Declaration of Original Work for SC2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed, and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Student ID	Course	Lab Group	Signature/Date
Yeo Yu Xuan Dazzel	U2423800E	SC2002	FDAD	dazzel 23/04/2025
Zhang Yuhe	U2422060C	SC2002	FDAD	Yuhe 23/04/2025
Mohamed Fahath Mohammed Adhil	U2423664G	SC2002	FDAD	Adhil 23/04/2025
Ng Zheng Da	U2322077H	SC2002	FDAD	Zhengda 23/04/2025
Huang Yitian	U2423017H	SC2002	FDAD	天 23/04/2025

Build-To-Order (BTO) Management System

1. Requirement Analysis & Feature Selection

1.1 Understanding the Problem & Requirements

We began by reading through the BTO document line-by-line, highlighting all use cases and system requirements. From this, we identified the main problem domain to be a centralised CLI-based system for applicants and HDB staff to view, apply for, and manage BTO projects.

We identified three distinct user roles:

1. Applicant: Regular users who view, enquire about, and apply for BTO projects
2. HDB Officer: Staff who respond to enquiries and process flat bookings
3. HDB Manager: Staff who create and manage BTO projects and approve applications

Explicit requirements include:

- User login via Singpass (NRIC and password)
- Role-based dashboard with access control and capabilities specific to each user type
- BTO project listing, filtering and visibility control
- Flat application system that follows eligibility rules and application status tracking
- Enquiry submission, management and response
- Flat booking by HDB Officers
- Project management by HDB Managers
- HDB Officer registration and approval workflow
- Receipt and report generation

Implicit expectations include:

- Intuitive CLI with clear menus and prompts to support user-friendly navigation
- Efficient filtering and searching mechanisms

Ambiguities and assumptions made:

- Workflow for flat booking after a successful application wasn't specified; we assumed that bookings are implemented through the Officer UI but applicants will be informed of the officer responsible for their booking.

- User filtering mechanism wasn't specified; we assumed user filter settings persist across different menus per session.
- Exact filtering options weren't specified; we decided upon the following: location, flat type, application opening and closing dates, manager(s) and officer(s).

1.2 Deciding on Features & Scope

We grouped features into three categories: core features explicitly required, optional features which we added to improve the system design, and more complex features which we excluded due to time constraints (but can be considered for future development).

Core Features	Optional Features	Excluded Features
● User authentication ● Project creation and viewing ● Project application system with status tracking ● HDB Officer registration and approval workflow ● Flat booking system with inventory management ● Enquiry and response system ● Receipt and report generation ● Role-based user interfaces	● View user profile ● Notifications ● Persistent filter settings	● Real-time booking availability updates ● Waitlisting and automated reallocation

A key additional feature we implemented to improve the usability and interactivity of our system is the notifications functionality. Applicants will be notified upon login of changes to their application status or new responses to their enquiries. This ensures users remain informed without needing to manually check their status. HDB Officers and Managers will be notified of pending tasks such as new applicant bookings to process or enquiries that require a response. This helps prioritise tasks and streamline daily workflows immediately upon system access.

2. System Architecture & Structural Planning

2.1 Planning the System Structure

We adopted a layered architecture structure inspired by the Model-View-Controller pattern, with Entity classes representing the model, Boundary classes serving as the view, and Controller classes handling the application logic.

Layer	Function	Classes

Entity	Domain objects	Applicant, Application, Enquiry, Filter, Flat, HDBManager, HDBOfficer, Project, Report, User
Controller	Business logic; mediates between boundary and entity objects	ApplicationController, AuthenticationController, EnquiryController, FilterController, ManagerController, OfficerController, ProjectController
Boundary	User interface and interaction	ApplicantUI, HDBManagerUI, HDBOfficerUI, UserUI
Utility	Common functions	CSVFileHandler

This structure encourages low coupling and high cohesion between classes as each layer has a clear, focused responsibility. The separation reduces interdependencies between layers, allowing for easier maintenance by isolating changes in one layer, preventing them from affecting others. For example, changes to the UI implementation would only impact the Boundary layer.

