ROOMIT

Application Design Document

Eylon Sade

Ofir Pominovsky

Noa Magrisso

A couple of women holding a glass of wine

Description automatically generated with low confidence

Abstract

Our project is a Matching system between roommates who live in apartments and looking for other roommates to join them and roommates who don’t live in apartments and looking for roommates who already leave in.

The system's target audience is all the people who share an apartment with others.

Our system will get the relevant information from the users who are roommates in each of the two positions.

After processing the data, the system will issue Matching reports: A Roommate-Matches report for the roommate who already lives in apartment and an Apartments-Matches report for the roommate who doesn’t leave in apartment.

- For the final project, our data is from questionnaires for students who have already experienced the roommate search stage or are experiencing now.

Table of Contents

[Chapter 1 – Use Cases 3](#_Toc135592327)

[Chapter 2 – System Architecture 3](#_Toc135592328)

[Chapter 3 – Data Model 3](#_Toc135592329)

[3.1 + 3.2 Description of Data Objects & Data Objects Relationships 3](#_Toc135592330)

[3.3 Databases 4](#_Toc135592331)

[Chapter 4 – Behavioral Analysis 10](#_Toc135592332)

[4.1 Sequence Diagrams 10](#_Toc135592333)

[4.2 Events 10](#_Toc135592334)

[4.3 States 10](#_Toc135592335)

[Chapter 5 – Object-Oriented Analysis 10](#_Toc135592336)

[5.1 Class Diagram 10](#_Toc135592337)

[5.2 Class Description 11](#_Toc135592338)

[5.3 Packages 12](#_Toc135592339)

[5.4 Testing 13](#_Toc135592340)

[Chapter 6 - User Interface Draft 25](#_Toc135592341)

# 

# Chapter 1 – Use Cases

For the sake of simplification, we will define the following two concepts as follows:

**Roommate A =** A roommate who is looking for a roommate to insert into the apartment who lives in.

**Roommate B =** A roommate who is looking to join an apartment that already has a tenant.

**Roommate A –**

1. *Roommate A inserts apartment description.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in.

Post-conditions: An apartment object was created and added to the Apartments table.

1. *Roommate A requests a matching report.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in, the apartment object exists, and at least one roommate is interested to join his apartment.

Post-conditions: A matching report was created.

1. *Roommate A asks to see Roommate B’s profile.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in, and the Roommate B object which Roommate A is ask for is exist and is active.

Post-conditions: Roommate B’s profile displays.

1. *Roommate A inserts new requirement.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in.

Post-conditions: The new requirement is added to his profile.

1. *Roommate A registers the system and creates an apartment profile.*

Actors: Roommate A

Pre-conditions: Roommate A username does not exist, and he chooses valid username and password.

Post-conditions: Roommate A was created with the username and the password he chose.

1. *Roommate A creates an apartment profile.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in, and he has not already added his apartment to the system.

Post-conditions: His apartment profile with created.

1. *Roommate A logs-in to the system.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged out.

Post-conditions: The object of Roommate A is active and logged in.

1. *Roommate A sees all matching Roommates B.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in, and he matched with at least one Roommate B.

Post-conditions: Roommates B which Roommate A was matched with display.

1. *Roommate A sees all Roommates B who have already chatted with him.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in, and he has chatted with at least one Roommate B.

Post-conditions: Roommates B which Roommate A was chatted with display.

1. *Roommate A edits his profile (description, his picture, requirements, etc.)*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in.

Post-conditions: Roommate A’s profile was updated.

1. *Roommate A edits his apartment profile (description, images, the apartment offers, etc.)*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in, and his apartment profile was created by him.

Post-conditions: The apartment’s profile of Roommate A was updated.

1. *Roommate A deletes his apartment from the system.*

Actors: Roommate A

Pre-conditions: The object Roommate A is active and logged in, and his apartment’ profile was already created.

Post-conditions: The apartment’s profile of Roommate A was deleted.

1. *Roommate A deletes his profile from the system.*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in, and his apartment profile doesn’t exist.

Post-conditions: Roommate A profile was deleted and all Roommate B who have chatted with were announced.

1. *Roommate A logs-out from the system*

Actors: Roommate A

Pre-conditions: The object of Roommate A is active and logged in.

Post-conditions: Roommate A is logged out.

**Roommate B -**

1. *Roommate B fills out personal information.*

Actors: Roommate B.

Pre-conditions: The object of Roommate B is active and logged in.

Post-conditions: Roommate B’s personal information was filled out.

1. *Roommate B answers apartments’ characters.*

Actors: Roommate B

Pre-conditions: The object of Roommate B is active and logged in, and he finished filling out his personal information and defined himself as “Roommate B”.

Post-conditions: The apartments’ characters were filled by Roommate B.

1. *Roommate B requests a matching report.*

Actors: Roommate B

Pre-conditions: The object of Roommate B is active and logged in.

Post-conditions: The matching report of Roommate B is displayed.

1. *Roommate B asks to see apartment’s profile.*

Actors: Roommate B, Roommate A

Pre-conditions: The object of Roommate B is active and logged in, Roommate A is active, and his apartment profile exists. Roommate B gets from Roommate A the apartment’s detail.

Post-conditions: The apartment’s profile is displayed.

1. *Roommate B asks to see Roommate A’s profile.*

Actors: Roommate B

Pre-conditions: The object of Roommate B is active and logged in, and Roommate A is active.

Post-conditions: Roommate A’s profile is displayed.

1. *Roommate B registers the system.*

Actors: Roommate B

Pre-conditions: Roommate B is logged in.

Post-conditions: Roommate B’s profile was created.

1. *Roommate B logs-in to the system.*

Actors: Roommate B

Pre-conditions: The object of Roommate B is active and logged out.

Post-conditions: Roommate B is active and logged in.

1. *Roommate B edits his profile (description, his picture, requirements, etc.)*

Actors: Roommate B

Pre-conditions: The object of Roommate B is active and logged in.

Post-conditions: Roommate B’s profile was updated.

1. *Roommate B deletes his profile from the system.*

Actors: Roommate B

Pre-conditions: The object of Roommate B is active and logged in, and his apartment profile doesn’t exist.

Post-conditions: Roommate B profile was deleted and all Roommate A who have chatted with were announced.

1. *Roommate B logs out from the system.*

Actors: Roommate B.

Pre-conditions: The object of Roommate B is active and logged in.

Post-conditions: Roommate B is active and logged out.

**Admin –**

1. *Admin approves creating a Roommate A/B profile.*

Actors: Admin

Pre-conditions: The Roommate profile was done and is waiting for the admin approval.

Post-conditions: The Roommate profile was approved and was published.

1. *Admin approves creating an apartment of Roommate profile.*

Actors: Admin

Pre-conditions: Roommate is logged-in and active, and his apartment profile was done and is waiting for the admin approval.

Post-conditions: The Roommate’s apartment profile was approved and was published.

