|  |  |
| --- | --- |
|  | **MINISTRY OF EDUCATION AND TRAINING** |

|  |
| --- |
| **FPT UNIVERSITY** |
| Capstone Project Document |
| FU House Finder |

|  |  |
| --- | --- |
| **SEP490\_G1** | |
| **Group Members** | Nguyễn Thế Giang HE153046  Bùi Ngọc Huyền HE150346  Nguyễn Thu An HE150432  Phùng Quang Thông HE150340  Nguyễn Trí Kiên HE150160 |
| **Supervisor** | Nguyen Tat Trung |
| **Ext Supervisor** |  |
| **Capstone Project code** | FHF |

- Hanoi, Sep/2022 -

**Table of Contents**

[Acknowledgement 4](#_Toc119391128)

[Definition and Acronyms 4](#_Toc119391129)

[I. Project Introduction 5](#_Toc119391130)

[1. Overview 5](#_Toc119391131)

[2. Product Background 5](#_Toc119391132)

[3. Existing Systems 6](#_Toc119391133)

[4. Business Opportunity 7](#_Toc119391134)

[5. Software Product Vision 7](#_Toc119391135)

[6. Project Scope & Limitations 7](#_Toc119391136)

[II. Project Management Plan 9](#_Toc119391137)

[1. Overview 9](#_Toc119391138)

[2. Management Approach 12](#_Toc119391139)

[3. Project Deliverables 13](#_Toc119391140)

[4. Responsibility Assignments 14](#_Toc119391141)

[5. Project Communications 14](#_Toc119391142)

[6. Configuration Management 15](#_Toc119391143)

[III. Software Requirement Specification 16](#_Toc119391144)

[1. Product Overview 16](#_Toc119391145)

[2. User Requirements 16](#_Toc119391146)

[3. Functional Requirements 20](#_Toc119391147)

[4. Non-Functional Requirements 41](#_Toc119391148)

[5. Requirement Appendix 42](#_Toc119391149)

[IV. Software Design Description 44](#_Toc119391150)

[1. System Design 44](#_Toc119391151)

[2. Database Design 46](#_Toc119391152)

[3. Detailed Design 49](#_Toc119391153)

[4. Database Tables 85](#_Toc119391154)

[V. Software Testing Documentation 91](#_Toc119391155)

[1. Scope of Testing 91](#_Toc119391156)

[2. Test Strategy 91](#_Toc119391157)

[3. Test Plan 91](#_Toc119391158)

[4. Test Cases 91](#_Toc119391159)

[5. Test Reports 92](#_Toc119391160)

[VI. Release Package & User Guides 92](#_Toc119391161)

[1. Deliverable Package 92](#_Toc119391162)

[2. Installation Guides 92](#_Toc119391163)

[3. User Manual 92](#_Toc119391164)

# Acknowledgement

# Definition and Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| PWM | Psychology website |
| AWS | Amazon Web Services |
| BA | Business Analysis |
| BR | Business Rule |
| ERD | Entity Relationship Diagram |
| GUI | Graphical User Interface |
| PM | Project Manager |
| SDD | Software Design Description |
| SPMP | Software Project Management Plan |
| SRS | Software Requirement Specification |
| UAT | User Acceptance Test |
| UC | Use Case |
| API | Application Program Interface |

# I. Project Introduction

## 1. Overview

### 1.1 Project Information

* Project name: FU House Finder
* Project code: FHF
* Group name: SWP490-G1
* Software type: Web App

### 1.2 Project Team

|  |  |  |  |
| --- | --- | --- | --- |
| **Full Name** | **Role** | **Email** | **Mobile** |
| Nguyen Tat Trung | Lecturer | Trungnt77@fe.edu.vn | 0904399139 |
| Nguyen The Giang | Leader | GiangNTHE153046@fpt.edu.vn | 0944961228 |
| Phung Quang Thong | Member | ThongPQHE150340 @fpt.edu.vn | 0987206969 |
| Bui Ngoc Huyen | Member | HuyenBNHE150346@fpt.edu.vn | 0346034217 |
| Nguyen Thu An | Member | AnNTHE150432@fpt.edu.vn | 0815709131 |
| Nguyen Tri Kien | Member | KienNTHE150160 @fpt.edu.vn | 0987999975 |

## 2. Product Background

Currently, the following major problems remain:

* The input data to the inn's website is currently not computerized to the smallest management unit, each room, to be included in the system. Therefore, the amount of high-quality data in existing web pages is not much.
* The current website searches for low-quality results for users because the status of the inn has not been updated to reflect reality. Due to the fact that the status of the room changes quite often and the host only puts general information on the Facebook group created by the admissions office, so it is difficult to find in large quantities to provide information for students to enroll with a large number of rooms to rent to students.

The student affairs department has a list of the names of the innkeepers who have been confirmed by the admissions office that the innkeeper exists in fact, the correct address and there may be a number of rooms in the area around FPT Hoa Lac University including 5 communes: Binh Yen, Thach Hoa, Tan Commune, Shareholders, Ha Bang save as excel file. The student affairs department also created 1 Facebook group for hostel owners to post and read about the rooms. With Facebook groups and a list of hostel owners, it still partially meets the rental information needs of a large number of old and new current students.

The problem situation is:

* The data of the motel rooms is not specific, the inn owner only posts general to the FB group as "Room for rent, xxx square meters, price, address, contact number ....". So, when enrolling new students, thousands of students want to find a hostel in 1 short time like when enrolling at the beginning of the year, the admissions office could not answer for thousands of students, parents' status of the large number of available rooms for rent around the campus area.
* The number of individual rooms for each student, the current way of doing it on the FB group meets but the large number of new students cannot rent a dormitory but having to outsource, the Facebook group method is not effective.

How to refresh:

* Digitize all accurate data to each room of all residents who have rooms for rent around Hoa Lac area. The smallest management unit is each room.

## 3. Existing Systems

### 3.1 FPT Can Tho

* Descriptions of the system: Helping students at FPT University Can Tho find accommodation
* Link: <https://dorm.fptucantho.vn/>
* System actors: guest, user, admin
* Features: Search for accommodation in Can Tho
* Pros:
  + There is a lot of information about the inn posted
  + Show full information about the accommodation people want to post
* Cons:
  + Pagination is still very weak
  + Difficult to attract users because the interface is not beautiful

### 3.2 Nhatot System

* Descriptions of the system: Helping people find accommodation
* Link: <https://www.nhatot.com/thue-phong-tro>
* System actors: guest, user, admin
* Features: Search for accommodation in Vietnam
* Pros:
  + Various hostels in many provinces
* Cons:
  + There are many ads inserted in eye-catching places that make it difficult for users to find the information they want to search

### 3.3 BatDongSan System

* Descriptions of the system: Helping people find accommodation
* Link: <https://batdongsan.com.vn/cho-thue-nha-tro-phong-tro-bac-tu-liem/gia-tu-1-trieu-den-3-trieu-dt-duoi-30m2>
* System actors: guest, user, admin
* Features: Search for accommodation in Vietnam
* Pros:
  + GUI is good
* Cons:
  + Can't show which rooms are available at the inn

### 3.4 Blog

* Descriptions of the system: Helping readers know how to convince customers
* Link: <https://timescityminhkhai.com/bi-quyet-luon-luon-lap-day-phong-trong.html>
* System actors: guest, user, admin

## 4. Business Opportunity

The system will be 1 tool for inn owners to regularly update the status of the motel room to reflect the reality or vacancy status in the future can also be searched. Therefore, the system helps innkeepers increase the annual rate of rental rooms and can collect money from the innkeeper on a monthly, or quarterly basis. If they find tool valuable, they are willing to pay.

## 5. Software Product Vision

The capstone team is expected to coordinate with the admissions department and the innkeeper to put the actual data of each room such as text description, photo of 3 each room on google sheet. Then the capstone group will run the excel file import function to put data into the data base in large quantities to serve the search. This works that want to finish into the 20/10/2022.

capstone group system intends to develop with the desire to put into practice with the most basic features by the end of 11/2022 in Hoa Lac. If it is ok, it is expected that the system can be deployed at 5 campuses Hoa Lac, Ho Chi Minh City, Can Tho, Da Nang, Quy Nhon.

## 6. Project Scope & Limitations

### 6.1 Major Features

FE-01: Log in to the system using Google or Facebook

FE-02: View list of available houses in an area around a school campus, search for a house by different criteria

FE-03: View the detail information of a house, view detail information of a room in a house

FE-04: A student can give rates and comments to review a house, or report it if it is violated

FE-05: A landlord can upload information of a house, including many rooms

FE-06: A landlord can view, create, update, delete his houses and each room in a house

FE-07: A landlord can view and update his profile

FE-08: A staff can view statistics and accept or decline landlord’s sign up request

FE-09: A staff can view and update his profile

FE-10: A staff can view reports to enable or disable accounts of landlords

FE-11: An admin can view, create, update, delete all staff’s accounts

FE-12: An admin change his password



### 6.2 Limitations & Exclusions

LI-1: Students cannot compare 2 houses or 2 rooms by its information

LI-2: Students cannot interact directly with the landlords via messaging in the system, they will have to make contact through phone or Facebook

LI-3: Students will not have a wish list to add their favorite houses into

LI-4: The system will not have the recommendation functionalities to recommend suitable houses to a student based on his history of views

# II. Project Management Plan

## 1. Overview

### 1.1 Scope & Estimation

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **WBS Item** | **Complexity** | **Est. Effort**  **(man-days)** |
| ***1*** | ***Login*** |  | **13** |
| 1.1 | ADMIN - ADMISSION - Login with username | Simple | 3 |
| 1.2 | Login with Facebook | Medium | 5 |
| 1.3 | Login with Google | Medium | 5 |
| ***2*** | ***LANDLORD - Sign Up*** |  | **17** |
| 2.1 | Sign Up with Facebook | Medium | 5 |
| 2.2 | Sign Up with Google | Medium | 5 |
| 2.3 | Upload ID Card | Simple | 3 |
| 2.4 | Choose Campus | Simple | 2 |
| 2.5 | Provide Basic Information | Simple | 2 |
| ***3*** | ***GUEST - List of houses*** |  | **36** |
| 3.1 | View List of Available Houses | Simple | 3 |
| 3.2 | Search Available House | Complex | 10 |
| 3.3 | View List of all Houses | Simple | 3 |
| 3.4 | View List of Reported Houses | Medium | 5 |
| 3.5 | House - Room Statistic | Medium | 5 |
| 3.6 | Filter Result | Complex | 10 |
| 3.6.1 | Filter by Type | Simple | 2 |
| 3.6.2 | Filter by Address | Simple | 2 |
| 3.6.3 | Filter by Price | Simple | 2 |
| 3.6.4 | Filter by Distance | Medium | 2 |
| 3.6.5 | Filter by Services | Medium | 2 |
| ***4*** | ***GUEST - House Details*** |  | **19** |
| 4.1 | House basic details | Medium | 5 |
| 4.2 | Maps position | Complex | 8 |
| 4.3 | View House Review | Simple | 2 |
| 4.4 | Available Room List | Simple | 2 |
| 4.5 | Landlord Information | Simple | 2 |
| ***5*** | ***GUEST - Room Details*** |  | **2** |
| 5.1 | View Room Details | Simple | 2 |
| ***6*** | ***GUEST - Review House*** |  | **8** |
| 6.1 | Star Rating | Medium | 4 |
| 6.2 | Comment | Medium | 4 |
| ***7*** | ***Report Violation*** |  | **4** |
|  | Report Violation | Medium | 4 |
| ***8*** | ***LANDLORD - List of Houses*** |  | **3** |
| 8.1 | View List of Houses | Medium | 3 |
| ***9*** | ***LANDLORD - Manage Houses*** |  | **25** |
| 9.1 | Update House Information | Medium | 4 |
| 9.2 | Delete House | Medium | 3 |
| 9.3 | Add New House | Complex | 10 |
| 9.3.1 | Download Templates | Medium | 2 |
| 9.3.2 | Import Data | Complex | 8 |
| 9.4 | Search House | Complex | 8 |
| 9.4.1 | Filter by Status | Simple | 2 |
| ***10*** | ***LANDLORD - Manage Rooms*** |  | **24** |
| 10.1 | Room List | Simple | 2 |
| 10.2 | View Room Details | Simple | 3 |
| 10.3 | Change Room status | Simple | 2 |
| 10.4 | Add Room | Complex | 8 |
| 10.5 | Delete Room | Medium | 4 |
| 10.6 | Update Room Information | Medium | 5 |
| ***11*** | ***LANDLORD - Profile*** |  | **6** |
| 11.1 | View Profile | Simple | 2 |
| 11.2 | Update Profile | Medium | 4 |
| ***12*** | ***ADMIN - Manage Staff Account*** |  | **18** |
| 12.1 | Change Password | Medium | 4 |
| 12.2 | Staff List | Simple | 2 |
| 12.3 | Create Staff Account | Medium | 5 |
| 12.4 | Change Active Status | Simple | 3 |
| 12.5 | Reset Password | Medium | 4 |
| ***13*** | ***STAFF - Landlord Sign Up Request*** |  | **6** |
| 13.1 | View Sign Up Request | Simple | 2 |
| 13.2 | Approve/Reject Sign Up Request | Medium | 4 |
| ***14*** | ***STAFF - Manage Landlord Account*** |  | **8** |
| 14.1 | Landlord List | Simple | 2 |
| 14.2 | Landlord Details | Medium | 4 |
| 14.3 | Change Active Status | Simple | 2 |
| ***15*** | ***STAFF - Manage Houses*** |  | **12** |
| 15.1 | View List of House | Simple | 2 |
| 15.2 | Filter list | Complex | 6 |
| 15.2.1 | Filter by Area | Medium | 3 |
| 15.2.2 | Filter by Availability Status | Medium | 3 |
| 15.3 | View List of Reported House | Medium | 4 |
| 15.3.1 | Change Active Status | Simple | 2 |
| 15.3.2 | List of Reports | Simple | 2 |
| ***Total Estimated Effort (man-days)*** | | | ***201*** |

### 1.2 Project Objectives

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Testing Stage** | **Test Coverage** | **No. of Defects** | **% of Defect** | **Notes** |
| 1 | Reviewing | 95% |  |  |  |
| 2 | Unit Test | 90% |  |  |  |
| 3 | Integration Test | 85% |  |  |  |
| 4 | System Test | 80% |  |  |  |
| 5 | Acceptance Test | 80% |  |  |  |

**Milestone Timelines (%): 100**

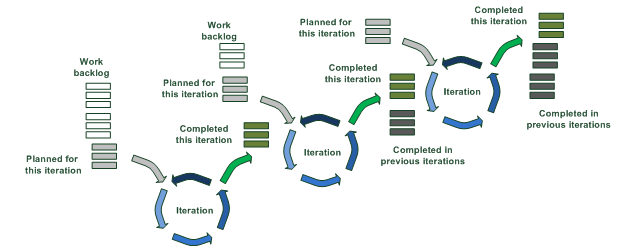
**Allocated Effort (man-days): 201**

### 1.3 Project Risks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Risk Description** | **Impact** | **Possibility** | **Response Plans** |
| 1 | Miss Deadline | Severe | High | Working over time |
| 2 | Requirement Change | Severe | Medium | Adjust Task and Schedule |

## 2. Management Approach

### 2.1 Project Process

**

The project is developed using the combination of Incremental Model and Iterative Model. Project requirements is divided into multiple standalone modules of software develop cycle. Each module goes through the requirements, design, implementation, and testing phase. Each subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

Some Definition:

- Work backlog is a list of parts of requirements for each phase

-

### 2.2 Quality Management

To improve the quality of the project, these approaches are included:

* Defect Prevention
* Unit Testing
* Integration Testing
* System Testing
* Acceptance Testing

### 2.3 Training Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Training Area | Participants | When, Duration | Waiver Criteria |
| Angular 12 | All Member | 11/09/2022 - 17/09/2022 | Mandatory |
| Git, Github | All Member | 09/09/2022 | Mandatory |

## 3. Project Deliverables

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Deliverable** | **Due Date** | **Notes** |
| 1 | Product Vision  Basic Use case  System actors  Report 1 | 10/09/2022 | Initiate Project |
| 2 | Project Scope  Requirements  Business Rules | 15/09/2022 | Customer Meeting  Functional Requirements  Non-functional Requirements |
| 3 | Plan and Schedule  Report 2 | 08/10/2022 |  |
| 4 | System Design  Report 3  Report 4 | 15/10/2022 | Screen Design  Architectural Design  Database Design  Etc. |
| 5 | Sprint 1 | 22/10/2022 | Coding  Unit Testing  Integration Testing |
| 6 | Sprint 2 | 05/11/2022 | Coding  Unit Testing  Integration Testing |
| 7 | Sprint 3 | 19/11/2022 | Coding  Unit Testing  Integration Testing  System Testing  User Acceptance Testing |
| 8 | Sprint 4 | 26/11/2022 | Coding  Unit Testing  Integration Testing  System Testing  User Acceptance Testing |
| 9 | Sprint 5 | 03/12/2022 | System Testing  User Acceptance Testing  Deploy Final Product |
| 10 | Final Report  Final Product  User Guides  Presentation | 10/12/2022 |  |

## 4. Responsibility Assignments

*D~Do; R~Review; S~Support; I~Informed; <blank>- Omitted*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Responsibility** | **GiangNTHE153046** | **HuyenBNHE150346** | **ThongPQHE150340** | **AnNTHE150432** | **KienNTHE150160** |
| Collect Requirements | R,D | D | D | D | D |
| Prepare Project Introduction Document | R,D | I | D | I | I |
| Prepare Project Management Plan | R | D | I | D | R |
| Prepare SRS Document (User Requirements) | R,D | I | I | I | D |
| Create Screen Mockups | R,D | D | D | D | D |
| Design Database | R,D | D | D | D | D |
| Collect Administrative Unit Data | D | I | I | I | I |
| Draw Use Case Diagram | R | D | I | I | I |
| Coge Function Login | R | I | I | D | I |
| Dram Entity Relationship Diagram | R | D | I | I | I |
| Draw Screen Flow Diagrams | D | I | I | I | I |

## 5. Project Communications

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Communication Item** | **Who/ Target** | **Purpose** | **When, Frequency** | **Type, Tool, Method(s)** |
| Daily Meeting | All team members | - Checking progress of assigned tasks  - Finding solutions for difficult problems | 9 p.m everyday | Discord |
| Assign Tasks | All team members | - Project Manager assigns tasks to other team members | Everyday | Jira |
| Weekly Meeting With Supervisor | All team members | - Checking progress of project  - Plan upcoming tasks  - Update requirements | Once a week | Offline |

## 6. Configuration Management

### 6.1 Document Management

* Management Tools:
  + Google Drive
  + OneDrive
  + GitHub
* Team leader assigns and describes tasks through Facebook & Discords. All Documents will be submitted to management tools to keep track of changes. Team leader then collects them all to submit to the mentor.

### 6.2 Source Code Management

* Management Tools:
  + GitHub
* Team leader assigns and describes tasks through Facebook & Discords. All source code will be pushed to Github for version control. Team leader then pull it back to run on deployment environment.