2.2 Reflection on Design Trade-offs

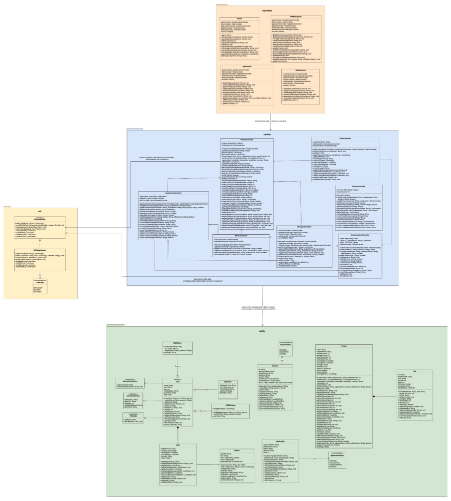
During the planning stage, there were a few design decisions that required our consideration:

1. We debated between creating a single UI class with conditional logic based on user roles or separate role-specific UI classes. We eventually chose the latter to improve cohesion and reduce coupling, making the system more maintainable and allowing for role-specific UI changes without affecting other roles.
2. Since persistence between sessions wasn't required, we opted for static in-memory objects to manage filter states, login sessions, and current user context, simplifying state management without external storage.
3. We debated whether to model Flat as a separate entity class or simply as attributes within the Project class. Creating a distinct Flat class increased the complexity of our model, but provided better encapsulation and enabled us to track individual flat statuses more effectively.

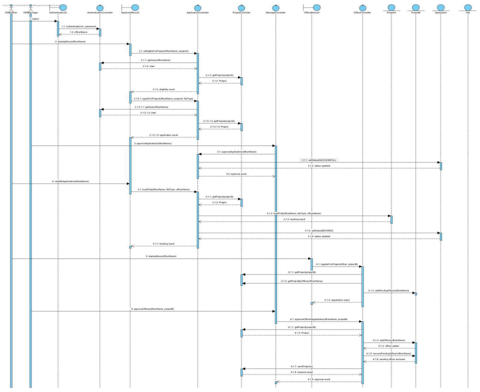
3. Object-Oriented Design

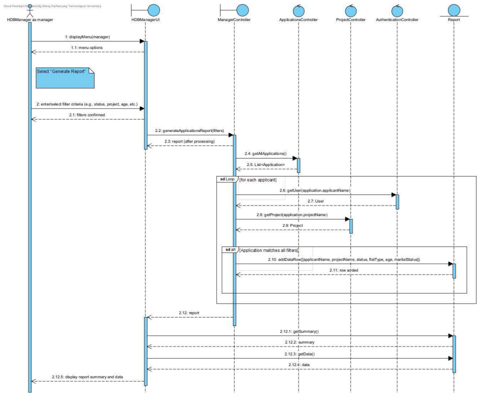
3.1 UML Class Diagram

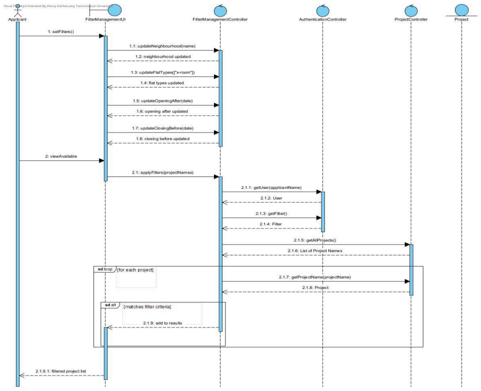
We identified the main classes based on user roles (Applicant, Officer, Manager), key nouns (e.g., Project, Application), and system use cases (e.g., login, apply, book, manage). In terms of considering the relationships between classes, we used inheritance used for user roles, composition for tightly coupled parts (e.g. user and filter), and associations and dependencies modeled where classes use or reference others (e.g. controllers use entities). We prioritized clarity and modularity in the design of our system. We also introduced interfaces and composition to support flexibility. Although more complex, we accepted it for better abstraction and maintainability.



3.2 UML Sequence Diagrams




HDBManager as manager
HDBManagerUI
ManagerController
ApplicationsController
ProjectController
AuthenticationController
Report
1: displayMenu(manager)
1.1: menu options
Select "Generate Report"
2: enter/select filter criteria (e.g., status, project, age, etc.)
2.1: filters confirmed
2.2: generateApplicationsReport(filters)
2.3: report (after processing)
2.4: getAllApplications()
2.5: List<Application>
sd Loop
[for each applicant]
2.6: getUser(application.applicantName)
2.7: User
2.8: getProject(application.projectName)
2.9: Project
sd alt
[Application matches all filters]
2.10: addDataRow([applicantName, projectName, status, flatType, age, maritalStatus])
2.11: row added
2.12: report
2.12.1: getSummary()
2.12.2: summary
2.12.3: getData()
2.12.4: data
2.12.5: display report summary and data


