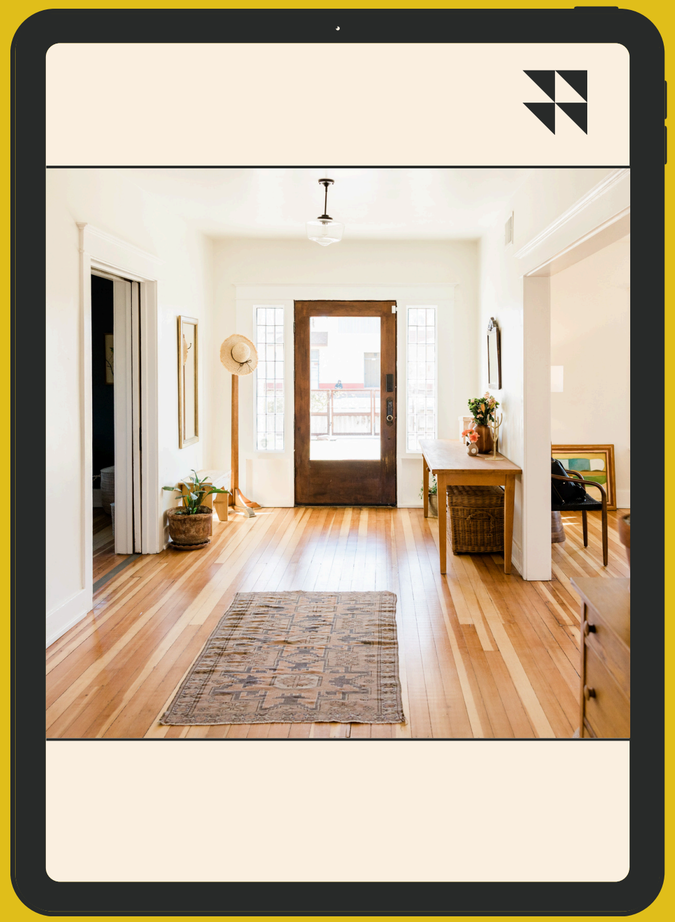
HDB PROJECT

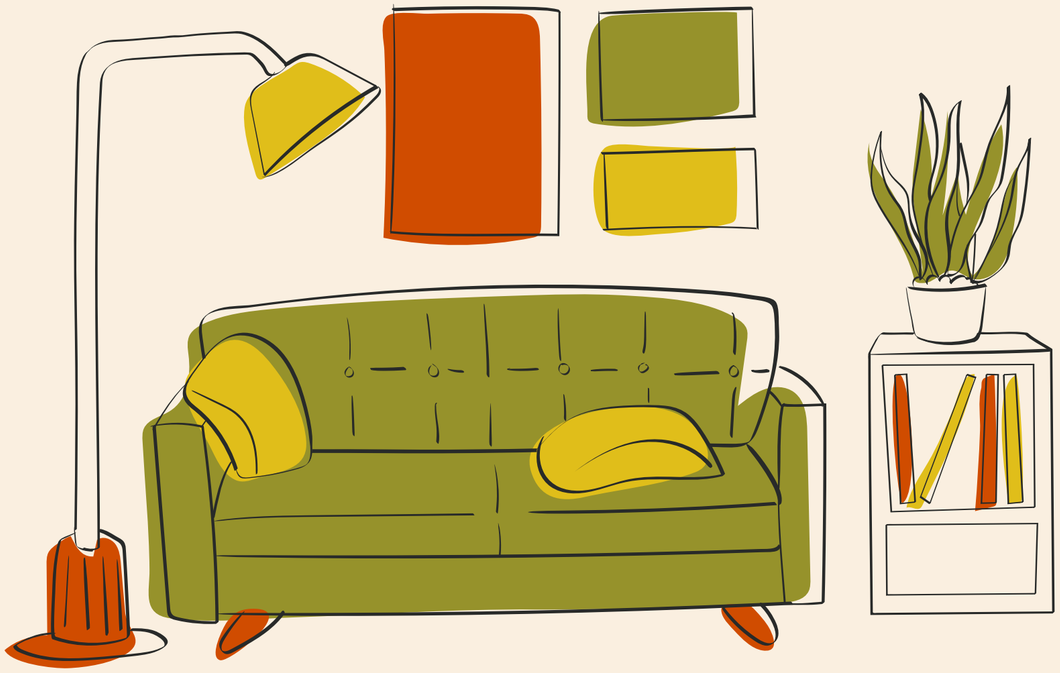
Presented By:
FCS2 Group 6

Content

- Introduction
- Features & Scope
- Structure Planning
- Trade-offs
- UML diagrams
- OOD Principles (SOLID)
- Implementation - Demo
- Reflection