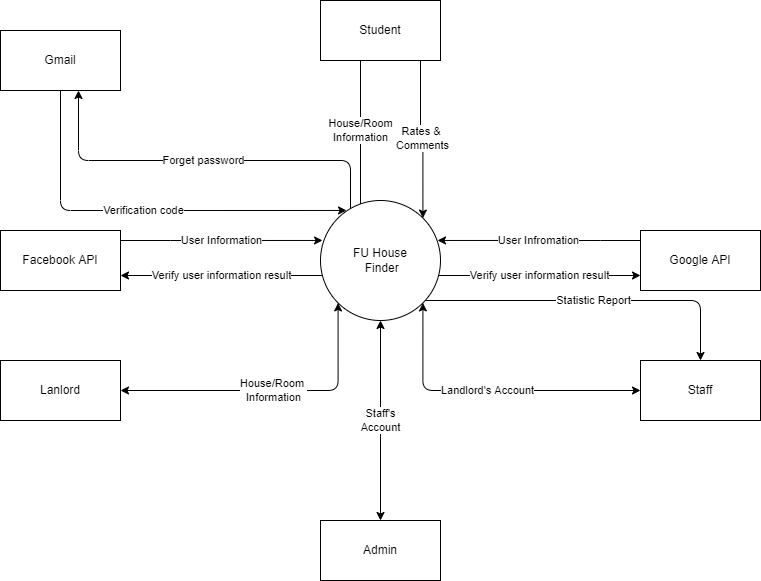
### 6.3 Tools & Infrastructures

|  |  |
| --- | --- |
| **Category** | **Tools / Infrastructure** |
| **Technology** | Angular 12 (FrontEnd); C# .NET 5 (BackEnd) |
| **Database** | Microsoft SQL Server |
| **IDEs/Editors** | Visual Studio Code; Visual Studio |
| **Diagramming** | DrawIO; Mindmeister |
| **Documentation** | Ms Offic;, Google Docs; Microsoft Office |
| **Version Control** | GitHub (Source Codes), Google Drive (Documents); OneDrive (Documents) |
| **Deployment server** |  |
| **Project management** | Jira (Schedule, Tasks, Defects) |
| **UI/UX Design** | Figma |

# III. Software Requirement Specification

## 1. Product Overview

The FU House Finder System is a new software system that helps the students and the landlords find each other through the Internet. The context diagram below illustrates the external entities and system interfaces for release 1.0. The system is expected to evolve over several releases, ultimately having the ability to recommend suitable houses for students based on view history and also earn money for the host.



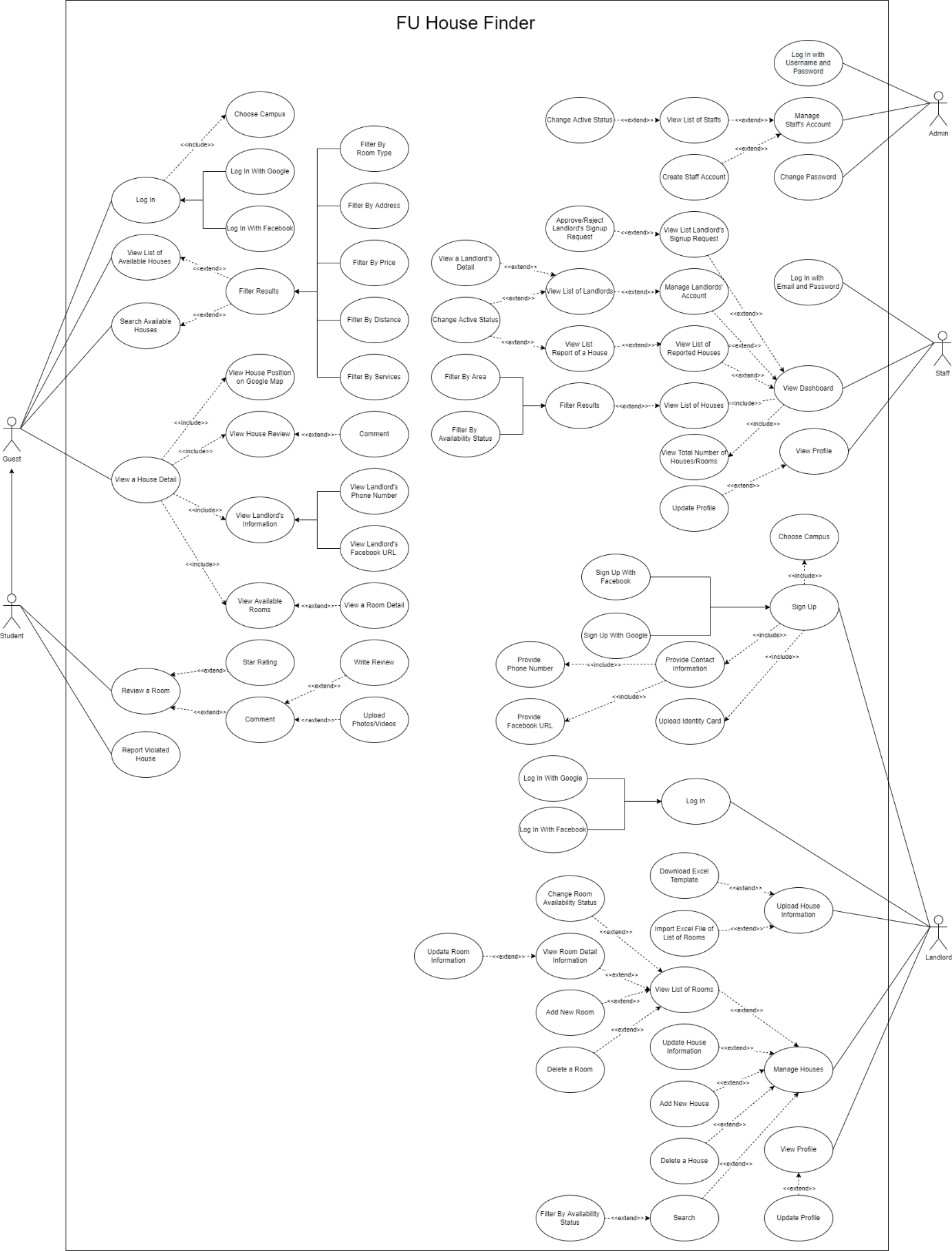
## 2. User Requirements

### 2.1 Actors

|  |  |  |
| --- | --- | --- |
| **#** | **Actor** | **Description** |
| 1 | Administrator | Administrator of the system; Manage accounts of Staffs |
| 2 | Staff | Staffs of the offices of the University; Manage accounts of Landlords and view statistics reports |
| 3 | Landlord | People having houses for rent; Manage their houses and rooms and their information |
| 4 | Student | People finding for houses to rent; Can search for available houses in the system |

### 2.2 Use Cases

#### 2.2.1 Diagram(s)



#### 2.2.2 Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Use Case** | **Actors** | **Use Case Description** |
| 01 | Log In | Guest | Guest logs into the system |
| 02 | View List Of Available Houses | Guest | Guest views list of houses |
| 03 | Search Available Houses | Guest | Guest searches for houses by name or filters it by category or by map |
| 04 | View a House Detail | Guest | Guest views the detail information of house |
| 05 | Review a House | Student | Student rates and comments the house |
| 06 | Report Violated House | Student | Student reports the violated house |
| 07 | Log In with Email and Password | Admin | Admin logs into the system |
| 08 | Manage Staff’s Account | Admin | Admin views the list of all staff accounts and can be change active status of the staff |
| 09 | Change Password | Admin | Admin changes login password |
| 10 | Log In with Email and Password | Staff | Staff logs into the system |
| 11 | View Dashboard | Staff | Staff views list of available houses and list of landlord’s request. Staff can be search for houses & statistics by name or filter it by category at the same time can also accept or decline landlord’s request |
| 12 | View Profile | Staff | Staff views and updates the profile |
| 13 | Sign Up | Landlord | Landlord registers for an account |
| 14 | Log In | Landlord | Landlord logs into the system |
| 15 | Upload House Information | Landlord | Landlord imports information of the house by excel file |
| 16 | Manage Houses | Landlord | Landlord views list of rooms, detail information of room at the same time can also add new, update, delete, change status the room |
| 17 | View Profile | Landlord | Landlord views and updates the profile |

## 3. Functional Requirements

### 3.1 System Functional Overview

#### 3.1.1 Screens Flow

Student

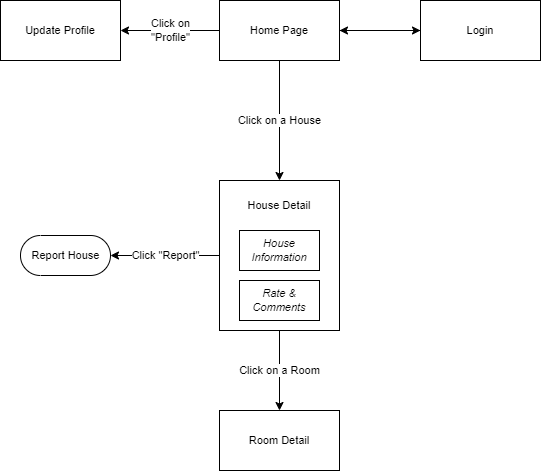


Figure 1-1: Screen Flow for Student

Landlord

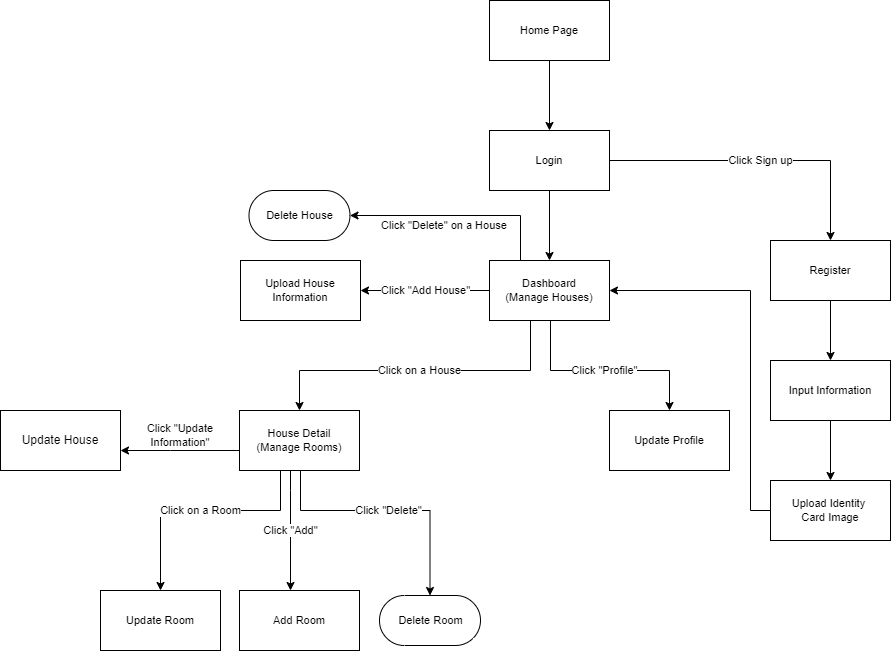


Figure 1-2: Screen Flow for Landlord

Staff

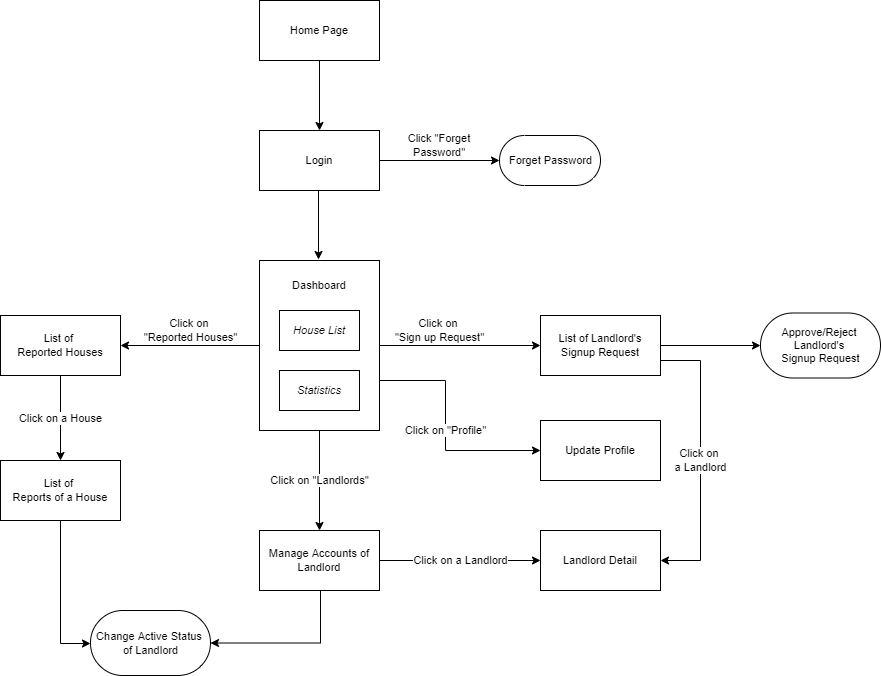


Figure 1-3: Screen Flow for Staff

Admin

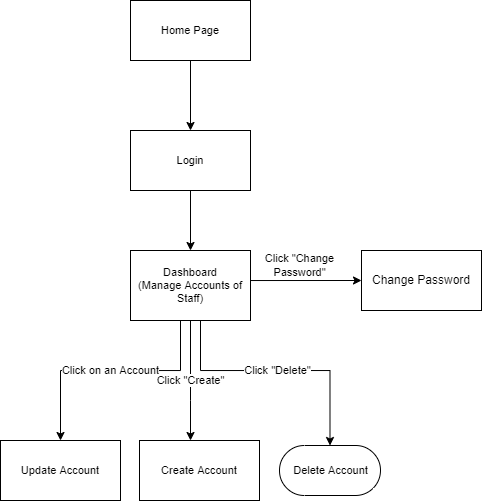


Figure 1-3: Screen Flow for Admin

#### 3.1.2 Screen Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Feature** | **Screen** | **Description** |
|  | View List Houses | Home Page | View list of houses |
|  | Search & Filter House | Home Page | Search for houses by name or filter it by category |
|  | Interact with map | Home Page | Use the map to find the location of the house |
|  | View House Detail | House Details | Display detail information of house |
|  | View Room Detail | Room Details | Display detail information of room |
|  | Report House | Report House | User reports the house |
|  | Login | User Login | User logs into the system |
|  | Register | User Register | User registers for an account |
|  | Register | Register [Landlord] | Landlord registers for an account |
|  | Verify Information Of Landlord | Input Information [Landlord] | Get information of landlord |
|  | Verify Identity Card Of Landlord | Upload Identity Card Image [Landlord] | Get identity card of landlord |
|  | View List Houses | Dashboard [Landlord] | Landlord views list of houses |
|  | Search & Filter House | Dashboard [Landlord] | Landlord searché for houses by name or filters it by category |
|  | Upload House Information | Upload House Information [Landlord] | Landlord imports information of the house |
|  | Manage Rooms | House Detail [Landlord] | Display detail information of house |
|  | Search & Filter Room | House Detail [Landlord] | Landlord searché for rooms by name or filters it by category |
|  | Update House | Update House [Landlord] | Landlord updates the house |
|  | Delete House | Delete House [Landlord] | Landlord deletes the house |
|  | Update Room | Update Room [Landlord] | Landlord updates the room |
|  | Add Room | Add Room [Landlord] | Landlord adds new the room |
|  | Delete Room | Delete Room [Landlord] | Landlord deletes the room |
|  | Update Proflie | Update Proflie [Landlord] | Landlord updates the proflie |
|  | View List Houses & Statistics | Dashboard [Staff] | Staff view list of available houses |
|  | Search & Filter House & Statistic | Dashboard [Staff] | Staff searches for houses & statistics by name or filter it by category |
|  | Accept Or Decline Landlord’s Signup Request | List Of Landlord’s Signup Request [Staff] | Staff views list of landlord’s request and accepts or decline |
|  | Update Proflie | Update Proflie [Staff] | Staff updates the proflie |
|  | View List Accounts Of Landlords | Landlords List [Staff] | Staff views list of landlords |
|  | View Landlord Detail | Landlord Detail [Staff] | Staff views detail information of landlord |
|  | View List Of Reported Houses | List Of Reported Houses [Staff] | Staff views list of reported houses |
|  | View List Accounts Of Staffs | Dashboard [Admin] | Admin views the list of all staff accounts |
|  | Search & Filter Staff | Dashboard [Admin] | Admin searches for staffs by name or filter it by category |
|  | Change Password | Change Password [Admin] | Admin changes login password |
|  | Create Staff Account | Create Account [Admin] | Admin creates account for staff |
|  | Update Staff Account | Update Account [Admin] | Admin updates account of staff |
|  | Detele Staff Account | Detele Account [Admin] | Admin deletes account of staff |

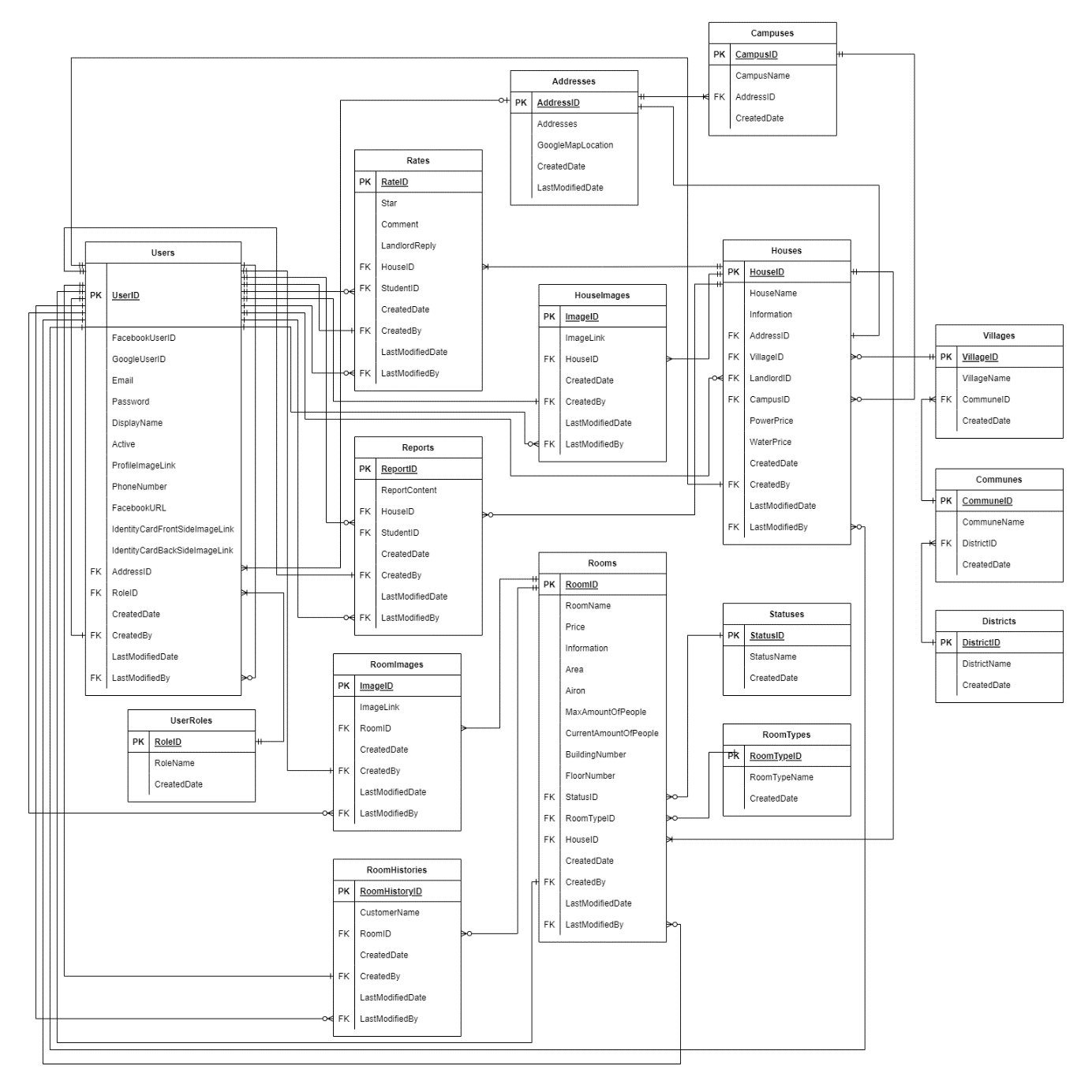
#### 3.1.3 Screen Authorization

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Screen** | **Guest** | **Student** | **Landlord** | **Staff** | **Admin** |
| Home Page | X | X | X | X | X |
| House Details | X | X | X | X | X |
| Room Details | X | X | X | X | X |
| Report House |  | X |  |  |  |
| User Login |  | X | X | X | X |
| User Register | X |  |  |  |  |
| Register [Landlord] |  |  | X |  |  |
| Input Information [Landlord] |  |  | X |  |  |
| Upload Identity Card Image [Landlord] |  |  | X |  |  |
| Dashboard [Landlord] |  |  | X |  |  |
| Upload House Information [Landlord] |  |  | X |  |  |
| House Detail [Landlord] |  |  | X |  |  |
| Update House [Landlord] |  |  | X |  |  |
| Delete House [Landlord] |  |  | X |  |  |
| Update Room [Landlord] |  |  | X |  |  |
| Add Room [Landlord] |  |  | X |  |  |
| Delete Room [Landlord] |  |  | X |  |  |
| Update Proflie [Landlord] |  |  | X |  |  |
| Dashboard [Staff] |  |  |  | X |  |
| List Of Landlord’s Signup Request [Staff] |  |  |  | X |  |
| Update Proflie [Staff] |  |  |  | X |  |
| Landlords List [Staff] |  |  |  | X |  |
| Landlord Detail [Staff] |  |  |  | X |  |
| List Of Reported Houses [Staff] |  |  |  | X |  |
| Dashboard [Admin] |  |  |  |  | X |
| Change Password [Admin] |  |  |  |  | X |
| Create Account [Admin] |  |  |  |  | X |
| Update Account [Admin] |  |  |  |  | X |
| Detele Account [Admin] |  |  |  |  | X |