Applicant
FilterManagementUI
FilterManagementController
AuthenticationController
ProjectController
Project
1: setFilters()
1.1: updateNeighbourhood(name)
1.2: neighbourhood updated
1.3: updateFlatTypes(["x-room"])
1.4: flat types updated
1.5: updateOpeningAfter(date)
1.6: opening after updated
1.7: updateClosingBefore(date)
1.8: closing before updated
2: viewAvailable
2.1: applyFilters(projectNames)
2.1.1: getUser(applicantName)
2.1.2: User
2.1.3: getFilter()
2.1.4: Filter
2.1.5: getAllProjects()
2.1.6: List of Project Names
sd loop
[for each project]
2.1.7: getProjectName(projectName)
2.1.8: Project
sd alt
[matches filter criteria]
2.1.9: add to results
2.1.9.1: filtered project list

While designing the system, we identified several key scenarios that represent core and complex functionalities central to the user experience and system architecture. To illustrate them, we created 3 separate UML Sequence Diagrams highlighting different scenarios. The LHS diagram shows a HDB officer applying for own flat and registering as an officer for another project, illustrating an officer acting as both applicant and officer, interacting with multiple controllers and demonstrating conditional logic. The 2nd diagram shows a manager generating a filtered applications report, highlighting the system's report generation logic, involving data aggregation, filtering, and output. The last diagram shows an applicant searching and filtering projects, illustrating the search and filtering logic, a core feature for user experience.

3.3 Application of OOD Principles (SOLID)

Single Responsibility Principle

Each class has a clear, single responsibility. For instance, the ProjectController class handles only project-related operations whereas the EnquiryController class handles only enquiry-related operations. The separation of concerns increases cohesion and improves code maintainability by ensuring changes to one class doesn't affect other classes.

Open/Closed Principle

We designed each class to be open for extension but closed for modification. For example, our User class is abstract and defines common behaviour but allows for specialised implementations as seen in Applicant, HDBOfficer, and HDBManager subclasses. We extended the subclasses without modifying the base User class whenever we needed to add role-specific functionality. This improves code extensibility as we can easily add new functions while minimising changes that need to be made to existing code as a result.

Liskov Substitution Principle

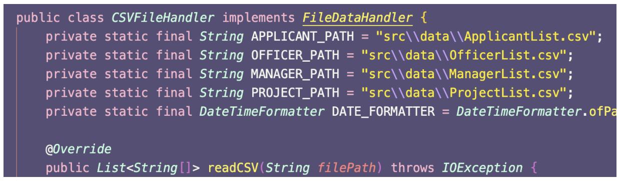
Our design ensures that subclasses are substitutable for their parent classes without affecting program functionality. For example, Applicant, HDBOfficer and HDBManager all extend User, and any method that expects a User can work with any of the subclasses appropriately. We did this by ensuring that for any instance of method overriding in the subclass, its expected precondition and postcondition are not stronger or weaker than in the base class respectively. This improves code maintainability.

Interface Segregation Principle

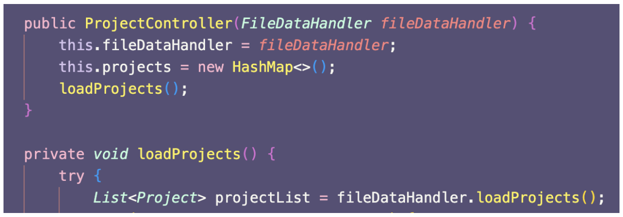
We created focused interfaces for specific functionalities rather than fat interfaces. This ensures no class is forced to implement method(s) it does not need. For example, we created specific interfaces like UserAuthentication for login functionality and ProjectManagerController for project management functionality, preventing classes from implementing unnecessary methods. This improves cohesion and reduces unnecessary dependencies, improving code maintainability.