Introduction



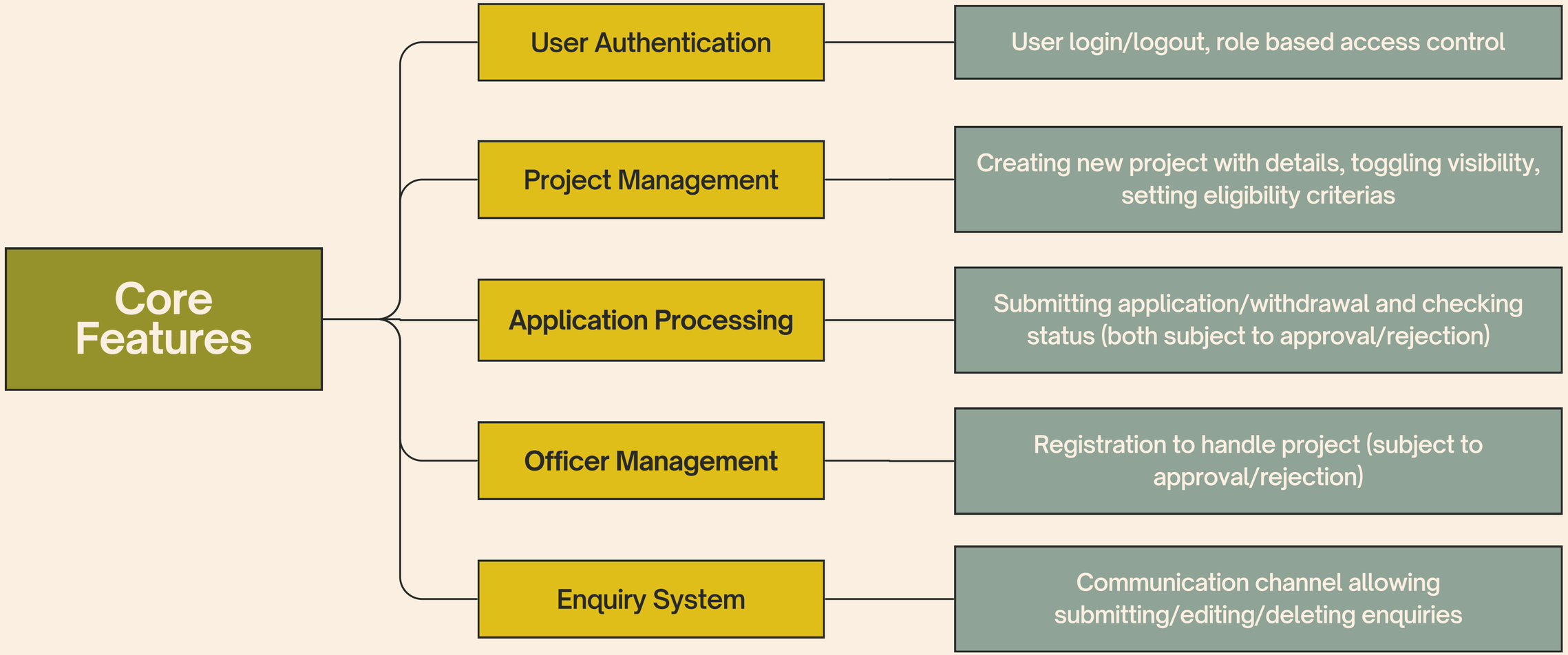
Understanding the Requirements

- BTO system would serve multiple stakeholders with different needs
- identified explicit requirements to implement
- identified user roles and system entities