#### 3.1.4 Non-Screen Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Feature** | **System Function** | **Description** |
| 1 | Login with Facebook | Login | User logs in the system with his Facebook account |
| 2 | Login with Google | Login | User logs in the system with his Google account |
| 3 | Import File Excel | Sign up & Manage Houses | Convert data from Excel file into house & room information |
| 4 | Forget Password | Login | System send verification code to user Email to reset password |

#### 3.1.5 Entity Relationship Diagram

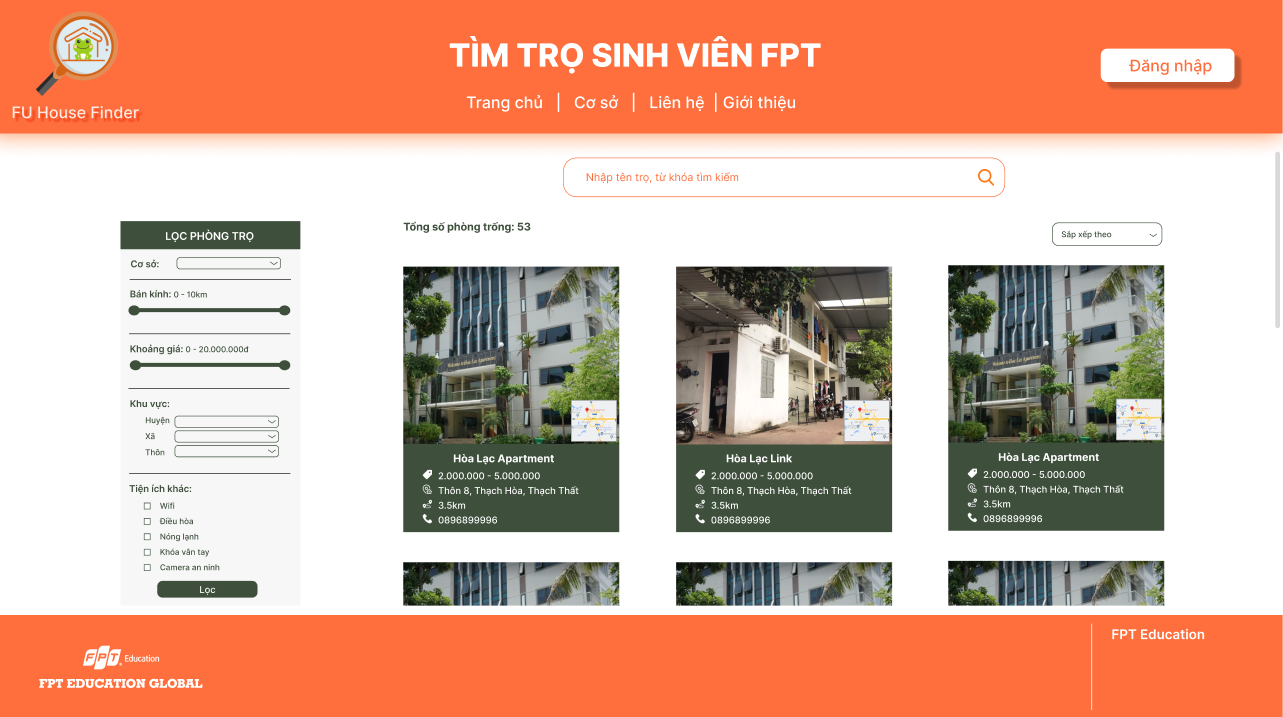


**Entities Description**

|  |  |  |
| --- | --- | --- |
| **#** | **Entity** | **Description** |
| 1 | Addresses | Used to store the address of User, House & Campus |
| 2 | Campuses | Used to store Campuses of FPT university |
| 3 | UserRoles | Used to store the role of each User |
| 4 | Users | Used to store Users in the system |
| 5 | Districts | Used to store Districts around the school |
| 6 | Communes | Used to store Communes around the school |
| 7 | Villages | Used to store Villages around the school |
| 8 | Houses | Used to store Houses |
| 9 | Statuses | Used to store Statuses of Room |
| 10 | RoomTypes | Used to store Types of Room |
| 11 | Rooms | Used to store Rooms |
| 12 | Rates | Used to store Rating & Comments of User |
| 13 | ImagesOfHouse | Used to store Images of House |
| 14 | ImagesOfRoom | Used to store Images of Room |
| 15 | Reports | Used to store student's Reports for House |
| 16 | RoomHistories | Used to store Histories of people staying in the House, for the landlord to voluntarily add if there is a need to manage & monitor |

### 3.2 Home page

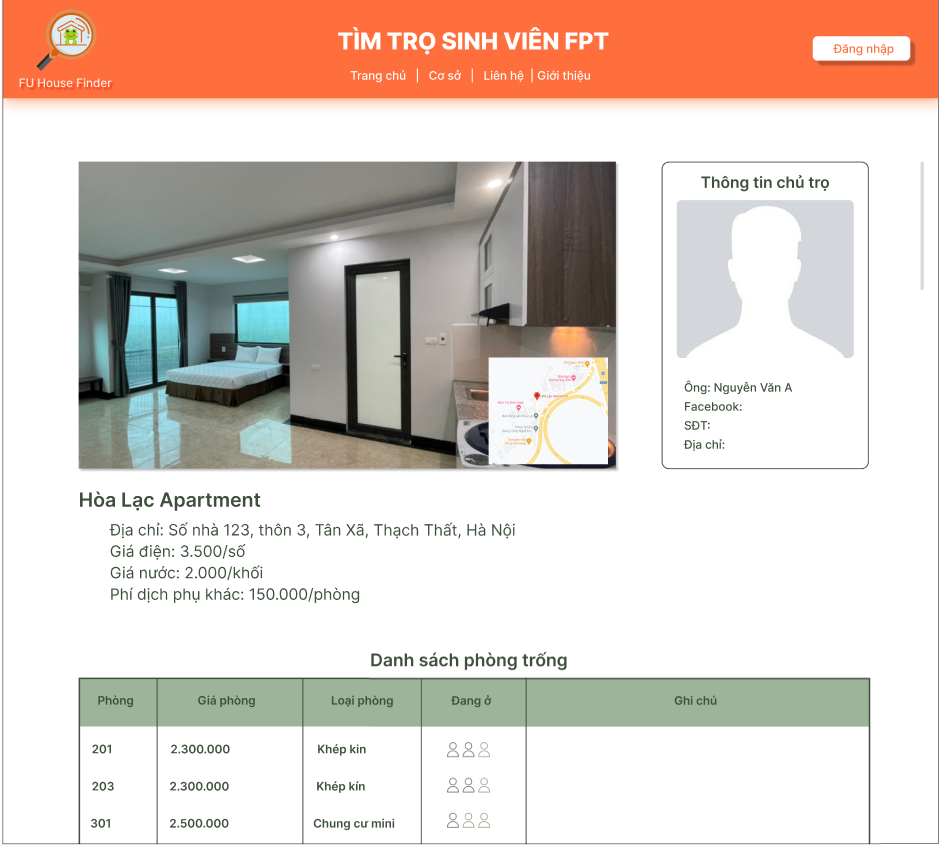
* Function trigger: User accesses the website
* Function description:
  + Role: Guest, Student
  + Purpose: User could view list of available houses and filter the results
* Screen layout:



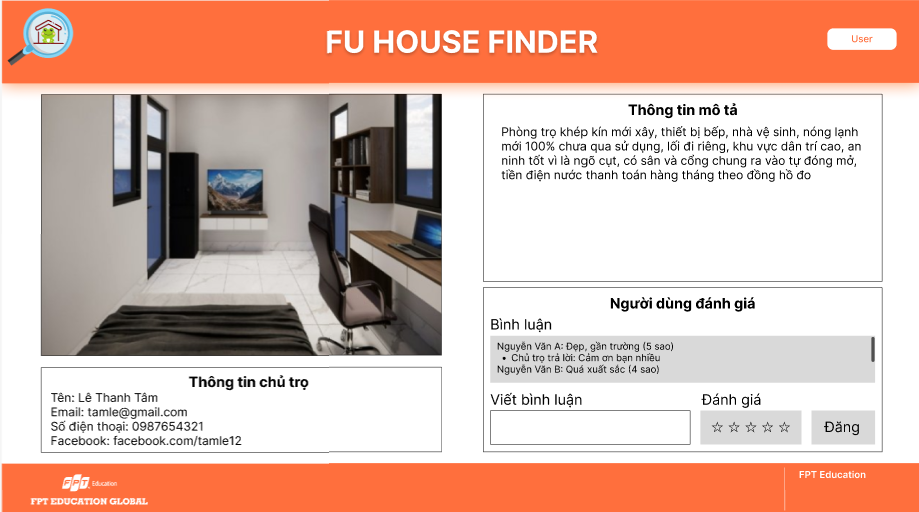
### 3.3 Login/Signup

* Function trigger: User clicks “Đăng nhập” from Home Page
* Function description:
  + Role: Guest
  + Purpose: Guest could sign up for an account or log in to access more functions
* Screen layout: 

### 3.4 House Detail

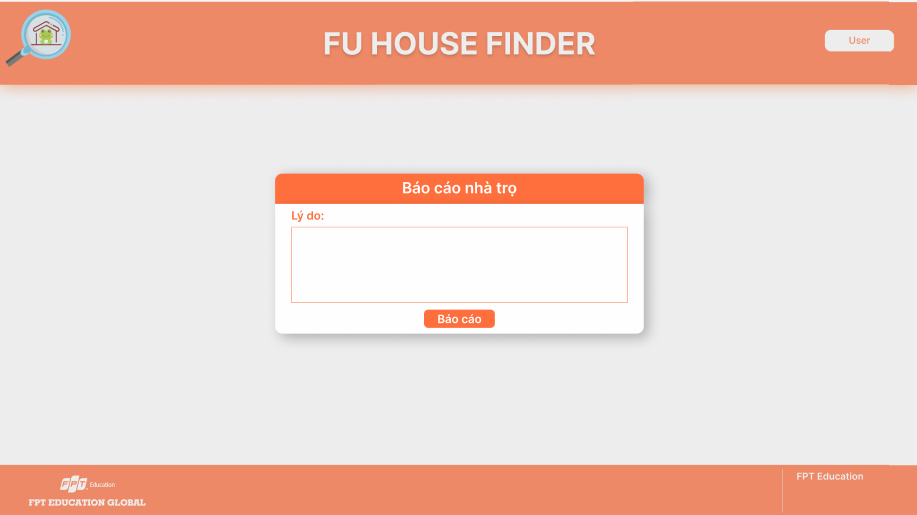
* Function trigger: User clicks on one house from Home page
* Function description:
  + Role: Guest, student
  + Purpose: View a house detail information (landlord’s information, available rooms, etc.)
* Screen layout: 

### 3.5 Room Detail

* Function trigger: User clicks on one Room from a House Detail
* Function description:
  + Role: Guest, student
  + Purpose: View a room detail information (images, price, etc.)
* Screen layout: 

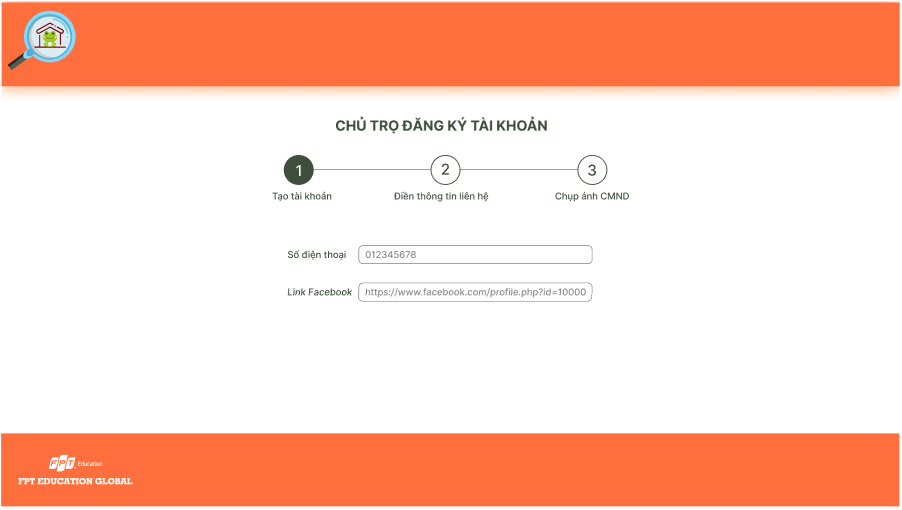
### 3.6 Report House

* Function trigger: Student clicks “Báo cáo” in one House Detail
* Function description:
  + Role: Student
  + Purpose: Student uses to report an unusual house
* Screen layout:



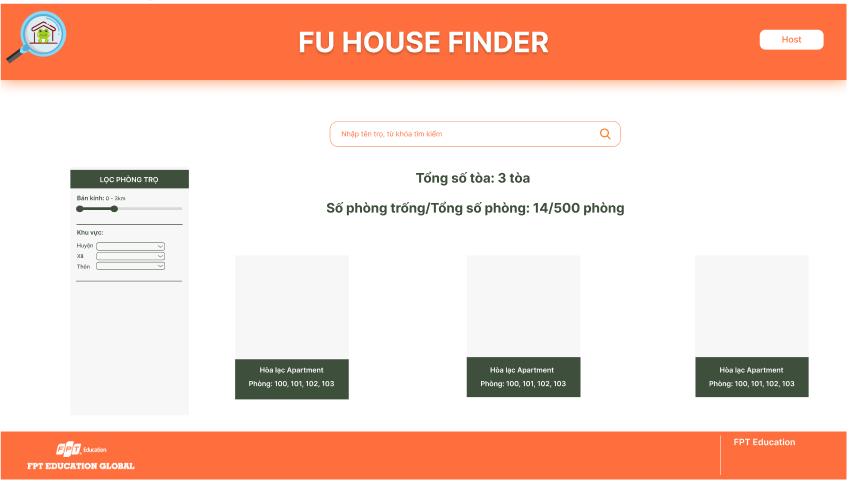
### 3.7 Register [Landlord]

* Function trigger: User click the landlord’s register button
* Function description:
  + Role: Guest
  + Purpose: Landlord register for an account
* Screen layout:

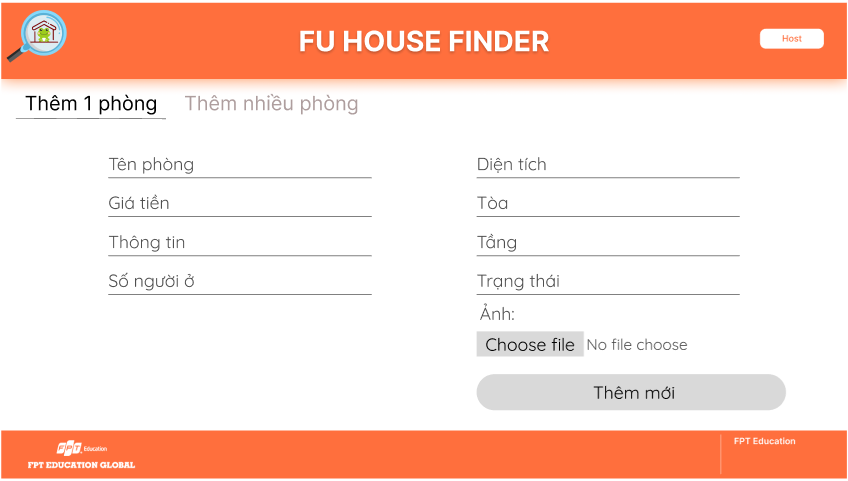
### 3.8 Dashboard [Landlord]

* Function trigger: Landlord accesses the website
* Function description:
  + Role: Landlord
  + Purpose: Landlord view list of houses and search for houses by name or filter it by category
* Screen layout:



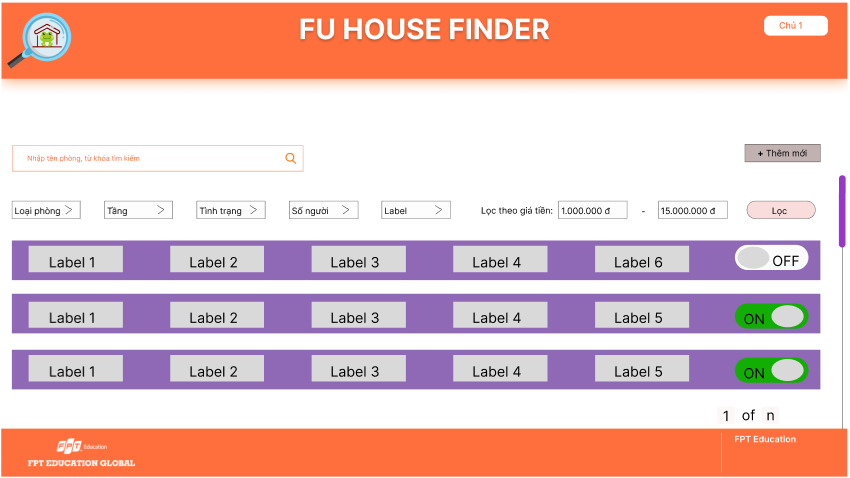
### 3.9 Upload House Information [Landlord]

