BTO MANAGEMENT SYSTEM

PROBLEM

We need to design a system that allows users

1. **Applicants to book BTO Projects**
2. **Managers create Projects**
3. **Officers to approve Booking Request**
4. **Officers to apply to work for a Project**
5. **Managers to approve BTO applications**
6. **Applicants ask Enquiries and are answered by Managers or Officers**

TEAM

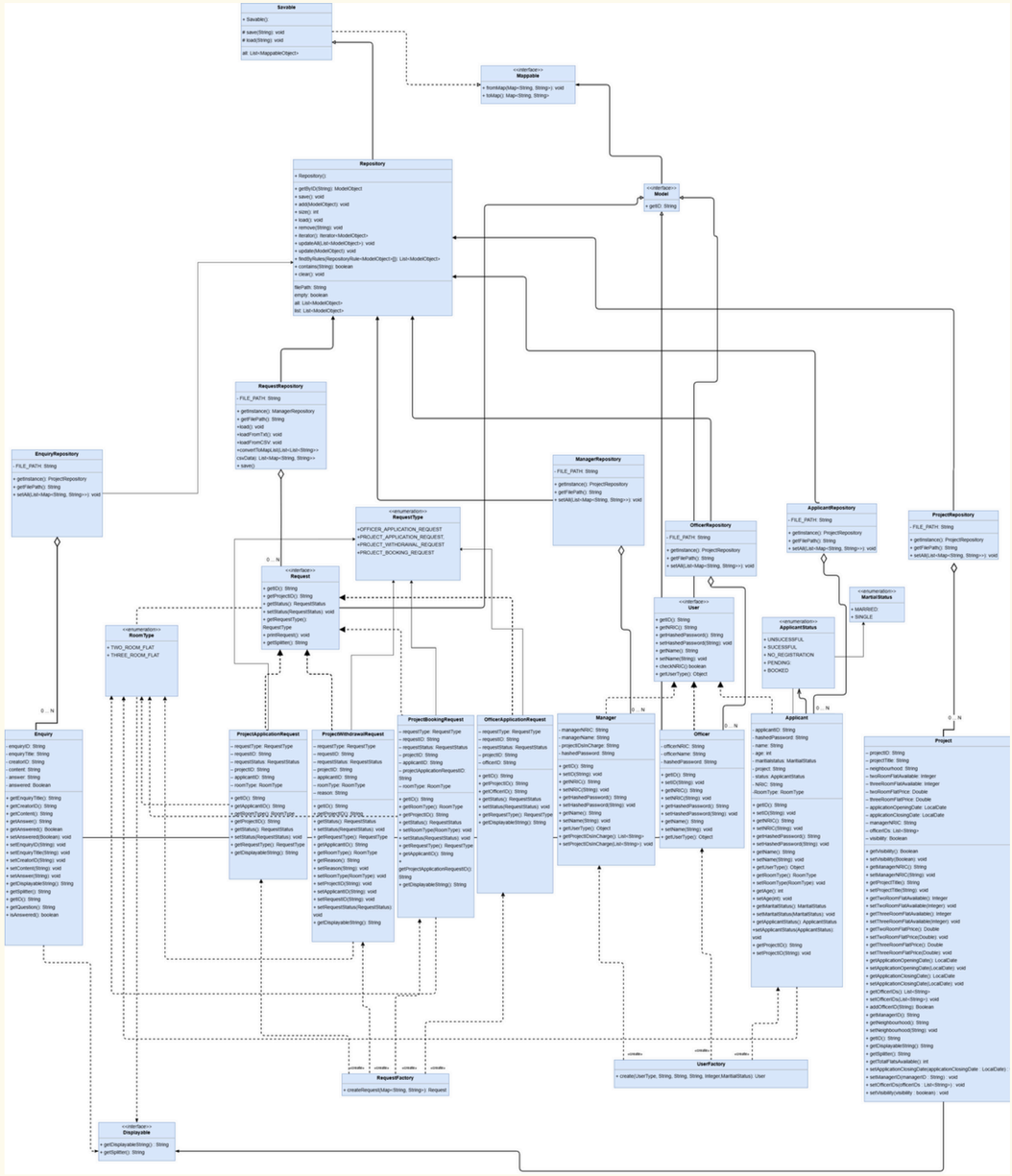
CONTRIBUTIONS

MODEL

Consists of the

1. **Applicants**
2. **Manager**
3. **Officer**
4. **Request**
5. **Enquiry**
6. **Project**

