **Info:** Provides details about the selected accommodation.

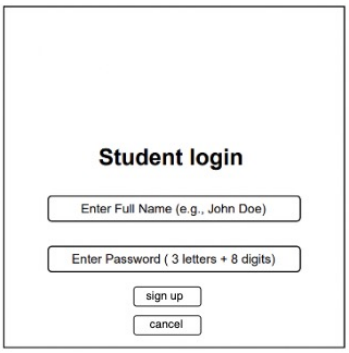
TextArea: Displays messages, confirmations, or errors.

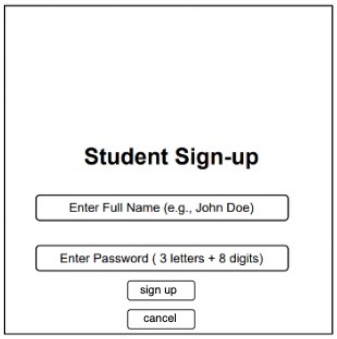
FXML

The FXML file is linked to the login controller and sign-up controller, which is responsible for managing and displaying all the information about accommodations.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.VBox?>

<VBox xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
      fx:controller="com.example.sprint3.LoginController"
      alignment="CENTER" spacing="15"
      style="-fx-background-color: #ffffff; -fx-padding: 20px; -fx-border-color: #cccccc; -fx-border-width: 2; -fx-border-radius: 10;
    <Label text="spareb&amp;B accmodation log in  " style="-fx-font-size: 20px; -fx-font-weight: bold; -fx-text-fill: #4CAF50;"/>
    <TextField fx:id="fullNameField" promptText="Full Name" style="-fx-pref-width: 300; -fx-font-size: 14px;"/>
    <PasswordField fx:id="passwordField" promptText="Password (Student ID)" style="-fx-pref-width: 300; -fx-font-size: 14px;"/>
    <Button text="Login" onAction="#handleLogin" style="-fx-pref-width: 150; -fx-font-size: 14px; -fx-background-color: #4CAF50; -fx-t
    <Button text="Sign Up" onAction="#switchToSignUpScreen" style="-fx-pref-width: 150; -fx-font-size: 14px; -fx-background-color: #21
</VBox>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<VBox alignment="CENTER" prefHeight="366.0" prefWidth="493.0" spacing="15" style="-fx-background-color: #ffffff; -fx-padding: 20px; -fx
    <Label prefHeight="19.0" prefWidth="118.0" style="-fx-font-size: 20px; -fx-font-weight: bold; -fx-text-fill: #4CAF50;" text="spareb
    <TextField fx:id="fullNameField" promptText="Full Name" style="-fx-pref-width: 300; -fx-font-size: 14px;" />
    <PasswordField fx:id="passwordField" promptText="Password (3 letters + 8 digits)" style="-fx-pref-width: 300; -fx-font-size: 14px;"
    <Button onAction="#handleSignUp" style="-fx-pref-width: 150; -fx-font-size: 14px; -fx-background-color: #4CAF50; -fx-text-fill: whi
    <Button onAction="#switchToLoginScreen" style="-fx-pref-width: 150; -fx-font-size: 14px; -fx-background-color: #f44336; -fx-text-fi
  <AnchorPane prefHeight="111.0" prefWidth="342.0" />
</VBox>
```





•Text Fields:

- **Full Name:** Input field for entering the user's full name .
- **Password:** Input field for entering the user's password 3 characters and digits only.

•Buttons:

- **Log In:** Authenticates the user using the Authenticate class if they have previously created an account, then transitions to the accommodation list upon successful validation.
- **Sign Up:** Redirects the user to the account registration screen.

•Input Fields:

- **Full Name:** Field for entering the user's full name.
- **Password:** Field for setting a secure password.

•Validation Rules:

- The password must consist of exactly three letters followed by eight digits.

•Buttons:

- **Sign-Up:** Validates the input based on the rules and registers the user if valid.
- **Back to Login:** Navigates the user to the login screen.