1. *Admin approves editing a Roommate A/B profile request.*

Actors: Admin

Pre-conditions: Roommate profile is logged-in and active, the Roommate profile editing was done and is waiting for the admin approval.

Post-conditions: The Roommate profile editing was approved, and it was updated.

1. *Admin approves editing an apartment of Roommate A profile request.*

Actors: Admin

Pre-conditions: Roommate A is logged-in and active, the Roommate’s apartment profile editing was done and is waiting for the admin approval.

Post-conditions: The Roommate’s apartment profile editing was approved, and it was updated.

1. *Admin approves deleting an apartment of Roommate A profile request.*

Actors: Admin

Pre-conditions: The Roommate who asks for it is active and logged in and asks for deleting his apartment profile.

Post-conditions: The Roommate’s apartment profile was deleted.

1. *Admin approves deleting a Roommate profile request.*

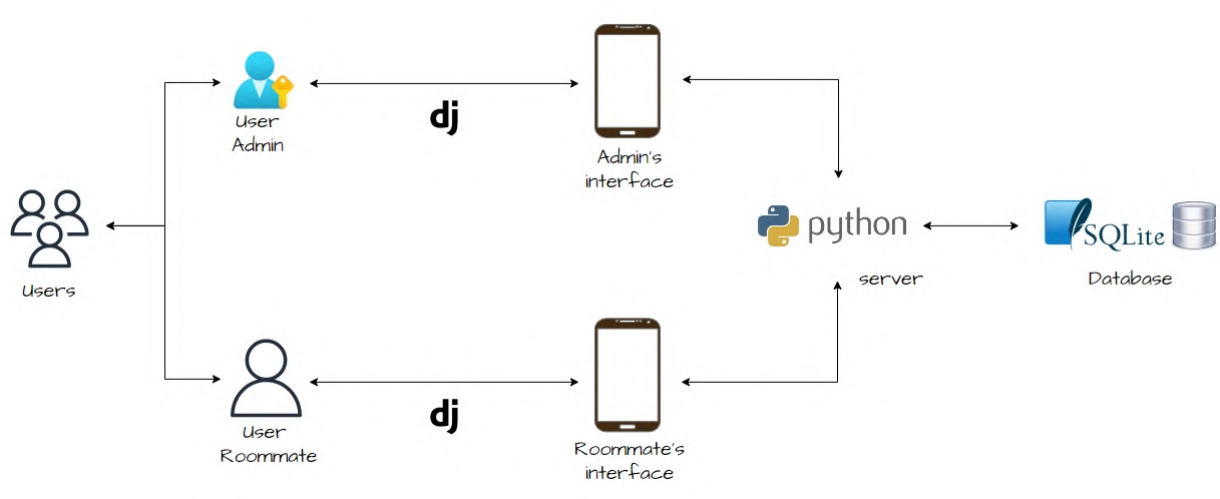
Actors: Admin

Pre-conditions: The Roommate who asks for it is active and logged in and asks for it deleting his profile.

Post-conditions: The Roommate’s profile was deleted.

# Chapter 2 – System Architecture

*Architecture Diagram*



This architecture diagram contains the main components as described below.

From left to right:

* There are users in the system, which are split into two types – Admin and Roommate.

The admin has all permissions, and few actions need its approval.

The roommate is the regular user, whom the application was created for. It is represent the both sides – the enter roommate and the insert roommate.

* The future interface will be a mobile application which we built by React Native, for now a temporary interface is presented built with HTML.
* All the users are using the web, so they can post/read any information by other roommates who are using this application.
* The database is relational tables, and the DBMS which we deal with is SQL Server named MySQL.

*Main components of our system architecture:*

Application Service:

Our application is running on **Windows**.  
Receives data from the user:

* Roommate A – Roommate personal details and apartment offers.
* Roommate B – Roommate personal details and apartment requirements.

The application stores the data in the database and accesses it as needed for preforming analyze and manipulations to export Matching-Reports for the users.

DBMS:

**MySQL**.

Store all relevant components attributes in the system – Users, Roommates A, Roommates B, Apartments, etc.

Backend Side:

Our algorithm and system were developed in **Python**. Export directly the information to the DB.

Frontend Side:

Mobile application developed in **React Native**. Each iOS/Android smartphone can use our application.

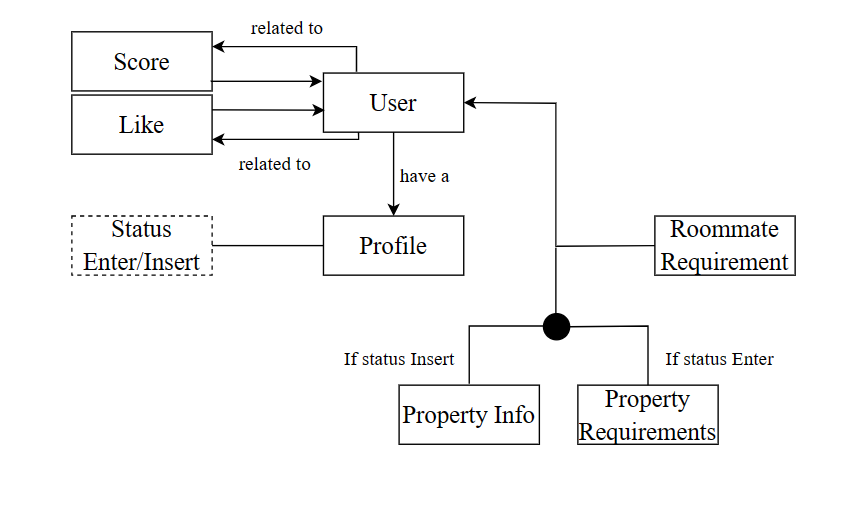
The Pipe Connecting the Different Sides (Web Framework):

We used **Django** for ORM (mapping objects to a relational database) and for handling HTTP requests from the frontend side to the backend side and vice versa.

# Chapter 3 – Data Model

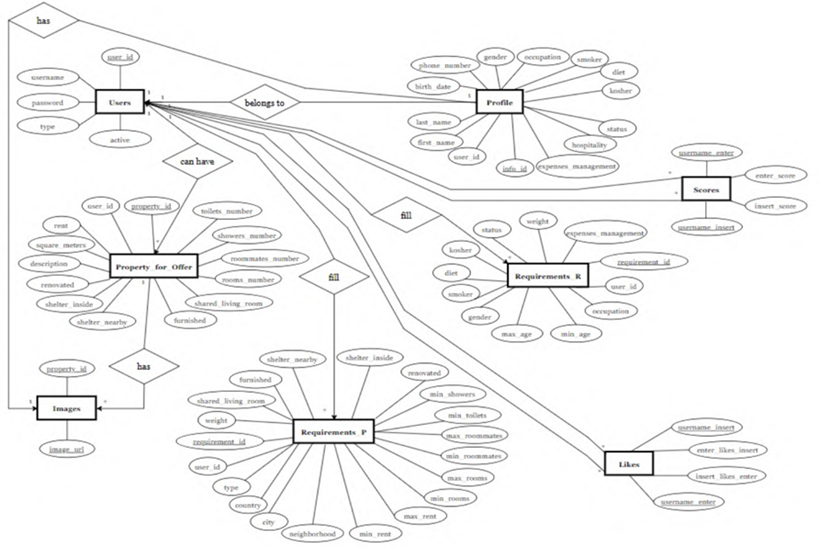
## 

## 3.1 + 3.2 Description of Data Objects & Data Objects Relationships



## 3.3 Databases

* *Entity-Relation Diagrams (ERD)*



These diagrams describe the data objects and the relationships among them.

