




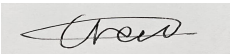

Project Title: Build-To-Order (BTO) Management System

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 | Course | Student ID | Signature/Date |
|--------------------|--------|------------|---|
| Ching Xin Wei | SC2002 | U2421032K |  |
| Lay Wei Jia | SC2002 | U2422945A |  |
| Li Qi Yue | SC2002 | U2421738A |  |
| Chew Shao Hong | SC2002 | U2420411H |  |
| Lee Ze Quan Joshua | SC2002 | U2423560B |  |

Important notes:

1. Name must EXACTLY MATCH the one printed on your Matriculation Card.
2. Student Code of Academic Conduct includes the latest guidelines on usage of Generative AI and any other guidelines as released by NTU

Chapter 1: Requirement Analysis & Feature Selection

1.1 Understanding the Problem and Requirements

We began by reading through the BTO document line-by-line, highlighting all use cases and system requirements. Based on this, we created a list of essential features and identified user roles and system entities such as Applicant, Officer, Manager, Project, Application, Registration, and Enquiry.

The main domain we identified was the management of Build-To-Order (BTO) housing projects, which included handling applications to these BTO projects from Applicants, Officer registering to be Officer-in-charge of selected projects, project creation and set-up, enquiry response and flat booking approval.

From the document, we extracted key explicit requirements:

- User login based on role (Applicant, Officer, Manager)
- Login authentication by NRIC and default password
- Role-specific display menus
- Applicants can apply for BTO projects, withdraw applications and send enquiries.
- Officers can register as OIC, respond to enquiries, and book flats for Applicants once their application is approved by Managers.
- Managers can create/edit projects, approve/reject applications, and view reports of applicants' applications.
- Data should be stored in respective CSV files for reading and loading as no database application or JSON or XML are to be used.

Ambiguities and our interpretation:

Whether Manager is allowed to edit project details after applications begin?

Ambiguity: The manager can “edit project details,” but there’s no mention of restrictions once applications are underway.

Our Approach: We allowed editing at any time, but in future extensions, locking certain fields, like price of flats and the location, in post-application could improve realism.

Report Filtering details

Ambiguity: The document mentions “various categories (e.g., flat type, marital status),” but does not list them exhaustively.

Our Approach: We implement filtering by:

1. Age groups (under 25, 25-35, etc)
2. Flat type (2-Room vs. 3-Room)

3. Marital status (Married/Single)
4. Project Name

Data storage

Ambiguity: The document allows CSV but doesn't specify how data should be structured or restored across sessions.

Our Approach: We built a centralized FileService with load and write methods for each entity from their respective CSV files to store and read data.

1.2 Deciding on Features and Scope

We grouped the features into three categories, the core features, bonus features and excluded features. The core features are features essential to the running of the program and its main requirements, the bonus features are those that are not necessary but useful if implemented, while the excluded features are those that were left out because of complexity and lack of time. Below are the implemented features grouped into their respective categories:

Core features:

- User login and password management
- Role-specific display menus
- Enquiry viewing, creation, editing, deletion and response (by Officers and Managers)
- Project viewing, creation, editing and deletion
- Project application for Applicants
- Project registration as Officer
- Officer flat booking for Applicants
- Generation of receipts and reports for Officer and Manager respectively
- Filters for projects and reports
- List manager for projects, applications, reports, officer registrations

Optional / Bonus features:

- Exporting reports to files

Excluded features:

- Password and NRIC encryption

We began by building a portion of the frontend first, focusing on the user display boundary classes with switch statements to identify the exact methods needed in the controller classes and define the correct return types. This allowed us to implement the backend logic in the controllers, which fetch the correct objects from

the repositories and process the data before returning it to the boundary for display. We prioritized implementing the core features and critical use cases first, such as the Manager adding a project and the Applicant applying for a project, ensuring the main functionality was working. Once these features were tested, we moved on to extend the system with additional use cases, like the Applicant making an enquiry about their applied project. This approach allowed us to systematically build and test the system, ensuring foundational features were functional before adding more complex ones.

To give more detail into the prioritisation of the core features, we split them based on the User roles with different members working on them. We prioritised the ones that were more specific to the User roles, like project creation with Manager, while features that overlapped with different roles like the booking system, we implemented later after further clarification. After implementing the essential core features, we worked on the bonus features and implemented them as time permitted.

