



SC2002 - OBJECT ORIENTED DESIGN & PROGRAMMING

Build-To-Order (BTO) Management System

Project Report AY 2024-25/SEMESTER 2

SDDA : Group 3

<u>Team Members</u>	<u>Matriculation Number</u>
Au Yi Teng	U2320100K
Chua Wen Jun	U2321352A
Hasini Lagatapathi	U2421663K
Kieran Mak	U2240494C
Rajkumar Saanvi	U2323082K

Table of Contents

Table of Contents.....	2
1. Declaration of Original Work.....	3
Declaration of Original Work for SC2002 Assignment.....	3
2. System Overview and Feature Planning.....	3
2.1 Understanding the Problem and Requirements.....	3
2.2 Deciding on Features and Scope.....	4
3. Design Considerations.....	4
3.1 Overview of the Approach.....	4
3.2 Assumptions Made.....	4
3.3 Object-Oriented Programming Principles.....	5
3.3.1 Abstraction.....	5
3.3.2 Encapsulation.....	5
3.3.3 Inheritance.....	5
3.3.4 Polymorphism.....	5
3.4 Application of SOLID Design Principles.....	5
3.4.1 Single Responsibility Principle (SRP).....	5
3.4.2 Open-Closed Principle (OCP).....	6
3.4.3 Liskov Substitution Principle (LSP).....	6
3.5 Design Trade-Offs and Extensibility.....	6
4. Additional Features.....	6
5. Detailed UML Class Diagram (uploaded separately for quality).....	7
6. Detailed UML Sequence Diagram (uploaded separately for quality).....	8
7. Testing.....	9
8. Reflection.....	11
8.1 Challenges Faced and How We Overcame Them.....	11
8.2 Key Takeaways.....	11
8.3 Future Improvements.....	12
9. Appendix.....	12


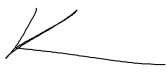

1. Declaration of Original Work

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	Lab Group	Signature/Date
Au Yi Teng	SC2002	SDDA	23 rd April 2025 yiteng
Chua Wen Jun	SC2002	SDDA	23 rd April 2025 
Hasini Lagatapathi	SC2002	SDDA	23 rd April 2025 Hasini
Kieran Mak	SC2002	SDDA	23 rd April 2025 
Rajkumar Saanvi	SC2002	SDDA	23 rd April 2025 

2. System Overview and Feature Planning

2.1 Understanding the Problem and Requirements

This project involves the development of a Build-To-Order (BTO) Management System, a command-line application aimed at streamlining housing workflows for HDB applicants, officers, and managers. Each user role has distinct permissions, such as applying for flats, booking units, managing projects, and handling enquiries.

Our team began by reviewing the assignment brief thoroughly, identifying key use cases and requirements. Core functionalities included NRIC/password login, role-based access, application and project management, officer registration, and data persistence using **.csv** files. Application statuses (e.g., Pending, Booked) and project visibility toggling were also explicitly required.

We also inferred several expectations: the system should enforce eligibility (based on age and marital status), support post-submission access regardless of project visibility, and validate NRIC formats. Officer registration needed to prevent overlapping project conflicts, and each action had to align with the assigned role.

To address ambiguities, we made clear design decisions: officer roles are fixed after registration, users can still view their submissions even if visibility is toggled off, and officer slots are capped and enforced. These interpretations helped ensure a robust, consistent system aligned with real-world logic.

2.2 Deciding on Features and Scope

Based on our analysis, we grouped the identified features into core, optional, and excluded categories to manage scope effectively.

Core features included login, password change, project listing with visibility filters, application handling, officer registration and flat booking, project creation/editing, enquiry management, and approval workflows. These directly supported the system's intended role-based flow and architectural integrity.

Optional features were usability enhancements, such as dynamic project filtering, real-time prompts, and advanced validation. These were implemented if time allowed but deprioritized in favor of core functionality.

Excluded features were intentionally left out due to scope limits or assignment constraints, such as GUI, multi-user concurrency, or external storage like databases or APIs.

This structured planning helped us stay on track, balance ambition with feasibility, and build a maintainable, well-functioning system within the required parameters.

3. Design Considerations

3.1 Overview of the Approach

The BTO Management System was developed using the Model-View-Controller (MVC) architecture to ensure a clean separation of concerns and to enhance the maintainability of the codebase. The Model layer defines core data classes such as User, Project, Application, and Enquiry. The View layer comprises CLI interfaces that facilitate user interaction, while the Controller logic is embedded within utility classes and menu handlers that coordinate user input and the model state.