MODEL


Savable
+ Savable()
# save(String): void
# load(String): void
all: List<MappableObject>
<<interface>>
Mappable
+ fromMap(Map<String, String>): void
+ toMap(): Map<String, String>
Repository
+ Repository()
+ getByID(String): ModelObject
+ save(): void
+ add(ModelObject): void
+ size(): int
+ load(): void
+ remove(String): void
+ iterator(): Iterator<ModelObject>
+ updateAll(List<ModelObject>): void
+ update(ModelObject): void
+ findBy(Rules(RepositoryRule<ModelObject>)): List<ModelObject>
+ contains(String): boolean
+ clear(): void
filePath: String
empty: boolean
all: List<ModelObject>
list: List<ModelObject>
<<interface>>
Model
+ getID(): String
RequestRepository
- FILE_PATH: String
+ getInstance(): ManagerRepository
+ getFilePath(): String
+load(): void
+loadFromCSV: void
+convertToMap(List): list<List<String>>
csvData: List<Map<String, String>>
+ save()
EnquiryRepository
- FILE_PATH: String
+ getInstance(): ProjectRepository
+ getFilePath(): String
+ setAll(List<Map<String, String>>): void
ManagerRepository
- FILE_PATH: String
+ getInstance(): ProjectRepository
+ getFilePath(): String
+ setAll(List<Map<String, String>>): void
OfficerRepository
- FILE_PATH: String
+ getInstance(): ProjectRepository
+ getFilePath(): String
+ setAll(List<Map<String, String>>): void
ApplicantRepository
- FILE_PATH: String
+ getInstance(): ProjectRepository
+ getFilePath(): String
+ setAll(List<Map<String, String>>): void
ProjectRepository
- FILE_PATH: String
+ getInstance(): ProjectRepository
+ getFilePath(): String
+ setAll(List<Map<String, String>>): void
<<enumeration>>
RequestType
+OFFICER_APPLICATION_REQUEST
+PROJECT_APPLICATION_REQUEST
+PROJECT_WITHDRAWAL_REQUEST
+PROJECT_BOOKING_REQUEST
<<interface>>
Request
+ getID(): String
+ getProjectID(): String
+ getStatus(): RequestStatus
+ setStatus(RequestStatus): void
+ getRequestType(): RequestType
+ getInRequest(): void
+ getSplitter(): String
<<enumeration>>
RoomType
+ TWO_ROOM_FLAT
+ THREE_ROOM_FLAT
<<interface>>
User
+ getID(): String
+ getNRIC(): String
+ getHashedPassword(): String
+ setHashedPassword(String): void
+ getName(): String
+ setName(String): void
+ checkNRIC(): boolean
+ getUserType(): Object
<<enumeration>>
MaritalStatus
+ MARRIED
+ SINGLE
<<enumeration>>
ApplicantStatus
+ UNSUCESSFUL
+ SUCESSFUL
+ NO_REGISTRATION
+ PENDING
+ BOOKED
Enquiry
- enquiryID: String
- enquiryTitle: String
- creatorID: String
- content: String
- answer: String
- answered: Boolean
+ getEnquiryTitle(): String
+ getCreatorID(): String
+ getContent(): String
+ getAnswer(): String
+ getAnswered(): Boolean
+ setAnswered(Boolean): void
+ setEnquiryID(String): void
+ setEnquiryTitle(String): void
+ setCreatorID(String): void
+ setContent(String): void
+ setAnswer(String): void
+ getDisplayable(String): String
+ getSplitter(): String
+ getID(): String
+ getQuestion(): String
+ isAnswered(): boolean
ProjectApplicationRequest
- requestType: RequestType
- requestID: String
- requestStatus: RequestStatus
- projectID: String
- applicantID: String
- roomType: RoomType
+ getID(): String
+ getApplicantID(): String
+ getProjectID(): String
+ getStatus(): RequestStatus
+ setStatus(RequestStatus): void
+ getRequestType(): RequestType
+ getDisplayable(String): String
ProjectWithdrawalRequest
- requestType: RequestType
- requestID: String
- requestStatus: RequestStatus
- projectID: String
- applicantID: String
- roomType: RoomType
- reason: String
+ getID(): String
+ getStatus(): RequestStatus
+ setStatus(RequestStatus): void
+ getRequestType(): RequestType
+ getApplicantID(): String
+ getRoomType(): RoomType
+ getReason(): String
+ setReason(String): void
+ setRoomType(RoomType): void
+ setProjectID(String): void
+ setApplicantID(String): void
+ setRequestID(String): void
+ setRequestStatus(RequestStatus): void
+ getDisplayable(String): String
ProjectBookingRequest