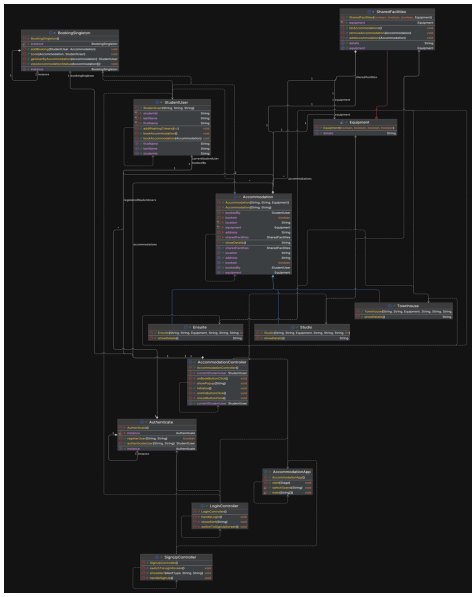
User stories

User Story 1:
As a student, I want to browse and view accommodations that match my preferences, so that I can find the best option for my needs.

User Story 2:
As a student, I want to select an accommodation and complete the booking process, so that I can secure my desired place to stay.

User Story 3:
As a student, I want to receive a confirmation message after booking, so that I can be assured that my reservation was successful and feel satisfied.

UML CLASS DIAGRAM



Relationships in the UML:

•Inheritance:
The Ensuite, Studio, and Townhouse classes inherit from the base class Accommodation.

•Composition:
The Accommodation class has a composition relationship with the SharedFacilities class.
Additionally, SharedFacilities contains Equipment as a nested class.

•Singleton Pattern:
The BookingSingleton class ensures centralized booking management by tracking all bookings in a single instance.

•Controller Classes:
These classes mediate interactions between the user interface and the application's core logic, serving as a bridge between the two.

AccommodationController:

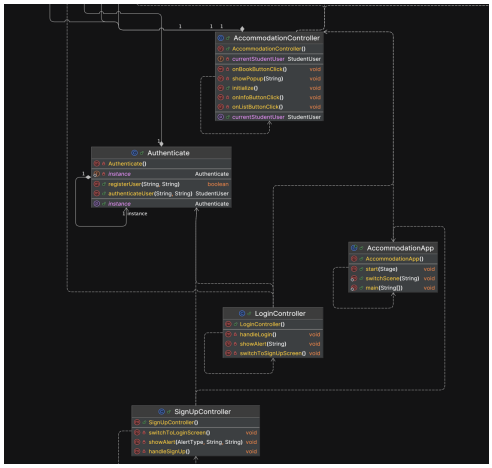
•Manages the accommodation list screen where users can view, book, and get details about accommodations.
•Populates the **TableView** (Location, Address, Status) using onListButtonClick().Handles bookings with onBookButtonClick() and displays accommodation details via onInfoButtonClick().Retrieves and updates booking data through BookingSingleton.

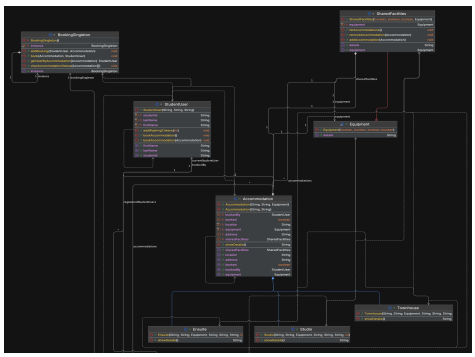
LoginController:

•Manages the login screen by authenticating user credentials using Authenticate. Redirects users to the Sign-Up screen via switchToSignUpScreen() and transfers authenticated users to the AccommodationController for personalized session tracking.

SignUpController:

•Handles user registration by validating inputs and creating accounts through Authenticate. Navigates back to the login screen after successful registration with switchToLoginScreen().





