<u>Declaration of Original Work for SC2002 Assignment</u>

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
IVAN YAP WEIWANG	SC2002	SCE3	[illegible] 20/04/2025
KOH JIA JUN	SC2002	SCE3	[illegible] 18/04/2025
ARPITA YOGISH	SC2002	SCE3	[illegible] 18/04/2025
LIN YIQIN	SC2002	SCE3	Lyiqin 18/04/2025
SYED RIDWAN KABIR	SC2002	SCE3	[illegible] 18/04/2025

Project Title: BTO 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.

We identified the main problem domain as the need to manage BTO applications efficiently while successfully streamlining actions by various users, namely applicants, officers and managers, who all have different roles and permissions in the system. Our system needed to handle applications for BTO flats for these 3 key stakeholders, ensuring our applicants can submit requests for the units available, officers can review and approve applications and managers can oversee the whole process.

Some explicit requirements that were provided are the secure login-logout system using user credentials, role-based access controls that grant different system features based on permissions assigned to the role, project and application management, application withdrawal and also an enquiry and reply system between applicants and officers.

Some implicit expectations we inferred was the menuing system. In order for users of the system to be able to intuitively navigate the system's capabilities, the user should be able to see all relevant options to them in each menu. For example, applicants should be able to apply for a project from the "View Project" menu, but they should also be able to apply for a project directly from the home page, similarly for viewing and modifying enquiries.

We also faced ambiguities with application management and history. For application (and officer application) management, it was unstated when applicants were allowed to withdraw their applications. We decided to allow users to withdraw at any time, as that allows for the smoothest user experience.To implement this, we created 2 separate attributes to ensure that withdrawals could occur regardless of status. We also faced ambiguities in application history. As applicants are only allowed to have 1 application, we realised that for applicants to be able to see their past applications, there was a need to be able to retrieve past applications, even if they were withdrawn or rejected.

1.2 Deciding on Features and Scope

We grouped features into three categories: core, optional, and excluded. Our goal was to ensure that the core features demonstrated strong object-oriented principles without making the system overly complex.

List of potential features considered are as follows: User login-logout function, role-based permissions for the 3 roles, application management (submission, approval or withdrawal), replying to enquiries, project management (for the HDB officers and managers) and search & filtering options for BTO projects and applications.

When deciding how we should prioritise these features, we considered the extent of necessity of the feature for the core functionality and main purposes of our system. The highest priority was given to those features which directly impact the application and approval, because these are the most fundamental and key operations. The next priority was given to features that can enhance and improve the user experience, but are not absolutely necessary (ie. optional features). Finally, those features which are outside our scope and would unnecessarily complicate the system if they were implemented were excluded.

1.2.1 Core features

Our first core feature is application management, encompassing the submission of BTO flat applications, with constraints (such as age and whether single or married), viewing of application status (pending, successful, unsuccessful or booked), requesting withdrawal and also submitting enquiries, and also HDB officers' capabilities such as selecting a BTO project and registering to handle it, applying as applicants, updating application status (validation ie. approval or rejection of application) and more. Application management is a core feature as it fulfills the fundamental purpose of our system.

Core feature 2 is HDB managers' management capabilities, namely through creating, editing, deleting projects (and also checking attributes like name, location or flat type), toggling the visibility of projects, approving or rejecting registrations from HDB officers and generating reports for applicants' flat booking details. This is fundamental as the HDB manager oversees almost every function on the interface, and marks the availability (start) and end of a BTO 6+booking.

1.2.2 Optional features

Optional feature 1 is Enhanced Search Features. An example would be Dynamic Filtering Based on Flat Availability Quota. Each BTO project has a limited number of units for 2-Room and 3-Room flats. As applicants book flats, the system updates the remaining count. This optional feature enables dynamic filtering based on those remaining units. Another example would be fuzzy searches for String searches such as Project Name and Neighbourhood. This feature is considered optional due to its reliance on additional filtering logic, and potential complexity of implementation.

Optional feature 2 is password hashing and salting. This would have been essential in a real-world application of the BTO management system, but it is optional for the purposes of this project as we are not handling real users' data and our main aim is to build and maintain backend system functionality rather than integrate with complicated external APIs.

Optional feature 3 is input validation with precise and descriptive error messages that guide them to correct their errors (specifying very precisely the mistake(s) in their entry in the field) instead of just throwing an error, improving user experience. We consider this optional as basic error messages are enough to inform the user of their mistake in input.

Optional feature 4 is analytics dashboard implemented through Tableau add-on for the reference of HDB officers and managers, showing statistics related to application approval based on flat types, districts with higher demand, approval rates by district, average time taken from application to approval, stats pertaining to nature of applicants (age and marital status), etc. We consider this optional as such a feature is not required by the specifications. but would be very useful in a real-world scenario.

1.2.3 Excluded Features

Excluded feature 1 is multiple concurrent user support due to its greater complexity and additional infrastructure that may be needed, such as threading and safeguards against race conditions. While this would allow our application to be more useful in a real-life scenario, this feature would make the scope of the project more complex.

Excluded feature 2 is an enhanced or customizable graphical user interface (GUI) as we wanted to focus on the core backend functionality and OODP of the BTO management system. Examples of such features would be dark/light mode toggle, or being able to customise menuing options. Our main goal was to demonstrate principles such as using classes, inheritance and encapsulation, introducing a GUI would have increased the complexity.

Excluded feature 3 is Singpass Login. Assignment requirements stated that "All users will need to login to this hub using their Singpass account." In a real-world scenario, logging in via Singpass authentication system would be an important feature to implement. However, realistically simulating Singpass is not possible due to API access and security. Therefore we decided to stick with traditional NRIC and password login.

Chapter 2: System Architecture & Structural Planning

2.1 Planning the System Structure

Before implementation, we broke down the system into 5 main logical components, Item (each representing an entity), ItemArray (each storing an entity type), User (and their subclasses), Manager (one for each entity and wherever else necessary), and Menu. We mapped user flows and use cases for each role; applicant, officer, and manager, to ensure all requirements were addressed and to clarify system boundaries and interactions. To start off, we used flowcharts and UML diagrams to visualise key relationships and data flow, so we could refine our architecture as we move along.

2.2 Reflection on Design Trade-offs

One unique design decision we made was to model our data storage system after a database, rather than using 2-way composition. In a traditional system with 2-way composition, there is a need to update 2 places every time one item changes. This is especially the case with many-to-many relationships, such as that between Officers and Projects, with the joining table represented by Officer Applications. In order to solve this, we decided to model our system's architecture after a database, where rather than using 2-way composition, we had the 'many' in the 'one-to-many' relationship store a reference to the 'one', similar to a foreign key, hence constituting one-way association. For example, storing the reference to a project only in the Applicant class but not in the Project class. In the case of a many-to-many such as Officers and Projects, we had the object representing the joining table, Officer Applications, store a reference to both sides.

This allowed for easy modification of entries, however, it came with the downside of needing to have 'SELECT' methods implemented, which we did by implementing various filters for each Array, for example Officer Applications being filtered by Officers or Applications. We also implemented methods such as getOfficers(Project P) which would filter the Officer Applications for P and return a List of Officers.

Another design choice we made was to use static classes for our arrays. This prevented us from needing to pass class instances around in our code, as we could simply refer to the static class to get the array. However, that came at the cost of needing to repeat code, as well as being unable to use interfaces for such static classes. Upon reflection, it may have been better to use a non-static class under the hood, and have a static .get() function to get the class instance, similar to our menuing system. This will be further explained in the section on SOLID principles.

Chapter 3: Object-Oriented Design

3.1 Class Diagram (Attached in Appendix A)

We identified the main classes by analyzing system requirements and key user roles, specifically the user hierarchy (Applicant, Officer, Manager) which reflects different access levels and actions. Project and Application classes are core entities, with Project to manage BTO listings and Application to handle applicant-project relationships through state tracking. Enquiry acts as a supporting(boundary) entity which addresses communication requirements between users.

Class	Responsibilities
Project	Manages BTO project details (name, dates, units).
Application	Tracks applicant-project relationships (status, flat type).
Enquiry	Handles applicant enquiries and replies.