- requestType: RequestType
- requestID: String
- requestStatus: RequestStatus
- projectID: String
- applicantID: String
- roomType: RoomType
+ getID(): String
+ getRoomType(): RoomType
+ getProjectID(): String
+ getStatus(): RequestStatus
+ setRoomType(RoomType): void
+ setStatus(RequestStatus): void
+ getRequestType(): RequestType
+ getApplicantID(): String
+ getDisplayable(String): String
OfficerApplicationRequest
- requestType: RequestType
- requestID: String
- requestStatus: RequestStatus
- projectID: String
- officerID: String
+ getID(): String
+ getProjectID(): String
+ getOfficerID(): String
+ getStatus(): RequestStatus
+ setStatus(RequestStatus): void
+ getRequestType(): RequestType
+ getDisplayable(String): String
Manager
- managerNRIC: String
- managerName: String
- projectDateCharge: String
- hashedPassword: String
+ getID(): String
+ setID(String): void
+ getNRIC(): String
+ setNRIC(String): void
+ getHashedPassword(): String
+ setHashedPassword(String): void
+ getName(): String
+ setName(String): void
+ getUserType(): Object
+ getProjectDateCharge(): List<String>
+ setProjectDateCharge(List<String>): void
Officer
- officerNRIC: String
- officerName: String
- hashedPassword: String
+ getID(): String
+ setID(String): void
+ getNRIC(): String
+ setNRIC(String): void
+ getHashedPassword(): String
+ setHashedPassword(String): void
+ getName(): String
+ setName(String): void
+ getUserType(): Object
Applicant
applicantID: String
hashedPassword: String
name: String
age: int
maritalstatus: MaritalStatus
project: String
status: ApplicantStatus
NRIC: String
RoomType: RoomType
+ getID(): String
+ setID(String): void
+ getNRIC(): String
+ setNRIC(String): void
+ getHashedPassword(): String
+ setHashedPassword(String): void
+ getName(): String
+ setName(String): void
+ getUserType(): Object
+ getRoomType(): RoomType
+ setRoomType(RoomType): void
+ getAge(): int
+ setAge(int): void
+ getMaritalStatus(): MaritalStatus
+ setMaritalStatus(MaritalStatus): void
+ getApplicantStatus(): ApplicantStatus
+ setApplicantStatus(ApplicantStatus): void
+ getProjectID(): String
+ setProject(String): void
Project
- projectID: String
- projectTitle: String
- neighbourhood: String
- twoRoomFlatAvailable: Integer
- threeRoomFlatAvailable: Integer
- twoRoomFlatPrice: Double
- threeRoomFlatPrice: Double
- applicationOpeningDate: LocalDate
- applicationClosingDate: LocalDate
- managerNRIC: String
- officerIDs: List<String>
- visibility: Boolean
+ getVisibility(): Boolean
+ setVisibility(Boolean): void
+ getManagerNRIC(): String
+ setManagerNRIC(String): void
+ getProjectTitle(): String
+ setProjectTitle(String): void
+ getTwoRoomFlatAvailable(): Integer
+ setTwoRoomFlatAvailable(Integer): void
+ getThreeRoomFlatAvailable(): Integer
+ setThreeRoomFlatAvailable(Integer): void
+ getTwoRoomFlatPrice(): Double
+ setTwoRoomFlatPrice(Double): void
+ getThreeRoomFlatPrice(): Double
+ setThreeRoomFlatPrice(Double): void
+ getApplicationOpeningDate(): LocalDate
+ setApplicationOpeningDate(LocalDate): void
+ getApplicationClosingDate(): LocalDate
+ setApplicationClosingDate(LocalDate): void
+ getOfficerIDs(): List<String>
+ setOfficerIDs(List<String>): void
+ addOfficerID(String): Boolean
+ getManagerID(): String
+ getNeighbourhood(): String
+ setNeighbourhood(String): void
+ getID(): String
+ getDisplayable(String): String
+ getSplitter(): String
+ getTotalFlatAvailable(): int
+ setApplicationClosingDate(applicationClosingDate : LocalDate): void
+ setManagerID(managerID : String): void
+ setOfficerIDs(officerIDs : List<String>): void
+ setVisibility(visibility : boolean): void
RequestFactory
+ createRequest(Map<String, String>): Request
UserFactory
+ create(UserType, String, String, String, Integer,MaritalStatus): User
<<interface>>
Displayable
+ getDisplayable(String): String
+ getSplitter(): String