* Function trigger: Landlord clicks the import button
* Function description:
  + Role: Landlord
  + Purpose: Landlord import information of the house
* Screen layout:

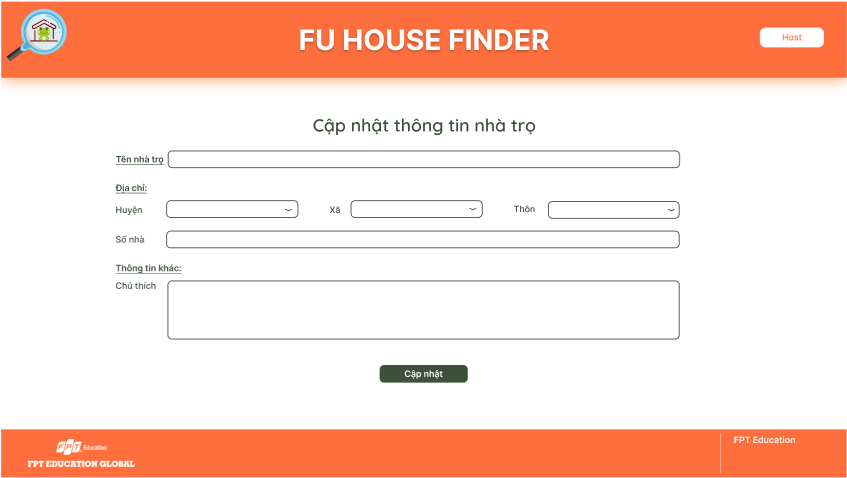
### 3.10 House Detail [Landlord]

* Function trigger: Landlord clicks each item of list houses
* Function description:
  + Role: Landlord
  + Purpose: Display detail information of house and search for rooms by name or filter it by category
* Screen layout:



### 3.11 Update House [Landlord]

* Function trigger: Landlord clicks the update button
* Function description:
  + Role: Landlord
  + Purpose: Landlord update the house
* Screen layout:



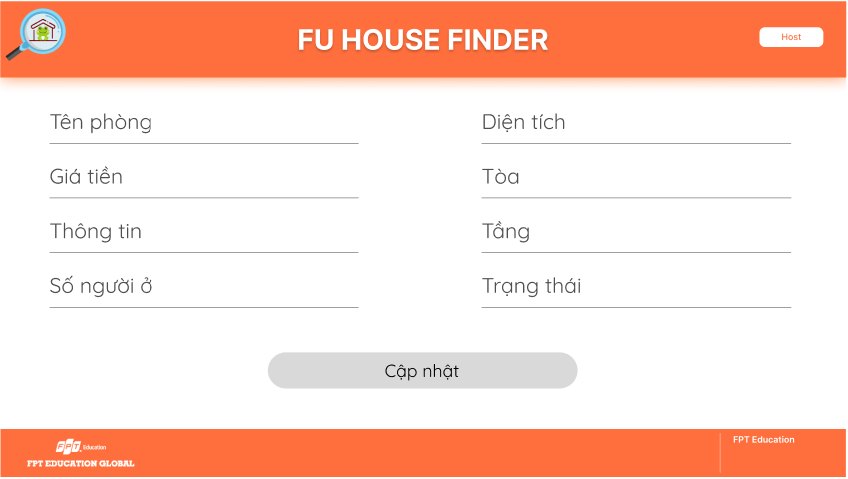
### 3.12 Delete House [Landlord]

* Function trigger: Landlord clicks the delete button
* Function description:
  + Role: Landlord
  + Purpose: Landlord delete the house
* Screen layout:



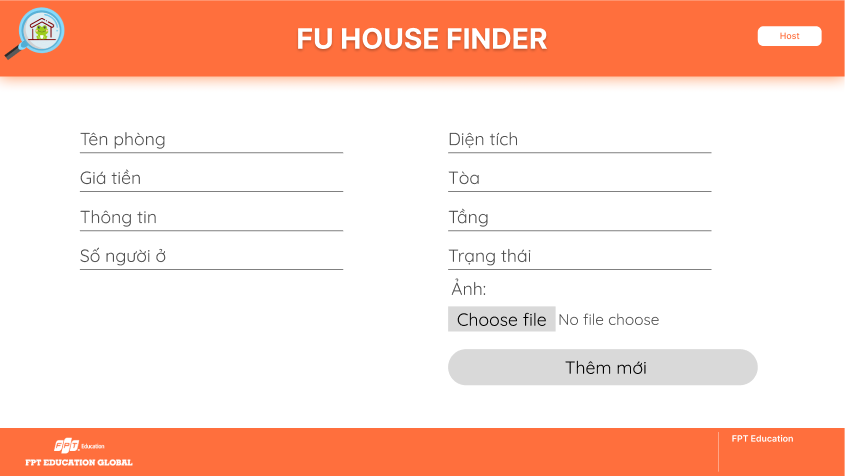
### 3.13 Update Room [Landlord]

* Function trigger: Landlord clicks the update button of each room in the list
* Function description:
  + Role: Landlord
  + Purpose: Landlord update the room
* Screen layout:



### 3.14 Add Room [Landlord]

* Function trigger: Landlord clicks the add button
  + Function description:
  + Role: Landlord
* Purpose: Landlord add the new room
* Screen layout:



### 3.15 Delete Room [Landlord]

* Function trigger: Landlord clicks the delete button of each room in the list
* Function description:
  + Role: Landlord
  + Purpose: Landlord delete the room
* Screen layout:



## 4. Non-Functional Requirements

### 4.1 External Interfaces

#### 4.1.1 User Interfaces

UI-1: The FU House Finder System screen displays shall conform to the User Interface Design and User Experience Design

UI-2: The website is designed with the feature of using Angular framework to provide a smooth user experience without having to reload the website many times.

#### 4.1.2 Software Interfaces

SI-1: FU House Finder Account Checking system

SI-1.1: Upload existing user data in the system through a programming interface

SI-1.2: The system automatically checks what state the account is in in the Active attribute of User table

SI-1.3: There will be 2 states including Active and Deactive. If the account is Active, you will be able to perform actions to the system (including managing Landlord’s accounts if you are Staff, and managing Houses if you are Landlord). If the account is Deactive, you will not be able to log in to perform any actions.

SI-2: FU House Finder Inventory System

SI-2.1: House Finder System shall transmit the quantities of house and room items to the House Finder Inventory System through a programmatic interface.

SI-2.2: House Finder System shall poll the House Finder Inventory System to determine whether a requested house item is available.

SI-2.3: The House Finder System will display the available houses left in system for the searching students. If the house is not available, the system will not display for the student to see.

#### 4.1.3 Hardware Interfaces

No hardware interfaces have been identified.

#### 4.1.4 Communication Interfaces

CI-1: FU House Finder shall send an email or send a message to a phone number (based on user account settings) to the Landlord to report any problems reported by students, the Landlord then will present at University campus to resolve.

### 4.2 Quality Attributes

#### 4.2.1 Availability

AVL-1: The FU House Finder website shall be available at least 98% of the time between 5:00 A.M. and midnight local time and at least 90% of the time between midnight and 5:00 A.M. local time, excluding scheduled maintenance windows.

#### 4.2.2 Usability

USB-1: The website shall be designed with user-friendly interfaces so that users could complete the main actions once they see the interface.

USB-2: Landlords shall import the list of their houses within 5 steps.

#### 4.2.3 Localization

LCL-1: The date format must be as follows: date/month/year.

#### 4.2.4 Performance

PE-1: The website must provide 7 seconds or less respond time in a Chrome browser in peak usage condition.

PE-2: The web pages shall fully load in an average of 5 seconds in normal condition.

#### 4.2.5 Security

SE-1: Only admin shall be able to create a new staff’s account and only staff shall be able to approve/reject landlords’ signup request.

SE-2: Landlords must provide their identity card image to be able to sign up a landlord account.

## 5. Requirement Appendix

### 5.1 Business Rules

|  |  |
| --- | --- |
| ID | Rule Definition |
| BR-01 | Only Staff can approve Landlord’s Sign up request and Deactive Landlord’s account. |
| BR-02 | Only Admin can create, modify, or deactive Staff’s account. |
| BR-03 | Landlord can only upload House information after sign up request being approved by Staff. |
| BR-04 | All Passwords require 256-bit encryption. |
| BR-05 | Student’s Search functionality requires at least searching by distance from campus, price and utility. |
| BR-06 | Landlord’s House information upload must adhere to the rules of the template |

### 5.2 Common Requirements

### 5.3 Application Messages List

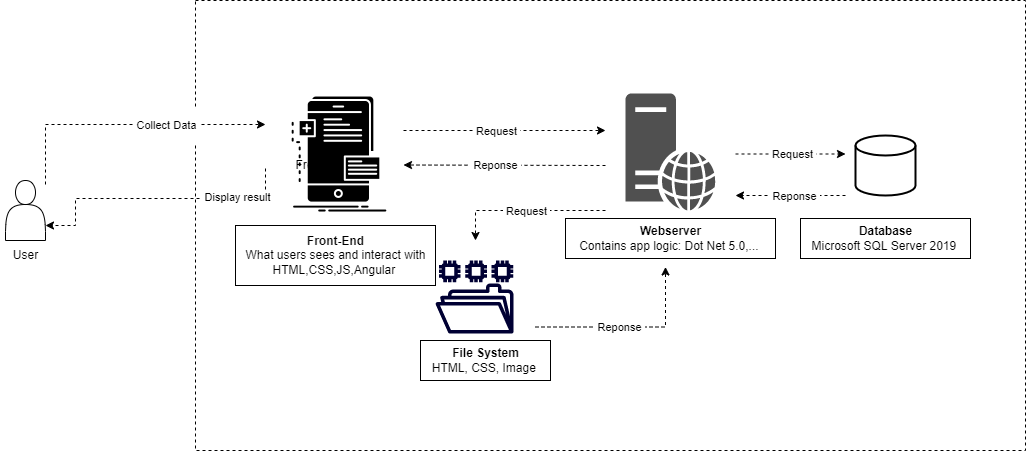
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Message code** | **Message Type** | **Context** | **Content** |
| 1 | MSG01 | In line | There is not any search result | *No search results.* |
| 2 | MSG02 | In red, under the text box | Input-required fields are empty | *The \* field is required.* |
| 3 | MSG03 | Toast message | Updating asset(s) information successfully | *Update asset(s) successfully.* |
| 4 | MSG04 | Toast message | Adding new asset successfully | *Add asset successfully.* |
| 5 | MSG05 | Toast message | Confirming email of asset hand-over is sent successfully | *A confirmation email has been sent to {email\_address}.* |
| 6 | MSG06 | Toast message | Resetting asset information successfully | *Return asset(s) successfully.* |
| 7 | MSG07 | Toast message | Deleting asset information successfully | *Delete asset(s) successfully.* |
| 8 | MSG08 | In red, under the text box | Input value length > max length | *Exceed max length of {max\_length}.* |
| 9 | MSG09 | In line | Username or password is not correct when clicking sign-in | *Incorrrect user name or password. Please check again.* |

### 5.4 Other Requirements…

# IV. Software Design Description

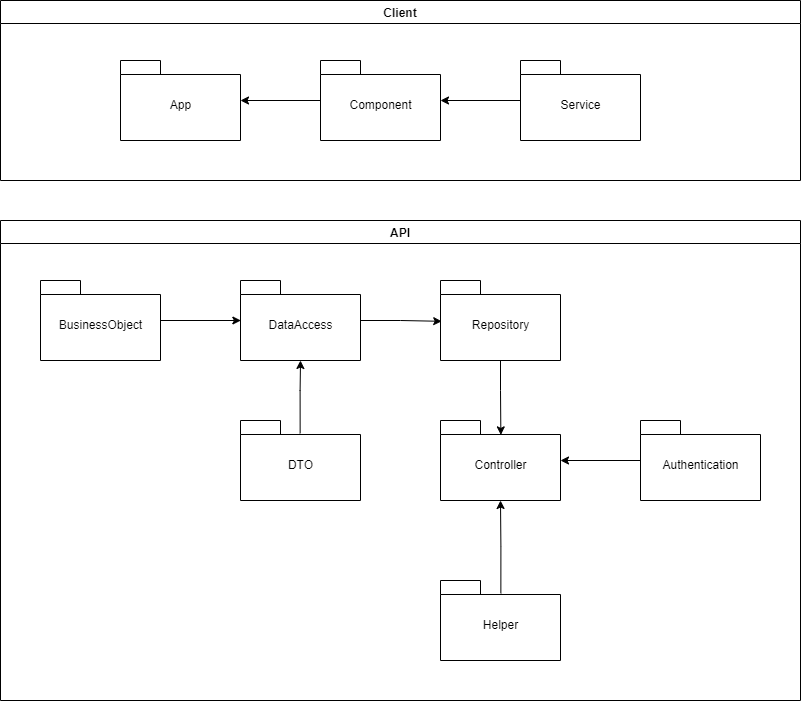
## 1. System Design

### 1.1 System Architecture



|  |  |  |
| --- | --- | --- |
| **No** | **Component** | **Description** |
| 1 | User | As a user, impact on the site |
| 2 | Front-End | What users sees and interact with HTML, CSS, JS, Angular |
| 3 | Webserver | Contains app logic: Dot Net 5.0, ... |
| 4 | File System | Contains HTML, CSS, Image in client. |
| 5 | Database | Microsoft SQL Server 2019, this contains all data of website. |

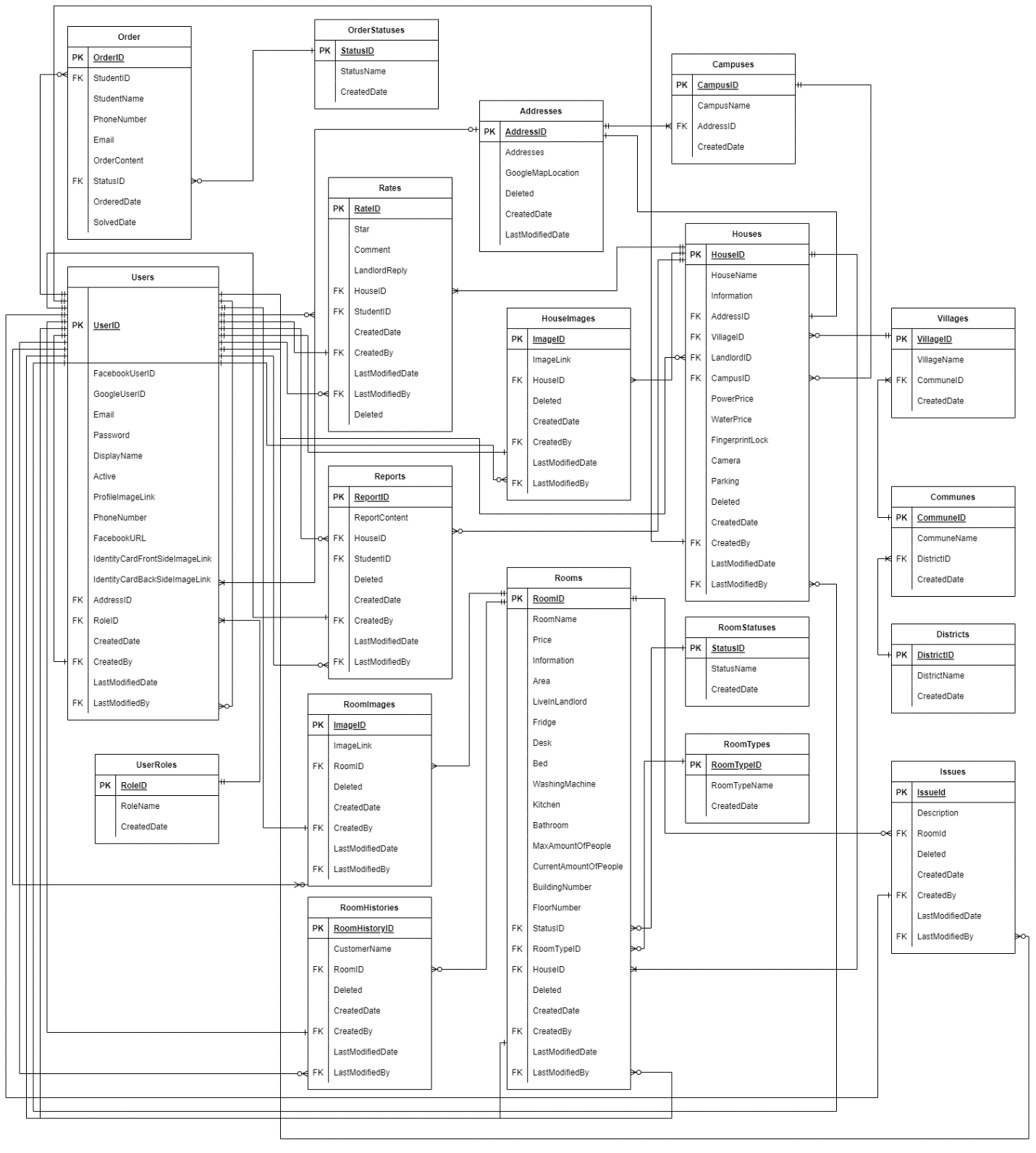
### 1.2 Package Diagram



***Package Descriptions***

|  |  |  |
| --- | --- | --- |
| **No** | **Package** | **Description** |
| 01 | BusinessObject | Define the logic of data extracted from Database |
| 02 | DataAccess | Provide the connection to the Database; Extract data for the system |
| 03 | Repository | Create an abstraction layer between the data access layer and the business logic layer of an application |
| 04 | DTO | Model with predefined validation in place for HTTP responses and requests |
| 05 | Authentication | Provide JWT Authentication methods for using in Controller |
| 06 | Helper | Provide helping methods for using in Controller |
| 05 | Controller | The controller takes the result of the model's processing (if any) and returns either the proper view and its associated view data or the result of the API call |
| 06 | App | Root component |
| 07 | Component | Components are the main building block for Angular applications. Each component consists of: An HTML template that declares what renders on the page. A TypeScript class that defines behavior |
| 08 | Service | Define code or functionalities that are then accessible and reusable in many other components in Angular project |