Fig. 1: Core Entity classes & respective responsibilities

Beyond core entities, we derived supporting classes by analyzing system workflows and design principles. Aligning with the **Single Responsibility Principle**, we created 5 distinct manager classes which isolates the business logic required for each domain(ie applying, approving, withdrawing etc.). Next, our data storage classes(Users, Projects, Applications) are to perform all **data persistence** tasks separately from business logic. We utilized Data Access Object pattern, where each storage class abstracts the details of how data is stored and retrieved away from what the rest of the application can access. As well as encapsulation design, where all our classes can keep their internal data structures private and expose only the necessary methods for interaction, such as filtering or updating records.

For our MenuSystem, it handled UI interactions through boundary classes. Aligning with the Boundary-Control-Entity (BCE) pattern, each menu class inherits from a common Menu boundary, separating user interface logic from core operations. This design ensures that all input handling and output rendering are encapsulated within dedicated boundary classes, promoting a clean architecture. Additionally, we followed the Open-Closed Principle (OCP) by designing the menu system so that new menu types (like FlatTypeFilterMenu) can be added as subclasses without altering existing menu code. This approach makes the UI layer highly extensible and maintainable, supporting future enhancements with minimal risk of introducing bugs into existing functionality. Lastly, the UserSystem package uses an inheritance hierarchy where User is the base class, extended by Applicant and Manager, and Officer further extends Applicant to add officer-specific functions. This design encapsulates users' sensitive data.

Package	Class	Responsibilities
User System	User	Base class for all users (stores NRIC, password, age, marital status).
	Applicant	Represents BTO applicants (applies for projects, submits enquiries).
	Officer	Manages project-specific tasks (approves bookings, replies to enquiries).
	Manager	Creates/manages projects, approves officer registrations.
Managers	ApplicationManager	Handles application workflows (apply, withdraw, approve).
	ProjectManager	Creates/edits projects, validates project constraints.
	Validator	Validates NRIC, dates, project overlaps, and application eligibility.
Data Storage	Users	Manages persistence of all user data (applicants, officers, managers).
	Projects	Stores/retrieves project data, filters by manager.
Menu System	Menu	Base class for UI navigation (display menus, handle input).
	ProjectListMenu	Displays filtered project lists (by name, flat type).

Fig. 2: Supporting classes & respective responsibilities

Regarding relationships between classes, inheritance forms the backbone for our User→Applicant→Officer and User→Manager hierarchies, enabling code reuse while maintaining role separation. Composition is evident in Project→Application relationships (1-to-many ownership) and

Project→Manager associations (exclusive ownership). Next, aggregation appears in Data Storage classes managing ItemArrays of entities. Dependency relationships connect MenuSystem to managers for operation execution while maintaining UI-business logic separation. The Stringifiable interface enables polymorphic serialization across Project, Application, and User classes.

Regarding trade-offs considered, static manager classes provide centralized control but introduce global state risks. Composition was prioritized over inheritance for flexibility, though this increased coupling between management classes. Notably, we practised strong encapsulation which added lots of boilerplate code, such as repetitive getter/setter methods. Lastly, isolating data storage improves persistence management but introduces synchronization challenges across the system. This becomes a problem in practical implementation when concurrent operations occur.

3.2 Sequence Diagrams (Attached in Appendix B)

The sequence diagram titled "Officer Applying As An Applicant" illustrates the use case of a HDB officer simulating their role as an applicant and applying as an applicant for a housing project. It shows the process of how the officer accesses the application menu, selects a project by choosing the ID, selects the type of flat and submits the application. In return, the system will validate the details in the application before confirming, creating and storing it. The system will also link back the details to the HDB officer's profile.

Our use case showcases our inheritance structure, as Officer extends Applicant and are hence able to participate in the process of application. It also shows how we have designed our system such that it is modular and allows for role-based handling. Finally, our diagram depicts how various different components of the system collaborate smoothly with the help of error handling to ensure data integrity and ensure applications meet pre-set criteria and rules.

The second sequence diagram titled "Officer Registering to Handle a Project" depicts the use case of HDB officers registering to handle projects. It outlines how the officer selects a project, system validates their registration, followed by it creating and storing an OfficerApplication linked to the officer. The system thus allows officers to apply to manage a Project of their choosing.

Both diagrams are designed to show the communication between the 5 main logical components of our app, with the Menu responsible for user input and displaying the final result, the Managers validating and creating new entries, the User system being checked for user type, the Item called to construct the new object, and the ItemArrays being used to store the final result.

3.3 Application of SOLID Principles

3.3.1 Single Responsibility Principle

In order to keep our classes handling one responsibility each, we had each entity have its own Item, Array, and Manager classes. We also have a class for each separate menu page. In doing so, we kept our code neat and segregated, while also allowing any bugs found to be easily located and resolved, based on which class in the chain had an issue. It also allowed us to keep easy track of what methods each class supported. One downside we incurred was the massive amounts of files that ended up being required for the menu system, where each file needed to subclass the Menu class in order to display the navigation options, for example having one menu file for each option in ProjectFilter.

3.3.2 Open/Closed Principle

In our menuing system, we declared an abstract method menu() which would be invoked by the display() method. This allows each individual menu to extend Menu and modify their menu contents, while still allowing them to reuse other methods such as navigate() and display(), and error handling. As such, this allows the Menu base class to be open for extension, but closed for modification. This allowed us to save on having to repeat code, and also allowed us to modify the base class to add any features we were missing, such as the ability to re-display the menu upon an error. However, this also resulted in some repeating code, such as in ItemArray and Menu's children classes, despite using base classes for each. This is due to the need to instantiate the baseclass of Menu for each class, and also partially due to choosing to implement these classes as static. This decision also partially contributed to the many menu file issue described above.

Another example would be Officer extending Applicant, as well as OfficerApplication extending Applications. This helped us to reduce repeating code, allowing Officer to have the same abilities as Applicant without needing to explicitly copy code. However, this also resulted in some redundant features in OfficerApplication that are not used, such as the ability to 'Book' an Officer Application. In hindsight, it may have been better to have both OfficerApplication and Application extend another base class BaseApplication with the methods that the two application types share.

3.3.3 Liskov Substitution Principle

Building on the previous example, we ensured that the Officer had all the features of Applicant. Additionally, all user types extend the base class User, which has an enum UserType describing the user's role. This system allows for safe upcasting of Officer to Applicant, in cases of classes where an officer chooses to apply for a project as an applicant. It also allows us to perform tasks regardless of user type, such as storing all users in a single Users array to filter, checking a user's password before logging in, and AppUserManager's getCurrentUser method to get the user regardless of user type. These would all be safe operations, as any of the child classes would all have a User's capabilities, and would thus be able to function as a User object. Finally, using UserType, we can safely perform runtime downcasting based on the current user's type.

We also ensured that OfficerApplication had all the features of Application, however this did result in the downside of redundant features as mentioned, as OfficerApplication did not use all the features that Application included, such as 'Book'.

3.3.4 Interface Segregation Principle

We segregated our two interfaces, Stringifiable and LongStringifiable, with stringifiable's toString() used to produce a 1-sentence summary of the item, and toLongString() used to produce a longer, multi-line display of the item. This allowed us to use one but not the other, for example for Enquiries where one line is always sufficient to explain its contents, and LongStringifiable is not necessary.

As a result of our decision to use static classes for our Menus and ItemArrays, we were severely limited in the options to use for our interfaces, as Interfaces cannot be created for static methods. In hindsight, we could have used a single static 'Factory' class to store all the instances of non-static Arrays as interfaces, then had each of the ItemArrays implement said interface, as well as a similar system for Menus. This may have helped us to improve the flexibility of our codebase and provided more opportunities to use interfaces.

3.3.5 Dependency Inversion Principle

The filter classes for Applicants and Projects, ApplicantFilterManager and ProjectFilterManager respectively, were designed to be able to take in an ArrayList as parameters, rather than getting the list directly in the code. Instead, the data in the ArrayList is gotten in the menu class, and passed to the Manager. This allows the two filter classes not to depend on the logic of ItemArray, and allowed them to be tested and used independently, hence showing off the Dependency Inversion Principle. However, this did come with the downside of simply passing down the dependency further down the code to the Menu classes, which would have to call the ItemArray class directly. In hindsight, it may have once again been better to use a single static 'Factory' class, which would present each Array as an interface. However, that may also come at the cost of having many interface classes, one for each Array, as different filter methods are required for each array.