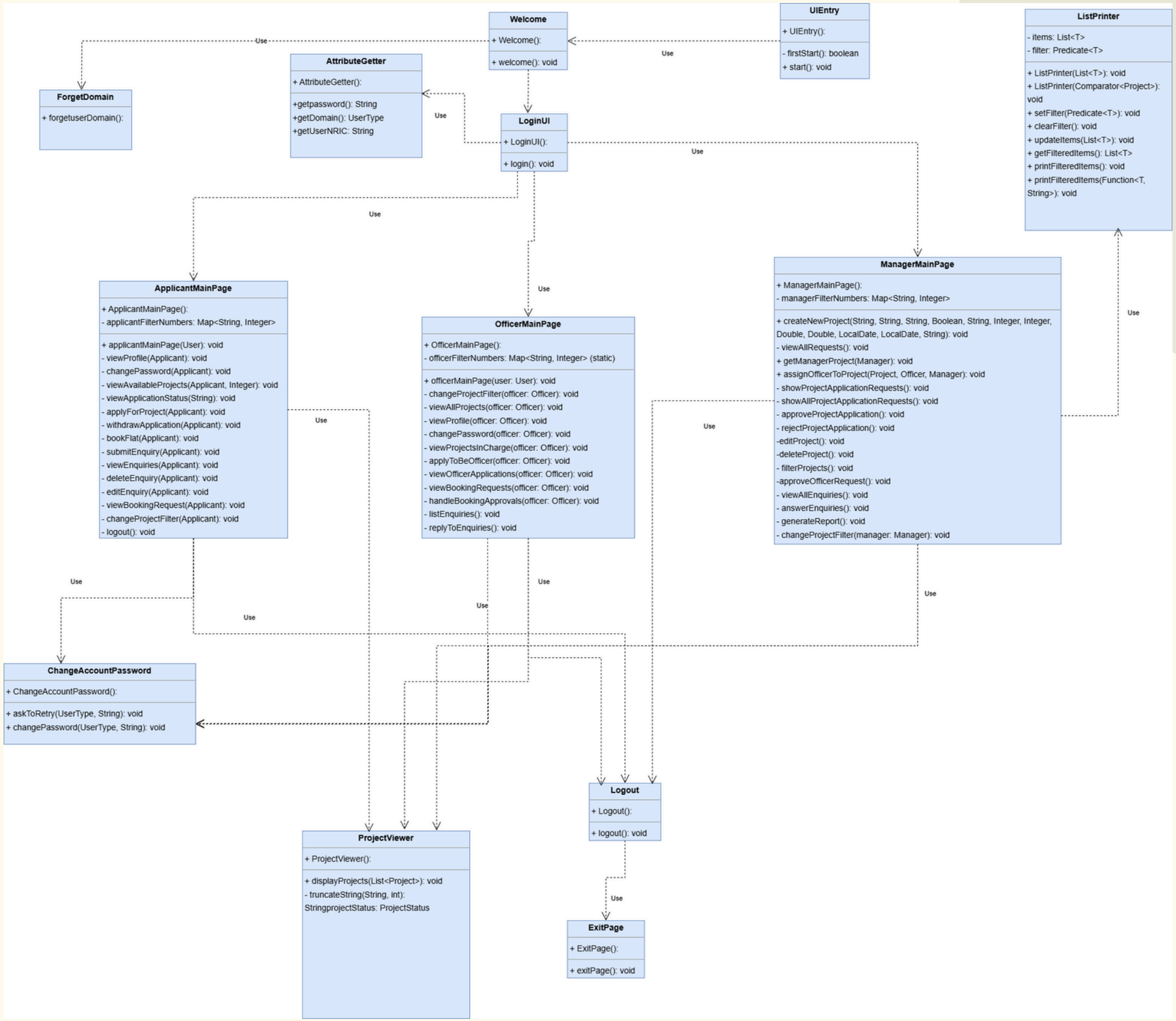
REPOSITORY PATTERN

Stores the various models

VIEW

Handles the printing of output and the user interaction flow

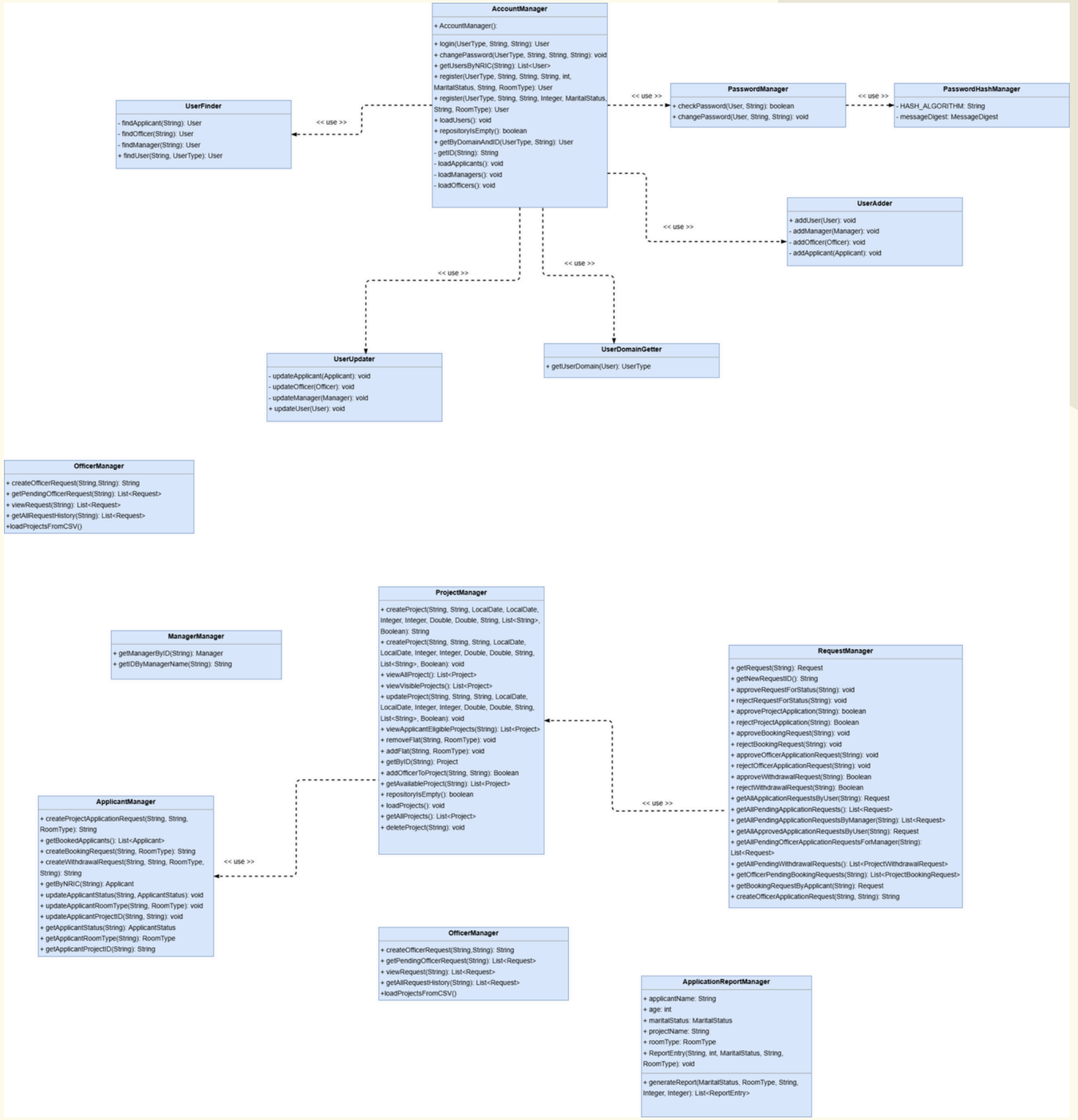
VIEW


Welcome
+ Welcome():
+ welcome(): void
UIEntry
+ UIEntry():
- firstStart(): boolean
+ start(): void
ListPrinter
- items: List<T>
- filter: Predicate<T>
+ ListPrinter(List<T>): void
+ ListPrinter(Comparator<Project>): void
+ setFilter(Predicate<T>): void
+ clearFilter(): void
+ updateItems(List<T>): void
+ getFilteredItems(): List<T>
+ printFilteredItems(): void
+ printFilteredItems(Function<T, String>): void
ForgetDomain
+ forgetuserDomain():
AttributeGetter
+ AttributeGetter():
+getpassword(): String
+getDomain(): UserType
+getUserNRIC: String
LoginUI
+ LoginUI():
+ login(): void
ApplicantMainPage
+ ApplicantMainPage():
- applicantFilterNumbers: Map<String, Integer>
+ applicantMainPage(User): void
- viewProfile(Applicant): void
- changePassword(Applicant): void
- viewAvailableProjects(Applicant, Integer): void
- viewApplicationStatus(String): void
- applyForProject(Applicant): void
- withdrawApplication(Applicant): void
- bookFlat(Applicant): void
- submitEnquiry(Applicant): void
- viewEnquiries(Applicant): void
- deleteEnquiry(Applicant): void
- editEnquiry(Applicant): void
- viewBookingRequest(Applicant): void
- changeProjectFilter(Applicant): void
- logout(): void
OfficerMainPage
+ OfficerMainPage():
- officerFilterNumbers: Map<String, Integer> (static)
+ officerMainPage(user: User): void