*The data objects:*

**Users –**

This table contains the system users’ information, so that each record represents a unique user with its information for log-in.

Each record contains: the user’s ID (Primary Key), username, password, last login, is super user, email, is staff, is active, and date joined.

**Profile –**

This table contains the Roommates details, so that each record represents a Roommate, which is a user in the system that is looking for another roommate.

Each record contains: the profile’s ID (Primary Key), its user’s ID (Foreign Key), the user’s image, first name, last name, birthdate, phone number, gender, occupation, if he’s a smoker, diet, status, hospitality, kosher, expense management preference, about me and his status (interested to enter/to insert into an apartment).

**Property for Offer –**

This table contains the properties’ information, so that each record represents a property that a roommate is offering for a new roommate to enter.

Each record contains: the property’s ID (Primary Key), its roommate’s id (the roommate who published the property), its rent, its square meters, its description, a Boolean answer if it is renovated, a Boolean answer if there is a shelter inside, a Boolean answer if there is a shelter nearby, a Boolean answer if it is furnished, a Boolean answer if the living room is shared, its rooms number, its roommates number, its showers number and its toilets number.

**Images –**

This table contains the properties images.

Each record contains: the property’s ID which the image belongs to (Foreign Key), and the image URL. The constraint for the Primary Key is the combination between them.

**Requirements\_R –**

This table contains the requirements of the ideal roommate.

Each record contains: the requirement’s ID (Primary Key), the roommate ID (Foreign Key – the roommate who ask for this requirement), and the all requirements – the roommate’s occupation, minimum age of roommate, maximum age of roommate, the roommate’s gender, smoker (yes/no/not important), diet (yes/no/not important), kosher (yes/no/not important), status (single/in a relationship/not important) and the weight that the roommate gives to the requirements of the ideal roommate.

**Requirements\_P –**

This table contains the requirements of the ideal property.

Each record contains: the requirement’s ID (Primary Key), the roommate ID (Foreign Key – the roommate who ask for this requirement), and all requirements – the property’s type, the country, the city, the neighborhood, the minimum and the maximum rent, the minimum and the maximum rooms number, the minimum and the maximum roommates number, the minimum toilets number, the minimum showers number, a Boolean answer if the property is renovated, a Boolean answer if there is a shelter inside, a Boolean answer if there is a shelter nearby, a Boolean answer if the property is furnished, a Boolean answer if the living room is shared and the weight that the roommate gives to the requirements of the ideal property.

**Scores –**

This table is a many-to-many relationship table, so that each record represents the score that is calculated for a matching between an enter roommate and an insert roommate.

Each record contains: the enter roommate’s ID (Primary Key & Foreign Key), the insert roommate’s ID (Primary Key & Foreign Key) and the following two grades:

1. Enter score –

The score is calculated for the potential enter user, according to the user’s property and roommate requirements and the insert user’s profile and property for offer.

1. Insert score –

The score is calculated for the potential insert user, according to the user’s roommate requirements and the enter user’s profile.

**Likes –**

This table is a many-to-many relationship table, for an enter roommate and an insert roommate, and its purpose is to save who likes the other and if it is mutual.

Each record contains: the enter roommate’s ID (Primary Key & Foreign Key), the insert roommate’s ID (Primary Key & Foreign Key), Boolean answers if the enter roommate likes the specific insert roommate and vice versa.

*The relationships among the data objects:*

**A Profile extends a User –**

This is a 1-to-1 relationship, between a profile object and a user object.

A profile has all characteristics of a user in the system.

A user in the system is meant to have a profile(except the admin user).

A user is created before the profile is created, so no profile can be in the system without being connected to a user.

**A Roommate can have a Property for Offer –**

This is a 1-to-1/0 relationship, between a roommate object and a property object.

A roommate has a property if he is in the side of “insert roommate”, and doesn’t have a property if he is in the side of “enter roommate”.

A property has exactly one roommate who rents it.

**A Property has Images –**

This is a 1-to-many relationship, for a Property and Images that belong it.

A property has at least one image but can have more than one image.

An image belongs to exactly one property.

**A User has Requirements (R/P) –**

This is a many-to-many relationship table, between a roommate objects and requirement objects.

A roommate answers at least one requirement.

A requirement is answered by a few roommates.

**Scores –**

This is a many-to-many relationship table, between an enter roommate and an insert roommate, for calculating the matching score between them.

An enter roommate can match with few insert roommates and see the scores.

An insert roommate can match with few enter roommates and see the scores.

**Likes –**

This is a many-to-many relationship table, between an enter roommate and an insert roommate, for saving the likes between them, so we can use it for the recommendation algorithm.

An enter roommate can like few insert roommates.

An insert roommate can like few enter roommates.