Another opportunity to use this principle would have been the AppUserManager's connections with the various other ItemManagers. Instead of directly linking the two, we could have used an interface to expose the method getCurrentUser(). However, at the time of planning, we failed to consider this, and hence failed to implement this feature.

Chapter 4: Implementation

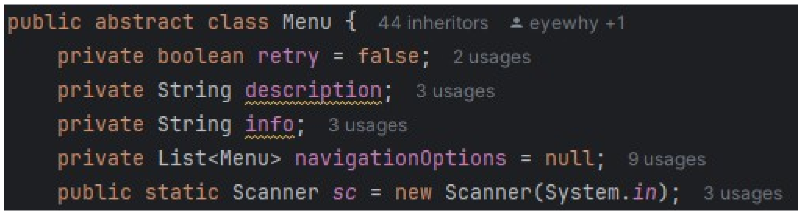
4.1 Tools Used

- Java 23
- Maven 4.0.0 (see Chapter 6 / pom.xml for dependencies)
- IDE: IntelliJ / VSCode (with Java Extension Pack)
- Git & Github
- PlantUML for diagrams

4.2 Code Snippets

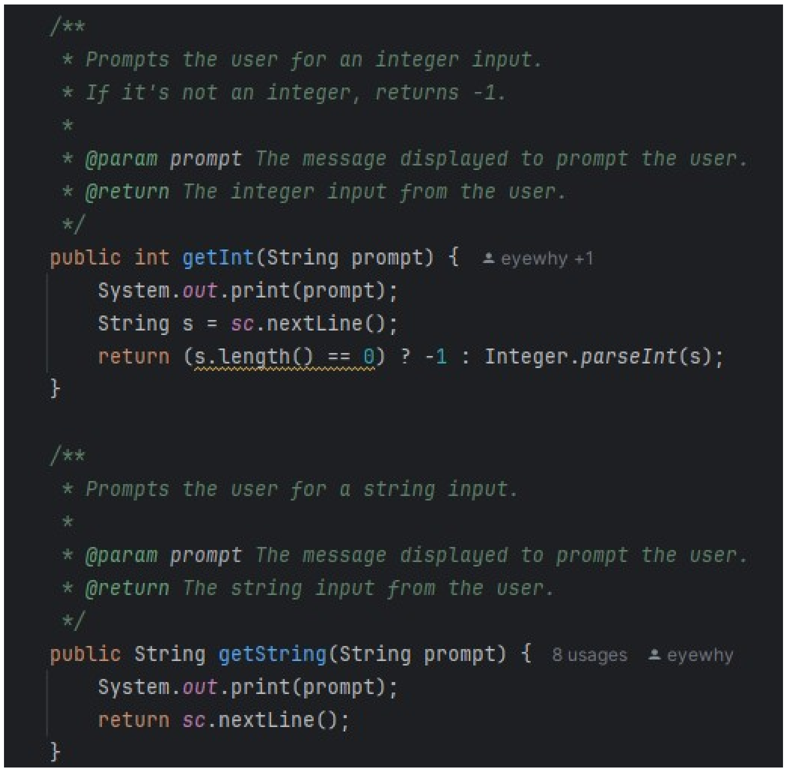
4.2.1 Encapsulation

One example of encapsulation being utilised in our code is the creation of our Menu class.



```
public abstract class Menu {  44 inheritors  eyewhy +1
    private boolean retry = false;  2 usages
    private String description;  3 usages
    private String info;  3 usages
    private List<Menu> navigationOptions = null;  9 usages
    public static Scanner sc = new Scanner(System.in);  3 usages
```

This is an example of variables being declared in our Menu class.

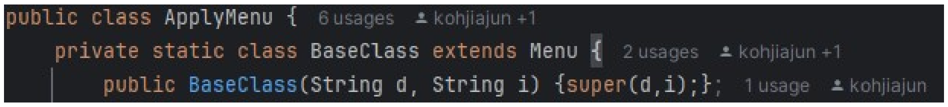


```
    /**
     * Prompts the user for an integer input.
     * If it's not an integer, returns -1.
     *
     * @param prompt The message displayed to prompt the user.
     * @return The integer input from the user.
     */
    public int getInt(String prompt) {  eyewhy +1
        System.out.print(prompt);
        String s = sc.nextLine();
        return (s.length() == 0) ? -1 : Integer.parseInt(s);
    }

    /**
     * Prompts the user for a string input.
     *
     * @param prompt The message displayed to prompt the user.
     * @return The string input from the user.
     */
    public String getString(String prompt) {  8 usages  eyewhy
        System.out.print(prompt);
        return sc.nextLine();
    }
```

This is an example of different methods being declared in our Menu class. These methods are then called and used by other portions of our code.

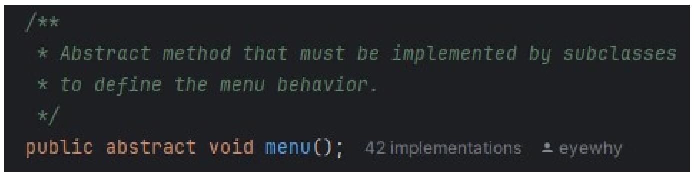
4.2.2 Inheritance

BaseClass inherits the variables and methods of Menu, and this was shown by the constructor of BaseClass having to declare the variables used in Menu through the super function.



```
public class ApplyMenu {  6 usages  kohjiajun +1
    private static class BaseClass extends Menu {  2 usages  kohjiajun +1
        public BaseClass(String d, String i) {super(d,i);};  1 usage  kohjiajun
```

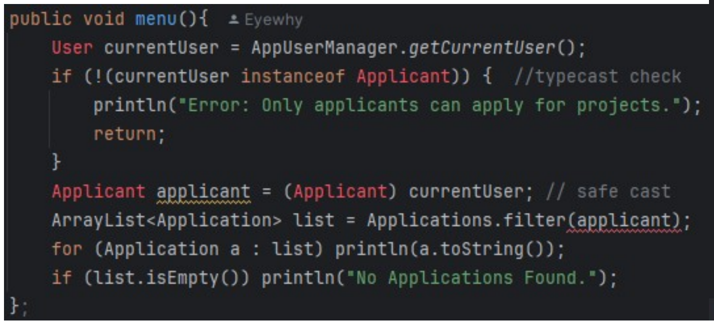
4.2.3 Polymorphism



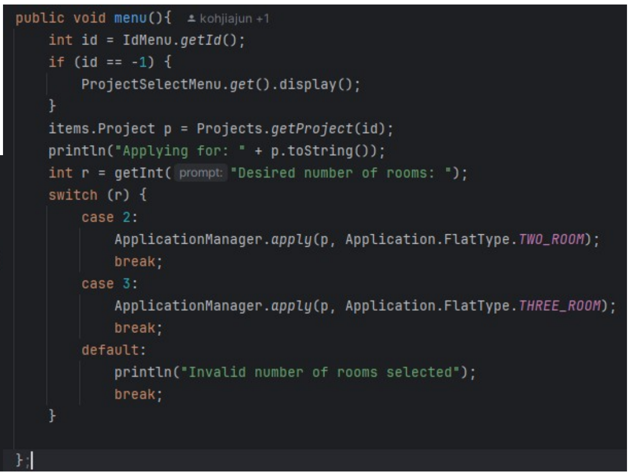
```
/**
 * Abstract method that must be implemented by subclasses
 * to define the menu behavior.
 */
public abstract void menu();  42 implementations  eyewhy
```

In the same class Menu, an abstract method, menu(), was declared. This abstract method must be implemented by its subclass.

The following 2 snippets are examples of how different Menu subclass implement the same menu() method differently.

