ROOMIT

Application Design Document

Eylon Sade
Ofir Povimonsky
Noa Magrisso

ROOMIT
Your ideal roommate awaits

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 the user his matching scores with the other users in the system and display them on his homepage feed.

For a roommate who already lives in an apartment, it will display optional roommates that looking for an apartment to enter, and for a roommate who looking for an apartment to enter in it will display optional roommates who own apartments.

We collect our data from questionnaires for students who have already experienced the roommate search stage or are experiencing it now.

TABLE OF CONTENTS

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 he lives in.

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

Roommate A –

1. *Roommate A inserts the 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.

2. *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.

3. *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 exists and is active.
   Post-conditions: Roommate B's profile displays.

4. *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.

5. *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.

6. *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.

7. *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.

8. *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.

9. *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.

10. *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.

11. *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.

12. *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.

13. *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.

14. *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.

2. *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.

3. *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.

4. *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.

5. *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.

6. *Roommate B registers the system.*
   Actors: Roommate B
   Pre-conditions: Roommate B is logged in.
   Post-conditions: Roommate B's profile was created.

7. *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.

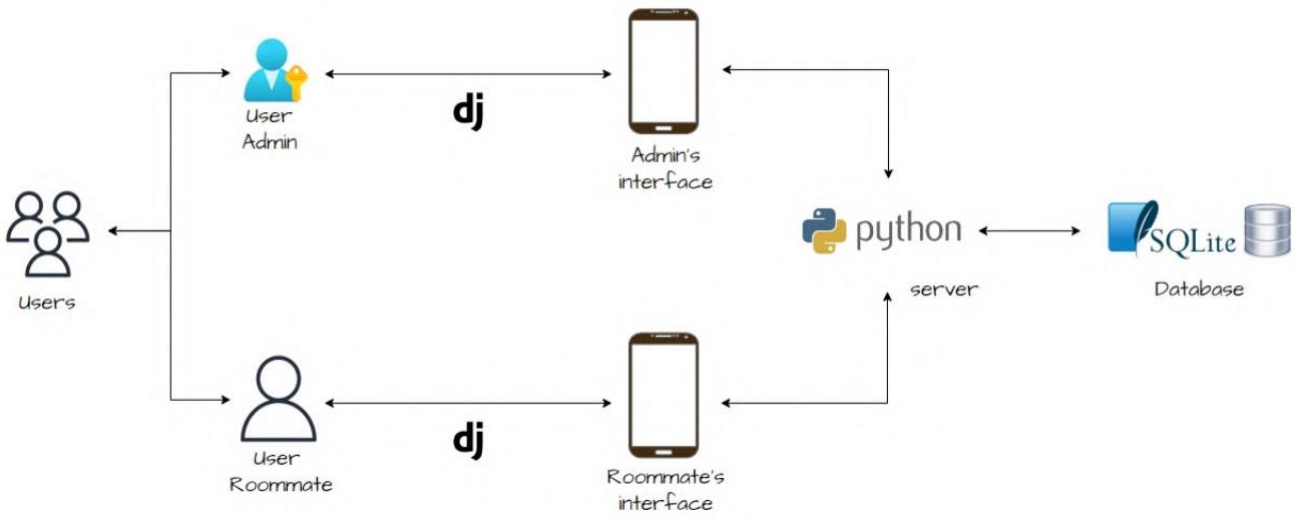
8. *Roommate B edits his profile (description, his picture, requirements, etc.)*
   Actors: Roommate B
   Pre-conditions: The object of Roommate B is active and logged in.
   Post-conditions: Roommate B's profile was updated.

9. *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.

10. *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.

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 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 that 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:

SQLite.