Dependency Inversion Principle

The High-level ProjectController class does not depend on the low-level CSVDataHandler class; both depend on the FileDataHandler interface. This reduces coupling between modules. This also improves code flexibility and extensibility as we can easily create alternative implementations (such as if we want to handle XML file data instead) without modifying the high-level code.



```java
public class CSVFileHandler implements FileDataHandler {
    private static final String APPLICANT_PATH = "src\\data\\ApplicantList.csv";
    private static final String OFFICER_PATH = "src\\data\\OfficerList.csv";
    private static final String MANAGER_PATH = "src\\data\\ManagerList.csv";
    private static final String PROJECT_PATH = "src\\data\\ProjectList.csv";
    private static final DateTimeFormatter DATE_FORMATTER = DateTimeFormatter.ofPa

    @Override
    public List<String[]> readCSV(String filePath) throws IOException {
```



```java
public ProjectController(FileDataHandler fileDataHandler) {
    this.fileDataHandler = fileDataHandler;
    this.projects = new HashMap<>();
    loadProjects();
}

private void loadProjects() {
    try {
        List<Project> projectList = fileDataHandler.loadProjects();
```

4. Implementation

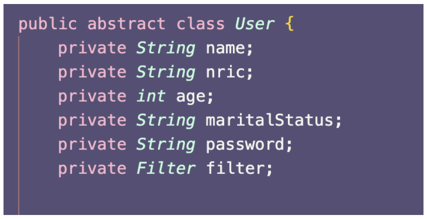
4.1 Tools Used

- Java 23
- IDE: Visual Studio Code / IntelliJ IDEA
- Version Control: GitHub
- UML: Visual Paradigm

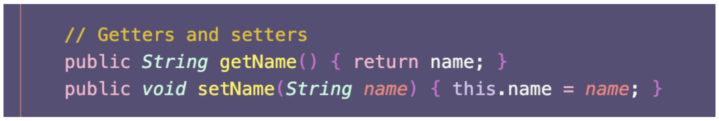
4.2 Sample Code Snippets

Encapsulation

All attributes are private, and can only be accessed or modified using getters and setters. This ensures data integrity (e.g. password can only be changed through proper methods) and that implementation details are hidden from other classes.



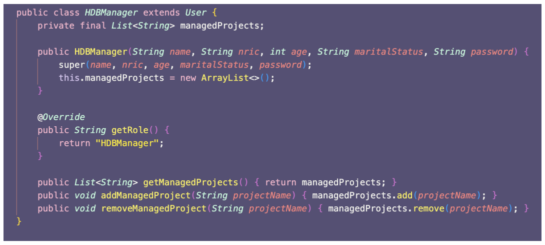
```java
public abstract class User {
    private String name;
    private String nric;
    private int age;
    private String maritalStatus;
    private String password;
    private Filter filter;
```



```java
    // Getters and setters
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
```

Inheritance

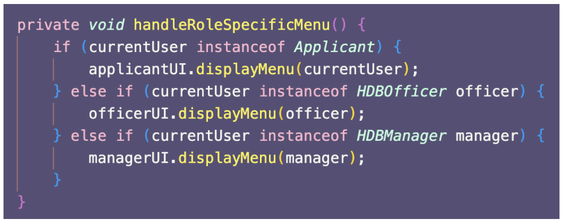
The User class serves as the base class, and each specific user role class extends from it. This enables code reuse as common attributes and methods defined in the User class are inherited by all subclasses. Role-specific attributes and methods can be further added to each subclass.



```java
public class HDBManager extends User {
    private final List<String> managedProjects;

    public HDBManager(String name, String nric, int age, String maritalStatus, String password) {
        super(name, nric, age, maritalStatus, password);
        this.managedProjects = new ArrayList<>();
    }

    @Override
    public String getRole() {
        return "HDBManager";
    }

    public List<String> getManagedProjects() { return managedProjects; }
    public void addManagedProject(String projectName) { managedProjects.add(projectName); }
    public void removeManagedProject(String projectName) { managedProjects.remove(projectName); }
}
```

Polymorphism

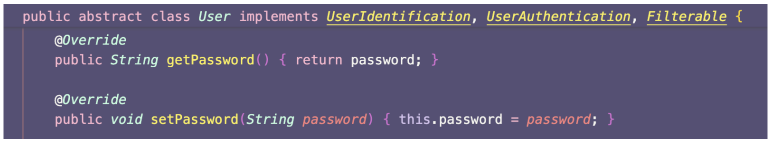
Our system is able to adapt its behaviour according to the specific role of the user. This improves code flexibility and maintainability as it allows us to add new user types in the future without changing the fundamental structure of the code.



