**Declaration of Original Work for SC2002/CE2002/CZ2002 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 |
| Kang Yi, Zen | SC2002 | FCS5 Grp 2 | 24/4/2025 |
| Jarrel Ng Tze Xuan | SC2002 | FCS5 Grp 2 | 24/4/2025 |
| Katta Rohita | SC2002 | FCS5 Grp 2 | 24/4/2025 |
| Leong Jun Rong | SC2002 | FCS5 Grp 2 | 24/4/2025 |
| Lim Hwee Jhee | SC2002 | FCS5 Grp 2 | 24/4/2025 |

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 the usage of Generative AI and any other guidelines as released by NTU.

**SC2002 Object-Oriented Design and Programming**

# Project Title: Build-To-Order Management System

## 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.

The system we needed to build was the Build-To-Order housing management platform. The platform was expected to simulate the real-world interactions between public members applying for flats and the HDB officers and managers managing these applications and projects.

|  |  |
| --- | --- |
| **Explicit Requirements** | **Implicit Requirements** |
| * Role-based user login using NRIC and password * Ability for applicants to apply, view status, and withdraw applications * HDB Officers can register to handle projects, assist with booking, and reply to enquiries * HDB Managers can create/edit/delete projects, handle applications, and manage officers * Receipt and report generation * CSV file-based data persistence * CLI-based interface | * Officers and Managers must be able to view filtered data, which is specific to their project * Only one flat can be booked per application * Based on the user’s age and marital status, we need to validate whether they are eligible or not for the project. * Role-based CLI navigation would need to be implemented to separate user experiences |

**Ambiguities and our interpretations**

|  |  |
| --- | --- |
| **Ambiguity** | **Interpretation** |
| Whether a manager can approve or reject BTO applications for projects they are not responsible for. | No, managers only approve applications for their own projects. |
| Whether to merge different user roles into a single CSV or keep them separate. | Choose separate CSVs for role abstraction and password security. Merging was considered but not used. |
| Should filters apply differently for officers viewing projects vs applicants? | Unified filter logic. Officers can't view/register for projects they manage or overlap with. |
| Do officers start as applicants and transition roles? | No. Officers and Applicants are initialized with fixed roles and remain that way. |

1.2 Deciding on Features and Scope

|  |  |  |
| --- | --- | --- |
| **Core features** | **Additional features** | **Excluded features** |
| * Role-based access control * Application, registration, and withdrawal workflows * Project creation and visibility settings * Filtering projects * Enquiry submission and replies * Receipt generation for flat bookings | * Formatted pretty print for details * Enforce strong Password when changing passwords * Concatenated filtering and sorting, filters are updated and saved in each session as well * Hashing of password using SHA-256 to enhance security * LRU Cache feature to speed up searching | * Combining the different user roles CSVs into one for login. We didn’t implement it as we opted for focusing on the abstraction of user data for the password. |

## Chapter 2: System Architecture & Structural Planning

2.1 Planning the System Structure

We used an ECB Design pattern to make sure our codebase was modular and reusable.

1. Entities: Applicant, Manager, User, Project, Enquiry, and ProjectApplication implement the IEntity interface and represent core data modules.
2. Boundaries: ApplicantBoundary, ManagerBoundary, and LoginBoundary handle CLI interactions and user flow, separating I/O from system logic.
3. Controllers: ApplicantController, ManagerController, and ProjectController contain the main logic, validate input, and link boundaries to repositories.
4. Repositories: The abstract Repository<T> class handles CSV data access. Specific repositories like ApplicantRepository and ProjectRepository manage entity-specific data operations.
5. Utilities: SafeScanner, PrettyPrint, and CsvUtils enhance input validation and output formatting.
6. We incorporated design patterns like Singleton pattern, Factory method and strategy pattern

2.2 Reflection of Design Trade-offs

We were initially hesitant to have controller and repository layers separately. However, we did not merge them to promote reusability, allow for better testing, and reduce coupling. We chose interfaces and abstract classes early so that our system could be easily extendable in the future. As a group, we had many debates, especially when we each understood the roles of the users differently, but as we started to discuss why each of us thinks that way, we concluded it.