Classes That Handle Data for the GUI:

Accommodation and Subclasses (Ensuite, Studio, Townhouse):
Serve as the data model for the **TableView** (Location, Address, Status).
bookedBy tracks which user booked the accommodation.
showDetails() displays detailed information in the **TextArea** when the **Info** button is clicked.

StudentUser:
Represents the logged-in user and displays a personalized welcome message.
Tracks booked accommodations and dynamically updates rewards (e.g., washing tokens).

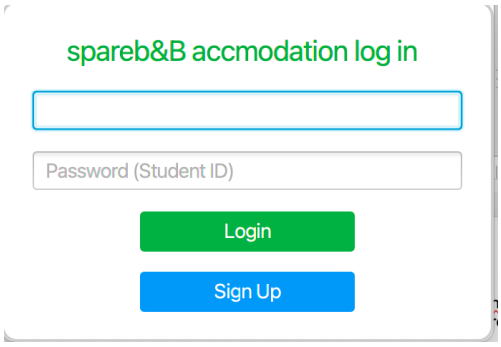
BookingSingleton:
Manages centralized booking data.
Allows AccommodationController to query booking status and identify which user booked a specific accommodation.
Ensures consistent booking data across the GUI.

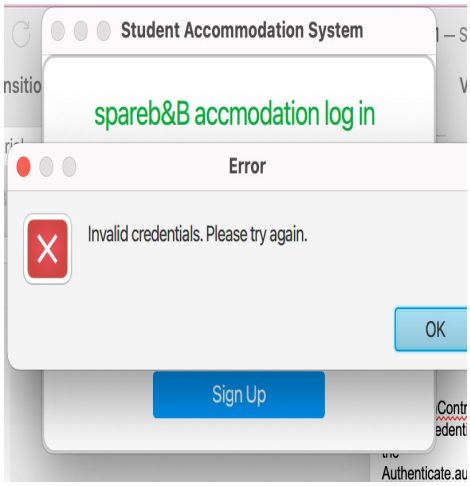
SharedFacilities and Equipment:
Provide additional accommodation details displayed in the **TextArea**.getDetails() generates summaries of shared facilities and equipment.

Authenticate:
Validates user credentials during login and registers new users during sign-up.
Connects with backend data for user management.

Screenshots of the GUI Application

The LoginController checks credentials using the Authenticate.authenticate User() method. If valid, the user is directed to the main screen, and their session is personalized using the StudentUser instance.

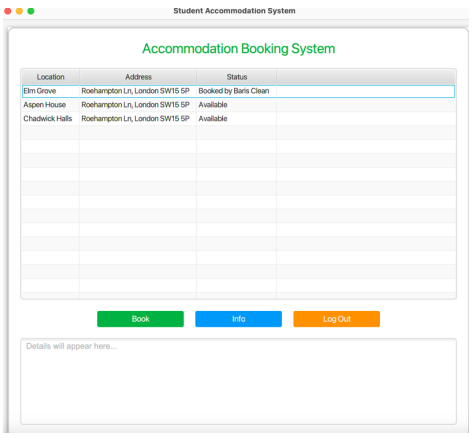


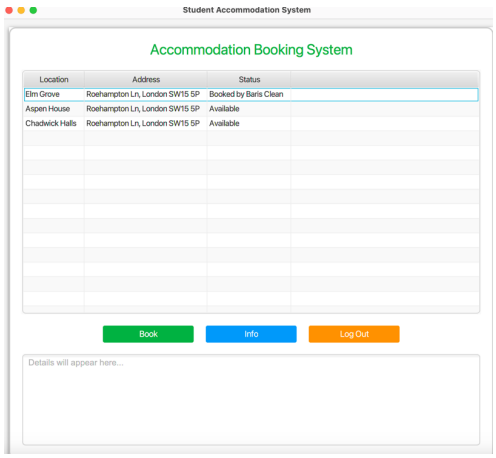


You need to sign up before logging in. If you try to log in without signing up, an alert will appear saying "Invalid credentials." this is error handling

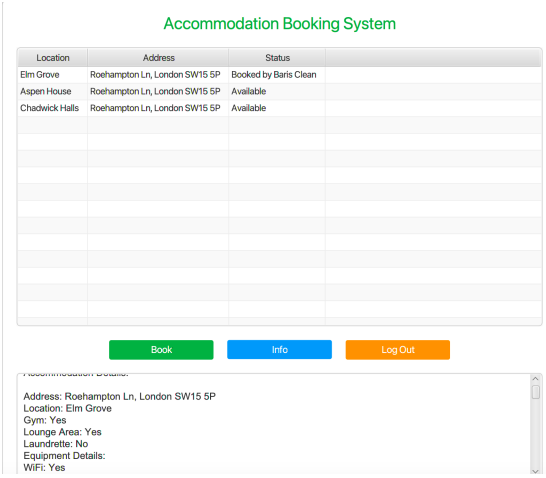
Once the sign-up is successful and you enter the correct combination of digits and letters, an alert will appear confirming that the login was successful.]