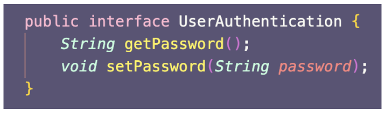
```java
private void handleRoleSpecificMenu() {
    if (currentUser instanceof Applicant) {
        applicantUI.displayMenu(currentUser);
    } else if (currentUser instanceof HDBOfficer officer) {
        officerUI.displayMenu(officer);
    } else if (currentUser instanceof HDBManager manager) {
        managerUI.displayMenu(manager);
    }
}
```

Interface Use

Instead of creating large interfaces with many unrelated methods, we defined smaller, specific interfaces. This lowers coupling, increases cohesion and improves code maintainability.



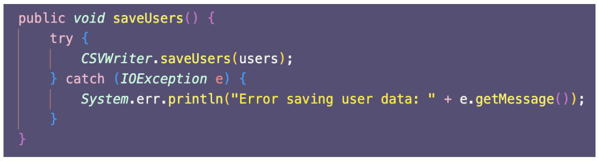
```java
public abstract class User implements UserIdentification, UserAuthentication, Filterable {
    @Override
    public String getPassword() { return password; }

    @Override
    public void setPassword(String password) { this.password = password; }
```



```java
public interface UserAuthentication {
    String getPassword();
    void setPassword(String password);
}
```

Error Handling

We ensured our system offers proper exception handling for better robustness.



```java
public void saveUsers() {
    try {
        CSVWriter.saveUsers(users);
    } catch (IOException e) {
        System.err.println("Error saving user data: " + e.getMessage());
    }
}
```

5. Testing

5.1 Test Strategy

We used manual functional testing — specifically black box testing — to test whether the functionality of our system meets the requirements. We created detailed test cases to cover a variety of possible scenarios, including normal operations, edge cases, and error conditions.

5.2 Test Case Table

No.	Test Case	Test Steps	Expected Result	Actual Result	Status
1	Valid User Login	1. Enter valid NRIC 2. Enter correct password	User successfully logs in and views appropriate dashboard	As expected	Pass
2	Invalid NRIC Format	1. Enter invalid NRIC 2. Enter any password	System displays error message	As expected	Pass
3	Invalid Password	3. Enter valid NRIC 4. Enter incorrect password	System displays error message	As expected	Pass
4	Password Change	1. Login as user 2. Change password 3. Re-login with new password	User successfully logs in with new password	As expected	Pass
5	Project Visibility - Single User	1. Login as a single user (age 35) 2. View available projects	Only 2-Room projects with visibility ON are displayed	As expected	Pass
6	Age Restriction	1. Login as single user (age 25) 2. View available projects	No projects displayed	As expected	Pass
7	Project Visibility - Married User	1. Login as a married user 2. View available projects	Both 2-Room and 3-Room projects with visibility ON displayed	As expected	Pass
8	Project Application - Married User	1. Login as a married user (age 25) 2. Apply for Flat	Able to choose between 2-room and 3-room flat type	As expected	Pass
9	Project Application - Single User	1. Login as single user 2. Apply for flat	Unable to choose flat type (automatically 2-room flat)	As expected	Pass
10	HDB Officer Cannot Apply for Handled Project	1. Login as HDB Officer 2. Register to handle a project 3. Apply for project	Registered project not in list of applicable projects shown	As expected	Pass
11	HDB Officer Cannot Register for Applied Project	4. Log in as HDB Officer 5. Apply for a project 6. Attempt to register to handle that project	System prevents registration and error message is shown	As expected	Pass
12	Project Application and Status	1. Login as applicant 2. Apply for eligible project	Applicant sees PENDING status	As expected	Pass