Store all relevant component 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:

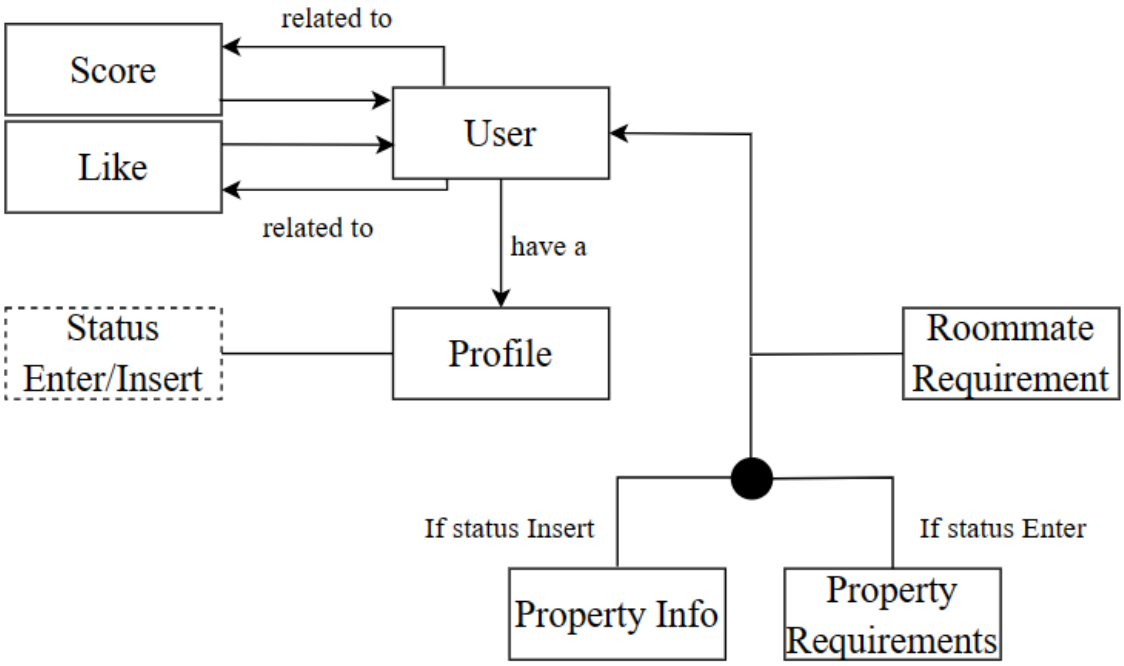
Web pages developed in **HTML** and **CSS**.

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.