* *Main Transactions*

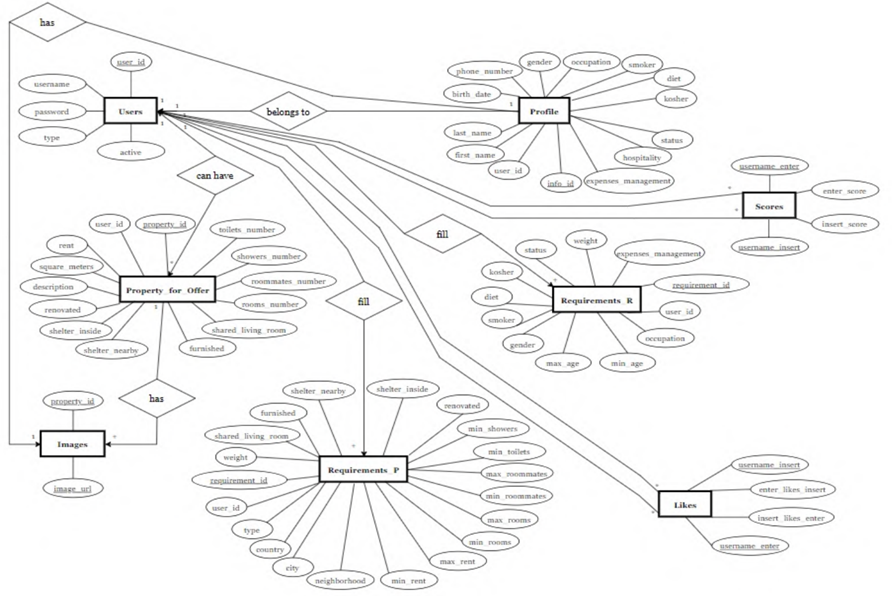
1. Deleting a user

1. **Select** the specific user from the **Users** table.
2. **Select** the profile from the **Profile** table that belongs to this user(join by user\_id).
3. **Delete** the selected record from the **Profile** table (Action b).
4. **If** the profile\_status is “statusInsert”, **Select** the property from the **Property\_for\_Offer** table which belongs to this user (join by user\_id).
5. **If** Action d has more than zero results, **Select** all images from the **Images** table that belong to this property (join by property\_id).
6. **If** Action e has more than zero results, **Delete** all selected images from the Images table (Action e).
7. **If** Action d has more than zero results, **Delete** the selected property from the Properties table (Action d).
8. **Select** all records from the **Scores** table that relate to that user (join by user\_id) and change their score to “-1”.
9. **Select** all records from the **Likes** table that relate to that user (join by user\_id) and change their score to “-1”.
10. **Select** all requirements from the **Requirements R** table that that relate to that user (join by user\_id).
11. **Delete** the selected records from **Requirements R** table (table j).
12. **Select** all requirements from the **Requirements P** table that that relate to that user (join by user\_id).
13. **Delete** the selected records from **Requirements P** table (table j).
14. **Delete** the user from the **Users** table.



2. Updating a property to a roommate

1. **Select** the specific user profile from the **Profile** table.
2. **If** the profile status is “StatusEnter”, **exit**.
3. **If** the profile status is “StatusInsert”, **Select** the property from the **Property\_for\_Offer** table which belongs to this user (join by user\_id).
4. **If** Action c has more than zero results, **Select** all images from the **Images** table that belong to this property (join by property\_id).
5. **If** Action d has more than zero results, **Delete** all selected images from the Images table (Action d).
6. **If** Action c has more than zero results, **Delete** the selected property from the Properties table (Action c).
7. **Add** the new property to the **Property\_for\_Offer** table.



# Chapter 4 – Behavioral Analysis

## 4.1 Sequence Diagrams

**Roommate register**

A picture containing text, diagram, plan, line

Description automatically generated

**Enter roommate asks to see insert roommate’s profile**

A picture containing text, diagram, line, screenshot

Description automatically generated

**Insert roommate edits a property’s description**

A picture containing text, screenshot, diagram, parallel

Description automatically generated

**Roommate edits personal information**

A picture containing text, diagram, screenshot, line

Description automatically generated

## 4.2 Events

Events which relevant to initial state only –

* Upon initialization, the system should initialize the database tables and, pre-built requirements, download and save all the users’ information and answers.

Events triggered by the insert roommate:

* Upon an insert user registration, the system should validate his username and save the user’s information in the database in a secure way.
* Upon an insert user Login, the system should validate his provided credentials and log him in to the system.
* Upon an insert user updating his status from an insert user to an enter user the system should update the user’s record accordingly and remove him from any insert score users listing.
* Upon an insert user updating his answers or details in the system, the system should update the relevant records saved in the database tables accordingly.
* Upon an insert user requesting to see an enter user description, the system should query the database for the enter user’s information and display it to the insert user.
* Upon an insert user editing his profile, the system should update the record saved the database tables accordingly.
* Upon an insert user logging out, the system should remove the active user and deny any further access to his protected information.
* Upon an insert user requesting to see an enter user description, the system should query the database for the enter user’s information and display it to the insert user.
* Upon an insert user requesting to see the home page, the system should calculate insert user’s scores according to the user’s preferences and export a page listing all the relevant enter users and their received scores.
* Upon an insert user filling in his personal information, the system should save the user’s information in the database in a secure way.
* Upon an insert user updating a property, the system saves the property’s information and images in the database in a secure way.

Events triggered by the enter roommate:

* Upon an enter user registration, the system should validate his username and save the roommate’s information in the database in a secure way.
* Upon an enter user Login, the system should validate his provided credentials and log him in to the system.
* Upon an enter user updating his status from an enter user to an insert user the system should update the user’s record accordingly and remove him from any enter score users listing.
* Upon an enter user updating his answers or details in the system, the system should update the relevant records saved in the database tables accordingly.
* Upon an enter user requesting to see an insert user description, the system should query the database for the insert user’s information and display it to the insert user.
* Upon an enter user editing his profile, the system should update the record saved on the database tables accordingly.
* Upon an enter user logging out, the system should remove the active user and deny any further access to his protected information.
* Upon an enter user requesting to see an insert user description, the system should query the database for the insert user’s information and display it to the insert user.
* Upon an enter user requesting to see the home page, the system should calculate enter user’s scores according to the user’s preferences and export a page listing all the relevant insert roommates and their received scores.
* Upon an enter user filling in his personal information, the system should save the user’s information in the database in a secure way.