## 2. Database Design



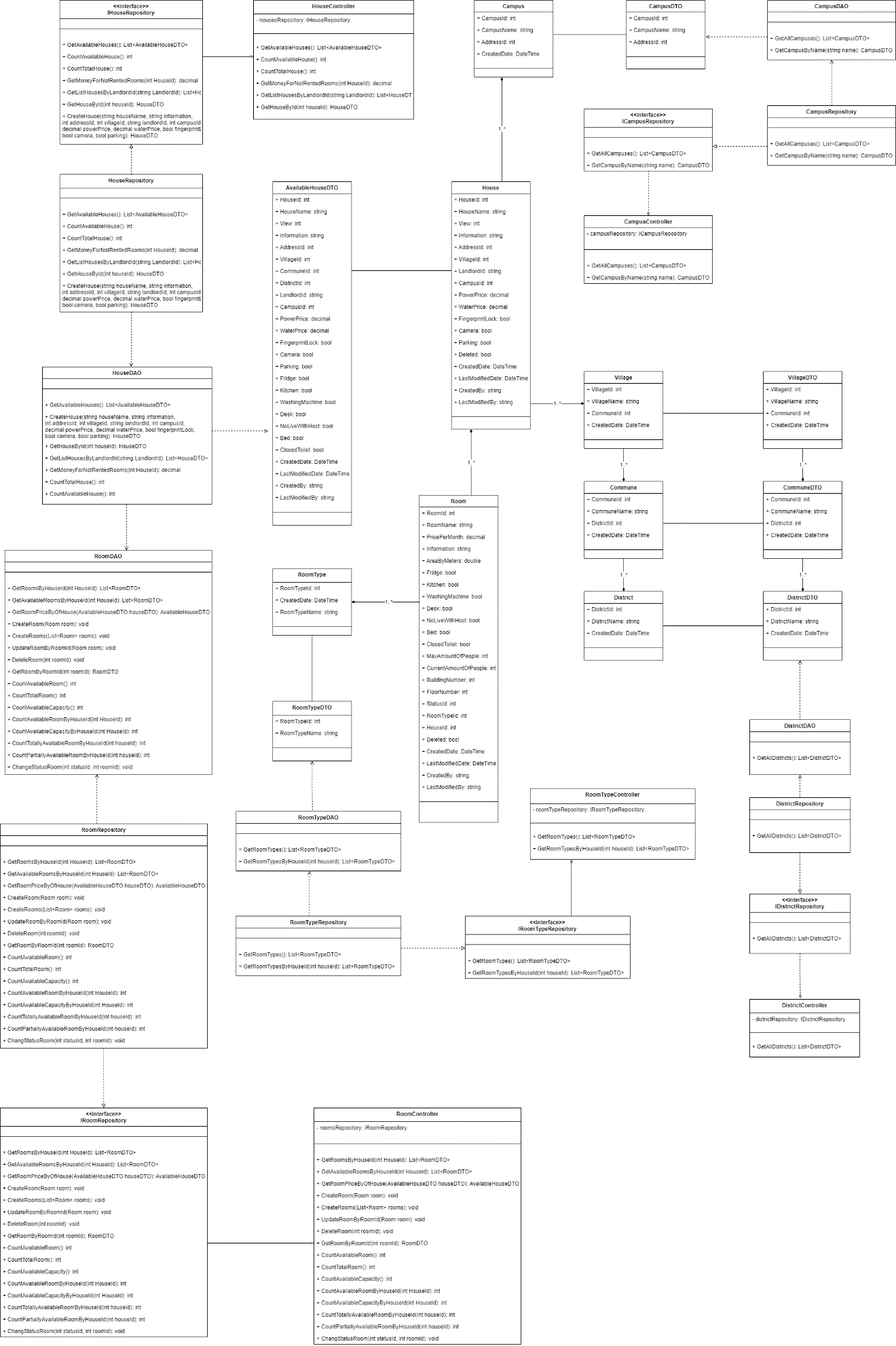
***Table Descriptions***

|  |  |  |
| --- | --- | --- |
| **No** | **Table** | **Description** |
| *01* | *Addresses* | *Store the information of addresses for Users (Landlords) and Houses*   * *Primary key: AddressId* |
| *02* | *Campuses* | *Store the information of all campus for users*   * *Primary key: CampusId* * *Foreign keys: AddressId* |
| *03* | *Communes* | *Store the information of all communes (for identifying house address)*   * *Primary key: CommuneId* * *Foreign keys: DistrictId* |
| *04* | *Districts* | *Store the information of all districts (for identifying house address)*   * *Primary key: DistrictId* * *Foreign keys: CampusId* |
| *05* | *Houses* | *Store the information of all houses*   * *Primary key: HouseId* * *Foreign keys: AddressId, LandlordId, VillageId, CampusId, CreatedBy, LastModifiedBy* |
| *06* | *ImagesOfHouse* | *Store the information of all pictures of house*   * *Primary key: ImageId* * *Foreign keys: HouseId, CreatedBy, LastModifiedBy* |
| *07* | *ImagesOfRoom* | *Store the information of all pictures of room*   * *Primary key: ImageId* * *Foreign keys: RoomId, CreatedBy, LastModifiedBy* |
| *08* | *Issues* | *Store the information of all records of Issues usage for Landlords to manage if they want to*   * *Primary key: IssueId* * *Foreign keys: RoomId, CreatedBy, LastModifiedBy* |
| *09* | *Order* | *Store the information of all records of all Orders*   * *Primary key: OrderId* * *Foreign keys: StudentId, StatusId* |
| *10* | *OrderStatuses* | *Store the information of all status of an Order*   * *Primary key: StatusId* |
| *11* | *Rates* | *Store the information of all rates and comments of users*   * *Primary key: RateId* * *Foreign keys: HouseId, StudentId, CreatedBy, LastModifiedBy* |
| *12* | *Reports* | *Store the information of all Reports of Students to Houses*   * *Primary key: ReportId* * *Foreign keys: HouseId, StudentId, CreatedBy, LastModifiedBy* |
| *13* | *RoomHistories* | *Store the information of all records of histories of a room*   * *Primary key: RoomHistoryId* * *Foreign keys: RoomId, CreatedBy, LastModifiedBy* |
| *14* | *Rooms* | *Store the information of all rooms of houses*   * *Primary key: RoomId* * *Foreign keys: StatusId, RoomTypeId, HouseId, CreatedBy, LastModifiedBy* |
| *15* | *RoomStatuses* | *Store the information of all status of a Room*   * *Primary key: StatusId* |
| *16* | *RoomTypes* | *Store the information of all types of a Room*   * *Primary key: RoomTypeId* |
| *17* | *UserRoles* | *Store the information of all roles of users*   * *Primary key: RoleId* |
| *18* | *Users* | *Store the information of all users*   * *Primary key: UserId* * *Foreign keys: RoleId, AddressId, CreatedBy, LastModifiedBy* |
| *19* | *Villages* | *Store the information of all villages (for identifying house address)*   * *Primary key: VillageId* * *Foreign keys: CommuneId* |

## 3. Detailed Design

### 3.1 Home Page

#### 3.1.1 Class Diagram



#### 3.1.2 Class Specifications

##### HouseDAO Class

This class performs action to data (CRUD) in House Table in Database

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### HouseController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### CampusDAO Class

This class performs action to data (CRUD) in Campus Table in Database

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetAllCampuses() | Return list of all Campuses in the Database |
| 02 | GetCampusByName() | Find a Campus by its Name |

##### CampusController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetAllCampuses() | Return list of all Campuses in the Database |
| 02 | GetCampusByName() | Find a Campus by its Name |

##### DistrictDAO Class

This class performs action to data (CRUD) in District Table in Database

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetAllDistricts() | Return list of all Districts in the Database |

##### DistrictController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetAllDistricts() | Return list of all Districts in the Database |

##### RoomTypeDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

##### RoomTypeController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

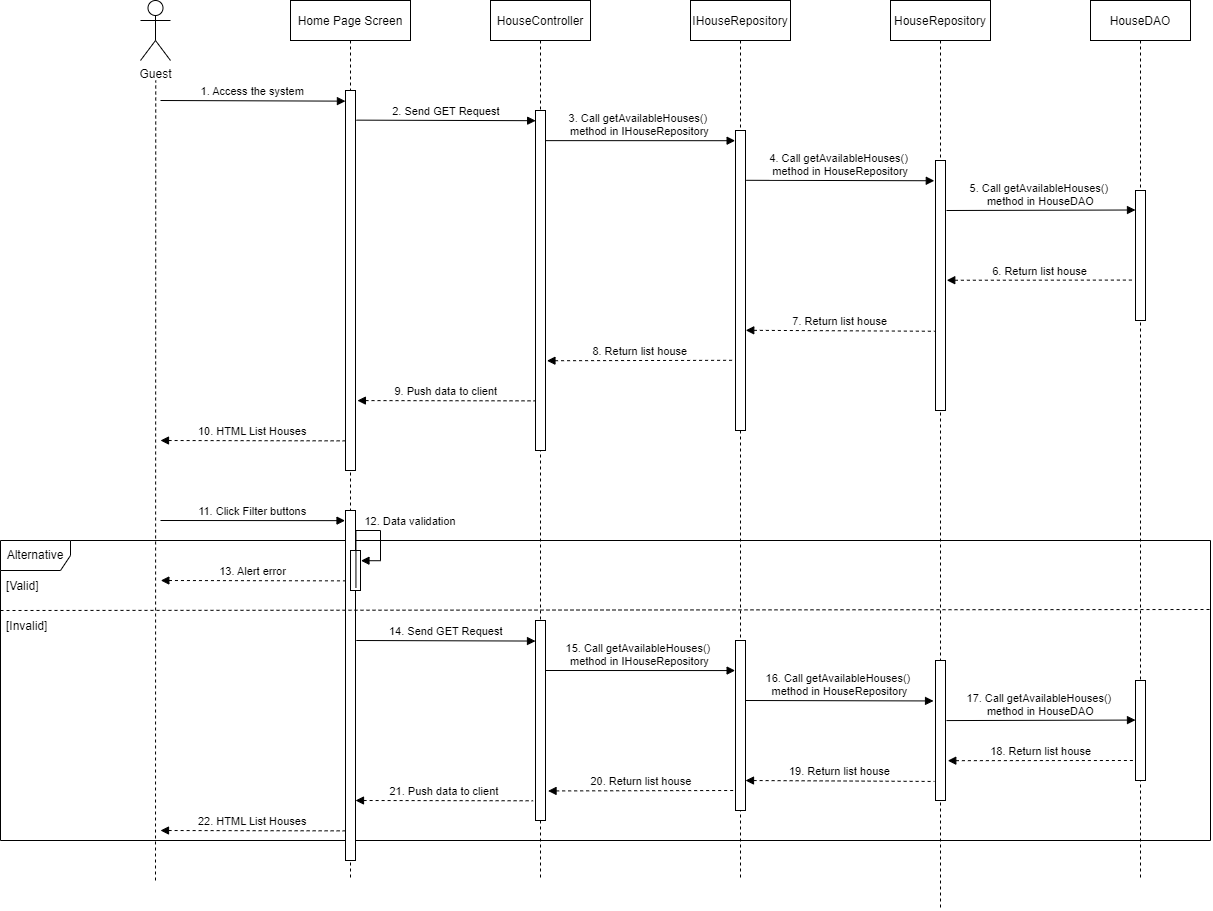
##### RoomDAO Class

This class performs action to data (CRUD) in Room Table in Database

**Class Methods**

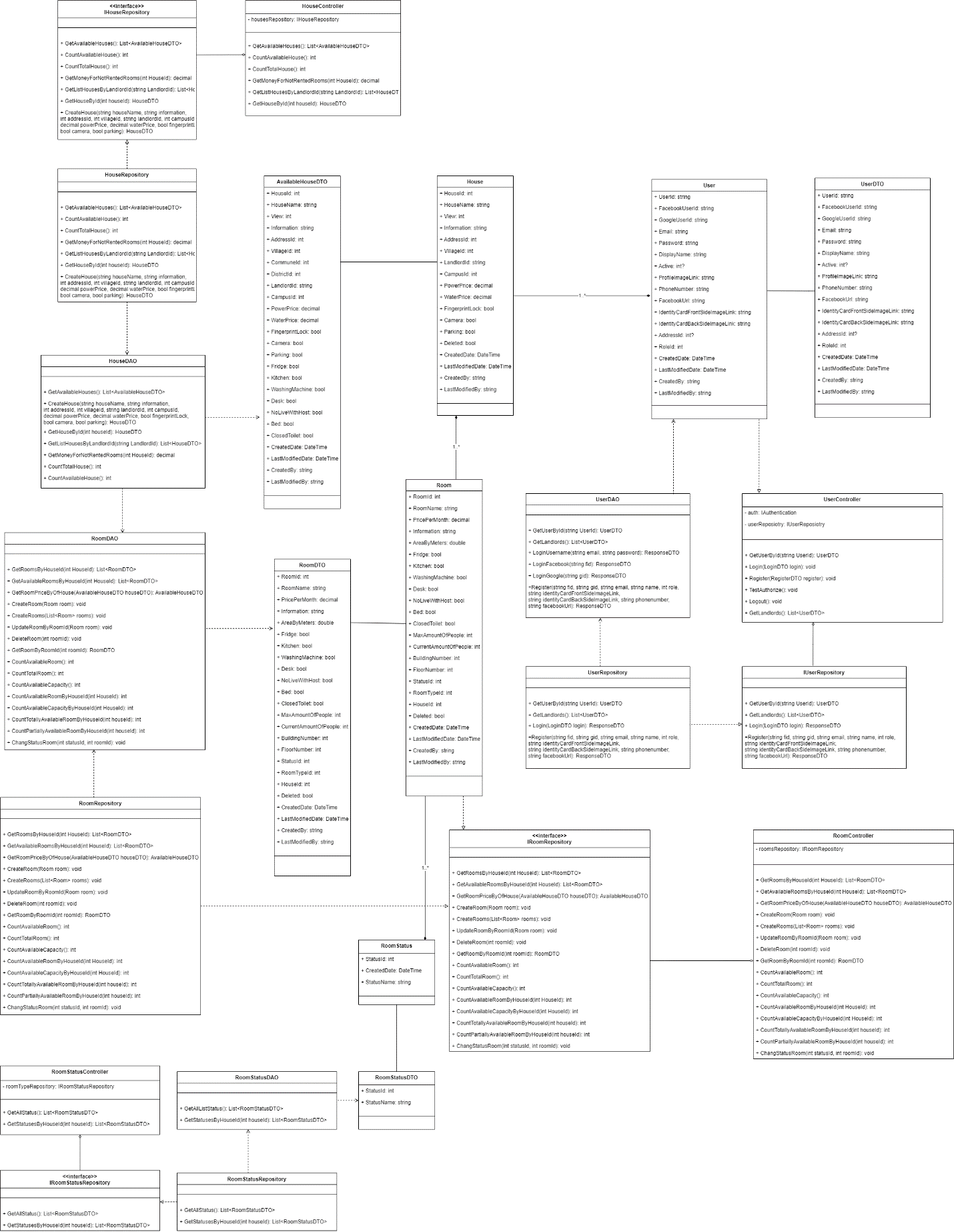
| **No** | **Method** | **Description** |
| --- | --- | --- |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

#### 3.1.3 Sequence Diagram



### 3.2 House Detail

#### 3.2.1 Class Diagram



#### 3.2.2 Class Specifications

##### HouseDAO Class

This class performs action to data (CRUD) in House Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### HouseController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### RoomStatusDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllListStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomStatusController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomDAO Class

This class performs action to data (CRUD) in Room Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### RoomController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### UserDAO Class

This class performs action to data (CRUD) in User Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

##### UserController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

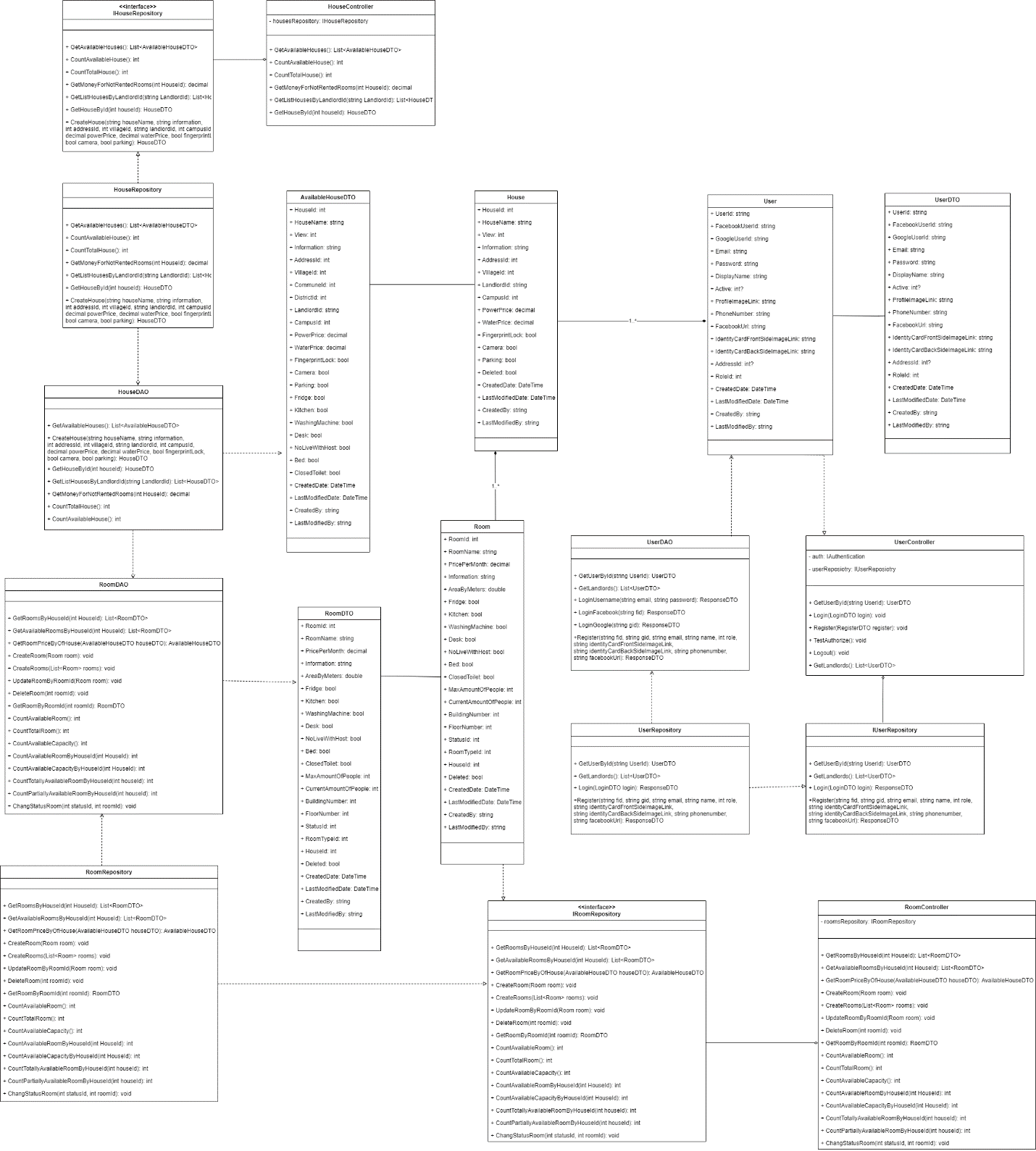
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

#### 3.2.3 Sequence Diagram



### 3.3 Room Detail

#### 3.3.1 Class Diagram



#### 3.3.2 Class Specifications

##### HouseDAO Class

This class performs action to data (CRUD) in House Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### HouseController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### RoomDAO Class

This class performs action to data (CRUD) in Room Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### RoomController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Find detail information of list Rooms by house Id |
| 02 | GetAvailableRoomsByHouseId () | Find detail information of list available Rooms by house Id |
| 03 | GetRoomsByRoomId () | Find detail information of list Room by its Id |
| 04 | CreateRoom () | Add a new Room into the Database |
| 05 | UpdateRoomByRoomId () | Update a Room into the Database |
| 06 | DeleteRoom () | Delete a Room |
| 07 | CountAvailableRoom () | Count number of available Rooms in the system |
| 08 | CountTotalRoom() | Count number of total Rooms in the system |
| 09 | CountAvailableCapacity() | Count number of capacity of available Rooms in the system |
| 10 | CountTotalCapacity() | Count number of capacity of Rooms in the system |
| 11 | CountTotallyAvailableCapacity() | Count number of total capacity of available Rooms in the system |
| 12 | CountTotallyAvailableRoomByHouseId() | Count number of total capacity of available Rooms in the system by house id |
| 13 | CountPartiallyAvailableRoomByHouseId() | Count number of total partially of available Rooms in the system by house id |
| 14 | CountAvailableCapacityByHouseId() | Count number of total available capacity of Rooms in the system by house id |
| 15 | ChangeStatusRoom() | Update a Room status into the Database |