For the excluded feature of password and NRIC encryption, although this may be essential when implemented in actual real-world setting, we decided to leave this feature as a possible extension of the system as we mainly focused on delivering the functional requirements first.

Chapter 2: System Architecture & Structural Planning

2.1 Planning the System Structure

We began by identifying core roles such as Applicant, HDB Officer, HDB Manager and key entities involved such as Project, Application, Registration and Enquiry. From there, we decomposed the system by separating them into Boundary, Control, Entity and Repository classes to maintain clear separation:

- Entity Classes (e.g., Project, Application, User, Enquiry) encapsulate the core data.
- Control Classes (e.g., Applicant Controller, Officer Controller) contain workflow logic and coordinate between entities and repositories.
- Boundary Classes(e.g., ApplicantDisplay, ManagerDisplay) handling all input/output and interact with users via CLI.

2.2 Reflection on Design Trade-offs

At first, we considered having both the methods and attributes for each user to be in the same class for simplicity. However, after further consideration, we decided to split the controller and logic layer to promote maintainability and organisation. Organising the methods by making things separate for the different types of methods like output and logic, made the code easier to read and modify the class methods for any extensions, especially as the program grew in size.

However, it did mean that we needed to be aware of which specific class the methods were in to be able to call them, which could be confusing when needing to implement code utilising such methods. Hence, we made sure not to go overboard with creating too many type-specific classes, making sure to keep it simple.

Chapter 3: Object-Oriented Design

3.1 Class Diagram (with Emphasis on Thinking Process):

Design Thinking Summary: Class Identification

We identified three main categories of classes:

- User Classes: Abstract User with concrete subclasses Applicant, Officer, and Manager, derived from user roles in requirements.
- Entity Classes: Project, Application, Enquiry, Registration, Report—based on recurring elements in workflows and documents.
- Utility/Service Classes: Supporting roles like FileService, LoginController, and repositories.

Design Structure & Responsibilities

We followed a layered architecture with clear separation:

- Entity classes store core data (e.g., Project holds prices, assigned officers).
- User classes manage login/display logic. Officer inherits from Applicant for shared functionality.
- Controllers coordinate logic, accessing repositories and returning results to displays.
- Repositories centralize data access (e.g., ApplicationRepository).
- Reports are utilities combining data from multiple sources; omitted in the core UML for clarity.

Relationships & Justifications

Inheritance: Officer and Manager extend Applicant/User to promote reuse.

Aggregation: Projects reference Applicants and Officers, but not as owned objects (non-composite).

Associations: Application, Enquiry, etc., link User and Project without managing their lifecycles.

Trade-Offs

Chose a hybrid logic structure: centralized via repositories, distributed via user displays.

Prioritized clarity over completeness by omitting transient or internal associations.

Focused on encapsulation and separation of concerns to improve maintainability and scalability.

3.2 Sequence Diagrams (with Emphasis on Thinking Process)

Design Thinking Process: Why These Diagrams?

The two main use cases demonstrated are officer registering for a project and officer applying for a project as an applicant. These use cases illustrate the overall logic flow and core functionalities of the system, specifically how officers interact with the project registration/application processes. We have included the preceding sequences, such as the officer logging in and the display of Officer and Applicant Menus, as they are vital steps before he can register or apply for a project.

Sequence Diagram 1: User Login (LoginDisplay.showDisplay())

This use case was selected because authentication is a fundamental and mandatory process all users undergo. It verifies that only existing officers in the database can access the officer menu, roles and responsibilities, ensuring security and also prevents potential bugs.

The error handling to trigger the reprompting of login details upon invalid details or choice is a user friendly design feature of the system as it does not automatically shut down the program should the user accidentally type in wrong details.

The key feature demonstrated is in the function call `user.display()` at the end of the sequence. `loginAuthenticator` returns either an applicant, officer or manager object, upcasted to a `User` type. This is the abstract superclass of all 3 user objects, which implement the `user.display()` method. By calling the respective user display classes through `user.display()`, new user types can be added in the future through extending the abstract `User` class, without edits to the `LoginDisplay` (e.g. if user instanceof `[newUserType]`...). This ensures clean, secure role-based access control, displaying the correct menu based on the logged-in user's role (e.g., an Applicant cannot access Officer or Manager options).