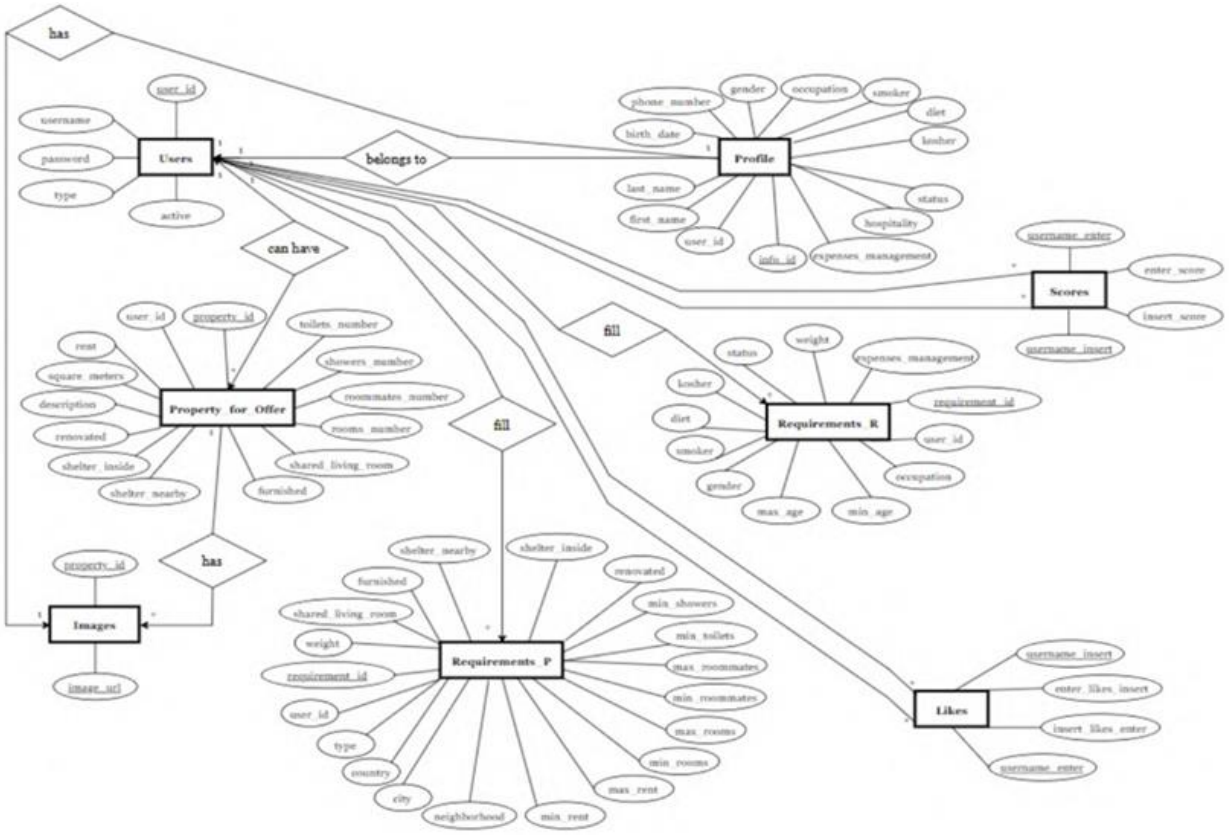
CHAPTER 3 – DATA MODEL

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 roommate's 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 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 matching between an enter roommate and an insert roommate.

Each record contains the entered 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.
2. Insert score –
   The score is calculated for the potential insert user, according to the user's roommate requirements and the entered 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 property if he is on the side of "insert roommate", and doesn't have 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 to 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 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 a few insert roommates and see the scores.
An insert roommate can match with a 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 a few insert roommates.
An insert roommate can like a few enter roommates.

➢ *Main Transactions*

1. Deleting a user

i. Retrieve the user's information and associated records from the database.
ii. Delete the user's profile, including their profile picture, personal information, and preferences.
iii. Delete any scores entered or inserted by the user, removing all score entries associated with them.
iv. Delete the user's requirements, whether they are property requirements or roommate requirements.
v. Remove any likes related to the user, including both the likes they initiated and the likes they received.
vi. Delete any properties listed by the user, along with the associated images.
vii. Finally, delete the user's account and all their data from the system.

2. Creating new Roommate-requirements

i. Retrieve the user's information from the database.
ii. Collect the requirements specified by the user, such as occupation, age range, gender, smoker status, etc.
iii. Create a new roommate requirement entry in the database, associating it with the user.

3. Creating new Property-requirements

i. Step 1: Retrieve the user's information from the database.
ii. Step 2: Collect the requirements specified by the user, such as rent range, number of rooms, maximum roommates, etc.
iii. Step 3: Create a new property requirement entry in the database, associating it with the user.

4. Updating a property to a roommate

i. Retrieve the user's information from the database.
ii. Fetch the property that needs to be assigned as the roommate property.
iii. Update the user's record to include the assigned property as their roommate property.