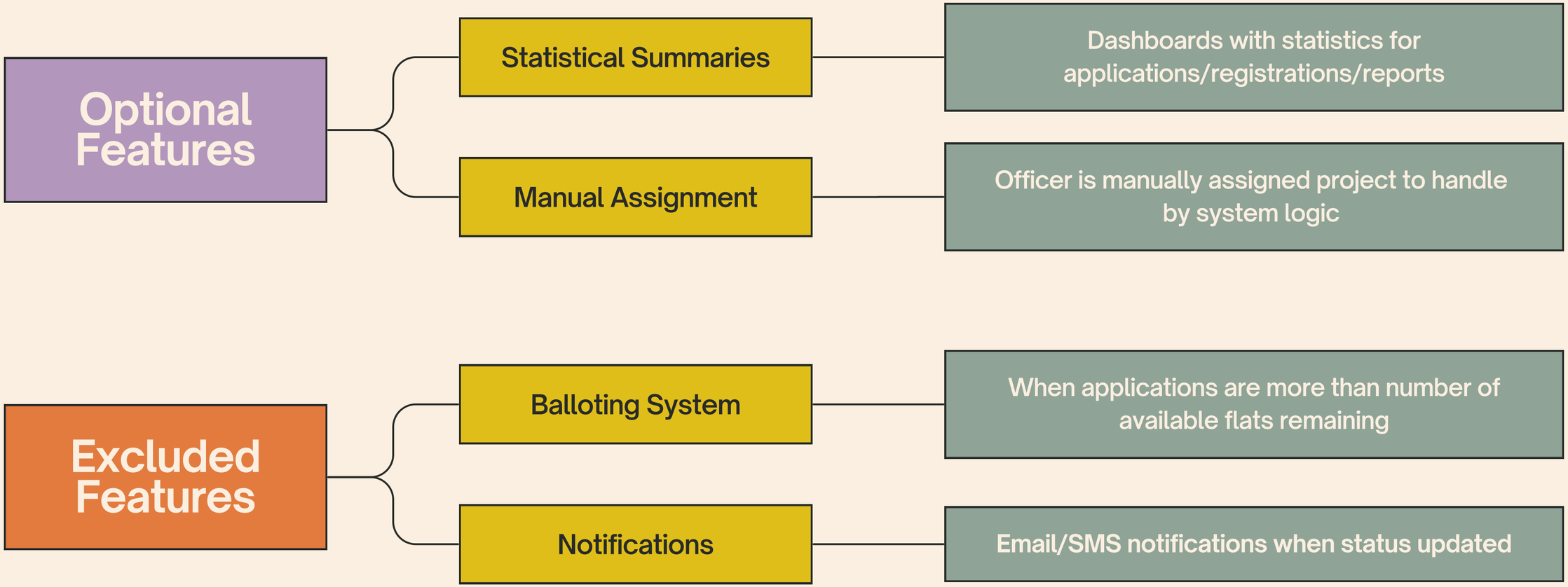
Key Users

- APPLICANT, OFFICER, MANAGER in order of increasing access to system capabilities

