SC2002 - Object Oriented Design and Programming

Build-to-Order Management System (BTOMS)

Declaration of Original Work for CE/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
Daniel Mark (U2420344L)	SC2002	FCSI	[illegible] 26/4/2025
William Notowibowo (U2423934D)	SC2002	FCSI	[illegible] 26/4/2025
Lim Hwee Woon (U2422201L)	SC2002	FCSI	[illegible] 26/4/2025
Soh Cek Cong (U2423500C)	SC2002	FCSI	[illegible] 26/4/2025

I. Design Considerations

1.1 Approach Taken

The Build-To-Order (BTO) Management System is a system for applicants and HDB staff to view, apply, and manage BTO projects. It will act as a centralized hub for all applicants and HDB staff.

The system employs the entity, controller, and boundary (ECB) class stereotypes. Applicants and HDB staff interact with the system via BtomsApp, represented as a **boundary class**, which acts as the interface between applicants and the system. It displays menus, collects user input, and calls appropriate **control classes** for processing. The Service and Controller classes handle logic implementation, error checking, and exception handling to ensure smooth system operation. The details of applicants, HDB staff, and projects are retrieved from **entity classes**, namely the User, Project, and Application classes.

We have split our system into different packages:

- **Models package:** Contains entity classes such as User, Applicant, HDB Officer, and HDBManager. They contain the data needed for the system to work properly.
- **Interfaces package:** Contains all the Interfaces needed. All interfaces defined in this program start with "I".
- **Controllers package:** Contains control classes that consist of logic to manage the flow of user interactions within the BTO System.
- **Service package:** Contains all the service classes that handle requests from users.
- **View package:** Contains all the boundary classes for users to view certain details.
- **Enumeration package:** Contains all the enumeration needed for the System.
- **Main:** The main program.

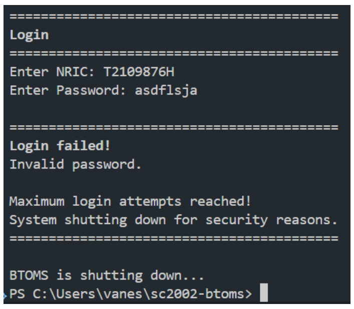
1.2 Assumptions made

Some of the assumptions we are making:

- Data is stored in CSV files, additional projects are added for testing.
- Withdrawal requests are always successful.
- Applicants can submit an enquiry for projects that they have/have not applied for. Once an enquiry has been replied to, the applicant cannot edit or delete it.
- HDB officers cannot cancel their registration after registering for a project.
- HDB managers can only manage one active project at any given point in time.

1.3 Functionalities and Additional features

In addition to the functionalities that were mentioned in the requirements, our team added the following additional features to enhance the usability and realism of the system. Our system automatically warns users and logs them out after a certain number of failed login attempts. This serves as a security measure to prevent any brute-force attempts.



```
=======================================
Login
=======================================
Enter NRIC: T2109876H
Enter Password: asdflsja

=======================================
Login failed!
Invalid password.

Maximum login attempts reached!
System shutting down for security reasons.
=======================================

BTOMS is shutting down...
PS C:\Users\vanes\sc2002-btoms>
```

II. Design Principles

2.1 Object Oriented Programming Principles

Abstraction

Abstraction involves hiding implementation details and exposing only the essential features of an object to simplify usage and reduce complexity. Abstraction is applied via the use of interfaces such as IBTOProjectService, IUserService, and IProjectView. By defining common behaviours through interfaces, our code becomes reusable and extensible. We can define the methods differently in different concrete classes that implement the interfaces, without affecting the rest of the program.

For example, the IUserService interface abstracts user-related operations such as changing of passwords. Our system can be extended in the event that we have to add a new user type (e.g System Developer) without needing to modify the existing user types.

Encapsulation / Information Hiding

Encapsulation is the concept of hiding data and only allowing access through getter and setter methods. It helps in data security and reducing unintended modifications. For example, in the User class, the attributes age and password are declared as private and hence hidden from the rest of the system. This data is protected and free from any accidental modification.