## 4.3 States

Login

Incorrect username or password

Correct username & password

Click add new property

Open system

Logout

Save new property

Display login page

Request username & password

Login

Incorrect username or password

User logged in

Display user’s possible action

Correct username & password

Click add property information

Open system

Add property information

Request property information

Logout

Save new property

**User logins and adds property information**

**Roommate logins and generates roommate list for home page**

Logout

Leave home page

Home page generated

Display home page with possible roommate matches and their match score

Display login page

Request username & password

Login

Incorrect username or password

User logged in

Display user’s possible action

Correct username & password

Clicks on home page

Open system

**Roommate logins and answers roommate requirements**

Display login page

Request username & password

Login

Incorrect username or password

Roommate logged in

Display roommate possible actions

Roommate requirements page opened

Display question form for roommate possible requirements

Correct username & password

Clicks answer roommate requirements

Open system

Logout

Answer & save

**Roommate logins and answers property requirements**

Display login page

Request username & password

Login

Incorrect username or password

Roommate logged in

Display roommate possible actions

Property requirements page opened

Display question form for property possible requirements

Correct username & password

Clicks answer property requirements

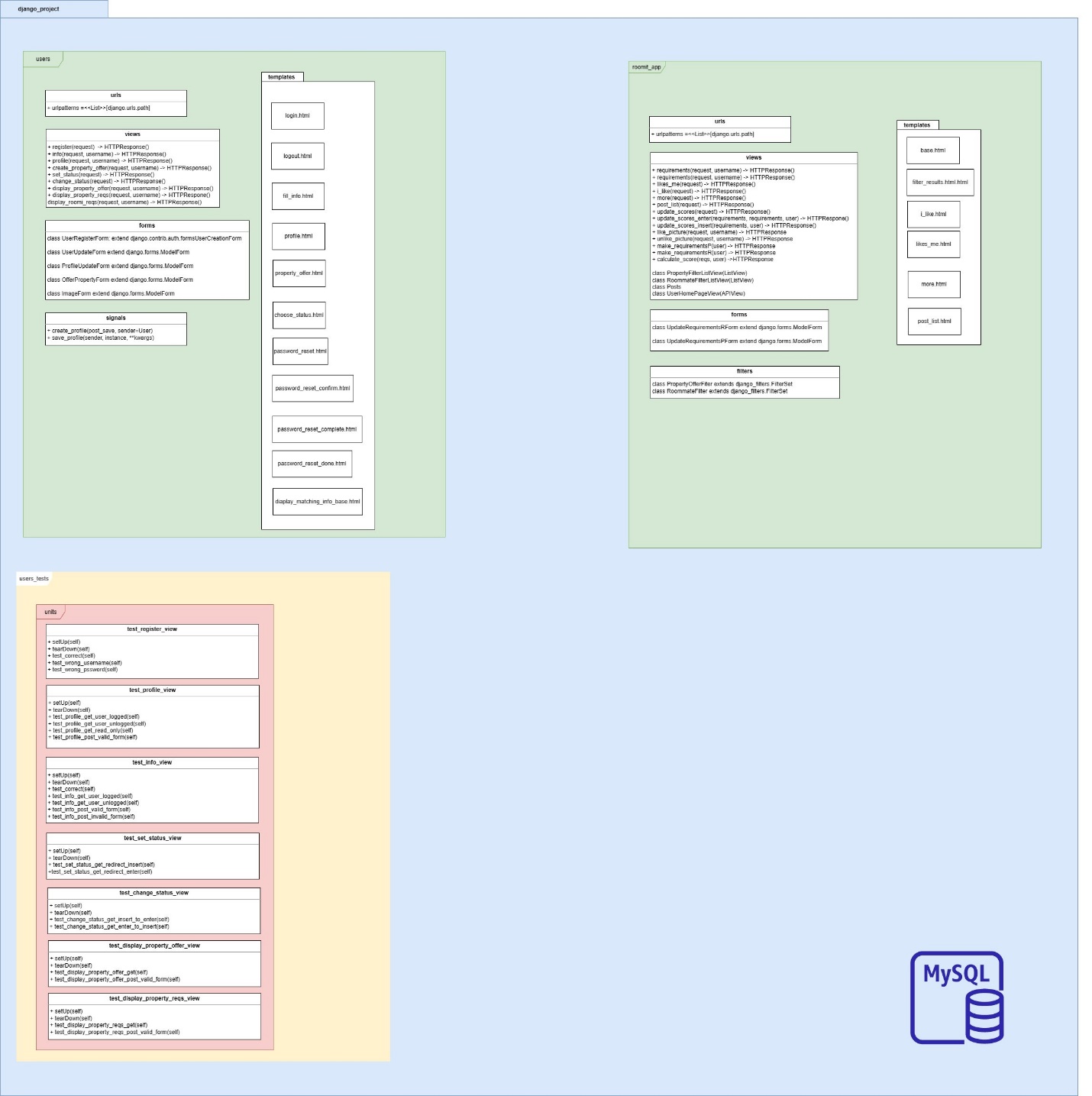
Open system

Logout

Answer & save

# Chapter 5 – Object-Oriented Analysis

## 5.1 Class Diagram



## 5.2 Class Description

**Users Package:**

**User** module is responsible for generating and organizing all the users in the system, for log-in and log-out of the system, and for saving their activeness status. This class is extended by the two next classes.

**Admin** module is responsible for a user who has all permissions, the admin can view all user in the system.

**Profile** module is responsible for the user’s profiles. The objects represent the users who have an apartment (which means “insert roommates”) and who don’t have an apartment (which means “enter roommates).In addition, it is responsible for all the personal details of a user (such as his name, birthdate, profile status etc.).

**Property for offer** module is responsible for the user’s property information. The objects represent a given property owned by a user.

**Roomit app Package:**

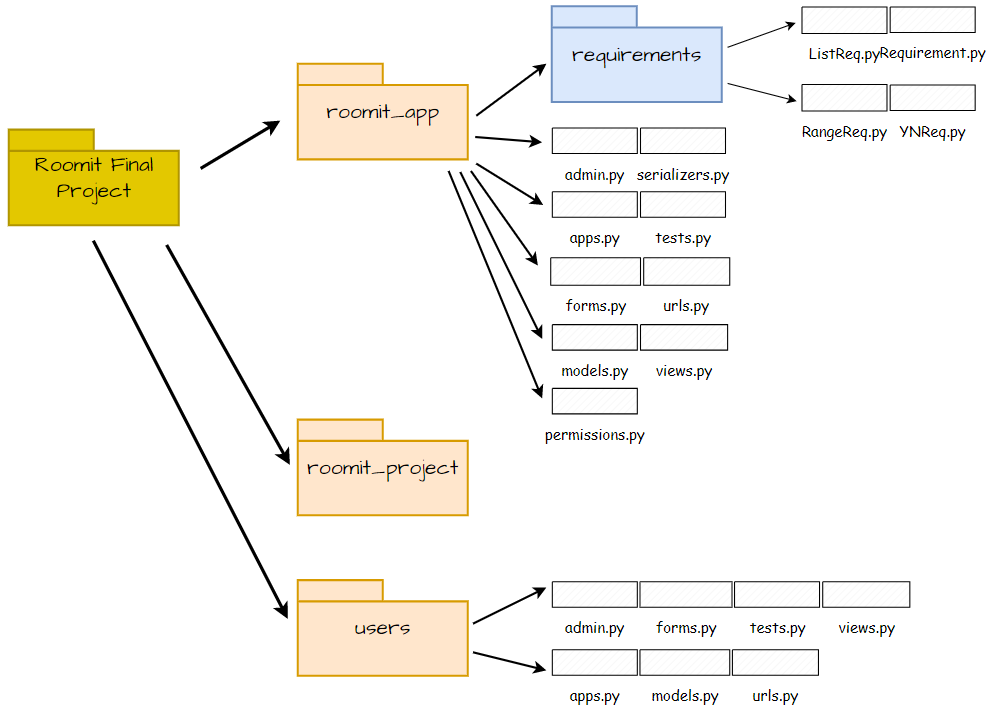
**RequirementsR** module is responsible for all roommate requirements required by the different users. The requirements are filled by the user (either at registration or later when the user is online), saved, and used to calculate the matching score between the different users.

**RequirementsP** module is responsible for all property requirements required by the different users. The requirements are filled by the user (either at registration or later when the user is online), saved, and used to calculate the matching score between the different users.

**Likes** module is responsible for holding all likes between the different users in the system. It hold likes from both directions (user a likes user b and vice versa).

**Score** module is the calculation itself. It contains the score of the property requirements (“property score”), the score of the roommate requirements (“roommate score”), and the final score which based on both and the weights that the roommate decided to give it.