Features & Scope



Features & Scope



SYSTEM STRUCTURE

- Planning
- Trade - Offs

Main Actors

Applicant

HDB Officer

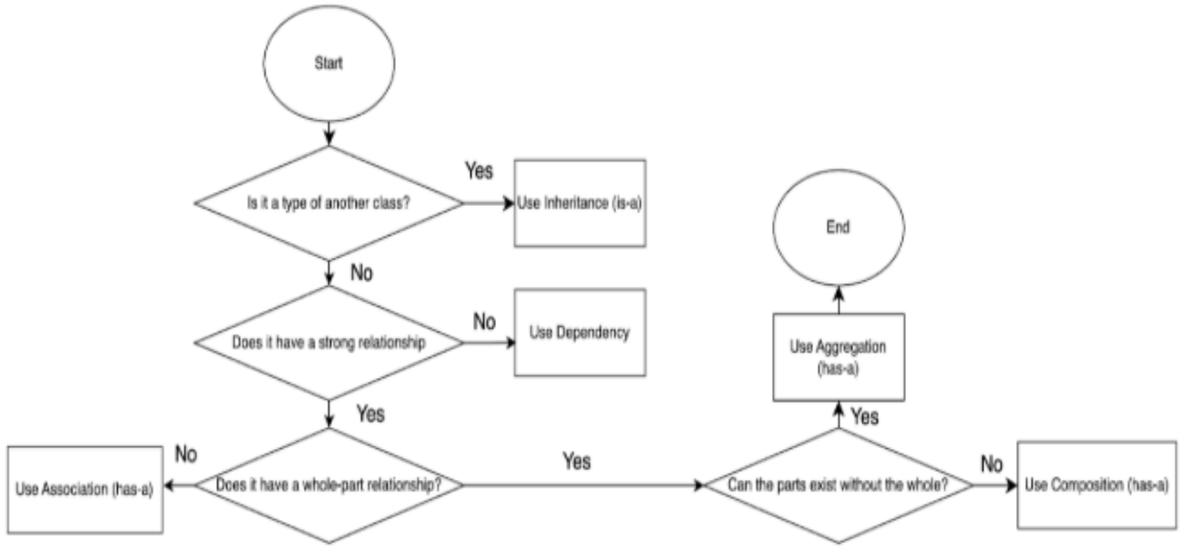
HDB Manager

MVCS Structure

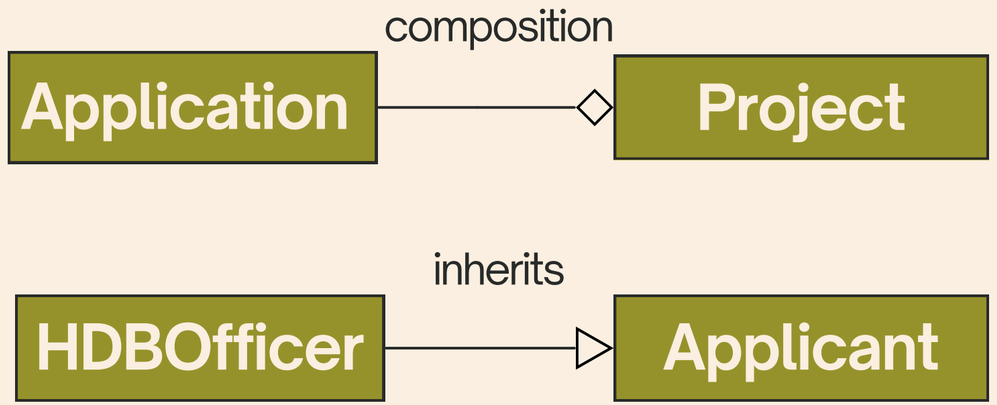
Model	**Defines the domain entities and their relationships.** • Core domain objects (Applicant, Project, Enquiry, Application)
View	**Contains all user interfaces** • Dashboard views for each user type • Specialized views for operations like registration, application and enquiry handling • Login view for authentication
Controller	**Acts as an intermediary between views and services** • User-specific controllers • Cross-cutting controllers • Operation-focused controllers
Service	**Implements core business logic and enforces business rules.** • Services for each domain area • Contains complex operations and validation logic
Repository	**Manages data persistence and retrieval operations.** • Entity-specific repositories • Encapsulates data access concerns