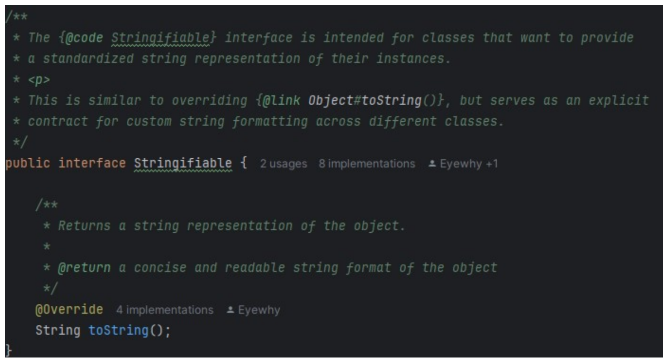


```java
public void menu(){  Eyewhy
    User currentUser = AppUserManager.getCurrentUser();
    if (!(currentUser instanceof Applicant)) {  //typecast check
        println("Error: Only applicants can apply for projects.");
        return;
    }
    Applicant applicant = (Applicant) currentUser; // safe cast
    ArrayList<Application> list = Applications.filter(applicant);
    for (Application a : list) println(a.toString());
    if (list.isEmpty()) println("No Applications Found.");
};
```

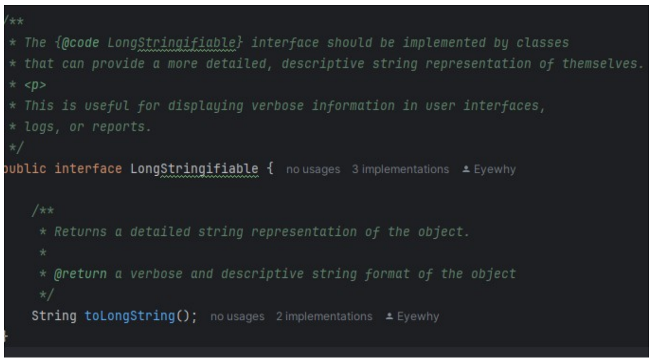


```java
public void menu(){  kohjiajun +1
    int id = IdMenu.getId();
    if (id == -1) {
        ProjectSelectMenu.get().display();
    }
    items.Project p = Projects.getProject(id);
    println("Applying for: " + p.toString());
    int r = getInt( prompt: "Desired number of rooms: ");
    switch (r) {
        case 2:
            ApplicationManager.apply(p, Application.FlatType.TWO_ROOM);
            break;
        case 3:
            ApplicationManager.apply(p, Application.FlatType.THREE_ROOM);
            break;
        default:
            println("Invalid number of rooms selected");
            break;
    }

};
```

4.2.4 Interface usage

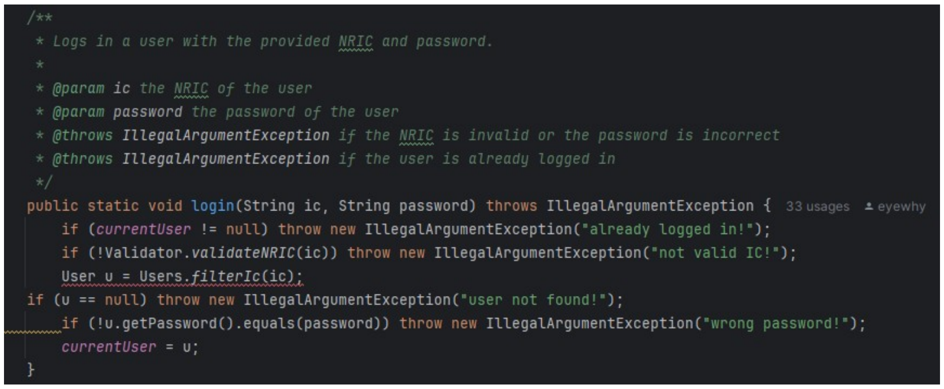


```java
/**
 * The {@code Stringifiable} interface is intended for classes that want to provide
 * a standardized string representation of their instances.
 * <p>
 * This is similar to overriding {@link Object#toString()}, but serves as an explicit
 * contract for custom string formatting across different classes.
 */
public interface Stringifiable {  2 usages  8 implementations  Eyewhy +1

    /**
     * Returns a string representation of the object.
     *
     * @return a concise and readable string format of the object
     */
    @Override  4 implementations  Eyewhy
    String toString();
}
```



```java
/**
 * The {@code LongStringifiable} interface should be implemented by classes
 * that can provide a more detailed, descriptive string representation of themselves.
 * <p>
 * This is useful for displaying verbose information in user interfaces,
 * logs, or reports.
 */
public interface LongStringifiable {  no usages  3 implementations  Eyewhy

    /**
     * Returns a detailed string representation of the object.
     *
     * @return a verbose and descriptive string format of the object
     */
    String toLongString();  no usages  2 implementations  Eyewhy
}
```

4.2.5 Error handling



```java
/**
 * Logs in a user with the provided NRIC and password.
 *
 * @param ic the NRIC of the user
 * @param password the password of the user
 * @throws IllegalArgumentException if the NRIC is invalid or the password is incorrect
 * @throws IllegalArgumentException if the user is already logged in
 */
public static void login(String ic, String password) throws IllegalArgumentException {  33 usages  eyewhy
    if (currentUser != null) throw new IllegalArgumentException("already logged in!");
    if (!Validator.validateNRIC(ic)) throw new IllegalArgumentException("not valid IC!");
    User u = Users.filterIc(ic);
if (u == null) throw new IllegalArgumentException("user not found!");
    if (!u.getPassword().equals(password)) throw new IllegalArgumentException("wrong password!");
    currentUser = u;
}
```

This is an example of error handling. This makes it so that bugs are easier to find as clear error messages are shown.

Chapter 5: Testing

5.1 Test Strategy

To test the accuracy and functionality of our HDB Management Program, we carried out manual testing for each of the test cases listed in Appendix C. Take note that the 'Expected Output' column has been omitted as the actual output is identical to the expected output for each test case. Words highlighted in RED in 'Test Input' are to be keyed in by the user.

Chapter 6: Documentation

6.1 Javadoc

- All public classes/methods documented
- HTML Javadoc generated and included in the Github Repo. See readme.md for detailed instructions.

6.2 Developer Guide

- Environment setup: See readme.md for detailed instructions.
- How to run the project: See readme.md for detailed instructions.
- Dependencies: See pom.xml for our dependencies.
  - Apache POI (to read XLSX file)
  - Maven Javadoc (to generate javadoc)

Chapter 7: Reflection & Challenges

Looking back on the project, we believe we did well with the planning and design. We clearly defined our system architecture before starting on code, by first planning out our features, then creating the Class and Sequence diagrams to design how classes would interact with each other. This allowed our coding process to be much smoother, and prevented us from having to backtrack and refactor, while allowing us to plan in advance how and where to apply SOLID principles to keep our code clean, modular, and easier to maintain. Having a clear design roadmap for everyone to follow allowed us to collaborate well as a team, and have different people working on different parts of the code at the same time.