Inheritance

Inheritance allows a class to inherit attributes and methods from another class . This promotes code reusability and hierarchical classification. Subclasses such as HDBManager, HDBOfficer and Applicant are extended from the superclass User. This allows shared attributes (such as name, age, NRIC) and shared methods (such as setPassword) to be defined once and reused. Similarly, AuthApplicantService, AuthHDBManagerService and AuthHDBOfficerService are extended from a common AuthService superclass.

Polymorphism

Polymorphism allows the same method to behave differently depending on the object that is calling it. This is achieved through method overriding and the use of interfaces. For example, our system has a view interface called IProjectView and is implemented by BTOProjectAvailableView class. Different user types—Applicants, HDBOfficers, and HDBManagers— interact with the view system in different ways.

Instead of writing separate methods for each user type, all user types can simply call the same method and the system will determine the behaviour. With polymorphism, our system becomes easily extensible as we can add new functionality by creating new classes inherited from a base class without modifying the base class and other classes derived from the base class.

2.2 SOLID Design Principles

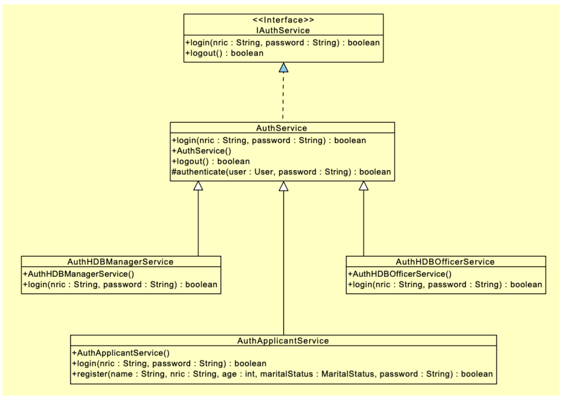
Single Responsibility Principle (SRP)

A class should have only one reason to change, meaning it should have one specific responsibility. SRP is observed in our system as each class only handles a specific function. For example, the AuthService class is responsible only for authentication-related operations such as login and logout. On the other hand, the ReportService class handles the generation of project, flat type and application report.

Open-Closed Principle (OCP)

Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. We have been utilising the concept of abstraction throughout our system via the use of interfaces.

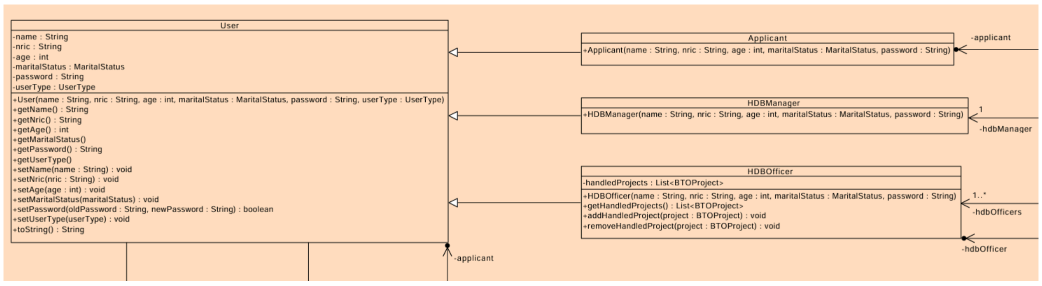
Our interfaces define the expected behaviours, while the concrete classes provide the actual functionality. The AuthService class implements the IAuthService interface, which defines methods such as login() and logout() methods. If we want to introduce a new login method, such as logging in using PIN instead of password, we can simply create a new class (e.g NewAuthService) that implements IAuthService. In this case, the IAuthService interface is closed for modification but allows classes to extend its functionality by implementing it.


<<Interface>>
IAuthService
+login(nric : String, password : String) : boolean
+logout() : boolean
AuthService
+login(nric : String, password : String) : boolean
+AuthService()
+logout() : boolean
#authenticate(user : User, password : String) : boolean
AuthHDBManagerService
+AuthHDBManagerService()
+login(nric : String, password : String) : boolean
AuthHDBOfficerService
+AuthHDBOfficerService()
+login(nric : String, password : String) : boolean
AuthApplicantService
+AuthApplicantService()
+login(nric : String, password : String) : boolean
+register(name : String, nric : String, age : int, maritalStatus : MaritalStatus, password : String) : boolean

<u>Liskov Substitution Principle (LSP)</u>

Subtypes should be <u>substitutable for their base types without altering the correctness of the program</u>. In other words, a <u>subclass must do all the things its superclass does without introducing unexpected behaviours</u>.