## 5.3 Packages



Main packages and files:

* “roomit\_app” –

This package contains an important package named “requirements” which contains inside four files:

“ListReq.py” (relate to a requirement in the pattern of list), “RangeReq.py”

(relate to a requirement in the pattern of range), “YNReq.py” (relate to a

requirement in the pattern of Yes/No questions) and “Requirement.py”.

In addition, this package contains few files:

“admin.py”, “apps.py” (contains the connection to Django), “forms.py” (the

forms which the roommates must fill), “models.py” (the templates of the

tables we created), etc.

* “users” –

This package contains information about the main users, the roommates.

In the file named “moduls.py” you can find the main classes which create the tables of Info (the personal information), the property of a roommate who his status is “insert” and not “enter”, and the images which are belonging to the roommate’s property.

## 5.4 Testing

**Unit Tests:**

User Module –

|  |  |  |
| --- | --- | --- |
| Test Method Name | Title | Description |
| test\_change\_status\_get\_insert\_to\_enter | Change Status - Get (Insert to Enter) | Tests the behavior of changing the user's status from "Insert" to "Enter" using a GET request. |
| test\_change\_status\_get\_enter\_to\_insert | Change Status - Get (Enter to Insert) | Tests the behavior of changing the user's status from "Enter" to "Insert" using a GET request. |
| test\_display\_property\_offer\_get | Display Property Offer - GET | Tests the behavior of retrieving the property offer display page using a GET request. |
| test\_display\_property\_offer\_post\_valid\_form | Display Property Offer - POST (Valid Form) | Tests the behavior of submitting a valid form for the property offer display page using a POST request. |
| test\_display\_property\_offer\_post\_invalid\_form | Display Property Offer - POST (Invalid Form) | Tests the behavior of submitting an invalid form for the property offer display page using a POST request. |
| test\_display\_property\_reqs\_get | Display Property Reqs - GET | Tests the behavior of retrieving the property requirements display page using a GET request. |
| test\_display\_property\_reqs\_post\_invalid\_form | Display Property Reqs - POST (Invalid Form) | Tests the behavior of submitting an invalid form for the property requirements display page using a POST request. |
| test\_info\_get\_user\_logged | Info - GET (User Logged) | Tests the behavior of retrieving the user's info page when the user is logged in. |
| test\_info\_get\_user\_unlogged | Info - GET (User Unlogged) | Tests the behavior of retrieving the user's info page when the user is unlogged. |
| test\_info\_post\_valid\_form | Info - POST (Valid Form) | Tests the behavior of submitting a valid form on the user's info page using a POST request. |
| test\_info\_post\_invalid\_form | Info - POST (Invalid Form) | Tests the behavior of submitting an invalid form on the user's info page using a POST request. |
| test\_profile\_get\_user\_logged | Profile - GET (User Logged) | Tests the behavior of retrieving the user's profile page when the user is logged in. |
| test\_profile\_get\_user\_unlogged | Profile - GET (User Unlogged) | Tests the behavior of retrieving the user's profile page when the user is unlogged. |
| test\_profile\_get\_read\_only | Profile - GET (Read-Only) | Tests the behavior of retrieving the profile page of another user in read-only mode. |
| test\_profile\_post\_valid\_form | Profile - POST (Valid Form) | Tests the behavior of submitting a valid form on the user's profile page using a POST request. |
| test\_info\_post\_invalid\_form | Profile - POST (Invalid Form) | Tests the behavior of submitting an invalid form on the user's profile page using a POST request. |
| test\_correct | Register - Correct Credentials | Tests the authentication of a user with correct credentials. |
| test\_wrong\_username | Register - Incorrect Username | Tests the authentication of a user with an incorrect username. |
| test\_wrong\_password | Register - Incorrect Password | Tests the authentication of a user with an incorrect password. |
| test\_set\_status\_get\_redirect\_insert | Set Status - Get (Redirect Insert) | Tests the behavior of redirecting the user to the "Insert" page when accessing the status page with the status set to "Insert". |
| test\_set\_status\_get\_redirect\_enter | Set Status - Get (Redirect Enter) | Tests the behavior of redirecting the user to the "Enter" page when accessing the status page with the status set to "Enter". |

roomit\_app module –

|  |  |  |
| --- | --- | --- |
| Method Name | Title | Description |
| test\_calculate\_score | Calculate Score Test | Tests the calculation of a score based on certain properties and requirements. |
| test\_update\_scores\_only\_creates\_new\_scores | Update Scores Test | Tests that updating scores creates new scores. |
| test\_update\_scores\_correctly\_calculates\_enter\_scores | Update Scores Enter Test | Tests that updating scores correctly calculates enter scores. |
| test\_update\_scores\_correctly\_calculates\_insert\_scores | Update Scores Insert Test | Tests that updating scores correctly calculates insert scores. |
| test\_update\_scores\_enter | Update Scores Enter Test | Tests the update of scores for the "enter" profile. |
| test\_update\_scores\_insert | Update Scores Insert Test | Tests the update of scores for the "insert" profile. |
| test\_make\_requirementsP | Make RequirementsP Test | Tests the creation of requirementsP object. |
| test\_make\_requirementsR | Make RequirementsR Test | Tests the creation of requirementsR object. |
| test\_requirementsP\_authenticated | Authenticated requirementsP Test | Tests the requirementsP view for an authenticated user. |
| test\_requirementsP\_unauthenticated | Unauthenticated requirementsP Test | Tests the requirementsP view for an unauthenticated user. |
| test\_requirementsR\_authenticated | Authenticated requirementsR Test | Tests the requirementsR view for an authenticated user. |
| test\_requirementsR\_unauthenticated | Unauthenticated requirementsR Test | Tests the requirementsR view for an unauthenticated user. |
| test\_likes\_me\_authenticated | Authenticated likes\_me Test | Tests the likes\_me view for an authenticated user. |
| test\_likes\_me\_unauthenticated | Unauthenticated likes\_me Test | Tests the likes\_me view for an unauthenticated user. |
| test\_i\_like\_authenticated | Authenticated i\_like Test | Tests the i\_like view for an authenticated user. |
| test\_i\_like\_unauthenticated | Unauthenticated i\_like Test | Tests the i\_like view for an unauthenticated user. |
| test\_more\_authenticated\_insert | Authenticated more (insert) Test | Tests the more view for an authenticated user with profile status "insert". |
| test\_more\_authenticated\_enter | Authenticated more (enter) Test | Tests the more view for an authenticated user with profile status "enter". |
| test\_more\_unauthenticated | Unauthenticated more Test | Tests the more view for an unauthenticated user. |
| test\_post\_list\_authenticated\_insert | Authenticated post\_list (insert) Test | Tests the post\_list view for an authenticated user with profile status "insert". |
| test\_post\_list\_authenticated\_enter | Authenticated post\_list (enter) Test | Tests the post\_list view for an authenticated user with profile status "enter". |

Requirements:

* **Y/N requirements –**

|  |  |  |
| --- | --- | --- |
| Test Method Name | Title | Description |
| test\_calculate\_score\_desired\_answer\_yes | Calculate Score - Desired Answer: Yes | Tests the calculation of the score when the desired answer is "Yes". |
| test\_calculate\_score\_desired\_answer\_no | Calculate Score - Desired Answer: No | Tests the calculation of the score when the desired answer is "No". |
| test\_calculate\_score\_no\_desired\_answer | Calculate Score - No Desired Answer | Tests the calculation of the score when there is no desired answer. |
| test\_calculate\_score\_answer\_yes\_not\_desired\_answer | Calculate Score - Answer: Yes, Not Desired Answer | Tests the calculation of the score when the answer is "Yes" but not the desired answer. |
| test\_calculate\_score\_answer\_no\_not\_desired\_answer | Calculate Score - Answer: No, Not Desired Answer | Tests the calculation of the score when the answer is "No" but not the desired answer. |
| test\_calculate\_score\_answer\_empty\_string | Calculate Score - Answer: Empty String | Tests the calculation of the score when the answer is an empty string. |
| test\_calculate\_score\_answer\_false | Calculate Score - Answer: False | Tests the calculation of the score when the answer is False. |
| test\_calculate\_score\_answer\_one | Calculate Score - Answer: 1 | Tests the calculation of the score when the answer is 1. |
| test\_calculate\_score\_answer\_empty\_list | Calculate Score - Answer: Empty List | Tests the calculation of the score when the answer is an empty list. |
| test\_calculate\_score\_answer\_non\_empty\_list | Calculate Score - Answer: Non-Empty List | Tests the calculation of the score when the answer is a non-empty list. |
| test\_calculate\_score\_with\_desired\_answer | Calculate Score - With Desired Answer | Tests the calculation of the score with a desired answer specified. |
| test\_calculate\_score\_without\_desired\_answer | Calculate Score - Without Desired Answer | Tests the calculation of the score without a desired answer specified. |
| test\_calculate\_score\_with\_non\_bool\_answer | Calculate Score - With Non-Boolean Answer | Tests the calculation of the score with a non-boolean answer. |
| test\_desired\_answer\_none | Desired Answer: None | Tests the desired answer as None. |
| test\_matching\_bool\_answer | Matching Boolean Answer | Tests the calculation of the score with a matching boolean answer. |
| test\_non\_matching\_bool\_answer | Non-Matching Boolean Answer | Tests the calculation of the score with a non-matching boolean answer. |
| test\_answer\_not\_bool | Answer: Not Boolean | Tests the calculation of the score with a non-boolean answer. |

* **Range requirements –**

|  |  |  |
| --- | --- | --- |
| Test Method Name | Title | Description |
| test\_calculate\_score\_when\_min\_and\_max\_are\_none | Calculate Score - Min and Max: None | Tests the calculation of the score when both the minimum and maximum values are None. |
| test\_calculate\_score\_when\_max\_is\_none | Calculate Score - Max: None | Tests the calculation of the score when the maximum value is None. |
| test\_calculate\_score\_when\_answer\_is\_none | Calculate Score - Answer: None | Tests the calculation of the score when the answer is None. |
| test\_calculate\_score\_when\_answer\_is\_not\_an\_int | Calculate Score - Answer: Not an Integer | Tests the calculation of the score when the answer is not an integer. |
| test\_calculate\_score\_when\_answer\_is\_within\_range | Calculate Score - Answer: Within Range | Tests the calculation of the score when the answer is within the specified range. |
| test\_calculate\_score\_when\_answer\_is\_outside\_  range\_and\_outside\_tolerance | Calculate Score - Answer: Outside Range and Tolerance | Tests the calculation of the score when the answer is outside the specified range and tolerance. |
| test\_both\_min\_max\_none | Min and Max: None | Tests the scenario when both the minimum and maximum values are none. |
| test\_answer\_not\_int | Answer: Not an Integer | Tests the scenario when the answer is not an integer. |
| test\_answer\_negative\_number | Answer: Negative Number | Tests the scenario when the answer is a negative number. |
| test\_answer\_outside\_range | Answer: Outside Range | Tests the scenario when the answer is outside the specified range. |
| test\_answer\_equal\_to\_min\_max | Answer: Equal to Min/Max | Tests the scenario when the answer is equal to the minimum or maximum value. |
| test\_answer\_is\_none | Answer: None | Tests the scenario when the answer is None. |
| test\_answer\_is\_not\_a\_number | Answer: Not a Number | Tests the scenario when the answer is not a number. |
| test\_answer\_is\_negative | Answer: Negative | Tests the scenario when the answer is a negative number. |
| test\_answer\_is\_greater\_than\_max | Answer: Greater than Max | Tests the scenario when the answer is greater than the maximum value. |
| test\_answer\_is\_less\_than\_min | Answer: Less than Min | Tests the scenario when the answer is less than the minimum value. |
| test\_answer\_is\_equal\_to\_min | Answer: Equal to Min | Tests the scenario when the answer is equal to the minimum value. |
| test\_answer\_is\_equal\_to\_max | Answer: Equal to Max | Tests the scenario when the answer is equal to the maximum value. |
| test\_answer\_is\_in\_range | Answer: In Range | Tests the scenario when the answer is within the specified range. |
| test\_answer\_birthdate\_is\_out\_of\_range | Answer: Birthdate out of Range | Tests the scenario when the answer (birthdate) is outside the specified range. |
| test\_answer\_birthdate\_is\_on\_range | Answer: Birthdate on Range | Tests the scenario when the answer (birthdate) is on the specified range. |
| test\_answer\_birthdate\_is\_in\_range | Answer: Birthdate in Range | Tests the scenario when the answer (birthdate) is within the specified range. |
| test\_answer\_is\_not\_birthdate | Answer: Not a Birthdate | Tests the scenario when the answer is not a birthdate. |
| test\_answer\_birthday\_is\_in\_future | Answer: Birthday in Future | Tests the scenario when the answer (birthday) is in the future. |
| test\_non\_datetime\_birthdate | Non-Datetime Birthdate | Tests the scenario when the answer (birthdate) is a non-datetime value. |
| test\_non\_numeric\_min\_or\_max | Non-Numeric Min/Max | Tests the scenario when the minimum or maximum value is non-numeric. |
| test\_non\_numeric\_answer | Non-Numeric Answer | Tests the scenario when the answer is non-numeric. |