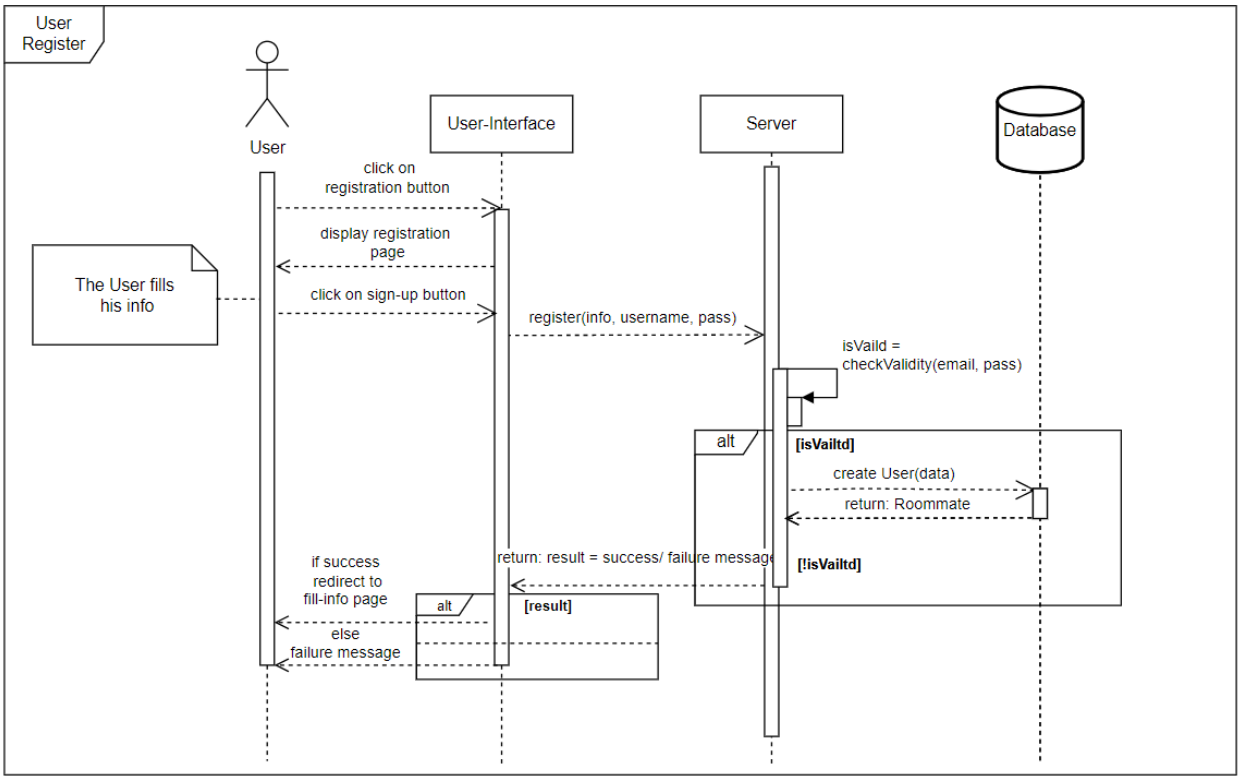
5. Uploading images for a property:

iv. Retrieve the property for which images need to be uploaded.
v. Allow the user to select and upload multiple images to showcase the property.
vi. Save the uploaded images to the appropriate location and associate them with the property in the database.