Visual Guides

To facilitate early design decisions, we created flowcharts

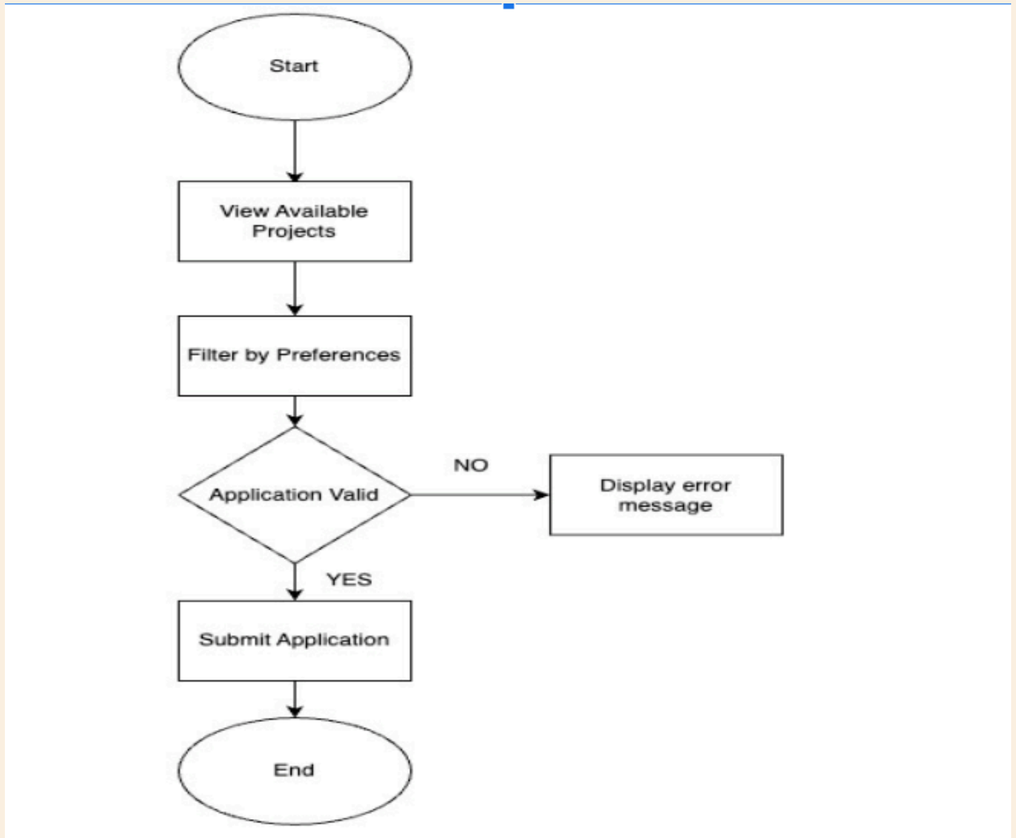


It helped us decide whether to use inheritance (is-a), dependency, or association types such as aggregation and composition based on the strength and nature of the relationships.



Visual Guides

To facilitate early design decisions, we created flowcharts



By using flowchart to map out use-cases, we come up with several main program flows

BTO Project Application Flow:
ApplicantProjectView → ApplicantProjectController → ApplicantProjectService → ProjectRepository/ApplicationRepository → Project/Application

HDB Officer Registration for Project Flow:
OfficerRegistrationView → OfficerRegistrationController → OfficerRegistrationService → OfficerRepository/ProjectRepository → HDBOfficer/Project
HDBManagerRegistrationView → HDBManagerRegistrationController → HDBManagerRegistrationService → OfficerRepository → HDBOfficer

Application Approval Flow
HDBManagerApplicationView → HDBManagerApplicationController → HDBManagerApplicationService → ApplicationRepository → Application

Trade - Offs

Centralised vs Distributed logic

Trade-off	Decision
Placing business logic in models make them self-contained but can lead to bloated classes	Decided to move complex logic operations to service classes while retaining basic validation in the models

Trade - Offs

General vs Role specific Controllers

Trade-off	Decision
Using General controller as it can reduce code duplication	Decided to use Tailored controllers for each user role to maintain clarity and isolate logic

Trade - Offs

Repository Patterns

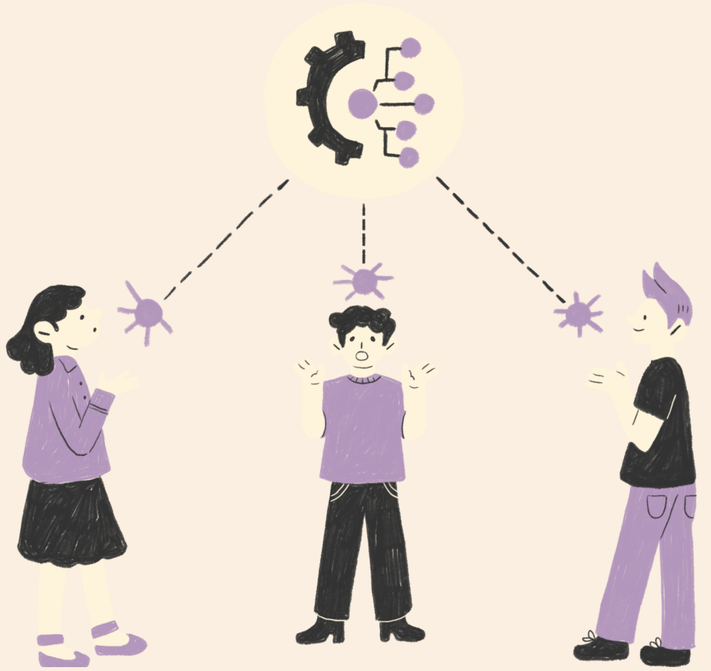
Trade-off	Decision
Adding the repository will add another layer of indirection	Decided to include it to allow database integration and made testing easier