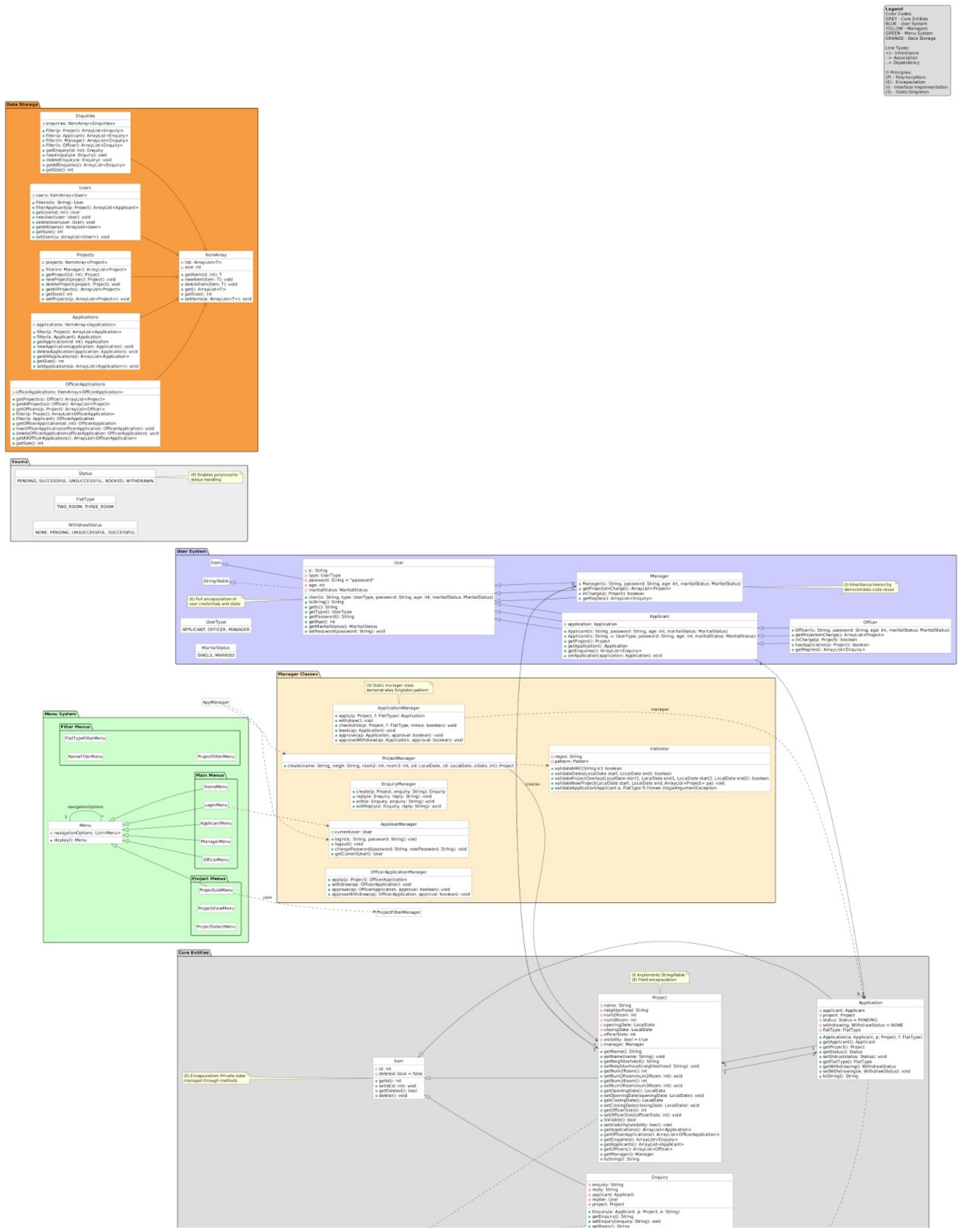
One area of improvement could be to utilise lesser static classes, which would have allowed us to better apply the SOLID principles in certain places, especially the ItemArrays, as explained in the SOLID principles section. Better planning could have led us to better utilise the flexibility provided by interfaces. We could also have done more thorough integration testing, especially to handle edge cases of interactions between user roles. One such example would be handling and reversing system errors, such as if multiple officers are assigned to the same project. The project filtering system could also be improved. While it functions and fulfils its purpose, it is still quite basic. We could add more dynamic filters, like remaining flat quotas, project application deadlines and more. This would allow applicants, officers, and managers to sort and identify ongoing projects (from soonest to latest) and upcoming ones more easily and in general, sort the listings with several more filters to view the most relevant BTO projects.

From this project, we learnt the importance of building the system iteratively. Focusing on drafting the system design before starting our code gave us a solid foundation to expand on, and helped avoid messy rewriting of code that would have happened if we made a fatal oversight, or tried to add too much complexity too soon. Overall, this project taught us that planning, teamwork and simplicity are equally important as technical development, in order to make a stable and manageable system.

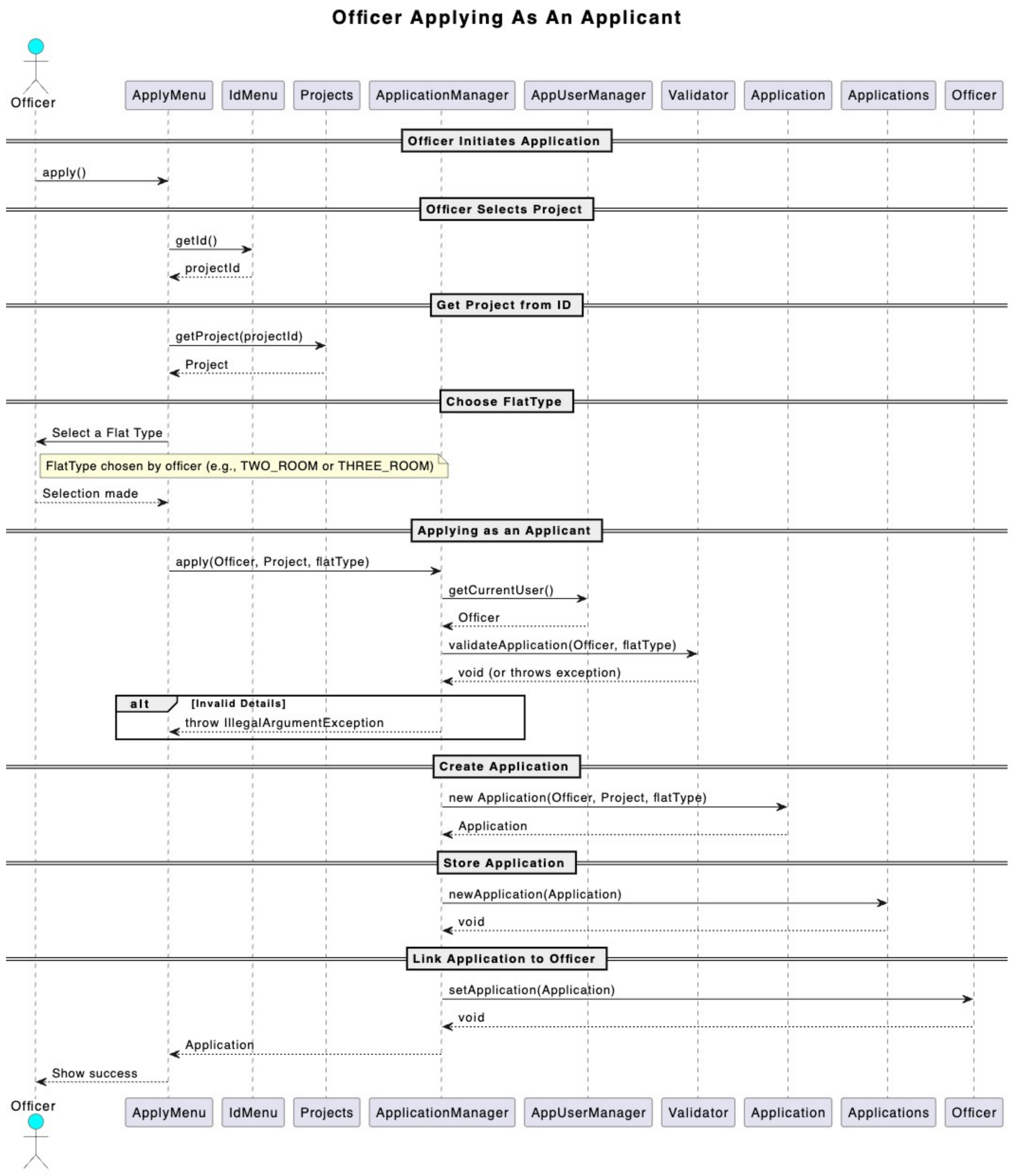
Chapter 8: References

- GitHub repo: https://github.com/Eyewhy/SC2002-Project
- Attachments:
  - Appendix A: Class Diagram
  - Appendix B: Sequence Diagrams
  - Appendix C: Testing Table
  - All appendixes are also uploaded to the GitHub repo (under SC2002 FOLDER).
  - Source code
  - Documentation (Javadoc)