- changeProjectFilter(officer: Officer): void
- viewAllProjects(officer: Officer): void
- viewProfile(officer: Officer): void
- changePassword(officer: Officer): void
- viewProjectsInCharge(officer: Officer): void
- applyToBeOfficer(officer: Officer): void
- viewOfficerApplications(officer: Officer): void
- viewBookingRequests(officer: Officer): void
- handleBookingApprovals(officer: Officer): void
- listEnquiries(): void
- replyToEnquiries(): void
ManagerMainPage
+ ManagerMainPage():
- managerFilterNumbers: Map<String, Integer>
+ createNewProject(String, String, String, Boolean, String, Integer, Integer, Double, Double, LocalDate, LocalDate, String): void
- viewAllRequests(): void
+ getManagerProject(Manager): void
+ assignOfficerToProject(Project, Officer, Manager): void
- showProjectApplicationRequests(): void
- showAllProjectApplicationRequests(): void
- approveProjectApplication(): void
- rejectProjectApplication(): void
-editProject(): void
-deleteProject(): void
- filterProjects(): void
-approveOfficerRequest(): void
- viewAllEnquiries(): void
- answerEnquiries(): void
- generateReport(): void
- changeProjectFilter(manager: Manager): void
ChangeAccountPassword
+ ChangeAccountPassword():
+ askToRetry(UserType, String): void
+ changePassword(UserType, String): void
ProjectViewer
+ ProjectViewer():
+ displayProjects(List<Project>): void
- truncateString(String, int):
Stringproject Status: ProjectStatus
Logout
+ Logout():
+ logout(): void
ExitPage
+ ExitPage():
+ exitPage(): void
Use