SYSTEM DESIGN

- UML diagrams
- OOD principles

Single Responsibility Principle

⇨ Ensure that each class had one distinct role



Examples

- Separated our classes into distinct layers
- i.e Models, View, Controller, Service, Repository

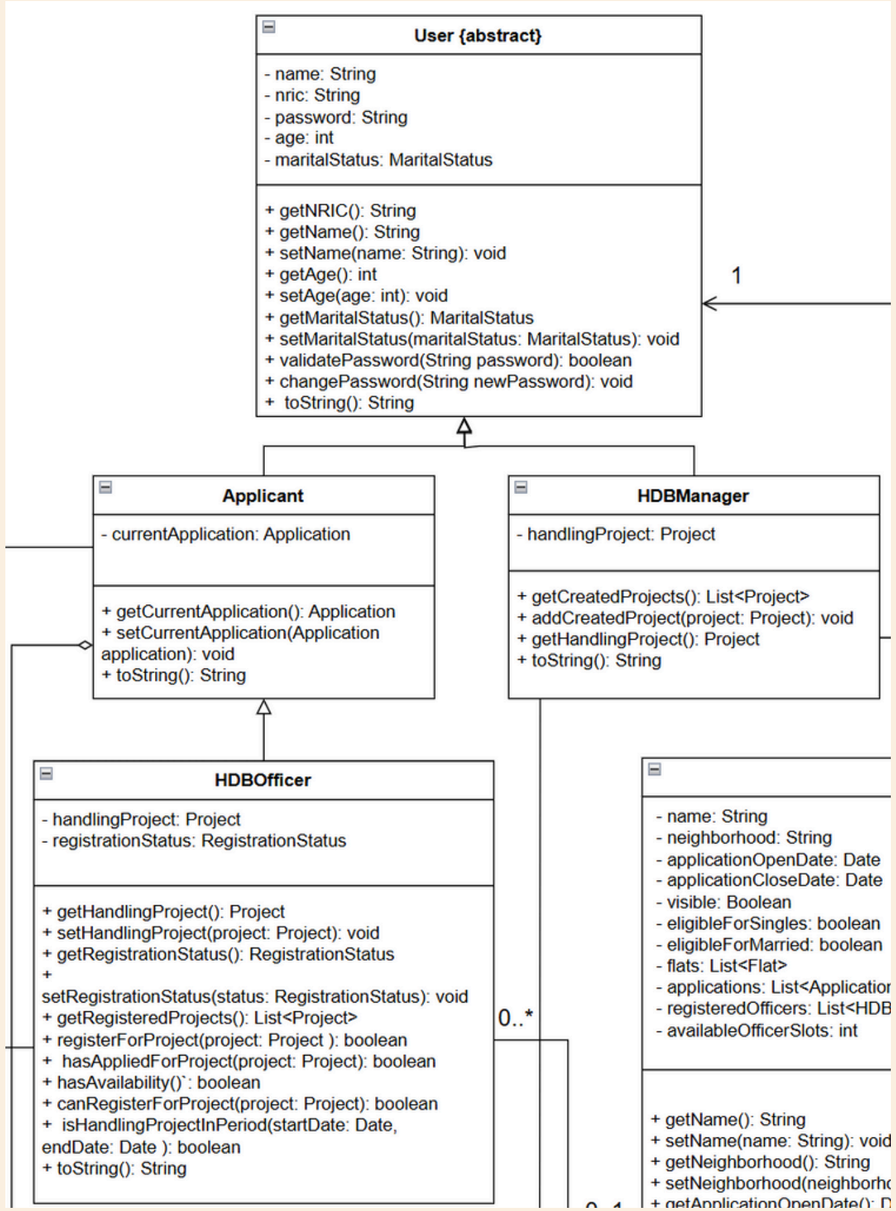
- Separated enquiry-related logic into 3 distinct class
- i.e. ApplicantEnquiryController, HDBOfficerEnquiryController, HDBManagerEnquiryController