	Tracking	3. Check application status			
13	Application Approval by Manager	1. Login as HDB Manager 2. View pending applications 3. Approve application 4. Login as applicant 5. Check application status	Applicant sees SUCCESSFUL status	As expected	Pass
14	Flat Booking After Successful Application	1. Login as HDB Officer 2. Process booking of successful applicant 3. Login as applicant 4. Check application status	System updates flat inventory and application status to BOOKED	As expected	Pass
15	Application Rejection by Manager	1. Login as HDB Manager 2. View pending applications 3. Reject application 4. Log in as applicant 5. Check application status 6. Attempt to apply for same project	System updates application status to UNSUCCESSFUL	As expected	Pass
16	Project Creation by Manager	1. Login as HDB Manager 2. Create new BTO project 3. Check project listing	New project appears in the system with correct details	As expected	Pass
17	Project Editing by Manager	1. Login as HDB Manager 2. Edit existing project's details 3. Verify changes	Project details correctly updated	As expected	Pass
18	Project Deletion by Manager	1. Login as HDB Manager 2. Delete existing project 3. Verify changes	Project is removed from the system	As expected	Pass
19	Officer Registration for Project	1. Login as HDB Office 2. Register to handle a project 3. View projects	Project visible with "pending approval" status	As expected	Pass
20	Approve Officer Registration	1. Login as HDB manager 2. View pending officer registration 3. Approve an officer's registration 4. Login as that HDB officer 5. View projects	Project visible as assigned project and officer is assigned to the project	As expected	Pass
21	Filtering Projects	1. Login as applicant 2. Select a filter (e.g. location) 3. View projects	Only projects in selected location are displayed	As expected	Pass
22	Filter Persistence Between Menus	1. Login as applicant 2. Select a filter 3. Logout and re-login as that applicant 4. View projects	Filter settings remain applied	As expected	Pass
23	Enquiry Submission and Response	1. Login as applicant 2. Submit enquiry 3. Log in as HDB Officer	Applicant notified of response and can see officer's response to enquiry	As expected	Pass

		4. Respond to enquiry 5. Login as applicant 6. View response			
24	Enquiry Editing	1. Login as applicant 2. Submit enquiry 3. Edit enquiry 4. View updated enquiry	Enquiry text is updated correctly	As expected	Pass
25	Enquiry Deletion	1. Login as applicant 2. Delete enquiry 3. Edit enquiry 4. View enquiries	Enquiry is no longer visible	As expected	Pass
26	Enquiry editing after response	1. Login as applicant 2. Try to edit replied enquiry	Unable to edit enquiry, error message displayed	As expected	Pass
27	HDB Officer Slot Limit Check	1. Login as a HDB Officer 2. Attempt to register for project with no empty slots	Unable to register, error message displayed	As expected	Pass
28	Report Generation with Filters	1. Login as HDB Manager 2. Generate report with filter 3. View report	Report shows only data matching the specified filters	As expected	Pass
29	Application Withdrawal Before Booking	1. Login as applicant with "SUCCESSFUL" status 2. Request withdrawal 3. Login as HDB Manager 4. Approve withdrawal 5. Login as applicant 6. Check application status	Application status updated to UNSUCCESSFUL	As expected	Pass
30	Application Withdrawal After Booking	1. Login as applicant with BOOKED status 2. Request withdrawal 3. Login as HDB Manager 4. Approve withdrawal 5. Login as applicant 6. Check application status 7. Check flat inventory	Application status updated to UNSUCCESSFUL, flat inventory updated and applicant can apply for new projects	As expected	Pass
31	Visibility OFF - HDB Officers	1. Login as HDB manager and toggle visibility of a registered project to OFF 2. Login as HDB Officer	Registered project with visibility OFF remains visible in HDB officer menu (not visible in applicant menu)	As expected	Pass
32	Visibility OFF - Applicant	3. Login as HDB manager and toggle visibility of a project to OFF 4. Login as applicant 5. View projects	Project with visibility OFF not visible	As expected	Pass
33	Notification	1. Login as applicant who has applied for project or as HDB officer or manager with applications or	Upon login, relevant notifications shown	As expected	Pass

		enquiries to handle			

6. Reflection

Our team did well in adopting a structured approach towards the system design and development. Our team's systematic methodology—analyzing requirements thoroughly before implementation—prevented major redesigns later. The layered architecture kept the codebase organized and made collaboration smoother. We were able to effectively apply OOD principles (SOLID) to reduce coupling and increase cohesion of modules. Creating UML diagrams before coding also helped visualize object interactions and clarify responsibilities.

However, we also faced challenges. Despite efforts to design a clean architecture, there were areas where coupling between components was higher than ideal. Some controllers had dependencies on multiple entity classes, reducing code maintainability. Our greatest setback was time constraints; some trade-offs were necessary, and we lacked the time for more extensive refactoring of tightly coupled components. We should have allocated more time for code review and refactoring to improve the quality of our code.

This experience allowed us to learn many things:

1. Planning is essential; taking our time to look through the requirements and plan the system design can reduce the amount of changes that need to be made in later stages.
2. Learned how to put SOLID design principles to practice to improve maintainability of code.
3. Testing is essential to development; developing test cases alongside features helped ensure functionality met requirements and identified edge cases we might have otherwise missed.

7. Appendix

- GitHub Repository: https://github.com/JamesHackerMP/SC2002-Group-Project/
- Refer to README.md for developer guide