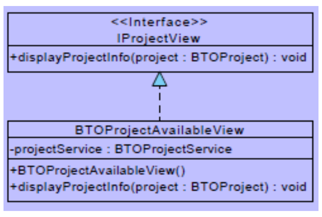
In our system, all user types (Applicant, HDBOfficer, HDBManager) inherit from a common User base class. The Applicant class uses getNric(), getMaritalStatus(), getPassword(), etc—all of which are defined in the base class. The subclass <u>extends the functionality without interfering with how the superclass is expected to behave</u>. Even with these added methods, Applicant still behaves as a User in contexts where only User is expected. This means LSP is preserved.

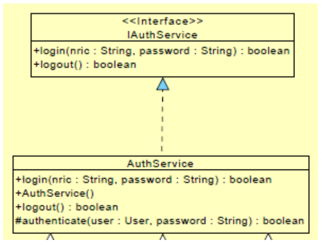

User
-name : String
-nric : String
-age : int
-maritalStatus : MaritalStatus
-password : String
-userType : UserType
+User(name : String, nric : String, age : int, maritalStatus : MaritalStatus, password : String, userType : UserType)
+getName() : String
+getNric() : String
+getAge() : int
+getMaritalStatus()
+getPassword() : String
+getUserType()
+setName(name : String) : void
+setNric(nric : String) : void
+setAge(age : int) : void
+setMaritalStatus(maritalStatus) : void
+setPassword(oldPassword : String, newPassword : String) : boolean
+setUserType(userType) : void
+toString() : String
Applicant
+Applicant(name : String, nric : String, age : int, maritalStatus : MaritalStatus, password : String)
-applicant
HDBManager
+HDBManager(name : String, nric : String, age : int, maritalStatus : MaritalStatus, password : String)
1
-hdbManager
HDBOfficer
-handledProjects : List<BTOProject>
+HDBOfficer(name : String, nric : String, age : int, maritalStatus : MaritalStatus, password : String)
+getHandledProjects() : List<BTOProject>
+addHandledProject(project : BTOProject) : void
+removeHandledProject(project : BTOProject) : void
1..*
-hdbOfficers
-hdbOfficer
-applicant

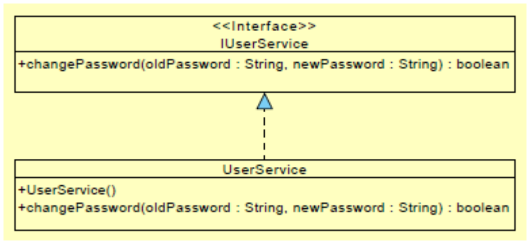
<u>Interface Segregation Principle (ISP)</u>

Classes should not depend on interfaces they do not use. Instead, <u>break down large interfaces into smaller, more specific ones</u>.

For example, our IBTOProjectService only contains methods related to the BTO application process while our IUserService interface only contains methods related to the change of passwords. By assigning methods into their respective smaller interfaces, we ensure that any classes that implement the interface will not have to implement any methods that they do not require.


<<Interface>>
IProjectView
+displayProjectInfo(project : BTOProject) : void
BTOProjectAvailableView
-projectService : BTOProjectService
+BTOProjectAvailableView()
+displayProjectInfo(project : BTOProject) : void


<<Interface>>
IAuthService
+login(nric : String, password : String) : boolean
+logout() : boolean
AuthService
+login(nric : String, password : String) : boolean
+AuthService()
+logout() : boolean
#authenticate(user : User, password : String) : boolean


<<Interface>>
IUserService
+changePassword(oldPassword : String, newPassword : String) : boolean
UserService
+UserService()
+changePassword(oldPassword : String, newPassword : String) : boolean

Dependency Injection Principle (DIP)

DIP states that high-level modules should not depend on low-level modules and both should depend on abstractions. Abstractions should not depend on details and details should depend on abstractions. Without DIP, a high level class like the Controller class would directly depend on a concrete implementation like BTOProjectAvailableView class, which tightly couples the two and makes it harder to extend the system. By applying DIP, the Controller class depends on abstraction—the IProjectView interface. The BTOProjectAvailableView class then implements this interface and can be modified without affecting the Controller classes.