##### UserDAO Class

This class performs action to data (CRUD) in User Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

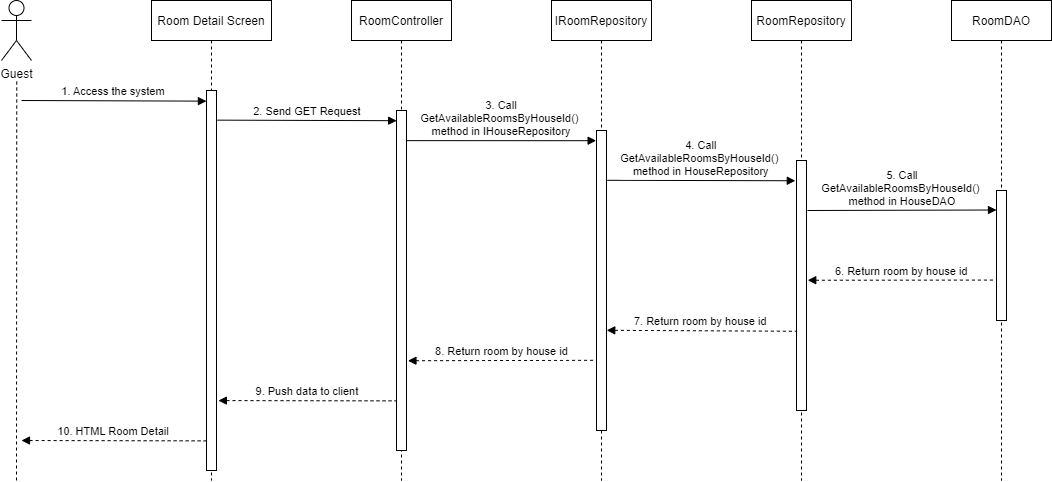
##### UserController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

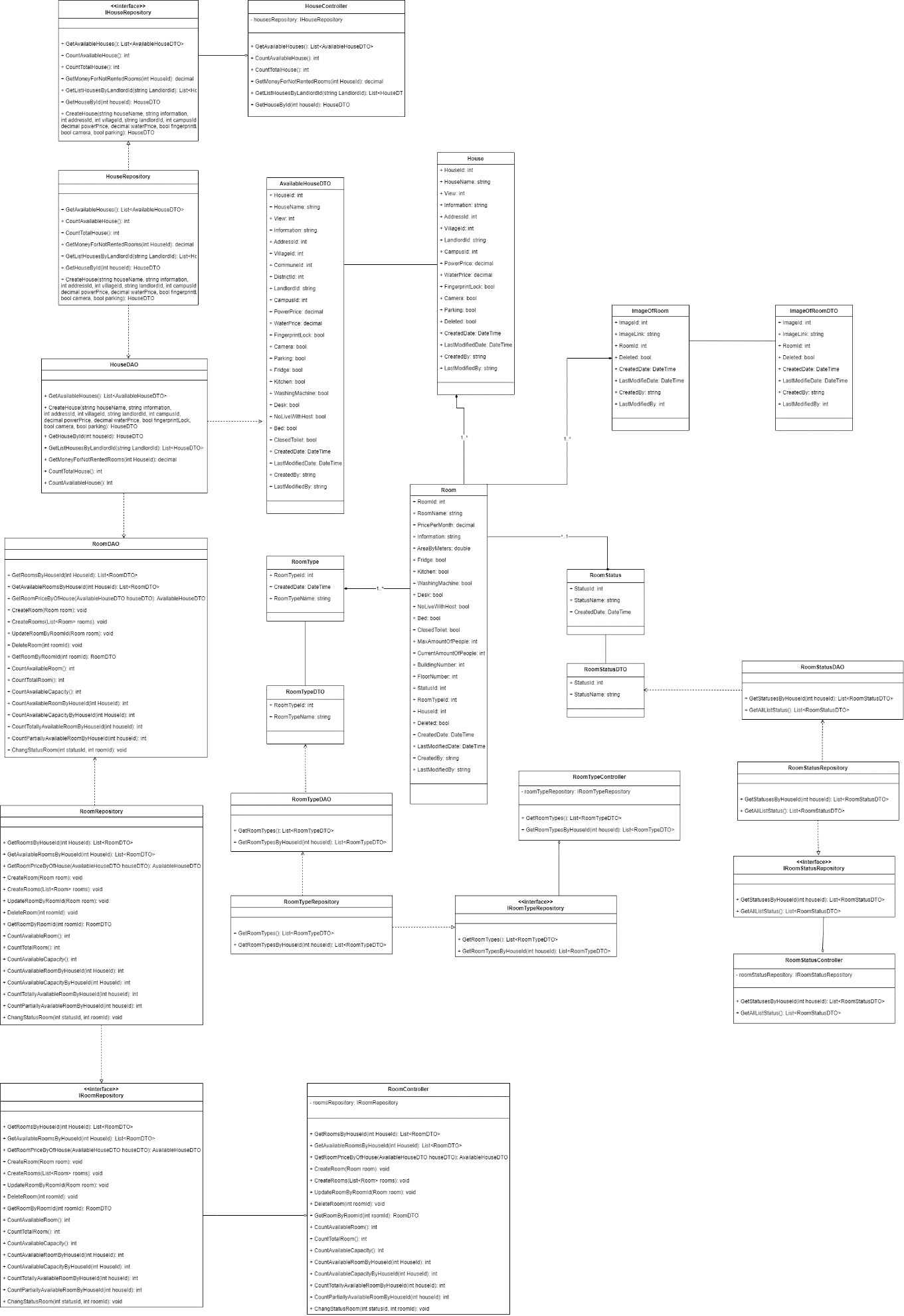
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

#### 3.3.3 Sequence Diagram



### 3.4 Landlord – List Room

#### 3.4.1 Class Diagram



#### 3.4.2 Class Specifications

##### HouseDAO Class

This class performs action to data (CRUD) in House Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### HouseController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### RoomDAO Class

This class performs action to data (CRUD) in Room Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### RoomController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### RoomStatusDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllListStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomStatusController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomTypeDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

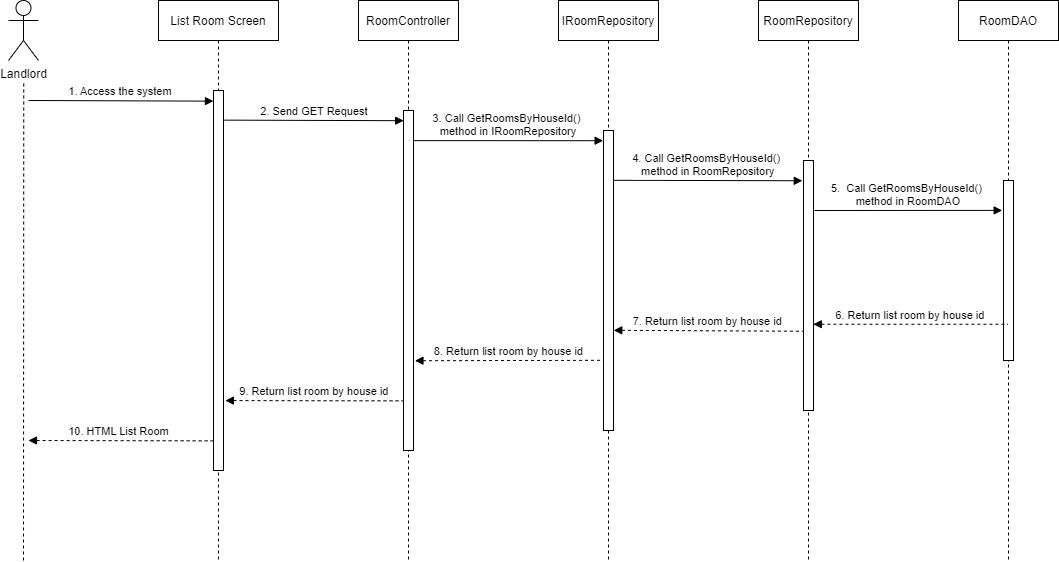
##### RoomTypeController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

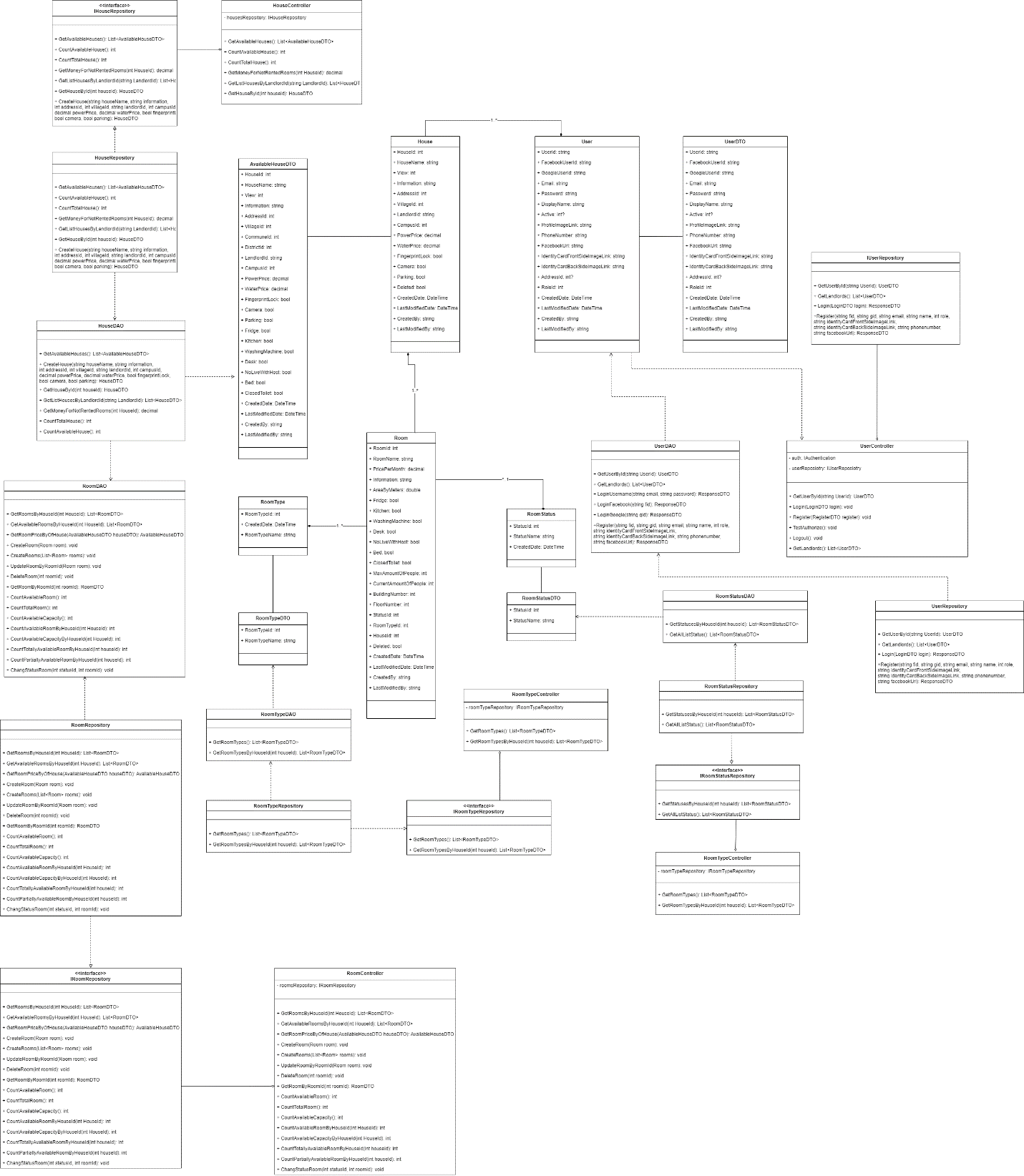
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

#### 3.4.3 Sequence Diagram



### 3.5 Landlord – Create Room

#### 3.5.1 Class Diagram



#### 3.5.2 Class Specifications

##### HouseDAO Class

This class performs action to data (CRUD) in House Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### HouseController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### RoomDAO Class

This class performs action to data (CRUD) in Room Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### RoomController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### UserDAO Class

This class performs action to data (CRUD) in User Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

##### UserController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

##### RoomStatusDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllListStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomStatusController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomTypeDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

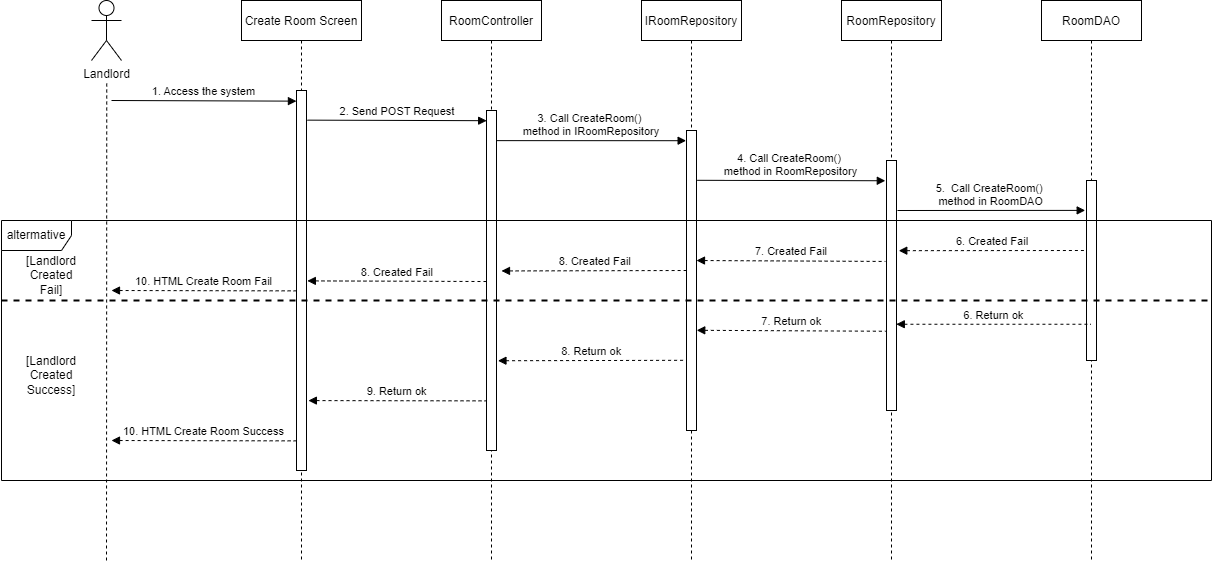
##### RoomTypeController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

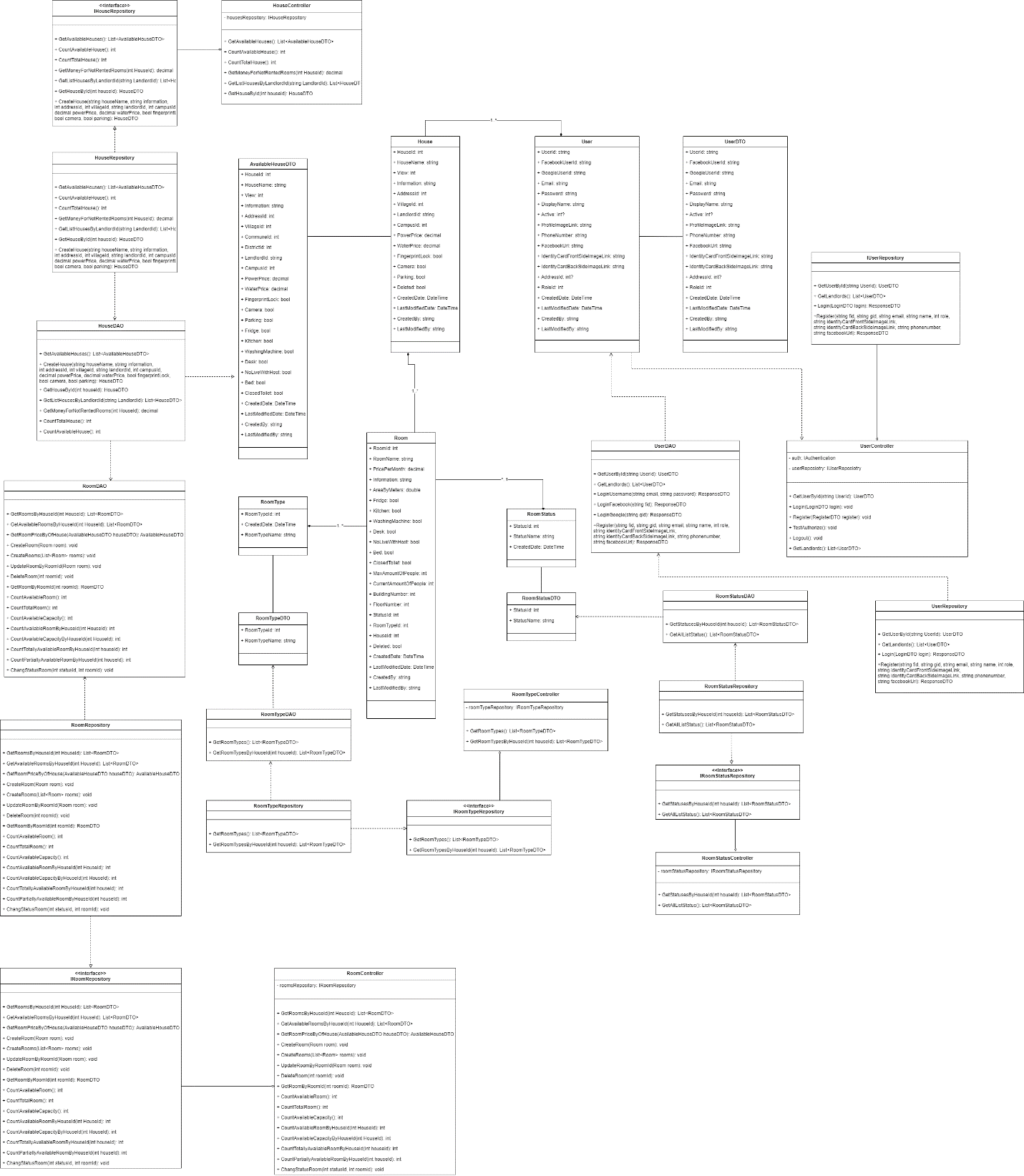
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

#### 3.5.3 Sequence Diagram



### 3.6 Landlord – Update Room

#### 3.6.1 Class Diagram



#### 3.6.2 Class Specifications

##### HouseDAO Class

This class performs action to data (CRUD) in House Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### HouseController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### RoomDAO Class

This class performs action to data (CRUD) in Room Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### RoomController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### UserDAO Class

This class performs action to data (CRUD) in User Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

##### UserController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

##### RoomStatusDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllListStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomStatusController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllStatus() | Return list of all RoomStatus in the Database |
| 02 | GetStatusesByHouseId() | Get all RoomStatus that this house has |