## Chapter 3: Object-Oriented Design

3.1 UML Class Diagram

Before creating our UML diagram, we had many meetings to discuss to ensure that our class structure would adhere to the BTO system specifications

**Identifying main classes:**

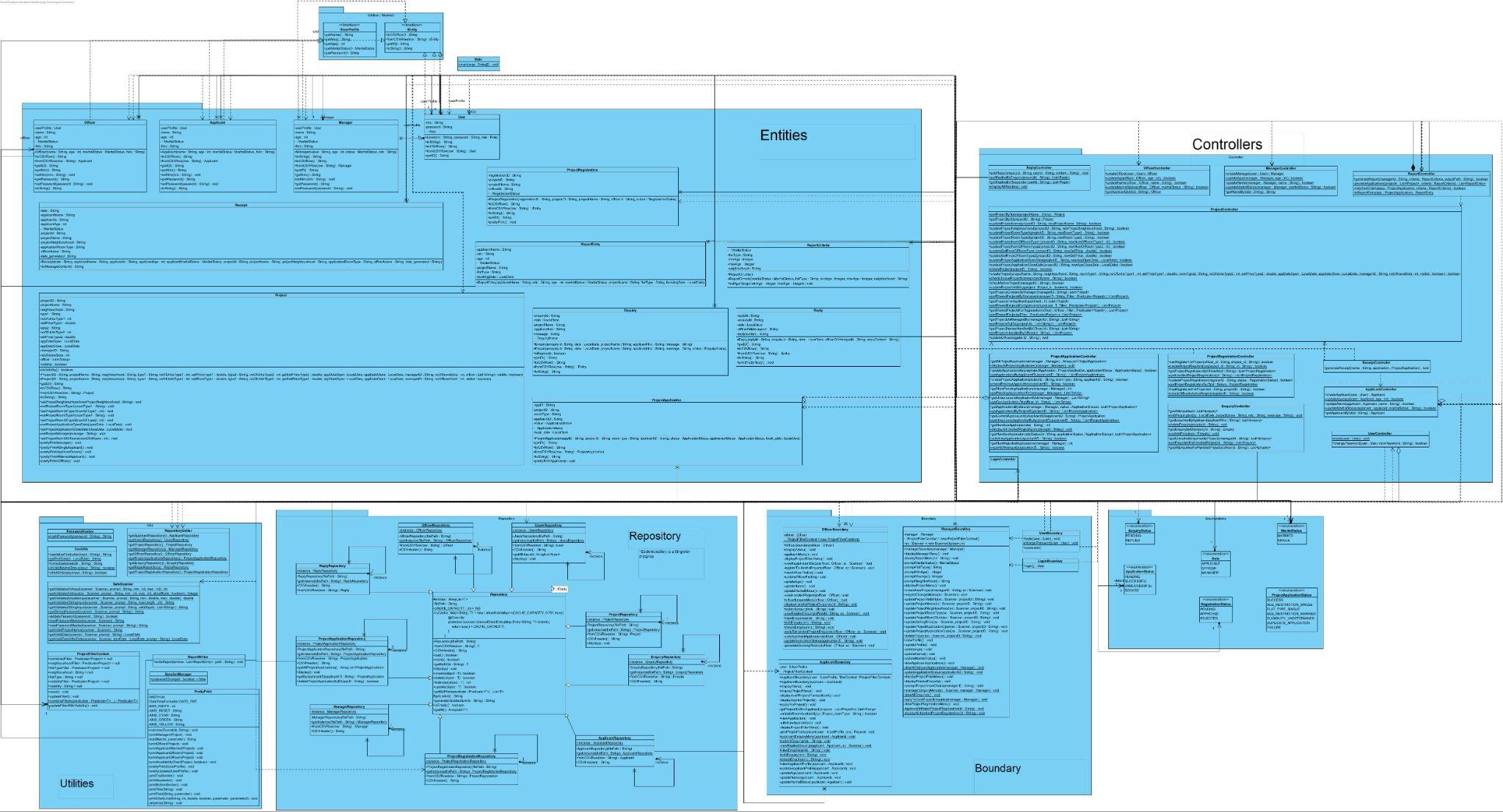
* Key roles like Applicant, Officer, and Manager extend User, while core entities include Project, Application, Registration, etc. Enums were used for and statuses.
* We evaluated relationships as attributes vs classes and mapped associations like ProjectApplication(ApplicantID, ProjectID).
* Manager control over projects is handled via controller logic.
* A generic Repository<T> with CRUD operations was used, with entity-specific subclasses (e.g., ApplicantRepository).
* We prioritized composition and controller delegation for flexibility.