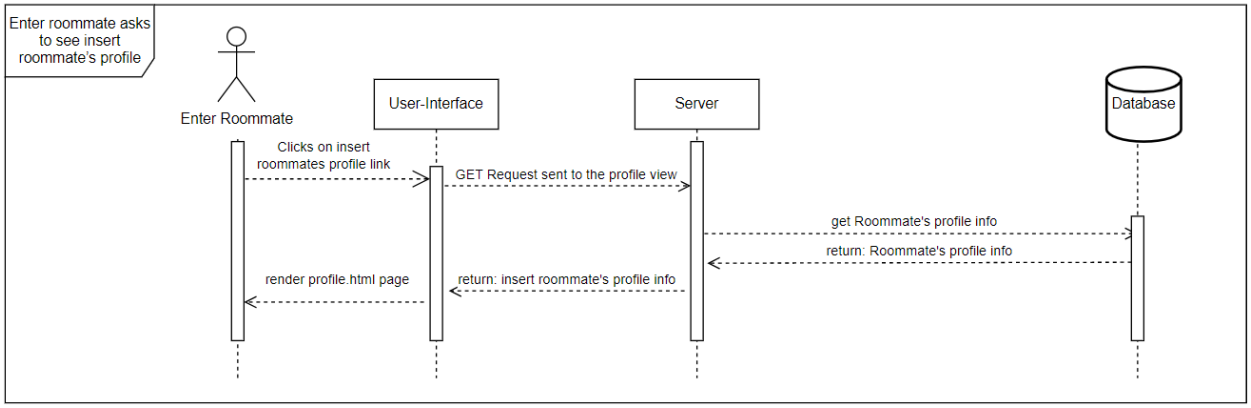
CHAPTER 4 – BEHAVIORAL ANALYSIS

4.1 Sequence Diagrams

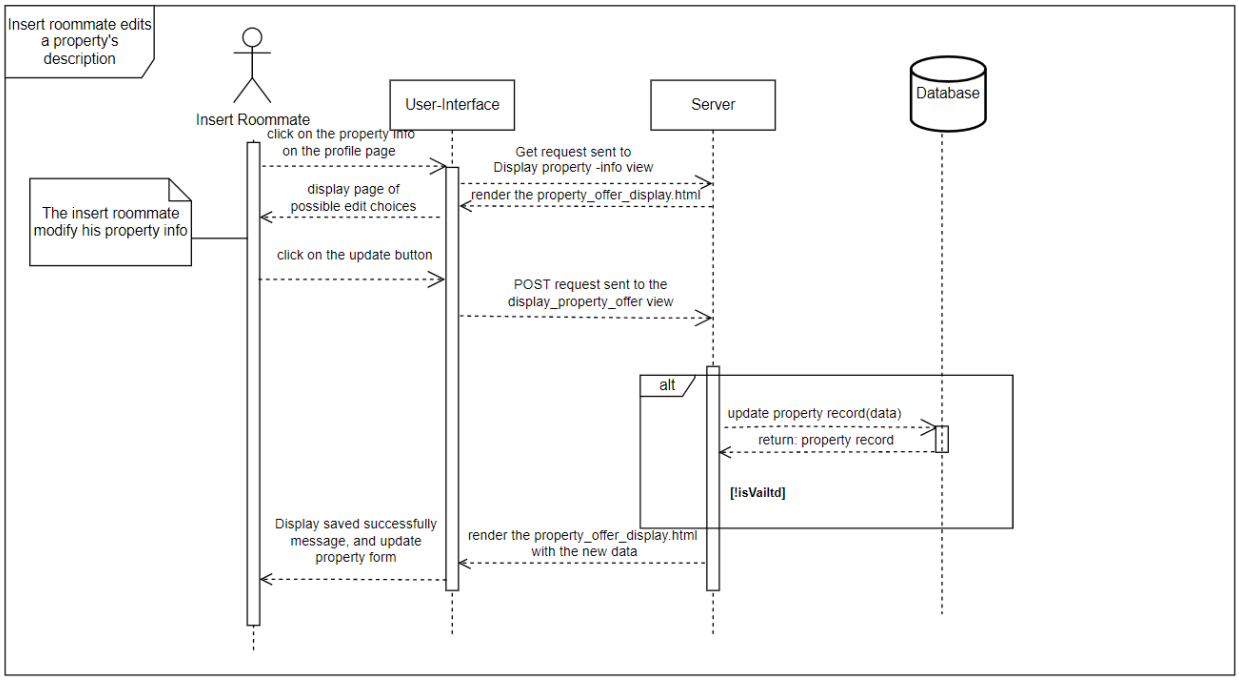
Roommate register



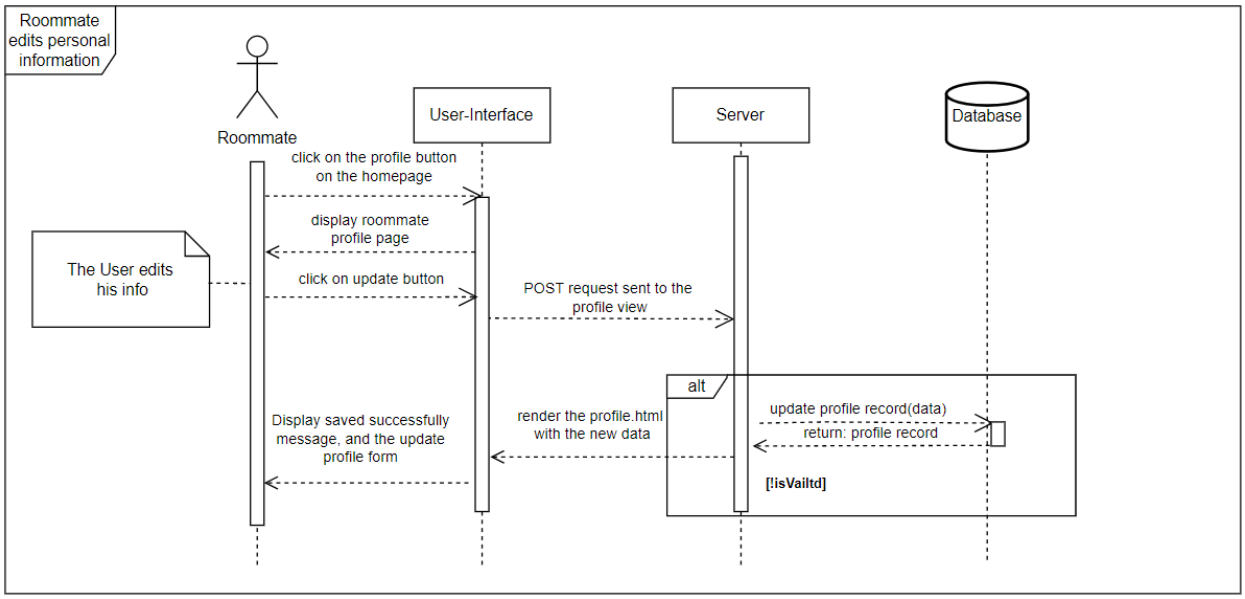
Enter roommate asks to see insert roommate's profile