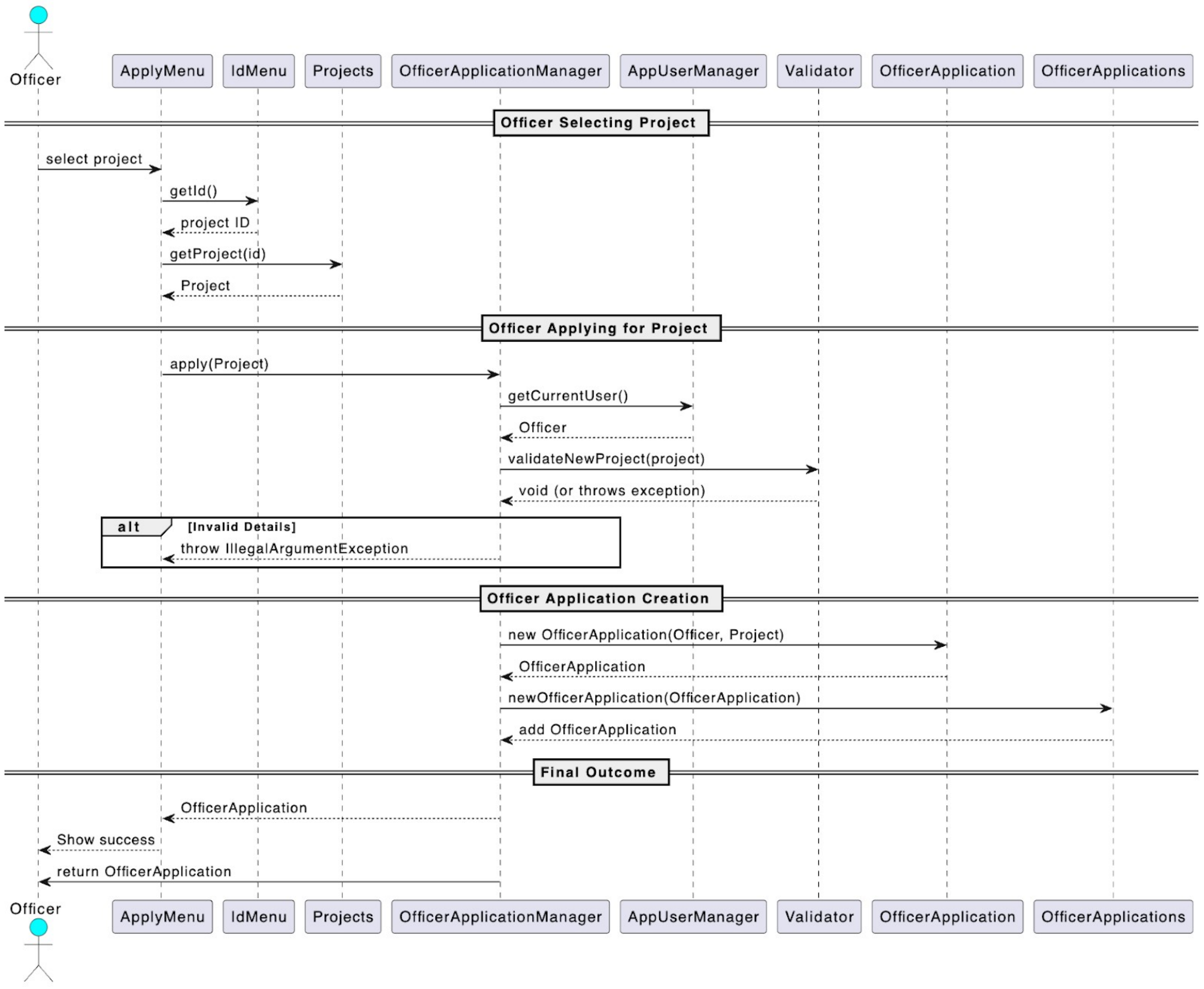
Appendix A: Class Diagram



Appendix B: Sequence Diagrams



Officer Registering to Handle A Project


Officer
ApplyMenu
IdMenu
Projects
OfficerApplicationManager
AppUserManager
Validator
OfficerApplication
OfficerApplications
Officer Selecting Project
select project
getId()
project ID
getProject(id)
Project
Officer Applying for Project
apply(Project)
getCurrentUser()
Officer
validateNewProject(project)
void (or throws exception)
alt
[Invalid Details]
throw IllegalArgumentException
Officer Application Creation
new OfficerApplication(Officer, Project)
OfficerApplication
newOfficerApplication(OfficerApplication)
add OfficerApplication
Final Outcome
OfficerApplication
Show success
return OfficerApplication

Appendix C: Test Table

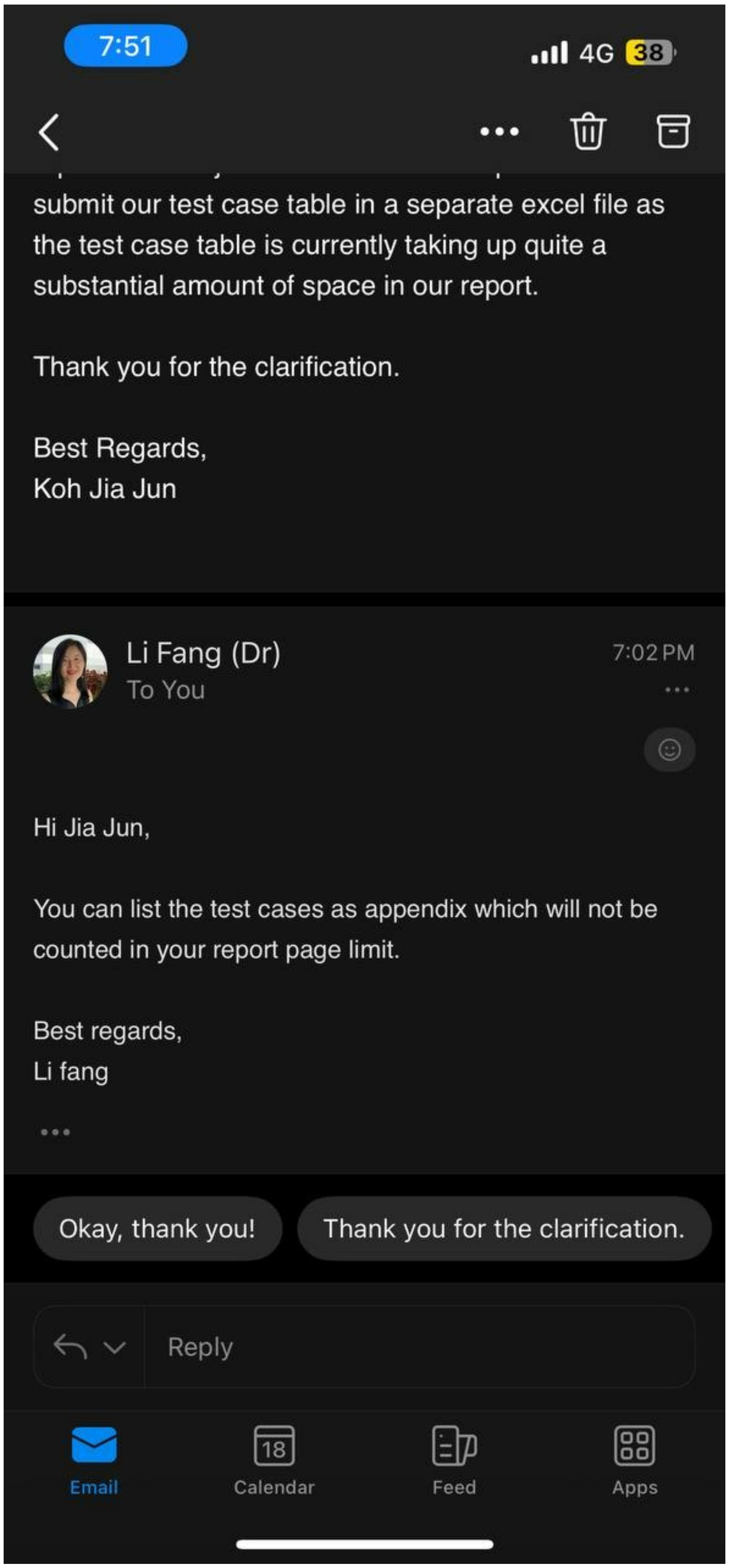
No.	Test Case	Expected Behavior	Test Input	Actual Output	Verdict	Remarks
1	Valid User Login	User should be able to access their dashboard based on their roles	NRIC: T7654321A Password: password	Welcome to HDB Management App, Applicant	Pass	Accesses the Applicant portal.
			NRIC: S1234567M Password: password	Welcome to HDB Management App, Manager	Pass	Accesses the Manager portal.
			NRIC: S7654321O Password: password	Welcome to HDB Management App, Officer	Pass	Accesses the Officer portal.
2	Invalid NRIC Format	User receives a notification about incorrect NRIC format	NRIC: T123456G Password: password	Error: not valid IC!	Pass	NRIC is too short.
			NRIC: S12345678A Password: password	Error: not valid IC!	Pass	NRIC is too long.
			NRIC: A1234567B Password: password	Error: not valid IC!	Pass	NRIC does not begin with 'S' or 'T'.
3	Incorrect Password	System should deny access and alert the user to incorrect password	NRIC: T7654321A Password: passwo	Error: wrong password!	Pass	Password is wrong (should either be "password" or whatever it has been changed to after changing password
4	Password Change Functionality	System updates password, prompt re login and allows login with new credentials	NRIC: T7654321A Password: password Select an option: 7 Current Password: password New Password: testpw Confirm Password: testpw Option: 0 NRIC: T7654321A Password: testpw	Password Changed Successfully. Welcome to HDB Management App, Applicant.	Pass	After changing the password, the user is only able to login using the new updated password. Old default password will not work anymore.
5	Project Visibility Based on User Group and Toggle	Projects are visible to users based on their age, marital status and the visibility setting	NRIC: T7654321F Password: password Option: 1	Projects: No Projects Found.	Pass	Single, under 21
			NRIC: T7654321C Password: password Option: 1	Projects: ID 4: 2 ROOM ONLY in D-street. Open from 2024-02-01 to 2024-03-02.	Pass	Single, 35 and above. Only can apply for 2-Room.
			NRIC: T7654321E Password: password Option: 2	Projects: ID 4: 2 ROOM ONLY in D-street. Open from 2024-02-01 to 2024-03-02. ID 5: 3 ROOM ONLY in D-street. Open from 2023-02-01 to 2023-03-02. ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10. ID 1: Bubbling Brooks in B-ville. Open from 2025-03-11 to 2025-03-13. ID 2: Cowering Canvases in C-ity. Open from 2025-03-01 to 2025-03-01. ID 3: Dancing Dames in D-street. Open from 2025-02-01 to 2025-03-02.	Pass	Married, above 40 years old. Can apply for any flat types.
6	Project Application	Users can only apply for projects relevant to their group or when visibility is off	NRIC: T7654321A Password: password Option: 1 ID: 1 Option: 1 Desired Number of Rooms: 3	Error: inapplicable flattype!	Pass	Single 35 year old CANNOT apply for a 3 room flat.
7	Viewing Application Status after Visibility Toggle Off	Applicants continue to have access to their application details regardless of project visibility.	NRIC: S1234567M Password: password Option: 4, ID: 0 Option: 4 <Logout> NRIC: T7654321B Password: password Option: 5	ID 6: B applying for Amazing Apricots, currently SUCCESSFUL and NOT withdrawing.	Pass	Manager toggles visibility of Amazing Apricots project to off. Applicant is still able to view application details after visibility is toggled off.
8	Single Flat Booking per	System allows booking one flat and restricts	NRIC: S7654321O Password: password Option: 3, ID: 0	Error: application not successful!	Pass	Officer is unable to book a second flat for