* **List requirements –**

|  |  |  |
| --- | --- | --- |
| Test Method Name | Title | Description |
| test\_calculate\_score\_desired\_answer\_none | Calculate Score - Desired Answer: None | Tests the calculation of the score when the desired answer is None. |
| test\_calculate\_score\_answer\_none | Calculate Score - Answer: None | Tests the calculation of the score when the answer is None. |
| test\_calculate\_score\_desired\_answer\_is\_disjoint | Calculate Score - Desired Answer: Disjoint | Tests the calculation of the score when the desired answer is disjoint from the actual answer. |
| test\_calculate\_score\_answer\_is\_subset | Calculate Score - Answer: Subset | Tests the calculation of the score when the actual answer is a subset of the desired answer. |
| test\_calculate\_score\_answer\_is\_superset | Calculate Score - Answer: Superset | Tests the calculation of the score when the actual answer is a superset of the desired answer. |
| test\_calculate\_score\_desired\_answer\_is\_empty | Calculate Score - Desired Answer: Empty | Tests the calculation of the score when the desired answer is an empty set. |
| test\_calculate\_score\_answer\_is\_none | Calculate Score - Answer: None | Tests the calculation of the score when the answer is None. |
| test\_calculate\_score\_answer\_does\_not\_  intersect\_desired\_answer | Calculate Score - Answer: No Intersection | Tests the calculation of the score when the answer does not intersect with the desired answer. |
| test\_calculate\_score\_answer\_intersects\_desired\_answer | Calculate Score - Answer: Intersects | Tests the calculation of the score when the answer intersects with the desired answer. |
| test\_calculate\_score\_no\_desired\_answer | Calculate Score - No Desired Answer | Tests the calculation of the score when there is no desired answer. |
| test\_calculate\_score\_desired\_answer\_dont\_care | Calculate Score - Desired Answer: Don't Care | Tests the calculation of the score when the desired answer is "D" (don't care). |
| test\_calculate\_score\_no\_answer | Calculate Score - No Answer | Tests the calculation of the score when there is no answer. |
| test\_calculate\_score\_answer\_in\_desired\_answer | Calculate Score - Answer in Desired Answer | Tests the calculation of the score when the answer is in the desired answer list. |
| test\_calculate\_score\_answer\_not\_in\_desired\_answer | Calculate Score - Answer not in Desired Answer | Tests the calculation of the score when the answer is not in the desired answer list. |
| test\_calculate\_score\_with\_empty\_desired\_answer\_list | Calculate Score - Empty Desired Answer List | Tests the calculation of the score when the desired answer list is empty. |
| test\_calculate\_score\_with\_single\_desired\_answer | Calculate Score - Single Desired Answer | Tests the calculation of the score when there is a single desired answer. |
| test\_calculate\_score\_with\_non\_list\_answer | Calculate Score - Non-List Answer | Tests the calculation of the score when the answer is not a list. |
| test\_calculate\_score\_with\_non\_string\_answer | Calculate Score - Non-String Answer | Tests the calculation of the score when the answer contains non-string elements. |

Security tests –

|  |  |  |
| --- | --- | --- |
| Test Method Name | Title | Description |
| test\_unauthorized\_access | Unauthorized Access | Verifies that when no user is logged in and attempts to access the admin page, they are redirected to the login page. |
| test\_invalid\_credentials | Invalid Credentials | Tests the scenario where the admin tries to log in with the wrong password and checks if the correct error message is displayed. |
| test\_password\_reset | Password Reset | Tests the functionality of password reset by sending a request and verifying the appropriate redirection. |
| test\_role\_permissions | Role Permissions | Tests the role-based permissions. Checks if a regular user is redirected to the login page when accessing the admin page and if the admin has access. |
| test\_input\_sanitization | Input Sanitization | Verifies that user input is sanitized to prevent cross-site scripting (XSS) attacks. |

Performance tests –

The **test\_performance** module is designed for performance testing using Locust, where multiple users simulate various tasks on a web application. This module focuses on simulating user registration, filling out user information, and performing tasks related to viewing the homepage and user profiles.

The performance test consists of three tasks:

* **registration:** Simulates user registration to the system and filling out their personal information and requirements.
* **view\_homepage:** Simulates users accessing the homepage of the web application by sending a GET request to the root URL.
* **view\_profile:** Simulates users viewing user profiles by sending a GET request to a randomly selected user's profile page. The username for the profile is chosen from the usernames list, which contains the usernames of all registered users.

The selected tasks for performance testing were intentionally chosen as they represent the heaviest operations in terms of database interactions and computational complexity. By focusing on these tasks, we aim to evaluate the system's performance under demanding scenarios that involve substantial database operations and computational calculations.

By running this performance test with Locust and scaling up the number of users, it becomes possible to measure the web application's performance under load. This module can help identify potential performance bottlenecks, such as slow response times or resource limitations, allowing developers to optimize and improve the application's scalability.

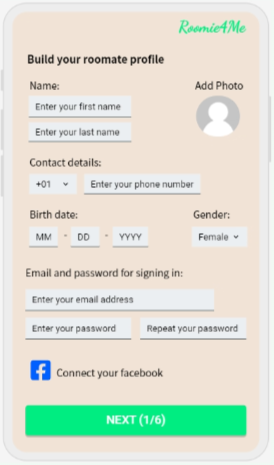
# Chapter 6 - User Interface Draft

**This is our first screen from the mobile application we built by React Native. This is a screenshot from an IOS smartphone:**

A couple of women holding a glass of wine

Description automatically generated with low confidence

**Since the application is still a work in progress, to maintain uniformity we will use our sketch. Here are the screens which are coming while sign up to the application:**

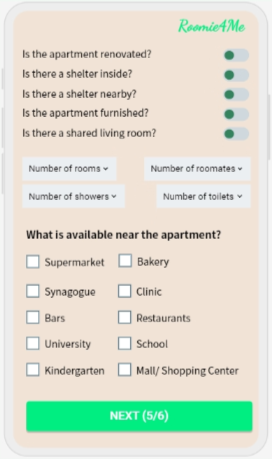
A screenshot of a survey

Description automatically generated with medium confidenceA screenshot of a phone

Description automatically generated with medium confidence

**If the roommate is an “insert roommate” here are the screens for continuing sign up:**

A screenshot of a phone

Description automatically generated with medium confidenceA screenshot of a survey

Description automatically generated with medium confidence

**If the roommate is an “enter roommate” here are the screens for continuing sign up:**

A screenshot of a phone

Description automatically generated with medium confidenceA screenshot of a survey

Description automatically generated with low confidenceA screenshot of a survey

Description automatically generated with low confidence

**The sign in screen:**

A screenshot of a login box

Description automatically generated with medium confidence

**If the roommate is an “insert roommate” here are the screens which he can see, his personal information, scrolling for other roommates and chatting in inbox with potential roommates:**

A screenshot of a phone

Description automatically generated with medium confidenceA screenshot of a phone

Description automatically generated with medium confidenceA screenshot of a phone

Description automatically generated with medium confidence

**If the roommate is an “enter roommate” here are the screens which he can see, his personal information, scrolling for properties and chatting in inbox with potential roommates:**

A screenshot of a phone

Description automatically generated with medium confidenceA screenshot of a cell phone

Description automatically generated with medium confidenceA screenshot of a phone

Description automatically generated with medium confidence