III. UML Class Diagram

Our UML Class Diagram shows the structure of our BTO System. We categorised our classes in different categories according to their functions and class types. This organisation makes it easier for us to show the relations between classes. The diagram is provided in a separate PDF file.

At its core, the system implements the Model-View-Controller (MVC) architectural pattern, clearly separating domain models (like BTOProject, BTOApplication, User and its subclasses) from controllers that handle user operations and views that render information. The extensive use of interfaces (IFileDataService, IAuthService, IBTOProjectService, IUserService) demonstrates strong adherence to the Interface Segregation Principle, allowing for loose coupling between components and facilitating future extensions or modifications. The system features specialized controllers for each user type (HDBManagerController, ApplicantController, HDBOfficerController), promoting separation of concerns. Data persistence is handled through the CsvDataService implementation, centralizing file operations while maintaining data integrity.

IV. UML Sequence Diagram

Our UML sequence diagram shows the dynamic behaviour of our BTO System by illustrating how a HDB Officer interacts with the system when applying for BTO, registering to handle a project and handling flat selection for successful applicants. The diagram is provided in a separate PDF file.

Key features include multiple verification stages where the HDB officer interacts with various system components to validate applicant information, process payments, and approve applications. The diagram shows how officers authenticate users and verify essential documents, with specific attention to payment processing (visible in multiple sections with payment validation sequences). The system has distinct phases for document submission, verification, and approval, with automated notifications at critical checkpoints. Security measures are evident through authentication sequences, while database interactions occur throughout to store and retrieve application information. The diagram also captures exception handling for scenarios like failed payments or incomplete documentation, showing alternative process flows when standard verification fails.

V. Testing and Demonstration

Our team utilises both unit testing and manual functional testing to ensure system reliability and correctness. Unit testing was done in the early development phase to test individual classes and methods. Afterwards, manual functional testing was done to test the user interface and check for any possible input errors.

Test Case 1: HDB Officer, Emily, Applying for BTO flats.

[Initial User Interface]
When the system first starts , Emily is presented with the following options:
1) Login as an Applicant
2) Login as a HDB Officer
3) Login as a HDB Manager
4) Register New Account
5) Shutdown the System.

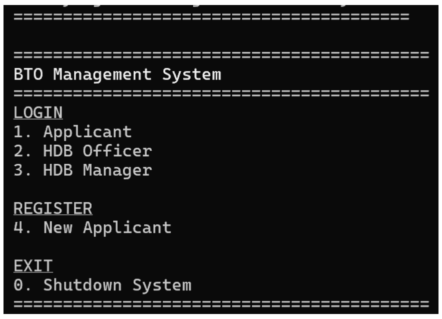
[Login and Navigation]
Emily selects "HDB Officer", enters her details, and successfully logs into the system.
She chooses to view her managed BTO projects and available BTO projects to apply for..

[Eligibility Check]
The system shows her a list of available BTO projects. Her eligibility is determined as follows:

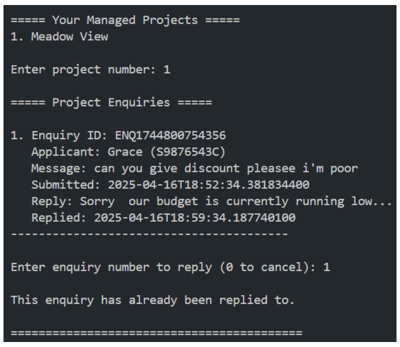
- Emily is married, and hence eligible for both 2-Room and 3-Room flats.
- She cannot apply for Binjai Hall as she is listed as one of the project officers for that development.
- She cannot apply for Rumah Daniel Baru as the application period has ended (she is applying on 17 April).