Open - Closed Principle

Allow classes to be open for extension but closed for modification

Examples

- Applicant, HDBOfficer and HDBManager subclasses extends from User superclass
- Allows for addition of new user types
- Existing User class logic remains unchanged

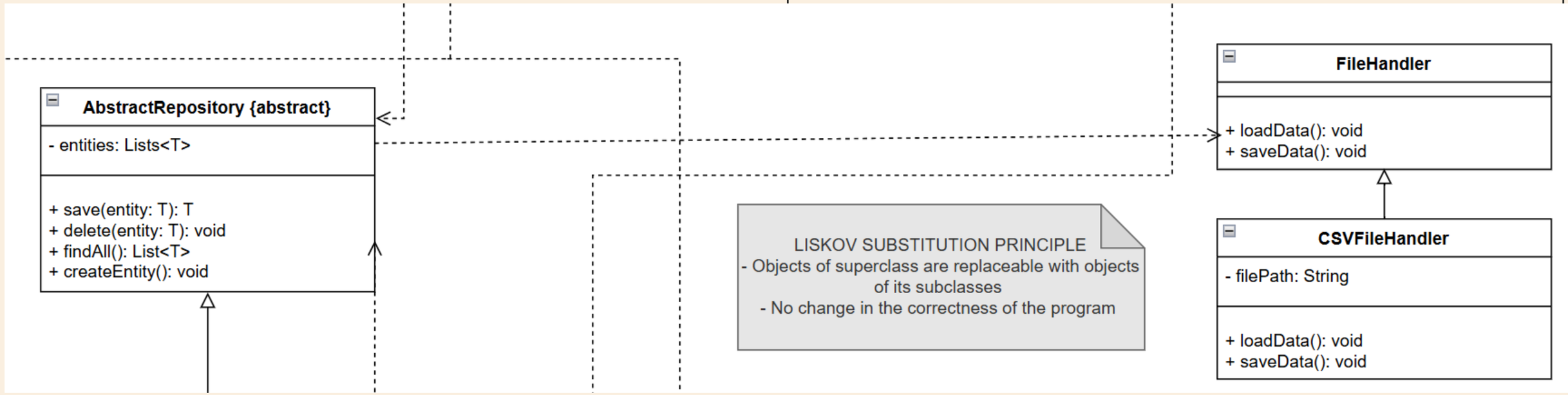


Liskov Substitution Principle

Ensure that objects of superclass are substitutable for subclass without change in correctness