	Successful Applicatio n	further bookings	Option: 4, Option: 1 Select an application by ID: 6, Option: 4, Option: 1 Select an application by ID: 6			the applicant, only 1 booking.
9	Applicant's enquiries manageme nt	Enquiries can be successfully submitted, displayed, modified, and removed.	NRIC: T7654321B Password: password Option: 1, ID: 1 Option: 2 Type your enquiry: How much? Option: 3 ID: 2 Edit your enquiry: How much!!! Option: 2 ID: 2	<After creating enquiry> ENQUIRIES ID 2: B: How much? <After editing enquiry> Enquiry edited. Enquiries: ID 2: B: How much!!! <After deleting enquiry> Enquiry deleted.	Pass	Applicant is able to create, edit, and delete his enquiry.
10	HDB Officer Registratio n Eligibility	System allows registration only under compliant conditions	NRIC: S7654329O Password: password Option: 6 Option: 7 ID: 2 NRIC: S7654321O Password: password Option: 6 Option: 7 ID: 3	No Applications Found. Applying for: ID 2: Cowering Canvases in C-ity. Open from 2025-03-01 to 2025-03-01. ID 2: O2 applying for Cowering Canvases, currently PENDING and NOT withdrawing. ID 5: O applying for Dancing Dames, currently PENDING and NOT withdrawing. Error: has applied for project!	Pass	Officer is able to register for a Project if he does not have any pending applications. Officer is unable to register for a Project if he is an applicant for another project.
11	HDB Officer Registratio n Status	Officers can view pending or approved status updates on their profiles.	NRIC: S7654321O Password: password Option: 1 ID: 0 Option: 6 NRIC: S1234567M Password: password Option: 2 ID: 0 Option: 5 Option: 1 Select an application by ID: 2 Option: 1 NRIC: S7654321O Password: password Option: 1 ID: 0 Option: 6	Applications: ID 2: C applying for Amazing Apricots, currently PENDING and NOT withdrawing. <After Manager approves applicant> ID 2: C applying for Amazing Apricots, currently SUCCESSFUL and NOT withdrawing.	Pass	After the Manager makes an update to an applicant's application, the Officer of that project is able to view pending or approved status on their profiles.
12	Project Detail Access for HDB Officer	Officers can always access full project details, even when visibility is turned off.	NRIC: S1234567M Password: password Option: 2 ID: 0 Option: 4 NRIC: S7654321O Password: password Option: 1 ID: 0	<Visibility set to no> PROJECT DETAILS Project ID 0: Name: Amazing Apricots Neighbourhood: A-hood 2 Room Units Available: 1 3 Room Units Available: 3 Application Period: 2025-03-02 to 2025-03-10 Visible: no STAFF DETAILS Manager: M Officers: O ENQUIRIES ID 0: A: Hello hello M: Good evening. ID 1: A: HiHi O: Hello?	Pass	Officer of the project is still able to view project details even when visibility is set to 'no'.
13	Restriction on Editing Project Details	Edit functionality is disabled or absent for HDB Officers.	NRIC: S1234567M Password: password Option: 2 ID: 0 Option: 1 NRIC: S7654321O Password: password Option: 1 ID: 0	<Manager> Editing Project Details… <Officer> <No option to edit project details>	Pass	HDB Officers are unable to access the edit functionality for Projects.
14	Response to Project Enquiries	Officers & Managers can access and respond to	NRIC: T7654321A Password: password Option: 1 ID: 1 Option: 2	<Applicant makes an enquiry> ENQUIRIES ID 2: A: How much?! <Manager or Officer replies to enquiry> ENQUIRIES	Pass	Managers are able to access and respond to enquiries made. Officers are also able to