##### RoomTypeDAO Class

This class performs action to data (CRUD) in RoomType Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

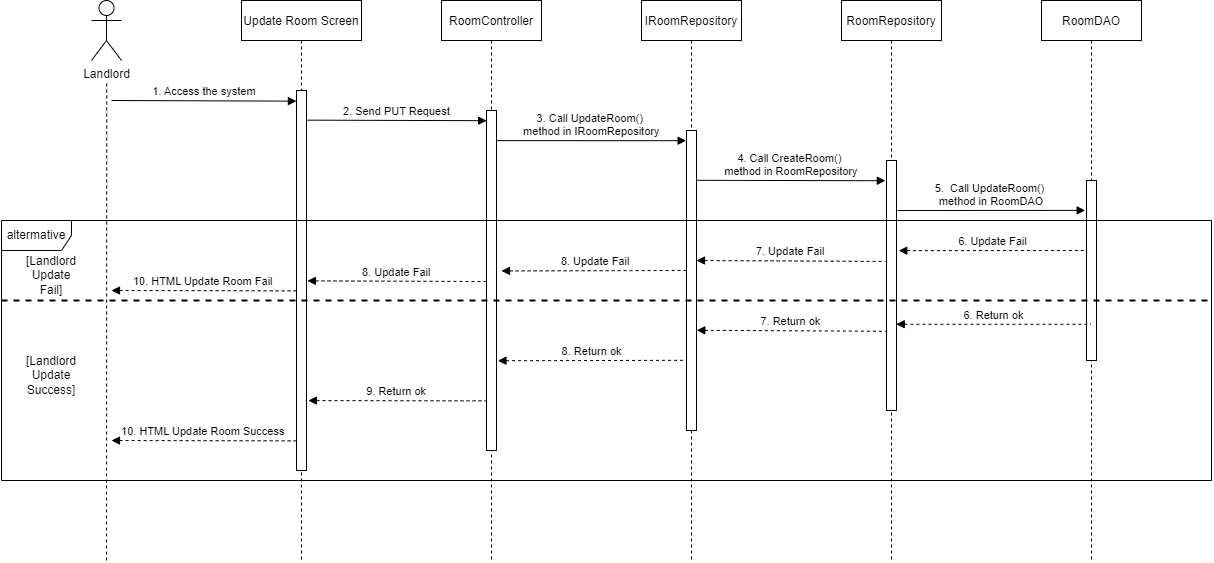
##### RoomTypeController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

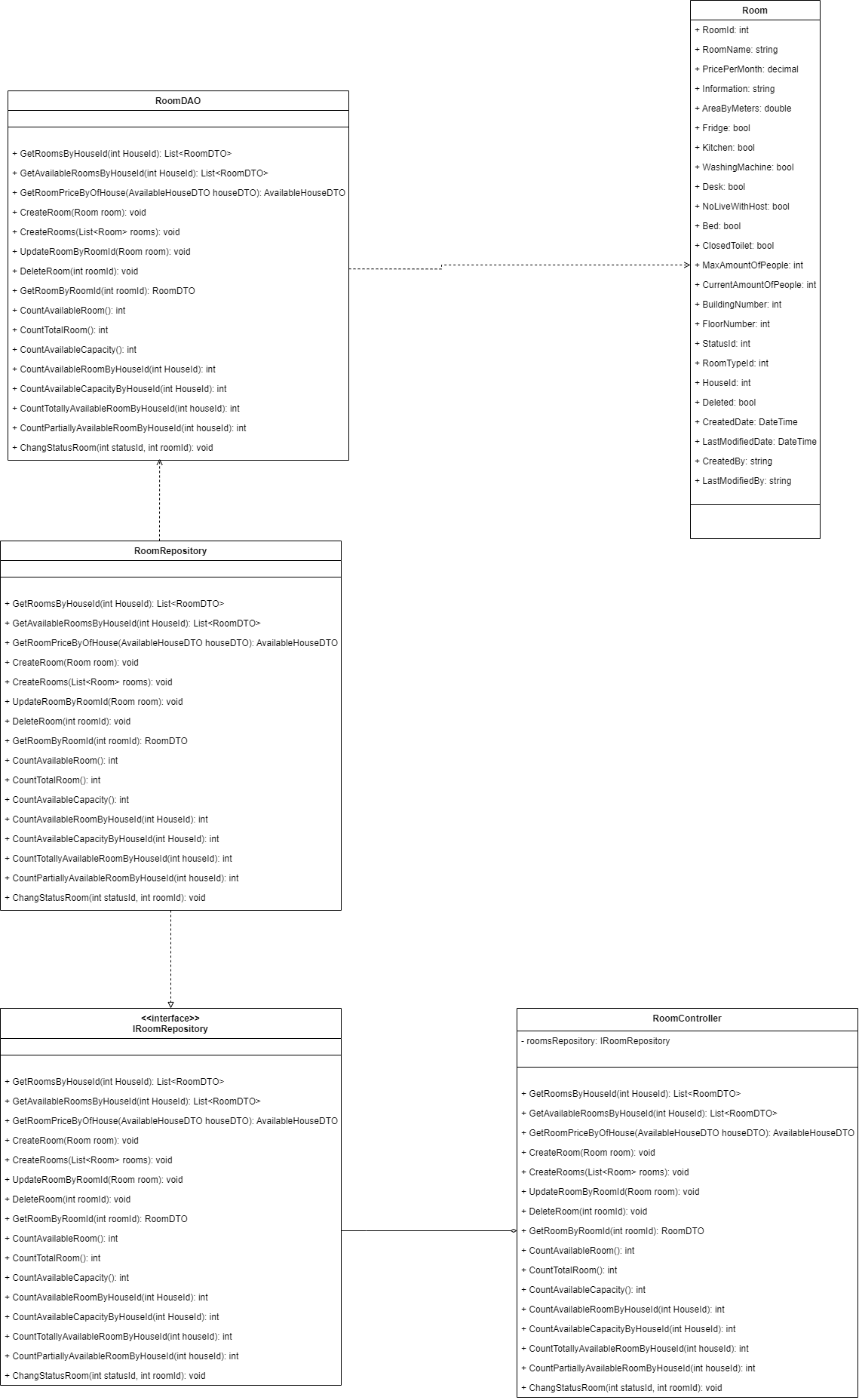
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomTypes() | Return list of all RoomTypes in the Database |
| 02 | GetRoomTypesByHouseId() | Get all roomTypes that this house has |

#### 3.6.3 Sequence Diagram



### 3.7 Landlord – Delete Room

#### 3.7.1 Class Diagram



#### 3.7.2 Class Specifications

##### RoomDAO Class

This class performs action to data (CRUD) in Room Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

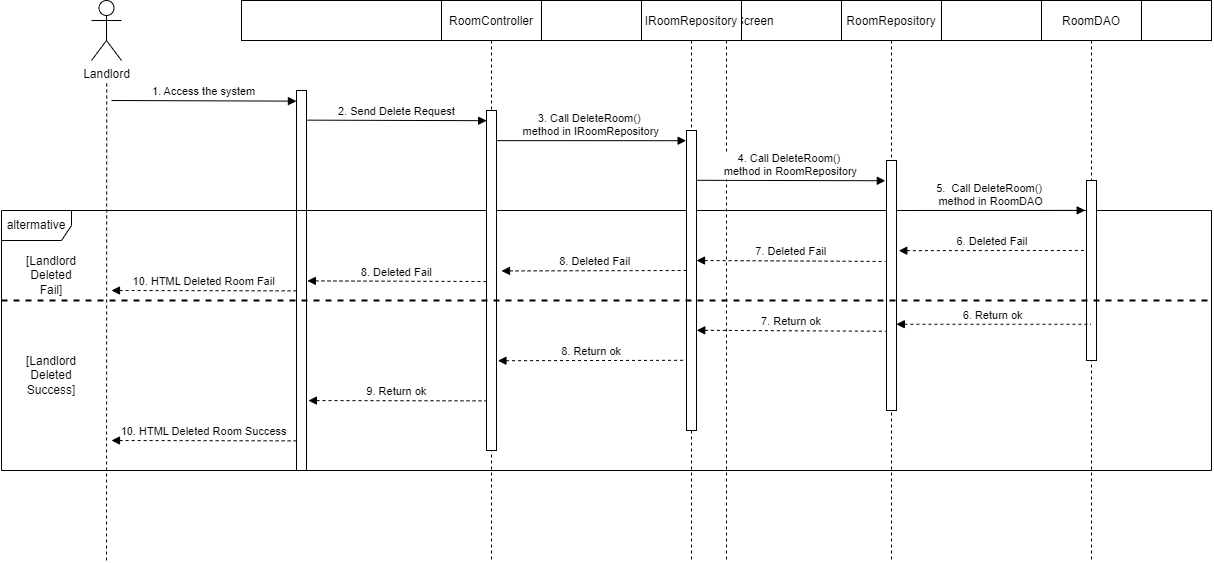
##### RoomController Class

This class takes data from DAO (Repository) and push it to API

**Class Methods**

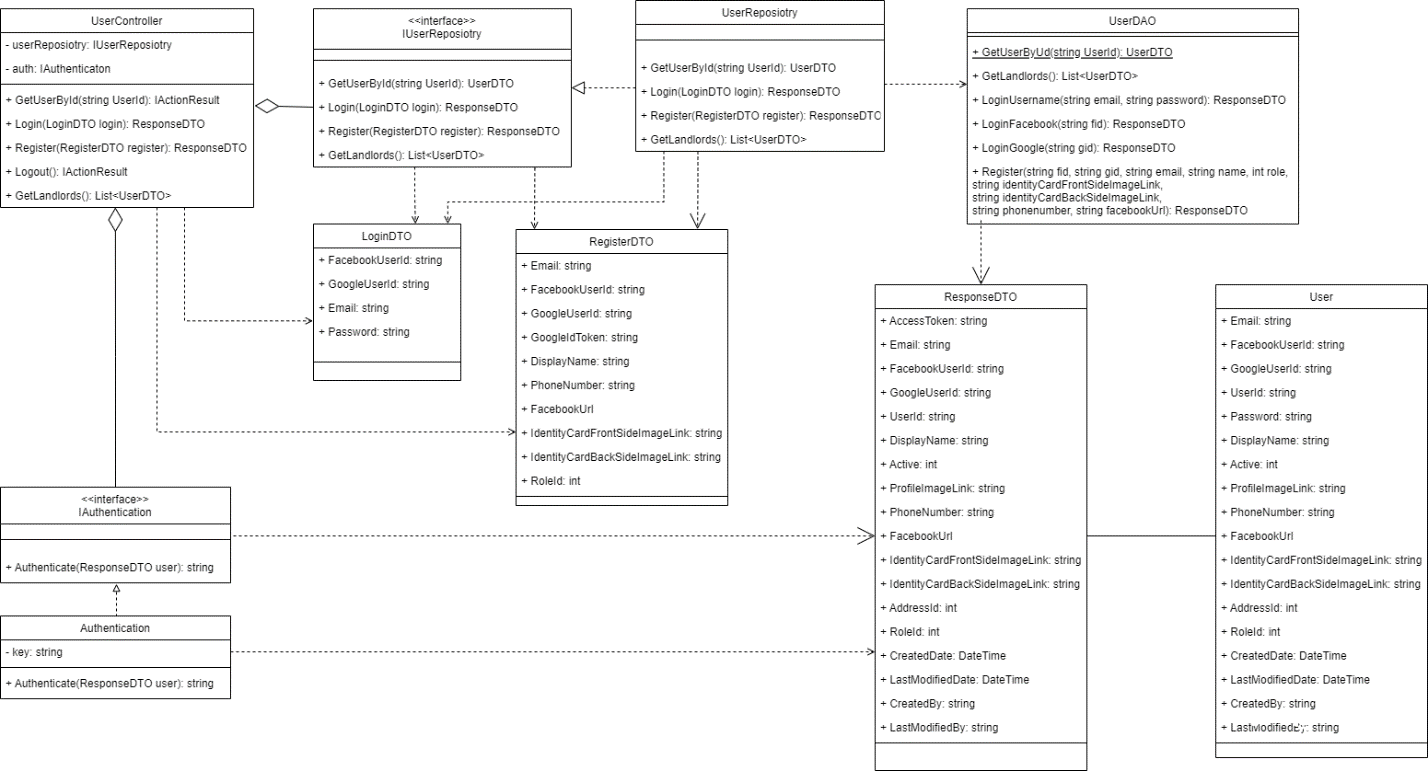
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

#### 3.7.3 Sequence Diagram



### 3.8 Login

#### 3.8.1 Class Diagram



#### 3.8.2 Class Specifications

##### UserController Class

This class takes request, get data from Repository and return Response

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

##### UserDAO Class

This class performs action to data (CRUD) in User Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | GetLandlords() | Get list of Landlords in the database |
| 03 | LoginUsername() | User login in the system |
| 04 | LoginFacebook() | User login with facebook in the system |
| 05 | LoginGoogle () | User login with google in the system |
| 06 | Register() | User create an account |

##### UserRepository Class

This class performs logic and get data from DAO to data (CRUD) in User Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetUserById () | Find detail information of User by its Id |
| 02 | Login() | Get user with login data provided |
| 06 | Register() | User create an account |

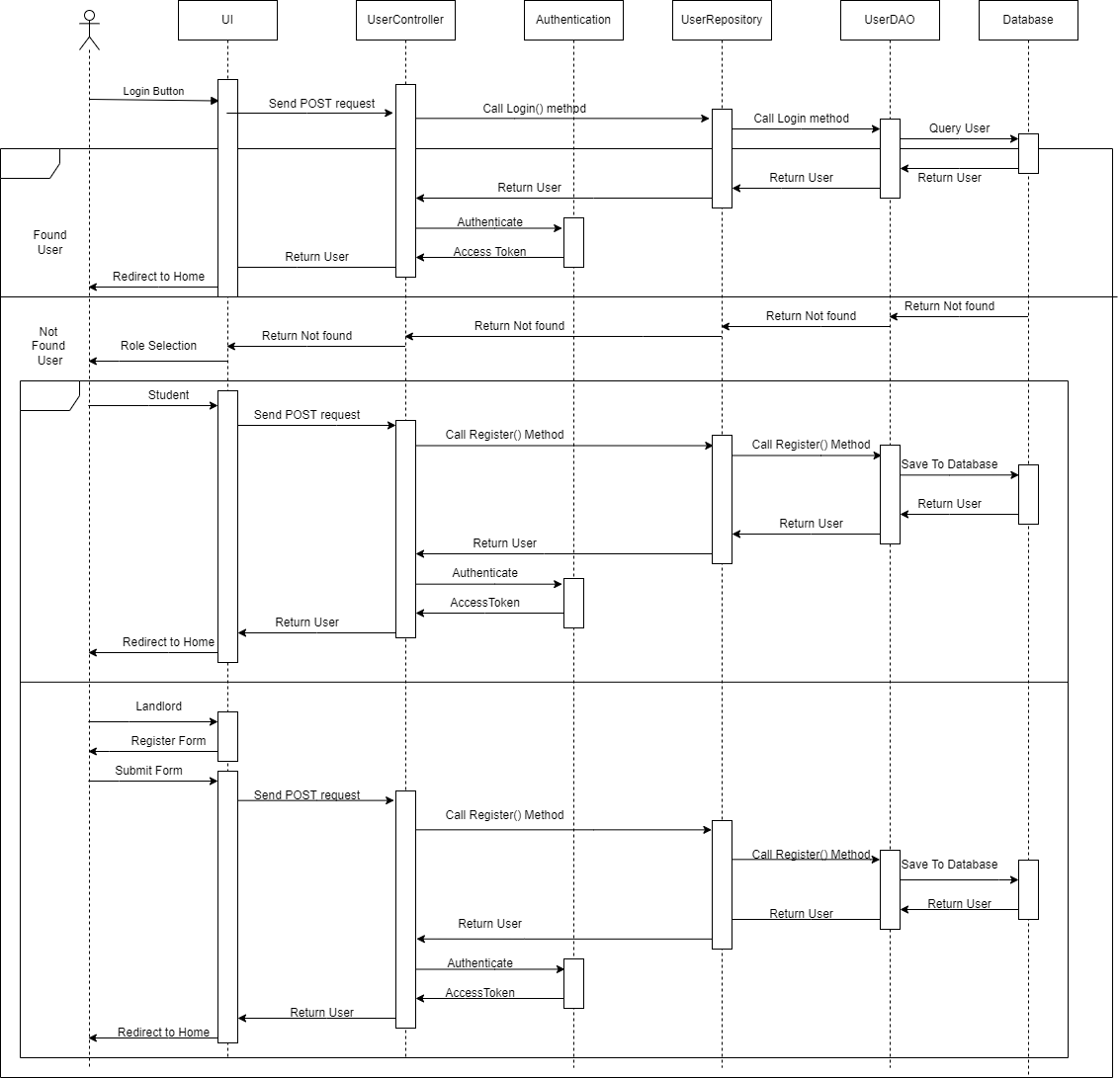
##### Authentication Class

This class create AccessToken for User

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | Authenticate() | Find detail information of User by its Id |

#### 3.8.3 Sequence Diagram



### 3.9 Landlord – Upload House Information

#### 3.9.1 Class Diagram



#### 3.9.2 Class Specifications

##### FileController Class

This class takes request, get data from Repository and return Response

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | DownloadFile() | Download Template file to User device |
| 02 | UploadFile() | Get data File from request |
| 03 | LoadDataAsync() | Create house, room data with Data from file |
| 04 | checkXlsxMimeType() | Check upload file mime type |

##### RoomDAO Class

This class class performs action to data (CRUD) in Room Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetRoomsByHouseId() | Return list of Rooms of 1 House |
| 02 | GetAvailableRoomsByHouseId() | Get list of available Rooms (partially or totally) of 1 House |
| 03 | GetRoomPriceByOfHouse() | Get Lowest room price and Highest room price for of 1 House (used in HouseDAO) |
| 04 | CreateRoom() | Add a new Room to the Database |
| 05 | CreateRooms() | Add a list of Rooms of 1 House to the Database |
| 06 | UpdateRoomByRoomId() | Update detail information of 1 Room |
| 07 | DeleteRoom() | Delete a Room from a system without removing its data from the Database |
| 08 | GetRoomByRoomId() | Find a Room by its Id |
| 09 | CountAvailableRoom() | Count the sum of all available Rooms (partially or totally) of all Houses in the system |
| 10 | CountTotalRoom() | Count the sum of all Rooms (available or not) of all Houses in the system |
| 11 | CountAvailableCapacity() | Count the sum of Capacity of all Rooms of all Houses in the system |
| 12 | CountAvailableRoomByHouseId() | Count the sum of all available Rooms (partially or totally) of 1 House |
| 13 | CountAvailableCapacityByHouseId() | Count the sum of Capacity of all Rooms of 1 House |
| 14 | CountTotallyAvailableRoomByHouseId() | Count the sum of all available Rooms (totally only) of 1 House |
| 15 | CountPartiallyAvailableRoomByHouseId() | Count the sum of all available Rooms (partially only) of 1 House |
| 16 | ChangeStatusRoom() | Change Status (Available or Occupied) of 1 Room |