The SignUpController validates user input and calls the Authenticate.registerUser() method to create a new StudentUser. A success alert confirms registration switches scene to the booking page





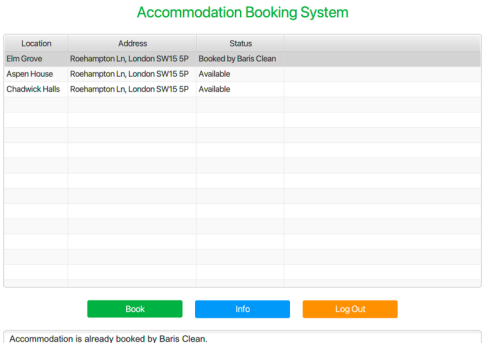
After verifying the user's credentials through authentication, the system navigates to the **Accommodation Booking Page**, which is managed by the AccommodationController. This page displays a list of available accommodations and indicates which ones have already been booked.



To view the details and prices of accommodations, users can select an option from the list and click the **Info** button. This provides detailed information about the selected accommodation, including its address, name, and available shared facilities, such as gym or lounge areas.
In this example:

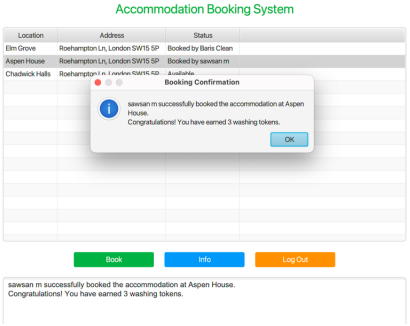
•**Elm Grove**: The details indicate it includes a gym and lounge area. The status confirms that it has already been booked by **Baris Clean**, as displayed in the table's **Status** column.
•**Aspen House** and **Chadwick Halls**: These are still marked as "Available" and can be booked.

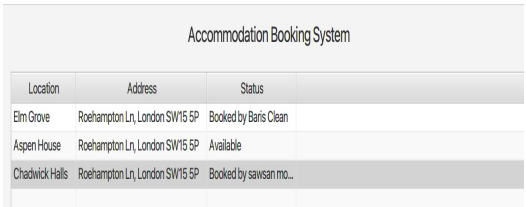
The system dynamically pulls and displays this information in the **TextArea** below the table. This interaction is managed by the AccommodationController, which retrieves the details from the Accommodation object and its associated SharedFacilities information. The showDetails() method is used to generate and display this information.



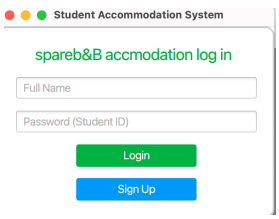
If you click on **Elm Grove** and then click the **Book** button, the system will notify you that this accommodation is already booked by Baris Clean and cannot be reserved again.

The table displays the list of accommodations with their status (Available or Booked). The **Book** button checks availability and provides feedback in the **TextArea**, while the **Info** button shows additional details about the selected accommodation.
This logic is managed by the AccommodationController and BookingSingleton to ensure accurate booking status and user feedback.