		enquiries efficiently.	Type your enquiry: How much?! NRIC: S1234567M Password: password Option: 2 ID: 1 Option: 7 Select an enquiry by ID: 2 Type your enquiry: 1 dollar	ID 2: A: How much?! M: 1 dollar		access and respond to enquiries made.
15	Flat Selection and Booking Manageme nt	Officers retrieve the correct application, update flat availability accurately, and correctly log booking details in the applicant's profile.	NRIC: S7654321O Password: password Option: 3 ID: 0 Option: 6 Option: 1 Select an application by ID: 6 Option: 7	PROJECT DETAILS Project ID 0: Name: Amazing Apricots Neighbourhood: A-hood 2 Room Units Available: 1 3 Room Units Available: 3 Application Period: 2025-03-02 to 2025-03-10 Visible: yes ID 6: B applying for Amazing Apricots, currently SUCCESSFUL and NOT withdrawing. Application booked. 2 Room Units Available: 0 APPLICATION ID 6 Name: B NRIC: T7654321B Age: 21 Marital Status: MARRIED Flat Type: TWO_ROOM Project: ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10. Status: BOOKED Withdrawing: NOT	Pass	After the Officer books a flat for a successful applicant, the remaining number of flats with the same room type is updated. Officer can retrieve the applicant's BTO application with applicant's NRIC, update applicant's project status, from "successful" to "Booked", and update applicant's profile with the type of flat (2-Room or 3 Room) chosen under a project.
16	Receipt Generation for Flat Booking	Accurate and complete receipts are generated for each successful booking	NRIC: S7654321O Password: password Option: 1 ID: 0 Option: 6 Option: 1 Select an application by ID: 6 Option: 7	Application booked. <Receipt Generation> APPLICATION RECEIPT APPLICATION ID 6 Name: B NRIC: T7654321B Age: 21 Marital Status: MARRIED Flat Type: TWO_ROOM Project: ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10. Status: BOOKED Withdrawing: NOT PROJECT DETAILS Project ID 0: Name: Amazing Apricots Neighbourhood: A-hood 2 Room Units Available: 0 3 Room Units Available: 3 Application Period: 2025-03-02 to 2025-03-10 Visible: yes STAFF DETAILS Manager: M Officers: O ENQUIRIES ID 0: A: Hello hello M: Good evening. ID 1: A: HiHi O: Hello?	Pass	After booking the flat for a successful applicant, the program is able to generate accurate receipts for each applicant. Receipts of other applicants have been omitted in this report due to page limit.
17	Create, Edit, and Delete BTO Project Listings	Managers should be able to add new projects, modify existing project details, and remove projects from the system	NRIC: S1234567M Password: password Option: 2 ID: 0 Option: 1 Enter name: BeepBoop Residence Enter neighbourhood: Serangoon	<Editing Existing Project> PROJECT DETAILS Project ID 0: Name: BeepBoop Residence Neighbourhood: Serangoon 2 Room Units Available: 2 3 Room Units Available: 4 Application Period: 2025-03-02 to 2025-03-10 Visible: yes	Pass	Managers are able to edit and delete existing projects, as well as create new project listings.

			Enter number of 2-room units: 2 Enter number of 3-room units: 4 Enter a date (yyyy-mm-dd): 2025-03-02 Enter a date (yyyy-mm-dd): 2025-03-10 Enter number of officer slot: 3 Option: 1 Enter the New Project's Details. Enter name: Cardiff Town Enter neighbourhood: Clementi Enter number of 2-room units: 3 Enter number of 3-room units: 3 Enter a date (yyyy-mm-dd): 2025-04-04 Enter a date (yyyy-mm-dd): 2025-04-11 Enter number of officer slot: 3 Option: 2 Confirm Delete? Type 1 to accept: 1	<Adding new Project> Project created with ID 6. PROJECT DETAILS Project ID 6: Name: Cardiff Town Neighbourhood: Clementi 2 Room Units Available: 3 3 Room Units Available: 3 Application Period: 2025-04-04 to 2025-04-11 Visible: yes <Deleting the project> Project Deleted.		
18	Single Project Management per Application Period	System prevents assignment of more than one project to a manager within the same application dates.	NRIC: S1234567M Password: password Option: 5 ID: 0 <Enter> Option: 5 ID: 1 Option: 1 Enter a date (yyyy-mm-dd): 2025-03-02 Enter a date (yyyy-mm-dd): 2025-03-10	PROJECT DETAILS Project ID 0: Name: Amazing Apricots Neighbourhood: A-hood 2 Room Units Available: 1 3 Room Units Available: 3 Application Period: 2025-03-02 to 2025-03-10 Visible: yes STAFF DETAILS Manager: M Error: overlapping dates!	Pass	System is unable to assign projects within the same application dates to the same manager.
19	Toggle Project Visibility	Changes in visibility should be reflected accurately in the project list visible to applicants	NRIC: T7654321B Password: password Option: 1 NRIC: S1234567M Password: password Option: 2 ID: 0 Option: 4 Option: 2 ID: 1 Option: 4 NRIC: T7654321B Password: password Option: 1	<Visibility On> Projects: ID 4: 2 ROOM ONLY in D-street. Open from 2024-02-01 to 2024-03-02. ID 5: 3 ROOM ONLY in D-street. Open from 2023-02-01 to 2023-03-02. ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10. ID 1: Bubbling Brooks in B-ville. Open from 2025-03-11 to 2025-03-13. ID 2: Cowering Canvases in C-ity. Open from 2025-03-01 to 2025-03-01. ID 3: Dancing Dames in D-street. Open from 2025-02-01 to 2025-03-02. <After turning visibility off> Projects: ID 4: 2 ROOM ONLY in D-street. Open from 2024-02-01 to 2024-03-02. ID 5: 3 ROOM ONLY in D-street. Open from 2023-02-01 to 2023-03-02. ID 2: Cowering Canvases in C-ity. Open from 2025-03-01 to 2025-03-01. ID 3: Dancing Dames in D-street. Open from 2025-02-01 to 2025-03-02.	Pass	After the Manager toggles visibility off, those Projects with visibility turned off will not show up in Applicant's list when he lists all Projects.
20	View All and Filtered Project Listings	Managers should see all projects and be able to apply filters to narrow down to their own projects.	NRIC: S1234567M Password: password Option: 2 Option: 3 Option: 3 Type your query: A-hood <Enter> Option: 2	<Before Filtering> Projects: ID 4: 2 ROOM ONLY in D-street. Open from 2024-02-01 to 2024-03-02. ID 5: 3 ROOM ONLY in D-street. Open from 2023-02-01 to 2023-03-02. ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10. ID 1: Bubbling Brooks in B-ville. Open from 2025-03-11 to 2025-03-13. ID 2: Cowering Canvases in C-ity. Open from 2025-03-01 to 2025-03-01.	Pass	Managers are able to view all their projects before filtering and view those select projects after filtering.

				ID 3: Dancing Dames in D-street. Open from 2025-02-01 to 2025-03-02. <After Filtering> Projects: ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10.		
21	Manage HDB Officer Registrations	Managers handle officer registrations effectively, with system updates reflecting changes accurately.	NRIC: S7654329O Password: password Option: 7 ID: 3 NRIC: S1234569M Password: password Option: 2 ID: 3 Option: 6 Option: 1 Select an application by ID: 2 Option: 1	OFFICER APPLICATIONS Remaining Officer Slots: 1 ID 2: O2 applying for Dancing Dames, currently PENDING and NOT withdrawing. Completed. OFFICER APPLICATIONS Remaining Officer Slots: 0 ID 2: O2 applying for Dancing Dames, currently SUCCESSFUL and NOT withdrawing.	Pass	After an officer registers for a Project as an Officer, the Manager of said project is able to register the Officer under the Project.
22	Approve or Reject BTO Applications and Withdrawals	Approvals and rejections are processed correctly, with system updates to reflect the decision	NRIC: S1234567M Password: password Option: 4 ID: 0 Option: 5 Option: 1 Select an application by ID: 2 Option: 1	<Before approving> ID 2: C applying for Amazing Apricots, currently PENDING and NOT withdrawing. <After approving> Completed. ID 2: C applying for Amazing Apricots, currently SUCCESSFUL and NOT withdrawing.	Pass	Managers are able to successfully approve and reject BTO applications and withdrawals.
23	Generate and Filter Reports	Accurate report generation with options to filter by various categories.	NRIC: S1234567M Password: password Option: 7 Option: 2 Option: 2 Option: 6	Current Filters: None. Current Filters: Marital Status: SINGLE Applications: APPLICATION ID 0 Name: A NRIC: T7654321A Age: 35 Marital Status: SINGLE Flat Type: TWO_ROOM Project: ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10. Status: WITHDRAWN Withdrawing: SUCCESSFUL APPLICATION ID 2 Name: C NRIC: T7654321C Age: 35 Marital Status: SINGLE Flat Type: TWO_ROOM Project: ID 0: Amazing Apricots in A-hood. Open from 2025-03-02 to 2025-03-10. Status: PENDING Withdrawing: NOT	Pass	Managers are able to apply filters such as Applicant’s Age, Marital Status, Flat Types, etc. After applying filters, only those select filtered projects will appear when listing all projects.



submit our test case table in a separate excel file as the test case table is currently taking up quite a substantial amount of space in our report.

Thank you for the clarification.

Best Regards,
Koh Jia Jun

Li Fang (Dr) 7:02 PM
To You

Hi Jia Jun,

You can list the test cases as appendix which will not be counted in your report page limit.

Best regards,
Li fang