Sequence Diagram 2: OfficerDisplay.showDisplay() and Sequence Diagram 3: ApplicantDisplay.showDisplay()

In Sequence Diagram 2, the system uses inheritance (`Officer` extends `Applicant`) to switch between roles. If the Officer role is chosen, the Officer menu is displayed and managed by the `OfficerController`. In Sequence Diagram 3, if the Applicant role is chosen, the Applicant menu is displayed, and control is passed to the `ApplicantController`. Both diagrams show role-based access control and a clean separation of responsibilities across classes, making the system flexible and easy to maintain.

Sequence Diagram 4: Registering as project OIC(OfficerController.registerAsOIC)

This use case demonstrates the boundary, controller, entity class design of the system. It shows the interaction between `OfficerController`, `ProjectController`, `RegistrationController`, `ApplicationController` (controllers), and `ProjectRepository`, `RegistrationRepository`, `ApplicationRepository` (entities) to validate and process the officer's registration request, with prompts and error or success messages displayed through `OfficerDisplay` (boundary).

The sequence includes conditional checks for:

1. Project Slot Availability: The controller filters available projects from the repository
2. Existing Registration: The controller ensures the officer isn't already registered
3. Role Validation: The controller checks if the officer has a pending registration
4. Time-based Validation: If officer has a registration, the controller ensures no date conflicts between projects

These validations are handled individually, with clear error messages returned to OfficerDisplay, ensuring accurate processing of the registration request.

Sequence Diagram 5: Displaying the project list (ApplicantController.viewProjects())

This use case demonstrates the filtering logic, as the list of all projects from the project repository is looped through and filtered based on Flat Type (dependent on Officer's age and marital status), Visibility (if off, it will not be displayed even if the officer is assigned to the project) and Assignment (if the officer is currently assigned to the project). The second and third conditions also act as a role based access control, as the officer is currently acting as an applicant, so their view and access to projects will be filtered to exclude what they might see when acting as an officer. After displaying the project list, the officer is prompted to select a project, before the applyProject() method is called (Sequence Diagram 6).

Sequence Diagram 6: Officer Applying for Project (applyProject(selected, flatTypeChoice))

This sequence performs conditional checks before application submission. It verifies that the applicant has no existing application and ensures the officer is not registered to the project (if the user is an officer). The Boundary-Controller-Entity model is prominent here, as OfficerController calls repository controllers (e.g., ProjectRepository, ApplicationRepository, RegistrationRepository) to retrieve information from repositories (entities). The controller then checks the request against the data and displays either an error message or a success message on the boundary (ApplicantDisplay).

3.3 Application of OOD Principles (SOLID)

1. Single Responsibility Principle (SRP)

SRP states that there should never be more than one reason that a class should change, and a class should handle only a single responsibility. In our design, the methods for each class are separated into distinct classes for each responsibility like Controller which holds the logic methods while Display holds the output methods for each class. For instance, our LoginController handles the logic for the login process, while LoginDisplay handles the output display that tells the user to input their details and such. This ensures greater maintainability and adheres to SRP.

2. Open-Closed Principle (OCP)

OCP states that a module should be open for extension but closed for modification. Our abstract User class follows this by allowing Applicant, Officer and Manager to extend functionality through inheritance, e.g. Officer inherits from Applicant and adds project registration without editing base user code, this design allows us to introduce future new roles without altering login or menu logic, showing extensibility.

3. Liskov Substitution Principle (LSP)

LSP states that subtypes must be substitutable for their base types, meaning that a user of the base class should continue to function properly if a derivative of the base class is passed to it. For instance, in our implementation, the Officer class extends the Applicant class, but taking into consideration the exception in which an Officer cannot apply for a project as an Applicant if it is a project he is already handling as an Officer. We made sure to implement this exception in the Applicant code logic for applying for projects. As such, when the Officer extends the Applicant class, its apply for project method adheres to the base class without needing to change anything, making it easy to implement the Applicant capabilities in Officer and fulfilling LSP.

4. Interface Segregation Principle (ISP)

ISP states that classes should not depend on methods that they do not use, only using specific class interfaces instead of a general all purpose interface. In our implementation, we applied ISP through our UserDisplay interface. This interface defines only the shared methods that are relevant to all user roles such as showDisplay() and changeUserPassword(). Each display class such as ApplicantDisplay, OfficerDisplay and ManagerDisplay implements this interface and only relies on what it actually uses. By avoiding a generic all purpose interface, we made sure each role does not implement methods that they do not use.