[Application Submission]
Emily chooses to apply for the Meadow project.

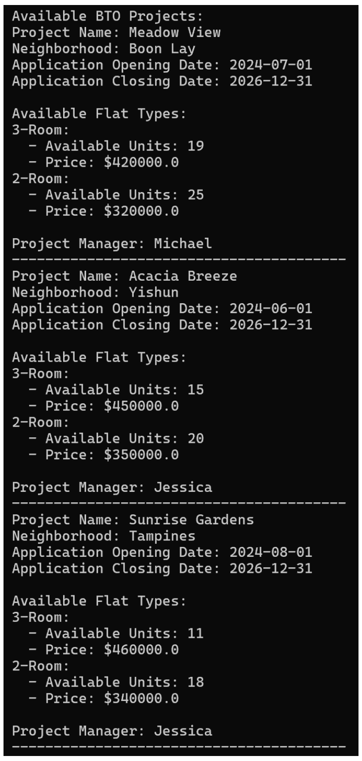
The system displays a confirmation message stating that her application was submitted successfully. It also informs her that another HDB officer will contact her for flat booking if her application is accepted.



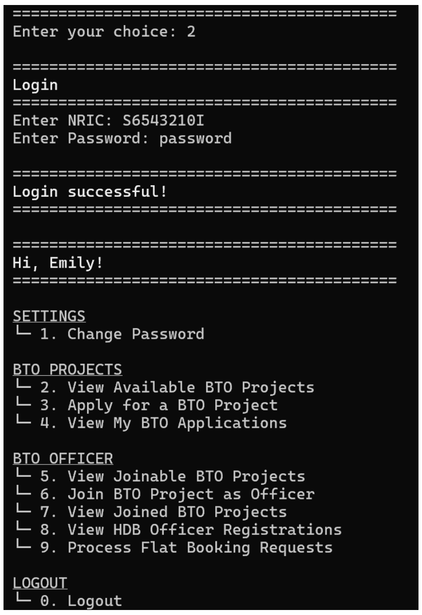
```
========================================

==========================================
BTO Management System
==========================================
LOGIN
1. Applicant
2. HDB Officer
3. HDB Manager

REGISTER
4. New Applicant

EXIT
0. Shutdown System
==========================================
```



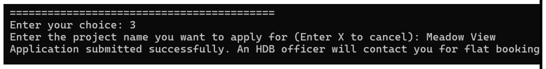
```
===== Your Managed Projects =====
1. Meadow View

Enter project number: 1

===== Project Enquiries =====

1. Enquiry ID: ENQ1744800754356
   Applicant: Grace (S9876543C)
   Message: can you give discount pleasee i'm poor
   Submitted: 2025-04-16T18:52:34.381834400
   Reply: Sorry  our budget is currently running low...
   Replied: 2025-04-16T18:59:34.187740100
----------------------------------------

Enter enquiry number to reply (0 to cancel): 1

This enquiry has already been replied to.

==========================================
```



```
Available BTO Projects:
Project Name: Meadow View
Neighborhood: Boon Lay
Application Opening Date: 2024-07-01
Application Closing Date: 2026-12-31

Available Flat Types:
3-Room:
  - Available Units: 19
  - Price: $420000.0
2-Room:
  - Available Units: 25
  - Price: $320000.0

Project Manager: Michael
----------------------------------------
Project Name: Acacia Breeze
Neighborhood: Yishun
Application Opening Date: 2024-06-01
Application Closing Date: 2026-12-31

Available Flat Types:
3-Room:
  - Available Units: 15
  - Price: $450000.0
2-Room:
  - Available Units: 20
  - Price: $350000.0

Project Manager: Jessica
----------------------------------------
Project Name: Sunrise Gardens
Neighborhood: Tampines
Application Opening Date: 2024-08-01
Application Closing Date: 2026-12-31

Available Flat Types:
3-Room:
  - Available Units: 11
  - Price: $460000.0
2-Room:
  - Available Units: 18
  - Price: $340000.0

Project Manager: Jessica
----------------------------------------
```