**Some tradeoffs that we had to consider include:**

* For better user interaction control, we have simplified UP logic via the Boundary class, even though it duplicated some controller logic
* We avoided direct inheritance between Applicant, Officer, and Manager to keep user-role management modular, however they all implement IUserProfile interface.

**Some highlights include:**

* We used enumerations for ApplicationStatus, RegistrationStatus, Role etc, to enforce valid values
* In our system, Controllers manage interaction logic, Repositories persist data to CSV, and Boundaries manage the CLI



3.2 Sequence Diagrams

#### **(i) Officer Registers to Handle a Project** (the sequence diagram is in the folder)

This scenario showcases a key role-specific action involving input, controller logic, and repository updates. It includes eligibility checks (e.g., overlapping projects) and role-based access, reinforcing ECB structure and the Single Responsibility Principle.

#### **(ii) Officer applies for a project** (the sequence diagram is in the folder)

This scenario demonstrates polymorphic role behavior—letting Officers access applicant features via shared interfaces. It highlights clear role delegation, controller-boundary collaboration, and enforcement of rules (e.g., officers can't apply to their own projects).

3.3 Application of SOLID principles

|  |  |
| --- | --- |
| **Single Responsibility Principle**  Each class has a single, well-defined responsibility, which makes the codebase modular and easier to maintain.  **Reasoning:** Separating storage logic (Repository), logic(Controller), and user interactions(Boundary), made the system more testable and allowed us to debug components independently | **ProjectRepository:**  Handles the persistence logic for Project  entities, such as **CRUD** operations (create, read, update,  delete).    **ProjectController:**  Implements logic like retrieving by visibility, managing bookings etc.    **SafeScanner:**  Provides input validation for various types of inputs.    **PasswordHasher:** Provides hashing functionality, adhering to the principle of a utility class having a singular focus |
| **Open/Closed Principle**  The codebase is open for extension but closed for modification  **Reasoning:** We could easily add new entity types (eg, Receipt, Reply) without modifying the existing logic. This helped in preserving compatibility and reducing bugs when adding features without modifying existing code. | **Repository<T>:**  The abstract base repository class provides a foundation for specific repositories like  UserRepository, ApplicantRepository, and ProjectRepository. New repositories can be added by extending Repository without modifying its base logic      ApplicantRepository, ReplyRepository:  Extend Repository<T> and override when individual logic is needed. |
| **Liskov Substitution Principle**  Subtypes should be able to replace base types without altering the program.  **Reasoning:** Code that interacts with entities via Repository<T extends IEntity>  works seamlessly regardless of the specific entity type (e.g., ProjectApplication, Officer, or Reply). | **IEntity Interface:**  All models (e.g., Applicant, Project, Reply) implement the IEntity interface,  ensuring they provide getID() and toCSVRow() and fromCSVRow() methods. |
| **Interface Segregation Principle**  Interface should provide only the methods relevant to the implementing class.  **Reasoning:** Preventing classes from being forced to implement unused methods. For example, Reply does not need the getName() method. | **IUserProfile:**  Segregated from IEntity for user-specific attributes.    Used by Applicant, Officer, Manager:  The roles only implement what they need. |
| **Dependency Inversion Principle**  High-level modules should not depend on low-level modules but instead on abstraction.  **Reasoning:** By decoupling repository creation from controller logic, we were able to allow for better testing and future migrations. | **Repository Abstractions:**  Code interacts with entities through abstract repositories (e.g., Repository<T>),  rather than concrete storage implementations.  For instance, the ProjectController depends on the ProjectRepository abstraction,  which can be replaced without altering the controller.    **RepositoryGetter Utility:**  Provides centralized access to repository instances, allowing higher-level  modules to remain decoupled from how repositories are instantiated. |