Our design strategy was guided by Object-Oriented Programming (OOP) principles and core SOLID design concepts. Each class has a well-defined responsibility, and dependencies between classes are minimized. The use of an abstract User class and polymorphic method calls, along with clearly separated logic for password handling, login validation, and file I/O, contributed to a modular and extensible system. Although interfaces and enums were not implemented in this version, the current architecture allows for easy integration of such components in the future.

3.2 Assumptions Made

- All users (Applicants, Officers, and Managers) are initialized via **.csv** data files and assigned default credentials (**password**).
- Each user is expected to understand their role's scope of functionality; thus, the CLI interface provides minimal guidance.
- Officers and Managers may act on BTO projects only if they are not already registered or assigned to conflicting ones.
- The system operates in a single-threaded manner; concurrency is not supported.
- Visibility toggling affects project listings but does not remove access to applications already made by users.

3.3 Object-Oriented Programming Principles

3.3.1 Abstraction

We employed abstraction to reduce complexity by isolating shared behavior in an abstract **User** class. This class includes common fields and method declarations such as **getCSVFilename()**, which is overridden in role-specific subclasses (**Applicant**, **Officer**, **Manager**).

3.3.2 Encapsulation

All model classes encapsulate internal data using private attributes with public getter and setter methods. Fields like **password**, **applicationStatus**, and **remainingFlats** are protected from unauthorized modification, maintaining data integrity.

3.3.3 Inheritance

Inheritance allowed us to reduce redundancy and promote code reuse. Each user role extends the **User** class, inheriting shared behavior such as authentication and profile management, while adding role-specific capabilities.

3.3.4 Polymorphism

Polymorphism is used to execute role-specific logic at runtime. For example, method calls like **user.getCSVFilename()** or behavior within the main menu are dynamically resolved based on the actual subclass instance of **User**.

3.4 Application of SOLID Design Principles

3.4.1 Single Responsibility Principle (SRP)

Each class in our system serves a focused purpose. **PasswordChanger** handles password updates, **LoginManager** performs NRIC-based user authentication, and **FileHandler** is solely

responsible for file operations. This separation of concerns enhances testability and code clarity.

3.4.2 Open-Closed Principle (OCP)

The system supports extension through subclassing without modifying existing logic. For instance, the abstract method `getCSVFilename()` is overridden in each subclass, allowing different user roles to be integrated seamlessly into shared workflows like login and CSV file access.

3.4.3 Liskov Substitution Principle (LSP)

All subclasses of `User`—such as `Applicant`, `Officer`, and `Manager`—can be used wherever a `User` object is expected. Methods like `FileHandler.writeUsersToCSV(List<? extends User>)` ensure compatibility across subclasses, preserving expected behavior.

3.5 Design Trade-Offs and Extensibility

Given the time and scope constraints, several design choices were made to balance clarity, modularity, and alignment with assignment expectations. We focused on a straightforward role-based logic structure rather than incorporating advanced design patterns like factories or dependency injection. This made the system easier to build and debug, especially during collaborative development.

A key constraint was the mandated use of `.csv` files for data storage, which limited the ability to implement structured relationships or scalable queries. To mitigate this, we implemented startup routines that verify and update missing headers or columns in user and project files, enabling the import of externally prepared `.csv` data (including those converted from Excel) with minimal manual intervention.

We chose to prioritize **role separation** over **deep feature encapsulation** to streamline logic and reduce abstraction overhead. The command-line interface was kept consistent across all roles, simplifying the user experience and reducing onboarding effort.

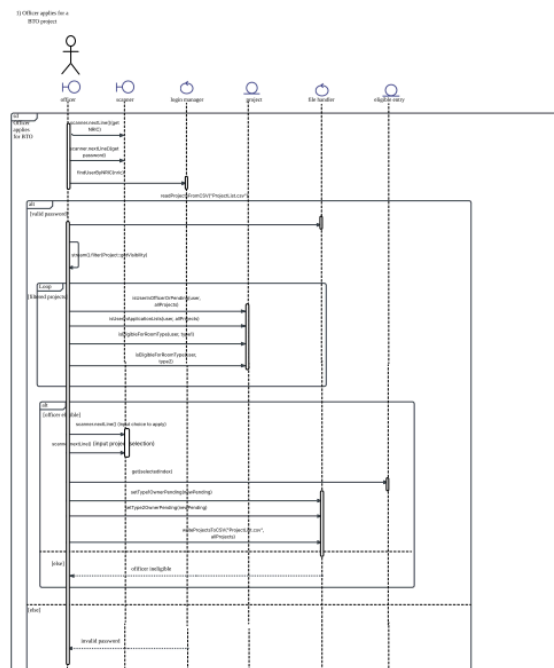
The system is also designed to be extensible. The centralized use of role checks and modular utility classes makes it straightforward to introduce new user roles such as Auditors or Developers, or new features that depend on additional `.csv` columns or logic layers. These decisions reflect a practical compromise that prioritizes stability, clarity, and extensibility within the project's constraints.