Insert roommate edits a property's description



Roommate edits personal information



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.

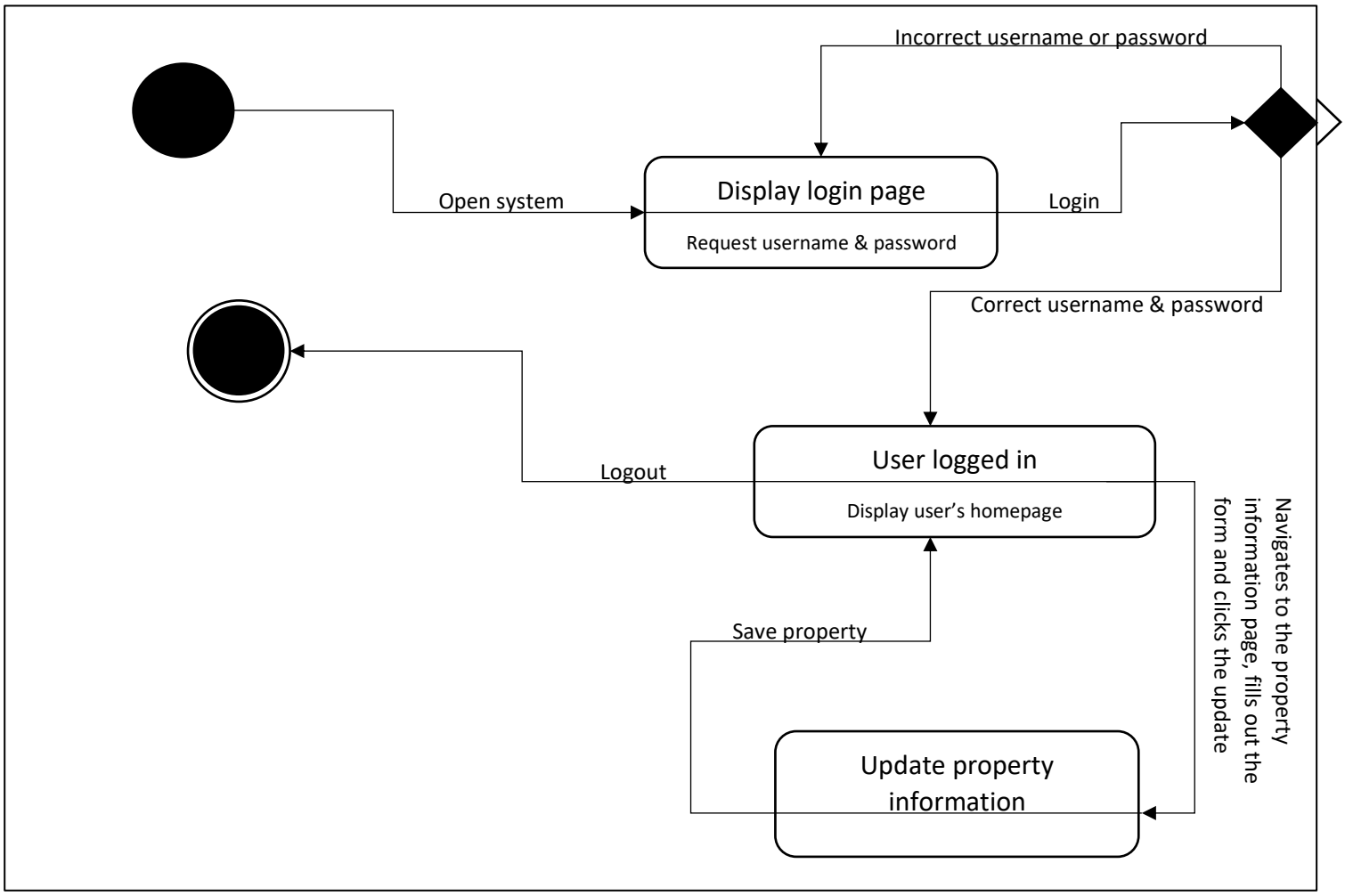
- Upon an enter user requesting to see an insert user description, the system should query the database for the insert user's information and display it to the insert user.