##### HouseDAO Class

This class class performs action to data (CRUD) in House Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAvailableHouses() | Return list of all available Houses in the system (Houses that have available Rooms) with additional information to support filtering |
| 02 | CreateHouse() | Add a new House into the Database |
| 03 | GetHouseById() | Find detail information of a House by its Id |
| 04 | GetListHousesByLandlordId() | Get list of Houses of 1 Landlord for him to manage |
| 05 | GetMoneyForNotRentedRooms() | Get total amount of money of rooms that has not been rented (of 1 House) |
| 06 | CountTotalHouse() | Count total number of Houses in the system |
| 07 | CountAvailableHouse() | Count number of available Houses in the system |

##### AddressDAO Class

This class performs action to data (CRUD) in Address Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | CreateAddress() | Create Address record and save to database |

##### CampusDAO Class

This class performs action to data (Read) in Campus Table in Database

**Class Methods**

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetAllCampuses() | Get List of All Campuses |
| 02 | GetCampusByName() | Get Campus by Campus Name |
| 03 | GetCampusById() | Get Campus by Campus Id |

##### VillageDAO Class

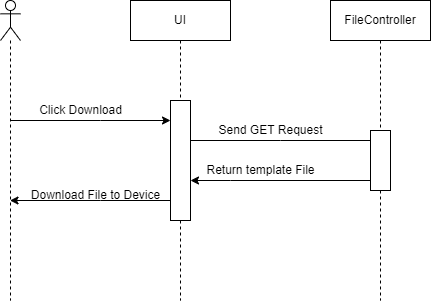
This class performs action to data (Read) in Village Table in Database

**Class Methods**

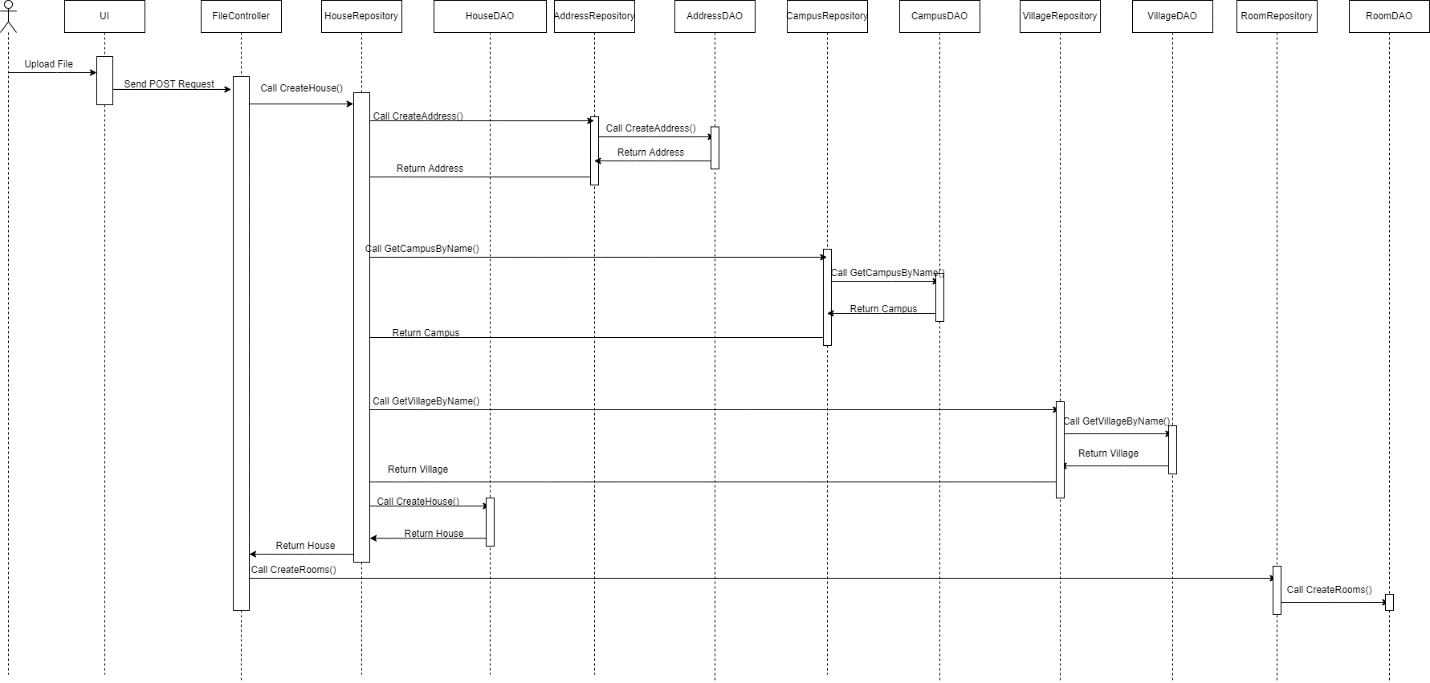
|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 01 | GetVillageByName() | Get Village by Village Name |
| 02 | GetVillageById() | Get Village by Village Id |

#### 3.9.3 Sequence Diagram

###### Download Template



###### Upload File



## 4. Database Tables

### 1. Campuses

*Store the information of all campus for users*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | CampusId | int |  |  | Yes | PK |  |
| 2 | CampusName | nvarchar | 100 |  | Yes |  |  |
| 3 | AddressId | int |  |  | Yes |  | To Table *Addresses* |
| 4 | CreatedDate | datetime |  |  | Yes |  |  |

### 2. UserRoles

*Store the information of all roles of users*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | RoleId | int |  |  | Yes | PK |  |
| 2 | RoleName | nvarchar | 100 |  | Yes |  |  |
| 3 | CreatedDate | datetime |  |  | Yes |  |  |

### 3. Addresses

*Store the information of addresses for Users (Landlords) and Houses*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | AddressId | int |  |  | Yes | PK |  |
| 2 | Addresses | nvarchar | 1000 |  | Yes |  |  |
| 3 | GoogleMapLocation | nvarchar | MAX |  |  |  | Location on Google Map |
| 4 | Deleted | bit |  |  | Yes |  |  |
| 5 | CreatedDate | datetime |  |  | Yes |  |  |
| 6 | LastModifiedDate | datetime |  |  |  |  |  |

### 4. Users

*Store the information of all users*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | UserId | nchar | 30 |  | Yes | PK | Include student id, host id and admission id;  Has specific convention to make |
| 2 | FacebookUserId | nchar | 300 |  |  |  | For people login with Facebook |
| 3 | GoogleUserId | nchar | 300 |  |  |  | For people login with Google |
| 4 | Email | nvarchar | 100 |  |  |  |  |
| 5 | Password | nvarchar | 100 |  |  |  | Hashed with PBKDF2 with SHA-512 |
| 6 | DisplayName | nvarchar | 500 |  |  |  | For Landlord and Staff |
| 7 | Active | int |  |  | Yes |  |  |
| 9 | ProfileImageLink | nvarchar | 500 |  |  |  | For Landlord and Staff |
| 10 | PhoneNumber | nvarchar | 50 |  |  |  | Only for Landlord |
| 11 | FacebookURL | nvarchar | 300 |  |  |  | Only for Landlord |
| 12 | IdentityCardFrontSideImageLink | nvarchar | 500 |  |  |  | Only for Landlord |
| 13 | IdentityCardBackSideImageLink | nvarchar | 500 |  |  |  | Only for Landlord |
| 14 | AddressId | int |  |  |  |  | Only for Landlord;  To Table *Addresses* |
| 15 | RoleId | int |  |  | Yes | FK | To Table *Roles* |
| 16 | CreatedDate | datetime |  |  | Yes |  |  |
| 17 | LastModifiedDate | datetime |  |  |  |  |  |
| 18 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 19 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |

### 5. Houses

*Store the information of all houses*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | HouseId | int |  |  | Yes | PK |  |
| 2 | HouseName | nvarchar | 100 |  | Yes |  |  |
| 3 | Information | nvarchar | MAX |  |  |  |  |
| 4 | View | int |  |  |  |  |  |
| 5 | AddressId | int |  |  | Yes |  | To Table *Addresses* |
| 6 | VillageId | int |  |  | Yes | FK | To Table *Villages* |
| 7 | LandlordId | nchar | 30 |  | Yes | FK | To Table *Users* |
| 8 | CampusId |  |  |  |  |  |  |
| 9 | PowerPrice | money |  |  | Yes |  |  |
| 10 | WaterPrice | money |  |  | Yes |  |  |
| 11 | FingerprintLock | bit |  |  |  |  |  |
| 12 | Camera | bit |  |  |  |  |  |
| 13 | Parking | bit |  |  |  |  |  |
| 14 | Deleted | bit |  |  | Yes |  |  |
| 15 | CreatedDate | datetime |  |  | Yes |  |  |
| 16 | LastModifiedDate | datetime |  |  |  |  |  |
| 17 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 18 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |

### 6. Rooms

*Store the information of all rooms of houses*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | RoomId | int |  |  | Yes | PK |  |
| 2 | RoomName | nvarchar | 50 |  |  |  |  |
| 3 | PricePerMonth | money |  |  | Yes |  |  |
| 6 | Information | nvarchar | MAX |  |  |  |  |
| 7 | AreaByMeters | float |  |  |  |  |  |
| 8 | Fridge | bit |  |  |  |  |  |
| 9 | Bed | bit |  |  |  |  |  |
| 10 | WashingMachine | bit |  |  |  |  |  |
| 11 | Desk | bit |  |  |  |  |  |
| 12 | Kitchen | bit |  |  |  |  |  |
| 13 | ClosedToilet | bit |  |  |  |  |  |
| 14 | NoLiveWithLandlord | bit |  |  |  |  |  |
| 15 | MaxAmountOfPeople | int |  |  |  |  |  |
| 16 | CurrentAmountOfPeople | int |  |  |  |  |  |
| 17 | BuildingNumber | int |  |  |  |  |  |
| 18 | FloorNumber | int |  |  |  |  |  |
| 19 | StatusId | int |  |  | Yes | FK | To Table *Statuses* |
| 20 | RoomTypeId | int |  |  | Yes | FK | To Table *RoomTypes* |
| 21 | HouseId | int |  |  | Yes | FK | To Table *Houses* |
| 22 | CampusId | int |  |  | Yes | FK | To Table *Campuses* |
| 23 | CreatedDate | datetime |  |  | Yes |  |  |
| 24 | LastModifiedDate | datetime |  |  |  |  |  |
| 25 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 26 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |

### 7. Rates

*Store the information of all rates and comments of users*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | RateId | int |  |  | Yes | PK |  |
| 2 | Star | int |  |  | Yes |  |  |
| 3 | Comment | nvarchar | MAX |  |  |  |  |
| 4 | LandlordReply | nvarchar | MAX |  |  |  |  |
| 5 | HouseId | int |  |  | Yes | FK | To Table *Houses* |
| 6 | StudentId | nchar | 30 |  | Yes | FK | To Table *Users* |
| 7 | CreatedDate | datetime |  |  | Yes |  |  |
| 8 | LastModifiedDate | datetime |  |  |  |  |  |
| 9 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 10 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |
| 11 | Deleted | bit |  |  | Yes |  |  |

### 8. ImagesOfHouse

*Store the information of all pictures of house*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | ImageId | int |  |  | Yes | PK |  |
| 2 | ImageLink | nvarchar | 500 |  | Yes |  |  |
| 3 | HouseId | int |  |  | Yes | FK | To Table *Houses* |
| 4 | CreatedDate | datetime |  |  | Yes |  |  |
| 5 | LastModifiedDate | datetime |  |  |  |  |  |
| 6 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 7 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |
| 8 | Deleted | bit |  |  | Yes |  |  |

### 9. ImagesOfRoom

*Store the information of all pictures of room*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | ID | int |  |  | Yes | PK |  |
| 2 | ImageLink | nvarchar | 500 |  | Yes |  |  |
| 3 | RoomId | int |  |  | Yes | FK | To Table *Rooms* |
| 4 | CreatedDate | datetime |  |  | Yes |  |  |
| 5 | LastModifiedDate | datetime |  |  |  |  |  |
| 6 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 7 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |
| 8 | Deleted | bit |  |  | Yes |  |  |

### 10. Districts

*Store the information of all districts (for identifying house address)*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | DistrictId | int |  |  | Yes | PK |  |
| 2 | DistrictName | nvarchar | 100 |  | Yes |  |  |
| 3 | CreatedDate | datetime |  |  | Yes |  |  |

### 11. Communes

*Store the information of all communes (for identifying house address)*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | CommuneId | int |  |  | Yes | PK |  |
| 2 | CommunetName | nvarchar | 100 |  | Yes |  |  |
| 3 | DistrictId | int |  |  | Yes | FK | To Table *Districts* |
| 4 | CreatedDate | datetime |  |  | Yes |  |  |

### 12. Villages

*Store the information of all villages (for identifying house address)*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | VillageId | int |  |  | Yes | PK |  |
| 2 | VillageName | nvarchar | 100 |  | Yes |  |  |
| 3 | CommuneId | int |  |  | Yes | FK | To Table *Communes* |
| 4 | CreatedDate | datetime |  |  | Yes |  |  |

### 13. RoomStatuses

*Store the information of all status of a Room*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | StatusId | int |  |  | Yes | PK |  |
| 2 | StatusName | nvarchar | 300 |  | Yes |  |  |
| 3 | CreatedDate | datetime |  |  | Yes |  |  |

### 14. RoomTypes

*Store the information of all types of a Room*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | RoomTypeId | int |  |  | Yes | PK |  |
| 2 | RoomTypeName | nvarchar | 300 |  | Yes |  |  |
| 3 | CreatedDate | datetime |  |  | Yes |  |  |

### 15. Reports

*Store the information of all Reports of Students to Houses*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | ReportId | int |  |  | Yes | PK |  |
| 2 | ReportContent | nvarchar | MAX |  | Yes |  |  |
| 3 | StudentId | nchar | 30 |  | Yes | FK | To Table *Users* |
| 4 | HouseId | int |  |  | Yes | FK | To Table *Houses* |
| 5 | CreatedDate | datetime |  |  | Yes |  |  |
| 6 | LastModifiedDate | datetime |  |  |  |  |  |
| 7 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 8 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |
| 9 | Deleted | bit |  |  | Yes |  |  |

### 16. RoomHistories

*Store the information of all records of Room usage for Landlords to manage if they want to*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | RoomHistoryId | int |  |  | Yes | PK |  |
| 2 | CustomerName | nvarchar | 800 |  | Yes |  | Name of the person stayed in the Room |
| 3 | RoomId | int |  |  | Yes | FK | To Table *Rooms* |
| 4 | CreatedDate | datetime |  |  | Yes |  |  |
| 5 | LastModifiedDate | datetime |  |  |  |  |  |
| 6 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 7 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |
| 8 | Deleted | bit |  |  | Yes |  |  |

### 17. Issues

*Store the information of all records of Room usage for Landlords to manage if they want to*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | IssueId | int |  |  | Yes | PK |  |
| 2 | Description | nvarchar | 800 |  | Yes |  |  |
| 3 | RoomId | int |  |  | Yes | FK | To Table *Rooms* |
| 4 | Deleted | bit |  |  | Yes |  |  |
| 5 | CreatedDate | datetime |  |  | Yes |  |  |
| 6 | LastModifiedDate | datetime |  |  |  |  |  |
| 7 | CreatedBy | nchar | 30 |  | Yes | FK | To Table *Users* |
| 8 | LastModifiedBy | nchar | 30 |  |  | FK | To Table *Users* |

### 18. Orders

*Store the information of all records of Order usage for Staffs to manage if they want to*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | OrderId | int |  |  | Yes | PK |  |
| 2 | StudentId | nchar | 30 |  | Yes | FK | To Table *Users* |
| 3 | StudentName | nvarchar | 100 |  | Yes |  |  |
| 4 | PhoneNumber | nvarchar | 50 |  | Yes |  |  |
| 5 | Email | nvarchar | 100 |  | Yes |  |  |
| 6 | OrderContent | nvarchar |  |  |  |  |  |
| 7 | StatusId | int |  |  | Yes | FK | To Table *OrderStatuses* |
| 8 | OrderedDate | datetime |  |  |  |  |  |
| 9 | SolvedDate | datetime |  |  |  |  |  |

### 19. OrderStatuses

*Store the information of all status of an Order*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **Field name** | **Type** | **Size** | **Unique** | **Not Null** | **PK/FK** | **Notes** |
| 1 | StatusId | int |  |  | Yes | PK |  |
| 2 | StatusName | nvarchar | 300 |  | Yes |  |  |
| 3 | CreatedDate | datetime |  |  | Yes |  |  |

# V. Software Testing Documentation

*[Provide final software testing information follow the template as part II in the Report #5]*

## 1. Scope of Testing

*[Describe the scopes of the test. Those include the target-of-test’s features, functions, and non-functional requirements that will or will not be tested.*

*Describe the stages/levels of testing that would be applied to your project­ - Unit, Integration, or System test. Each includes the in-charge, inputs/time, focuses, acceptance criteria.*

*List any constraints or assumptions made during the development of this document that may impact the design, development or implementation of testing]*

## 2. Test Strategy

*[List out and describe all testing types (you can refer the test types listed below or any other test types to selected the suitable ones for the project; for each selected test types you need to provide the following information: test objective, technique, completion criteria, etc.), test levels that those test types would be performed, & the details of test supporting tools would be used in the project]*

### 2.1 Testing Types

*[List out and describe here the testing types which you would apply in your project. You need to mention following information for each type of testing: objective, technique, completion criteria]*

### 2.2 Test Levels

*<List out and describe here the testing levels which you would execute in your project. Besides, clearly state the test types which are performed in each test level that you plan for this project>*

### 2.3 Supporting Tools

*<List of the test supporting tools which will be employed for this project>*

## 3. Test Plan

### 3.1 Human Resources

*[List and provide the details on roles and responsibilities of the project members who would involve in testing works]*

### 3.2 Test Environment

*[List and provide the details about the tools (software, hardware, infrastructure) which the project would use for testing]*

### 3.3 Test Milestones

*[Separate test milestones, which should be identified to communicate project status accomplishments]*

## 4. Test Cases

*[Prepare the details on the test cases following the provided template*

* *Unit Test Cases: Report5\_Unit Test.xls*
* *Other Test Cases (IT, ST, AT): Report5\_Test Report.xls]*

## 5. Test Reports

*[Provide the test result, statistics and the relevant test analysis for your testing in the project]*

# VI. Release Package & User Guides

*[Provide final software testing information follow the template as part II in the Report #6]*

## 1. Deliverable Package

*[The section will list all source programs, scripts, documents with version number in this release. You can see the example following table for reference, can customize or delete if not using belong to each project characteristics]*

| **No.** | **Deliverable Item** | **Description** |
| --- | --- | --- |
| 1 | Schedule/Task Tracking |  |
| 2 | Project Backlog |  |
| 3 | Source Codes |  |
| 4 | Database Script(s) |  |
| 5 | Final Report Document |  |
| 6 | Test Cases Document |  |
| 7 | Defects List |  |
| 8 | Issues List |  |
| 9 | Slide |  |

## 2. Installation Guides

### 2.1 System Requirements

*[Define any system requirements necessary to support the application, including the software and relevant configurations]*

### 2.2 Installation Instruction

*[Includes installation instructions and configuration guidelines]*

## 3. User Manual

### 3.1 Overview

*[Descript the overview of the application and if could, insert the features workflow to help user has the overview of all the features in this application]*

### 3.2 Workflow 1

*[Describe the purpose of this workflow, draw workflow diagram and other relevant diagrams]*

*[Describe the detailed guides for the workflow by providing the brief description, step by step guides (attached with user interface) of how to use that function]*

### 3.3 Workflow 2

…