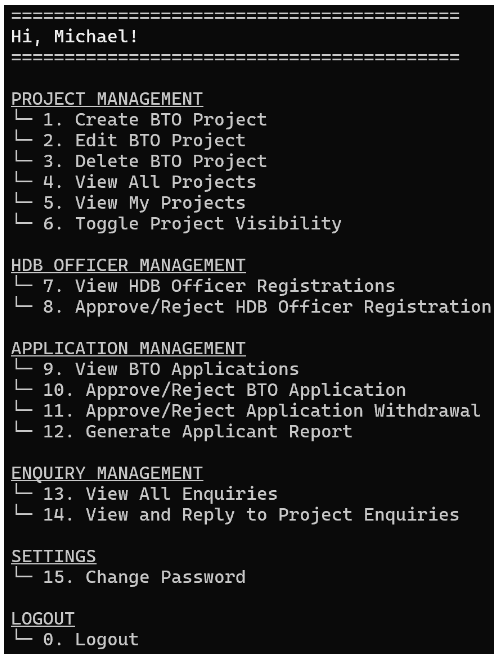
```
========================================
Enter your choice: 2

========================================
Login
========================================
Enter NRIC: S6543210I
Enter Password: password

========================================
Login successful!
========================================

========================================
Hi, Emily!
========================================

SETTINGS
└─ 1. Change Password

BTO PROJECTS
└─ 2. View Available BTO Projects
└─ 3. Apply for a BTO Project
└─ 4. View My BTO Applications

BTO OFFICER
└─ 5. View Joinable BTO Projects
└─ 6. Join BTO Project as Officer
└─ 7. View Joined BTO Projects
└─ 8. View HDB Officer Registrations
└─ 9. Process Flat Booking Requests

LOGOUT
└─ 0. Logout
```



```
========================================
Enter your choice: 3
Enter the project name you want to apply for (Enter X to cancel): Meadow View
Application submitted successfully. An HDB officer will contact you for flat booking
```

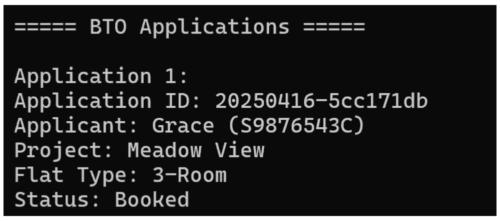
<u>Test Case 2: HDB manager Michael Views BTO Applications and responds to Enquiries</u>

[Initial Scenario]
Michael logs into the system and is presented with many options that are related to his project management.



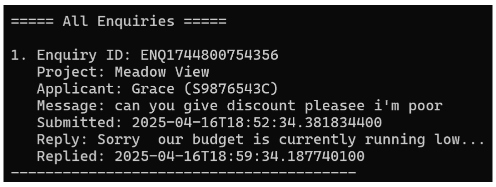
```
==========================================
Hi, Michael!
==========================================

PROJECT MANAGEMENT
└─ 1. Create BTO Project
└─ 2. Edit BTO Project
└─ 3. Delete BTO Project
└─ 4. View All Projects
└─ 5. View My Projects
└─ 6. Toggle Project Visibility

HDB OFFICER MANAGEMENT
└─ 7. View HDB Officer Registrations
└─ 8. Approve/Reject HDB Officer Registration

APPLICATION MANAGEMENT
└─ 9. View BTO Applications
└─ 10. Approve/Reject BTO Application
└─ 11. Approve/Reject Application Withdrawal
└─ 12. Generate Applicant Report

ENQUIRY MANAGEMENT
└─ 13. View All Enquiries
└─ 14. View and Reply to Project Enquiries

SETTINGS
└─ 15. Change Password

LOGOUT
└─ 0. Logout
```

[Viewing BTO Applications]
Michael views the list of applications and finds that Grace has applied for his BTO project, Meadow View, and has already booked a flat with a HDB officer.



```
===== BTO Applications =====

Application 1:
Application ID: 20250416-5cc171db
Applicant: Grace (S9876543C)
Project: Meadow View
Flat Type: 3-Room
Status: Booked
```