4. Additional Features

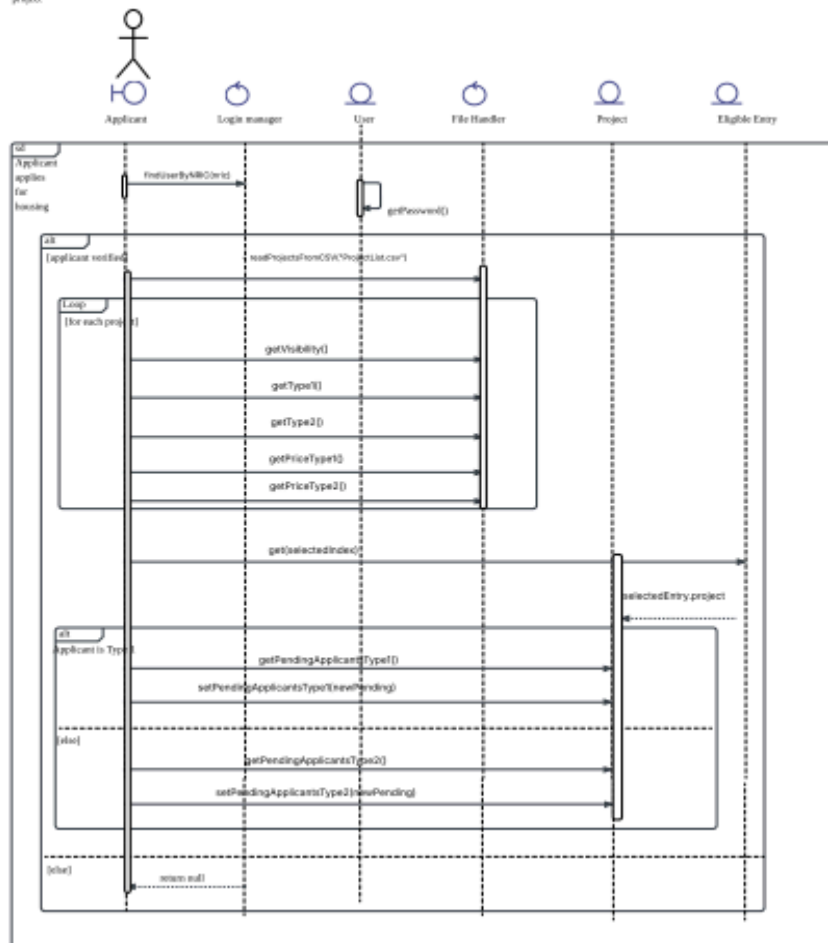
1. We implemented user account creation for new user onboarding
2. We implemented base64 encoding to avoid storing password in plaintext
3. We implemented a receipt to the `.txt` file generator for easy emailing by officers.

[illegible]

6. Detailed UML Sequence Diagram (uploaded separately for quality)



2) Applicant applies for a project



7. Testing

No	Test Cases	Expected Behaviour	Screenshot
1	Valid User Login	User can login with correct username and password	
2	Invalid NRIC Format	User can only use his/her NRIC for username to login	
3	Incorrect Password	Access is denied if password is wrong	
4	Password Change Functionality	Allow changing of password and re login with new	