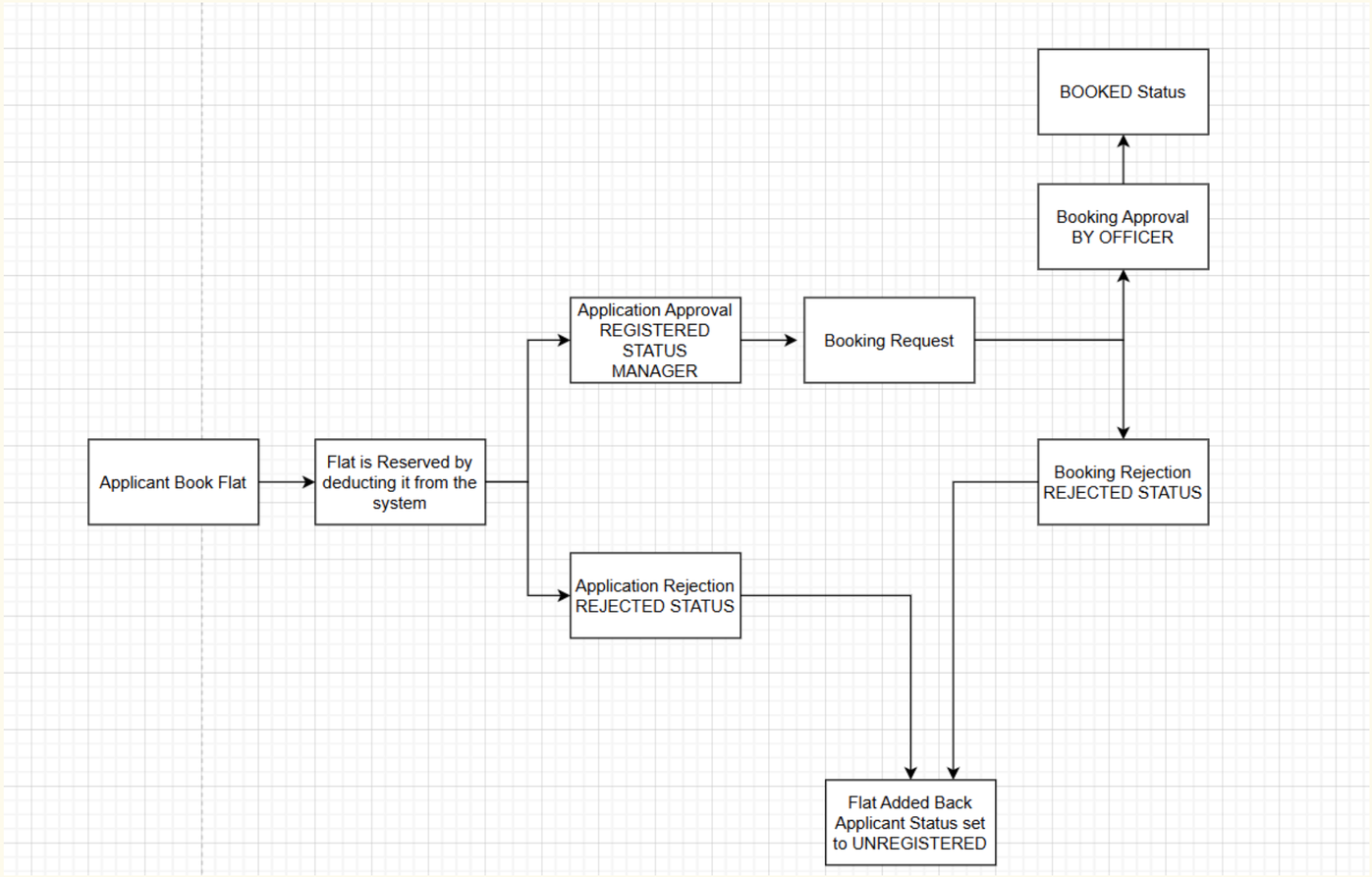
CONTROLLER

Handles the business logic, manipulation of the Models and saves them in their repository

CONTROLLER


AccountManager
+ AccountManager():
+ login(UserType, String, String): User
+ changePassword(UserType, String, String, String): void
+ getUsersByNRIC(String): List<User>
+ register(UserType, String, String, String, int, MaritalStatus, String, RoomType): User
+ register(UserType, String, String, Integer, MaritalStatus, String, RoomType): User
+ loadUsers(): void
+ repositoryIsEmpty(): boolean
+ getByDomainAndID(UserType, String): User
- getID(String): String
- loadApplicants(): void
- loadManagers(): void
- loadOfficers(): void
<< use >>
PasswordManager
+ checkPassword(User, String): boolean
+ changePassword(User, String, String): void
<< use >>
PasswordHashManager
- HASH_ALGORITHM: String
- messageDigest: MessageDigest
UserFinder
- findApplicant(String): User
- findOfficer(String): User
- findManager(String): User
+ findUser(String, UserType): User
<< use >>
UserAdder
+ addUser(User): void
- addManager(Manager): void
- addOfficer(Officer): void
- addApplicant(Applicant): void
<< use >>
<< use >>
<< use >>
UserUpdater
- updateApplicant(Applicant): void
- updateOfficer(Officer): void
- updateManager(Manager): void
+ updateUser(User): void
UserDomainGetter
+ getUserDomain(User): UserType
OfficerManager
+ createOfficerRequest(String,String): String
+ getPendingOfficerRequest(String): List<Request>
+ viewRequest(String): List<Request>
+ getAllRequestHistory(String): List<Request>
+loadProjectsFromCSV()
ProjectManager
+ createProject(String, String, LocalDate, LocalDate, Integer, Integer, Double, Double, String, List<String>, Boolean): String
+ createProject(String, String, String, LocalDate, LocalDate, Integer, Integer, Double, Double, String, List<String>, Boolean): void
+ viewAllProject(): List<Project>
+ viewVisibleProjects(): List<Project>
+ updateProject(String, String, String, LocalDate, LocalDate, Integer, Integer, Double, Double, String, List<String>, Boolean): void
+ viewApplicantEligibleProjects(String): List<Project>
+ removeFlat(String, RoomType): void
+ addFlat(String, RoomType): void
+ getByID(String): Project
+ addOfficerToProject(String, String): Boolean
+ getAvailableProject(String): List<Project>