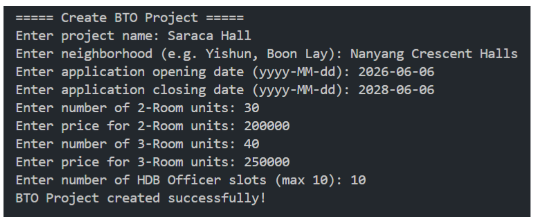
[View Enquiries]
Michael accesses the Enquiries section and finds that an applicant has submitted a query.
Michael reads the enquiry and replies to the applicant's question through the system.



```
===== All Enquiries =====

1. Enquiry ID: ENQ1744800754356
   Project: Meadow View
   Applicant: Grace (S9876543C)
   Message: can you give discount pleasee i'm poor
   Submitted: 2025-04-16T18:52:34.381834400
   Reply: Sorry  our budget is currently running low...
   Replied: 2025-04-16T18:59:34.187740100
----------------------------------------
```

Test Case 3: HDB manager Creates BTO

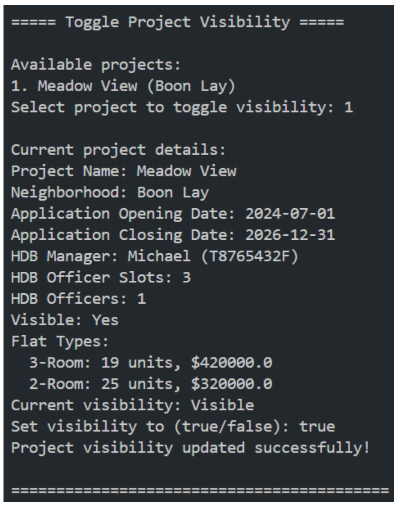
This allows HDB Managers to add new BTO projects into the system. This allows HDB officers and applicants to have new projects to apply for. The system will prompt the manager to enter details of the projects such as name, place and price etc.



```
===== Create BTO Project =====
Enter project name: Saraca Hall
Enter neighborhood (e.g. Yishun, Boon Lay): Nanyang Crescent Halls
Enter application opening date (yyyy-MM-dd): 2026-06-06
Enter application closing date (yyyy-MM-dd): 2028-06-06
Enter number of 2-Room units: 30
Enter price for 2-Room units: 200000
Enter number of 3-Room units: 40
Enter price for 3-Room units: 250000
Enter number of HDB Officer slots (max 10): 10
BTO Project created successfully!
```

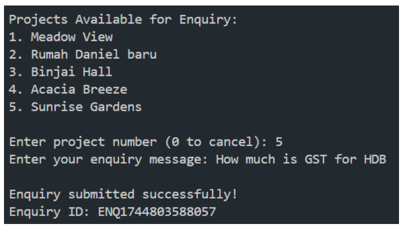
Test Case 4: Manager Toggles Project Visibility

Projects are visible to users based on their age, marital status and the visibility setting.

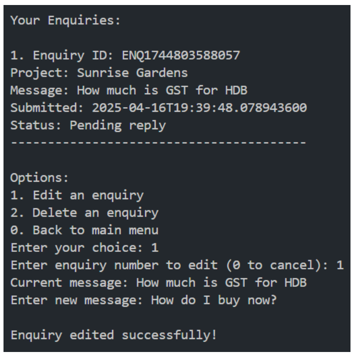


```
===== Toggle Project Visibility =====

Available projects:
1. Meadow View (Boon Lay)
Select project to toggle visibility: 1

Current project details:
Project Name: Meadow View
Neighborhood: Boon Lay
Application Opening Date: 2024-07-01
Application Closing Date: 2026-12-31
HDB Manager: Michael (T8765432F)
HDB Officer Slots: 3
HDB Officers: 1
Visible: Yes
Flat Types:
  3-Room: 19 units, $420000.0
  2-Room: 25 units, $320000.0
Current visibility: Visible
Set visibility to (true/false): true
Project visibility updated successfully!

========================================
```

Test Case 5: Applicant submits enquiry and edits enquiry.