## Chapter 4: Implementations (JAVA)

4.1 Tools used

* Java 23
* IntelliJ IDE
* Version control: GitHub

4.2 Sample code snippets:

|  |  |  |
| --- | --- | --- |
| **Encapsulation** | **Inheritance** | **Polymorphism** |
|  |  |  |
| **Interface use** |  | **Error Handling** |
|  | |  |

## Chapter 5: Testing

5.1 Test Strategy:

**Manual Testing Strategy:** We did the testing using CLI by creating realistic accounts and projects.

5.2 Test Case Table: (please zoom in all 23 test cases are there)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case 1:  Valid user login | Test Case 2:  Invalid NRIC format | Test Case 3:  Incorrect Password | Test Case 4:  Password Change Functionality | Test Case 5:  Project Visibility Based on User Group and Toggle    For single aged 35 these are projects visible    For singles aged 26 this is what they can see. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case 6:  Project Application    (below 35 and single)    (married) | Test Case 7:  Viewing Application Status after Visibility Toggle Off | Test Case 8: Single Flat Booking per Successful Application | Test Case 9: Applicant’s enquiries management | Test Case 10:  HDB Officer Registration Eligibility      Cannot apply as the dates of the project handled by the officer collide with the dates of the project that she wants to apply to. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case 11  HDB Officer Registration Status  : | Test Case 12:  Project Detail Access for HDB Officer | Test Case 13:  Restriction on Editing Project Details | Test Case 14:  Response to Project Enquiries | Test Case 15:  Flat Selection and Booking Management |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case 16:  Receipt Generation for Flat Booking | Test Case 17:  Create, Edit, and Delete BTO Project Listings | Test Case 18:  Single Project Management per Application Period | Test Case 19:  Toggle Project Visibility | Test Case 20:  View All and Filtered Project Listings |

|  |  |  |
| --- | --- | --- |
| Test Case 21:  Manage HDB Officer Registrations | Test Case 22:  Approve or Reject BTO Applications and Withdrawals | Test Case 23:  Generate and Filter Reports |

## Chapter 6: Documentation

6.1 Javadoc:

Javadoc was generated for all public methods and classes. The javadoc is present in the GitHub Repository.

6.2 Developer guide:

Step 1: Clone the repository from GitHub

Step 2: Open with IntelliJ or your preferred IDE

Step 3: Run Main.java

## Chapter 7: Documentation

### **What Went Well**

* Applied SOLID design principles across modules like Repository<T>, RepositoryGetter, SafeScanner, and user creation.
* Core features—login, application, registration, status updates, and filtering—were fully implemented, tested, and integrated.

### **What Could Be Improved**

* Late feature completion left little time for documentation and polish.

### **Lessons Learned**

* Finalize architecture early to avoid rework.
* Document alongside development to reduce pressure and increase collaboration.
* Use comments consistently for clarity and team coordination.

## Chapter 8: Appendix

* GitHub repository link: <https://github.com/zenkang/BTOproject>
* All functionalities were implemented using Java’s standard libraries and custom-written utility classes.
* UML Class diagram, as well as the sequence diagram, are present in the folder
* Source code