+ repositoryIsEmpty(): boolean
+ loadProjects(): void
+ getAllProjects(): List<Project>
+ deleteProject(String): void
ManagerManager
+ getManagerByID(String): Manager
+ getIDByManagerName(String): String
RequestManager
+ getRequest(String): Request
+ getNewRequestID(): String
+ approveRequestForStatus(String): void
+ rejectRequestForStatus(String): void
+ approveProjectApplication(String): boolean
+ rejectProjectApplication(String): Boolean
+ approveBookingRequest(String): void
+ rejectBookingRequest(String): void
+ approveOfficerApplicationRequest(String): void
+ rejectOfficerApplicationRequest(String): void
+ approveWithdrawalRequest(String): Boolean
+ rejectWithdrawalRequest(String): Boolean
+ getAllApplicationRequestsByUser(String): Request
+ getAllPendingApplicationRequests(): List<Request>
+ getAllPendingApplicationRequestsByManager(String): List<Request>
+ getAllApprovedApplicationRequestsByUser(String): Request
+ getAllPendingOfficerApplicationRequestsForManager(String): List<Request>
+ getAllPendingWithdrawalRequests(): List<ProjectWithdrawalRequest>
+ getOfficerPendingBookingRequests(String): List<ProjectBookingRequest>
+ getBookingRequestByApplicant(String): Request
+ createOfficerApplicationRequest(String, String): String
<< use >>
ApplicantManager
+ createProjectApplicationRequest(String, String, RoomType): String
+ getBookedApplicants(): List<Applicant>
+ createBookingRequest(String, RoomType): String
+ createWithdrawalRequest(String, String, RoomType, String): String
+ getByNRIC(String): Applicant
+ updateApplicantStatus(String, ApplicantStatus): void
+ updateApplicantRoomType(String, RoomType): void
+ updateApplicantProjectID(String, String): void
+ getApplicantStatus(String): ApplicantStatus
+ getApplicantRoomType(String): RoomType
+ getApplicantProjectID(String): String
<< use >>
OfficerManager
+ createOfficerRequest(String,String): String
+ getPendingOfficerRequest(String): List<Request>
+ viewRequest(String): List<Request>
+ getAllRequestHistory(String): List<Request>
+loadProjectsFromCSV()
ApplicationReportManager
+ applicantName: String
+ age: int
+ maritalStatus: MaritalStatus
+ projectName: String
+ roomType: RoomType
+ ReportEntry(String, int, MaritalStatus, String, RoomType): void
+ generateReport(MaritalStatus, RoomType, String, Integer, Integer): List<ReportEntry>