5. Dependency Injection Principle (DIP)

DIP states that a) high level modules should not depend upon low level modules, both should depend upon abstractions and b) abstractions should not depend upon details, details should depend upon abstractions. We applied this by using a UserDisplay interface with showDisplay() method, allowing the system to interact with all user roles through the interface rather than specific display classes, making our design more extensible.

Chapter 4: OOP Principles (with sample code)

Below are some snippets of our source code that demonstrate OOP principles in our system design

Encapsulation

Encapsulation is shown in how ApplicantController accesses private fields of Applicant, Application through getters like applicant.getApplication and

```
public void applyProject(Project project, int flatTypeChoice) {
    if (applicant.getApplication() != null) {
        System.out.println(x:"Cannot apply for a second project.");
        return;
    }
    //if officer, check if registered for the project
    if (applicant instanceof Officer) {
        Officer o = (Officer) applicant;
        if (RegistrationController.isOfficerAlreadyPending(project,o)){
            System.out.println(x:"You cannot apply for a project you are registered for.");
            return;
        }
    }

    Application app = ApplicationController.addApplication(applicant, project, flatTypeChoice);
    System.out.println("Successfully applied for " + project.getName());
    app.print();
}
```


project.getName(), rather than accessing directly. It also interacts with other controllers via public methods like isOfficerAlreadyPending(), without handling internal repository details, protecting data and keeping logic modular and maintainable.

Inheritance

In our system, User is the base class, inherited by Applicant, which is further inherited by Officer. They all share common attributes (e.g., name, NRIC) while adding role-specific ones like application details for Applicant and assigned project for Officer, this reduces code duplication and allows easy future extensions.

```
public abstract class User {
    protected String name;
    protected int age;
    protected boolean married;
    protected String password;
    protected String nric;
}

public class Officer extends Applicant {
    private Project assignedProject;
}

public class Applicant extends User {
    private Application application;
}
```

Polymorphism

Polymorphism is used by upcasting all users to User created in LoginController, then calling the respective overridden display() method, allowing new user types to be added without changing login logic.

Interface Use

We use the UserDisplay interface to define showDisplay() and a default changeUserPassword() method. This provides a shared structure for all user roles and avoids repeating logic.

Error Handling

Login errors are handled gracefully by catching invalid NRIC or password attempts without throwing exceptions. It shows an error message, tracks remaining attempts and exits after too many failed attempts, ensuring smooth authentication without disrupting user flow.

```
public class LoginDisplay {
    public static void showDisplay(Scanner scanner) {
        String role = LoginController.checkRole(scanner);
        scanner.nextLine();
        User user = null;
        int attempts = 3;
        while (attempts > 0 && user == null) {
            user = LoginController.loginAuthenticator(scanner, role);
            if (user == null) {
                attempts--;
                if (attempts > 0) {
                    System.out.println("Attempts remaining: " + attempts);
                } else {
                    System.out.println(x: "Too many failed attempts. Exiting application.");
                    return;
                }
            }
        }
        System.out.println("Login Successful for " + user.getName());
        user.display(); //allows for new users to be created without amending this class
    }
}
```

```
public interface UserDisplay {
    public void showDisplay();

    default void changeUserPassword(Scanner scanner, User user) {
        System.out.print(s: "Enter current password: ");
        String currentPassword = scanner.nextLine().trim();
        System.out.print(s: "Enter new password: ");
        String newPassword = scanner.nextLine().trim();
        if (UserController.changePassword(user, currentPassword, newPassword)) {
            System.out.println(x: "Password changed successfully.");
        } else {
            System.out.println(x: "Password change failed. Please check your current password and try again.");
        }
    }
}
```

```
if ("Applicant".equalsIgnoreCase(role)) {
    Applicant applicant = ApplicantRepository.findApplicantByNRIC(nric);
    if (applicant != null && applicant.getPassword().equals(password)) {
        return applicant;
    }
} else if ("Officer".equalsIgnoreCase(role)) {
    Officer officer = OfficerRepository.findOfficerByNRIC(nric);
    if (officer != null && officer.getPassword().equals(password)) {
        return officer;
    }
} else if ("Manager".equalsIgnoreCase(role)) {
    Manager manager = ManagerRepository.findManagerByNRIC(nric);
    if (manager != null && manager.getPassword().equals(password)) {
        return manager;
    }
}
System.out.println(x: "Invalid NRIC or password. Please try again.");
return null;
```

```
while (attempts > 0 && user == null) {
    user = LoginController.loginAuthenticator(scanner, role);
    if (user == null) {
        attempts--;
        if (attempts > 0) {
            System.out.println("Attempts remaining: " + attempts);
        } else {
            System.out.println(x: "Too many failed attempts. Exiting application.");
            return;
        }
    }
}
```