Examples

- CSVFileHandler subclass is substitutable for FIleHandler superclass

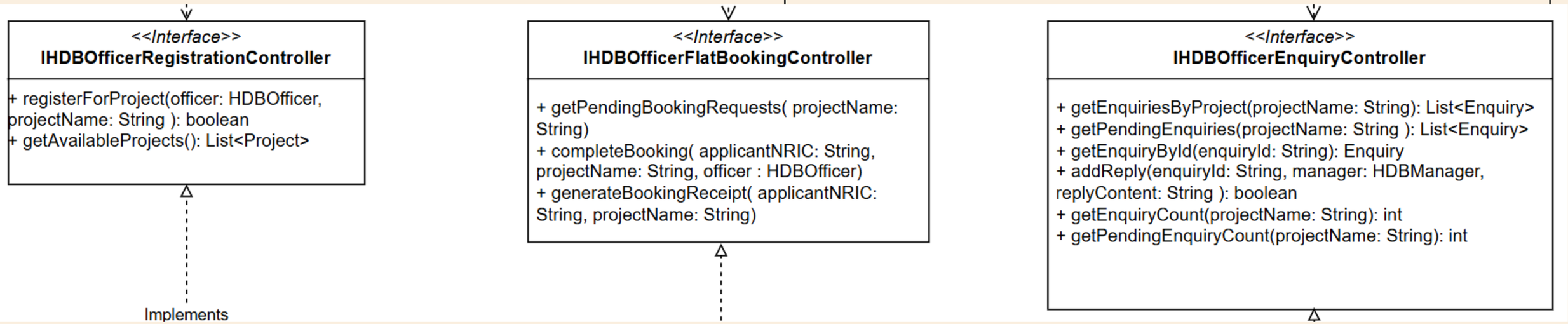


Interface Segregation Principle

Spit into separate interfaces that are role-specific

Examples

- Segregate general HDB Officer interface into IHDBOfficerController, IHDBOfficerFlatController and IHDBOfficerEnquiryController

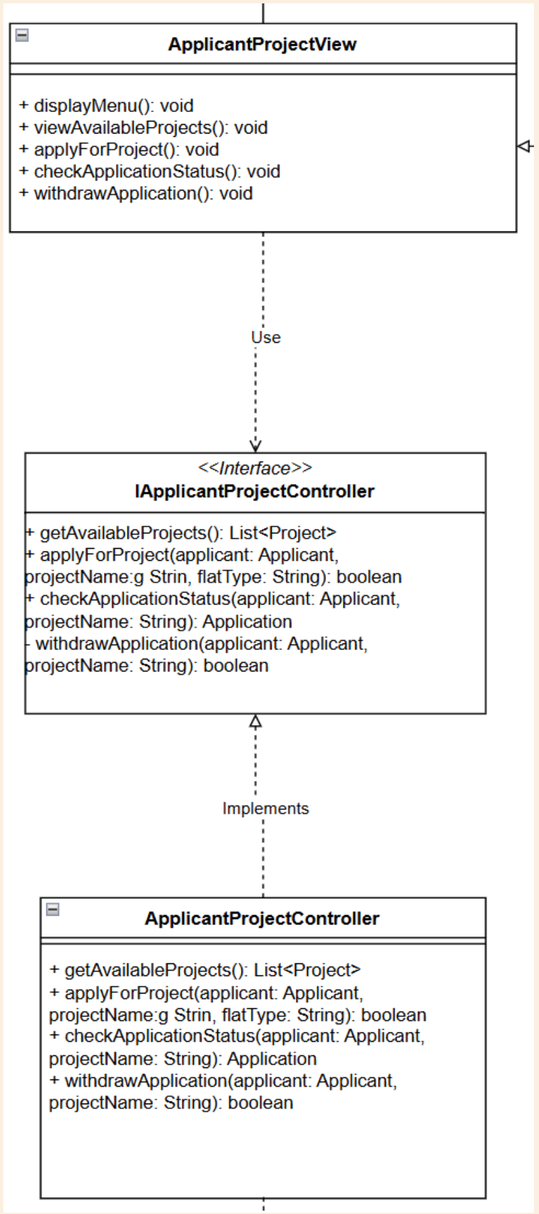


Dependency Injection Principle

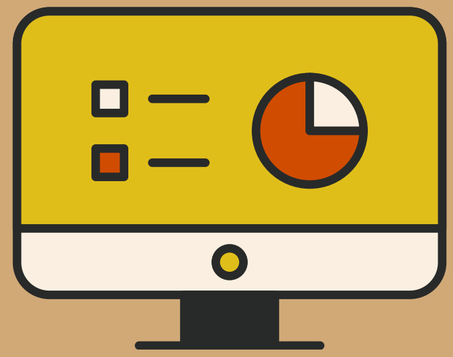
Ensure that high level modules depend on abstractions (interface) rather than low level modules

Examples

- ApplicantProjectView depend on IApplicantProjectController rather than directly depending on ApplicantProjectController



LIVE DEMO



PROJECT REFLECTION

- Success
- Improvements

What went well....

we were able to build a modular and maintainable system that mirrors actual processes in SIngapore's BTO flat booking

- Implemented all core features
- Adopted useful design patterns
- Validated input/flows well
- Robust error handing

Difficulties & improvements

we were able to build a modular and maintainable system that mirrors actual processes in SIngapore's BTO flat booking

Keeping method signatures consistent

Integration of code

Stronger database-backed system

Additional integration/stress testing

Further breaking down of classes

Thank You!