BOOKING FLOW



BOOKING FLOW

Ensure that Applicants and Officers cannot enter illegal flow by not allowing them to select the option

SOLID - SRP

The Single Responsibility principle recommends that each class should have a clear and singular responsibility, avoiding unrelated tasks.

SOLID - SRP

We ensure SRP by splitting the classes into MVC classes, where the model implements the data structures, the repository classes handle the saving and persistence, the controller classes execute logic to handle what has to be viewed along with updating User,Projects,Enquiry and Request and the boundary class handles only the printing of Objects and interactions.

Example: Applicant Class only defines what a applicant is

SOLID - OCP

The Open/Closed Principle (OCP) states that classes should be open for extension but closed for modification, allowing for the addition of new functionality without changing existing code. OCP can be implemented through abstraction, inheritance, and polymorphism.

SOLID - OCP

In our project, OCP is applied by providing an abstract class, Repository<Model>, which serves as a base for creating specialized repositories like ProjectRepository and RequestRepository. Each subclass provides its own implementation of the getFilePath() method, making it easy to extend the repository system without modifying existing logic.

SOLID - LSP

LSP states that the subtypes can be a replacement for the base types.

SOLID - LSP

In our system, this is used in the Request class.

All subclasses of Request include the different types of requests(eg ProjectApplicationRequest,OfficerApplicationRequest etc). All of these subclasses are interchangeable with the Request interface, ensuring they can be used wherever a Request object is expected while maintaining correct behavior. When determining the type of a request, different request instances can invoke the getRequestType() method.

SOLID - ISP

The interface segregation principle states that we should use many smaller specific interfaces rather than 1 general interface.

SOLID - ISP

In our project, we split the ‘Model’ into User and Request interfaces so that the different entities can implement the interfaces respectively. For example, Request has a requestType that is different from that of the UserType.

SOLID - DIP

DIP states that both high and low level modules should depend on abstractions.

SOLID - DIP

We implement this by providing a User Interface. Instead of depending on the concrete classes of Applicant,Manager or Officer, we can call User.getID() from the User Interface to get the userID/NRIC.

UserFactory used to create new users during testing

TESTING

Manual + Unit Testing

After implementation, write a small unit test to verify output

TESTING

```
public class ManagerSaveTest {  Goh Zhen Rong
    public static void setUp() {  Goh Zhen Rong
        managers = new Manager[3];
        // Using the full constructor for clarity
        managers[0] = new Manager(
                managerNRIC: "M1234567A", // managerNRIC
                "hashed_password_1", // hashedPassword (placeholder)
                "Alice Wonderland", // managerName
                new ArrayList<>(Arrays.asList("P101", "P102")) // projectIDsInCharge
        );
        managers[1] = new Manager(
                managerNRIC: "M7654321B", // managerNRIC
                "hashed_password_2", // hashedPassword (placeholder)
                "Bob The Builder", // managerName
                new ArrayList<>() // Empty project list
        );
        managers[2] = new Manager(
                managerNRIC: "M9876543C", // managerNRIC
                "hashed_password_3", // hashedPassword (placeholder)
                "Charlie Chaplin", // managerName
                new ArrayList<>(Arrays.asList("P201")) // projectIDsInCharge
        );
```

EDGE CASES + NEW TEST CASES

Officer Applies to project they are already officer of

Officer applies to Project they are incharge of

Enquiries only deleted by creator and if not answered

Data Persistance issues

DEMO

THANK YOU