The user experience is designed to be seamless, intuitive, and rewarding. When a user books an accommodation, such as **Chadwick Halls**, a pop-up confirmation congratulates them and awards 3 washing tokens, enhancing the sense of achievement. The **Status** column updates in real-time to reflect the booking (e.g., **"Booked by Sawsan Mounes"**), providing immediate feedback. This process is powered by the BookingSingleton class, which ensures consistent tracking and management of bookings across the system, creating a smooth and reliable experience for the user.



When you click the log out button it logs out ands goes back to the log in page

Programming Paradigms Used in the Application

Key OOP Concepts:

In this application, Object-Oriented Programming (OOP) principles—such as encapsulation, polymorphism, and inheritance—are seamlessly applied. These principles are evident in the Accommodation class and its subclasses (Ensuite, Studio, and Townhouse), as well as in the StudentUser and SharedFacilities classes. Together, they provide a modular, reusable, and maintainable code structure.

1.Inheritance:
The Accommodation class uses inheritance to allow subclasses to share common properties and methods, reducing code duplication and enabling functionality reuse.
2.Polymorphism:
The showDetails() method demonstrates polymorphism by being overridden in each subclass to display specific details unique to each accommodation type.
3.Encapsulation:
Encapsulation is implemented through getters and setters in the Accommodation class. Attributes such as location, address, and status are protected to ensure they are accessed and modified in a controlled and secure manner.

Declarative Programming in Practice:
Declarative programming focuses on describing what the program should achieve, not how to achieve it. It emphasizes desired outcomes, leaving implementation to the programming language, and is ideal for computational logic.
In this application, declarative techniques are used in FXML to define UI components like table views and buttons, keeping the design clean and organized. For example, scene transitions like moving from login to sign-up focus on the result rather than the process.
This combination of declarative programming, functional programming with streams, and OOP principles ensures an efficient, maintainable, and user-friendly application.

Functional programming features

Functional programming features, such as Java Streams, are also utilized. For example, in the AccommodationController class, streams are used to dynamically filter and display accommodation details. This is demonstrated in the onInfoButtonClick method, which processes accommodation data efficiently.

```
@FXML
private void onInfoButtonClick() {
    // Generate detailed information about all accommodations.
    String details = accommodations.stream()
            .map(Accommodation::showDetails) // Get the details of each accommodation.
            .collect(Collectors.joining( delimiter: "\n--------------------------------\n")); //
    infoArea.setText("Accommodation Details:\n\n" + details); // Display the details in the i
}
```

1.Mapping (map):
1. Converts each Accommodation object into a formatted string using the showDetails() method.

Example: For **Elm Grove**,
Location: Elm Grove
Address: Roehampton Ln, London SW15 5P
Gym: Yes
Lounge: Yes

2.Reducing (collect):
1. Combines all the strings into a single block of text, separating them with:

example: --------------------------------

3.Stream Pipeline:
1. Processes the list of accommodations step-by-step:
   stream(): Starts processing.
   map: Converts objects to details.
   collect: Combines the results into a single output.

Example: Outputs formatted details for all accommodations, displayed in the **TextArea**.

This combination of declarative programming, functional programming with streams, and OOP principles ensures an efficient, maintainable, and user-friendly application.

Conclusions

- The SpareB&B Student Accommodation System exemplifies a robust application of Object-Oriented Programming principles, functional programming features, and declarative design methodologies to deliver a seamless and efficient user experience. By leveraging key OOP concepts such as inheritance, polymorphism, and encapsulation, the application ensures maintainability and modularity.
- Functional programming techniques like Java Streams enhance data processing efficiency, while the declarative FXML approach ensures a clean and organized user interface. The system not only meets the core user requirements of browsing, booking, and managing accommodations but also incorporates features like real-time updates, intuitive navigation, and a rewarding user experience. Collectively, these elements demonstrate a well-rounded and scalable solution tailored to student needs, paving the way for future enhancements and applications