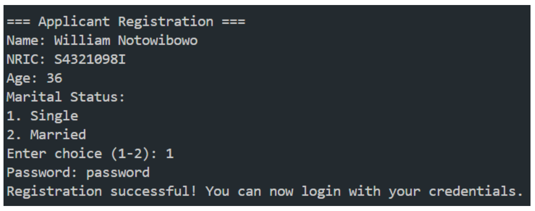


```
Projects Available for Enquiry:
1. Meadow View
2. Rumah Daniel baru
3. Binjai Hall
4. Acacia Breeze
5. Sunrise Gardens

Enter project number (0 to cancel): 5
Enter your enquiry message: How much is GST for HDB

Enquiry submitted successfully!
Enquiry ID: ENQ1744803588057
```

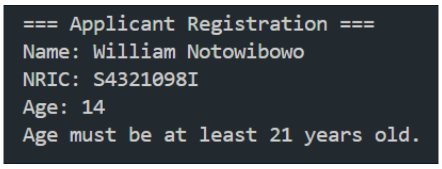


```
Your Enquiries:

1. Enquiry ID: ENQ1744803588057
Project: Sunrise Gardens
Message: How much is GST for HDB
Submitted: 2025-04-16T19:39:48.078943600
Status: Pending reply
---------------------------------------

Options:
1. Edit an enquiry
2. Delete an enquiry
0. Back to main menu
Enter your choice: 1
Enter enquiry number to edit (0 to cancel): 1
Current message: How much is GST for HDB
Enter new message: How do I buy now?

Enquiry edited successfully!
```

Test Case 6: Register new applicant (failed and successful test case)



```
=== Applicant Registration ===
Name: William Notowibowo
NRIC: S4321098I
Age: 36
Marital Status:
1. Single
2. Married
Enter choice (1-2): 1
Password: password
Registration successful! You can now login with your credentials.
```



```
=== Applicant Registration ===
Name: William Notowibowo
NRIC: S4321098I
Age: 14
Age must be at least 21 years old.
```

<u>Test Case 7: Manager Generates Comprehensive Report</u> **[Initial Scenario]** Manager Priya logs in and selects "Generate Applicant Report" **[Report Configuration]** Priya chooses to include all her projects in the report. She selects to include both SUCCESSFUL and BOOKED applications.	
<u>Test Case 8: Project Date Overlap Prevention Test</u> **[Initial Scenario]** Manager Chen has created "Emerald Gardens" with application period May 1-30, 2025. **[Creating New Project]** Chen attempts to create "Ruby Heights" with application period May 15-June 15, 2025. **[System Response]** ● System detects overlap with existing project dates ● System displays warning: "You already have a project with overlapping application dates" ● System prevents creation of the new project [Alternative Dates] Chen modifies the dates to June 1-30, 2025 (no overlap). System allows creation of "Ruby Heights" with the non-overlapping dates.	

VI. Reflection

4.1 Challenges/ Solutions undertaken/ Possible Improvements

It was difficult to showcase how we applied various OOP and SOLID principles in every single scenario. Instead, our team cherry-picked key examples that best illustrate our understanding throughout the project. We encountered some trade offs in our design. At one point, we considered combining certain packages to simplify the structure. However, we ultimately chose to separate the classes into distinct Model, Service, View, and Controller packages. This makes our system more organised and maintainable in the long run.

For future enhancements, we plan to implement several specific improvements to address the system's current limitations. Firstly, the Data Layer which includes a DataStore for all entity classes, can follow the SRP more by utilizing the Repository Pattern - a repository for each entity class. Secondly, security in the personal information of users can be improved. In particular, passwords should be hashed before storing into the data files. Finally, CLI features such as autocomplete, as well as more detailed descriptions, can make the interface more user-friendly and intuitive.

4.2 Knowledge Learnt

We now have a better understanding of the different design principles such as OOP and SOLID principles. Moreover, we have always assumed valid user input when working on our lab assignments. This project was different as we had to incorporate exception handling into our program. This made our code more robust and brought us closer to the kind of real-world development practices expected in the industry.

When we first started learning how to code, we were introduced to flowcharts, which became impractical as our programs grew in size and complexity. This project taught us how to utilize the Visual Paradigm in creating UML class diagrams and sequence diagrams. These tools are essential for helping us plan our code and for enabling others to understand the structure and behavior of our system.