		password	
5 + 6	Project Visibility Based on User Group and Toggle + Users can apply for BTO if they meet requirements	When visibility is off, users who meet requirements are able to see the project, when visibility is on, users who meet requirements are able to see the project + Users who meet requirement can apply to application	<pre> ===== Index Project Name Visibility 1 Acacia Breeze False ===== Enter number to toggle: Success! Welcome Sarah, 40, Married Applicant. 1) Logout 2) Change Password 3) View Eligible Projects 4) View Enquiries Choice: 3 No Eligible Projects Welcome Sarah, 40, Married Applicant. ===== Index Project Name Visibility 1 Acacia Breeze True ===== Success, Acacia Breeze is now Visible! Welcome Jessica, 26, Married Manager.. ===== Project Name Type Price Acacia Breeze 2-Room 20000 Acacia Breeze 3-Room 30000 ===== Choice (a to apply, e to enquiry, c to cancel): </pre>
7	Single Flat Booking per Successful Application	User cannot apply for more than 1 flat	<pre> ===== Project Name Type Price Acacia Breeze 2-Room 20000 Acacia Breeze 3-Room 30000 ===== You cannot apply for any projects, you are PENDING for Acacia Breeze type 2-Room </pre>
8 + 9	Enquiries + Replying Enquiries	Applicants can make and view enquiries, officers in charge and manager can reply	<pre> Enquiries Project Enquiry Reply Acacia Breeze Can I have a discount [Pending] ===== Enquiries Project Enquiry Reply Acacia Breeze Can I have a discount Yes ===== </pre>
10	HDB Officer Registration Eligibility	Officers who meet requirements and not assigned to the project can apply for the project	<pre> Success! Welcome Daniel, 36, Single Officer. 1) Logout 2) Change Password 3) View Eligible Projects 4) Reply to Enquiries (Assigned Projects) 5) View Enquiries 6) Book Owner Choice: 3 ===== Project Name Type Price Acacia Breeze 2-Room 20000 ===== Choice (a to apply, e to enquiry, c to cancel): Project No Neig1 Type 1 Nui Selling pri Typ Nu Sel Apr Ap Manager OI Officer Visibility Acacia Breeze 2-Room 2 20000 3-R 3 ### # Jessica 2 Emily,David TRUE </pre>

11 + 12	HDB Officer Registration Status + Project Detail Access for HDB Officer	Officers assigned to the project can approve the application regardless of visibility	<pre> Success! Welcome David, 29, Married Officer. 1) Logout 2) Change Password 3) View Eligible Projects 4) Reply to Enquiries (Assigned Projects) 5) View Enquiries 6) Book Owner Choice: 6 ===== Index Project Name Type Owner 1 Acacia Breeze 2-Room Sarah ===== Enter index of Owner to Book (enter c to cancel): </pre>
13	Restriction on Editing Project Details	Only Managers can have the option to edit the project details	<pre> Success! Welcome Jessica, 26, Married Manager. 1) Logout 2) Change Password 3) Create Project 4) Toggle Visibility 5) Edit Project 6) Approve Owners 7) Reply to Enquiries (Any Project) 8) View Enquiries 9) Approve Withdrawals Choice: 5 ===== Index Project Name Visibility 1 Acacia Breeze False ===== Enter number to edit: 1 Neighbourhood (, empty to keep): </pre>

8. Reflection

8.1 Challenges Faced and How We Overcame Them

One of the main challenges our team encountered was dividing work by user roles (Applicant, Officer, Manager), which at first seemed straightforward but soon created overlapping logic across modules. For instance, application and flat booking workflows involved both Applicants and Officers, which led to repeated code and unclear ownership of responsibilities. To address this, we shifted to a modular approach—treating key features like project management, enquiry handling, and application processing as standalone components shared by multiple roles. This improved integration and code reusability.

Another challenge arose due to the absence of a unified architecture early in development. Without clear structure, we initially faced inconsistent naming, duplicated logic, and difficulties merging contributions. To resolve this, we adopted the **Model-View-Controller (MVC)** framework and anchored our design around **Object-Oriented Programming (OOP)** and **SOLID principles**, improving consistency and maintainability. Time management was also crucial, and by clearly distinguishing between core and optional features, we managed to deliver a functional and stable system within the project timeline.

8.2 Key Takeaways

This project deepened our understanding of core OOP concepts such as inheritance, encapsulation, abstraction, and polymorphism. Applying these, alongside the SOLID design principles, helped us build a flexible and extensible system. We also learned the value of upfront planning—particularly through rough UML diagrams—which gave us a shared structural vision and reduced misunderstandings during integration.

Teamwork played a key role in our success. By working in parallel yet collaboratively, we ensured our code was interoperable and aligned with the overall design. Through this experience, we grew more confident in handling collaborative coding projects and in applying software design theory in practice.

8.3 Future Improvements

While our system met the required functionality, there is room for enhancement. Adding real-time user session handling, implementing automated data backups, and improving error handling and feedback messages would make the system more robust. A future version could also explore transitioning to a GUI or web-based interface and incorporate security features like password encryption or session timeouts.

9. Appendix

github link : <https://github.com/yitenglol/SDDA-grp3/>