Chapter 5: Testing

5.1 Test Strategy

We used manual functional testing to test our system. We first simulated successful runs of use cases for each user type, ensured correct output and changes saved to persisting data in the csv files. The full test case table demonstrating the functional requirements of the project are all fully met can be found in a separate document. Below, we have selected a few key test cases to show key functionalities and certain edge cases.

5.2 Test Case Table

| Test Case | Expected Behaviour | Failure Indicator | Output | Functionality shown |
|--|--|--|--|---|
| Incorrect Password | Login fails with “Invalid NRIC or password. Please try again” message and “Attempts remaining: “ message | User logs in with a wrong password | <pre> Welcome to the BTO System Please enter your role: applicant / officer / manager (Enter 1, 2 or 3) 1. Applicant 2. HDB Officer 3. HDB Manager 1 Please input NRIC: 51234567A Please input password (default password is 'password'): wrong Invalid NRIC or password. Please try again. Attempts remaining: 2 </pre> | Error handling: Ensures login fails on incorrect credentials and handles multiple attempts gracefully |
| Viewing Application Status after Visibility Toggle Off | Applicants continue to have access to their application details regardless of project visibility. | Application details become inaccessible once visibility is off. | <p>Projects you're in charge of:</p> <pre> 1. Tengah Choose project to toggle visibility of: 1 Project 'Tengah' visibility is set to false </pre> <p>Manager toggling visibility off</p> <pre> ===== Applicant Menu ===== (1) View Projects (2) Apply for Project (3) View Applied Project/View Withdrawal Outcome (4) View Enquiries (5) Create Enquiry (6) Edit Enquiry (7) Delete Enquiry (8) Change Password (9) Withdraw Application (0) Exit 3 ===== BTO Application ===== Application Status: Pending ----- Applicant: Sarah NRIC: T7654321B Age: 40 Marital Status: Married Project: Tengah Flat-type: 2-Room Price: \$10000 Number of units available: 4 ----- Withdrawal Requested: false Withdrawal Status: Pending </pre> <p>Still able to see application details</p> | Access control: Applying to the project gives a “higher access” for viewing Verifies that Applicants retain access to their application status, regardless of visibility settings |
| HDB Officer Registration Eligibility | System allows registration only under compliant conditions | System allows registration while the officer is an applicant or registered | <pre> ===== Officer Menu ===== (1) Register as Officer-in-Charge (2) View Assigned Project (3) View Registration Status (4) Reply to Enquiry (5) Book flat for Applicant (6) Generate Receipt (7) Change Password (0) Exit 1 Available Projects: 1. Azzada Breeze 2. Sky Park 3. Tengah Choose project to request OSC role: 2 Registration submitted and pending manager approval. </pre> <p>Officer registering for a project when all conditions are cleared</p> | Criteria validation: Ensures Officers can only register for projects under valid conditions, preventing duplicate registrations |

| | | | | |
|--|---|--|---|--|
| | | for another project in the same period | <div><div>===== Officer Menu ===== (1) Register as Officer-In-Charge (2) View Assigned Project (3) View Registration Status (4) Reply to Enquiry (5) Book flat for Applicant (6) Generate Receipt (7) Change Password (8) Exit 2 Available Projects: 1. Acacia Breeze 2. Sky Park 3. Tengah Choose project to request OIC role: 2 You have already registered for this project</div><p>Officer registering for a project that they registered for already.</p><div>===== Officer Menu ===== (1) Register as Officer-In-Charge (2) View Assigned Project (3) View Registration Status (4) Reply to Enquiry (5) Book flat for Applicant (6) Generate Receipt (7) Change Password (8) Exit 2 Available Projects: 1. Acacia Breeze 2. Sky Park 3. Tengah Choose project to request OIC role: 2 You cannot register for a project you have applied for</div><p>Officer registering for project that they applied for as an Applicant</p></div> | |
| Project Detail Access for HDB Officer | Officers can always access full project details, even when visibility is turned off. | Project details are inaccessible when visibility is toggled off | <div><div>===== Officer Menu ===== (1) Register as Officer-In-Charge (2) View Assigned Project (3) View Registration Status (4) View Project Enquiries (5) Reply to Enquiry (6) Book flat for Applicant (7) Generate Receipt (8) Change Password (9) Exit 2 Assigned Project: Project Name: Sky Park Neighborhood: Sembawang Visibility: false 2-Room Units: 4 Price of 2-Room units: \$250000 3-Room Units: 5 Price of 3-Room units: \$250000 Application Period: 4/12/2025 to 5/26/2025 Available Officer Slots: 9</div><p>Details still accessible even after visibility is false</p></div> | Role-based access: Ensures HDB Officers retain access to project details regardless of visibility settings |
| Single Project Management per Application Period | System prevents assignment of more than one project to a manager within the same application dates. | Manager is able to handle multiple projects simultaneously during the same period. | <div><p>Manager trying to create and handle a project with the clashing application dates as another project.</p><div><div>===== Manager User Menu ===== (1) Create a project (2) Edit a project (3) Delete a project (4) View all my projects (5) Approve/Reject Officer Registration (6) Approve/Reject Applicant Application (7) View all Enquiries (8) View Project Enquiries (9) Reply to an Enquiry (10) Approve/Reject Withdrawal of App (11) Generate Report (12) Toggle Visibility (13) Change Password (9) Exit Please enter your choice: 1 Enter Project Name: Sunshine Heights Enter Neighborhood: Pioneer Enter Visibility (true/false): true Enter number of 2-Room units: 10 Enter price of 2-Room units: 500000 Enter number of 3-Room units: 5 Enter price of 3-Room units: 400000 Enter Application Opening Date (MM/DD/YYYY): 04/05/2025 Enter Application Closing Date (MM/DD/YYYY): 05/05/2025 Enter available HDB Officer Slots: 10 The new project 2 has been successfully created with the project ID: 2025-04-05-02</div><div>===== Manager User Menu ===== (1) Create a project (2) Edit a project (3) Delete a project (4) View all my projects (5) Approve/Reject Officer Registration (6) Approve/Reject Applicant Application (7) View all Enquiries (8) View Project Enquiries (9) Reply to an Enquiry (10) Approve/Reject Withdrawal of App (11) Generate Report (12) Toggle Visibility (13) Change Password (9) Exit Please enter your choice: 1 Enter Project Name: Sunshine Heights Enter Neighborhood: Pioneer Enter Visibility (true/false): true Enter number of 2-Room units: 10 Enter price of 2-Room units: 500000 Enter number of 3-Room units: 5 Enter price of 3-Room units: 400000 Enter Application Opening Date (MM/DD/YYYY): 04/05/2025 Enter Application Closing Date (MM/DD/YYYY): 05/05/2025 Enter available HDB Officer Slots: 10</div></div></div> | Project management constraints: Ensures Managers cannot handle multiple projects within the same application period |

| | | | | |
|--|--|--|---|--|
| | | | <div>The new project's date range (2025-04-05 to 2025-05-05)</div> <div>overlaps with the project 'Sky Park' date range (2025-04-12 to 2025-05-26).</div> | |
|--|--|--|---|--|

Chapter 6: Reflection & Challenges

• What went well

The code worked as intended with not many bugs to fix. Coordination and communication of tasks was good to ensure we could finish the project on time. Everyone made sure to participate and contribute to the project, providing ideas and feedback.

• What could be improved

Planning behind the implementation of the code could have been more intentional, to better fit the SOLID design principles, making it such that any later changes were easier and not as tedious. Time was a bit of a struggle as the system had many methods that needed to be implemented, which meant the code may not be as polished as we would have liked.

• Individual contributions

Xin Wei, Wei Jia, Joshua, Qiyue: Code Implementation, Report

Shao Hong: Javadoc Code Implementation

• Lessons learned about OODP

We learnt good coding practices by following the SOLID principles, creating Class and Sequence diagrams to help structure the code, and work and communicate as a group to implement the system.

Chapter 7: Appendix

Github Link: <https://github.com/Qiyueyue123/SC2002.git>

Separate submissions (zip everything in a zip file):

1. UML Class Diagram
2. UML Sequence Diagram
3. Test Case Table
4. JavaDocs
5. Source code