- 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.

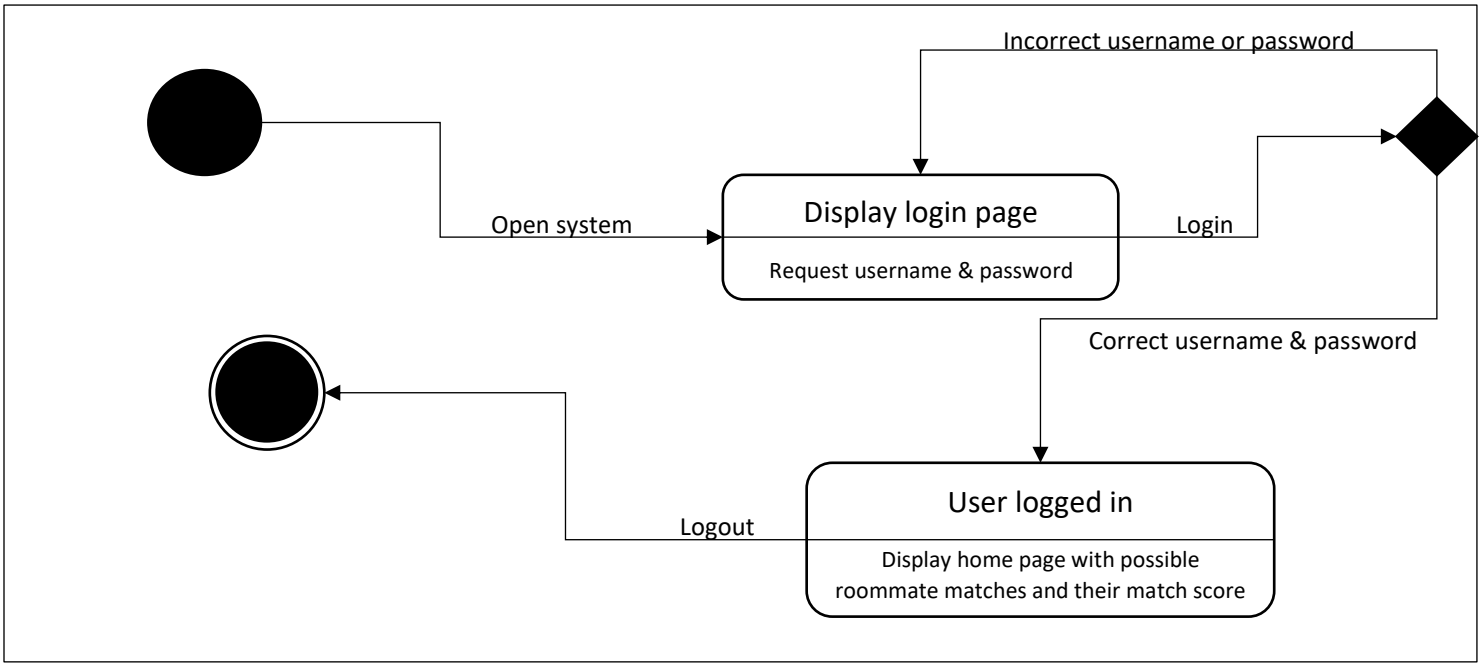
- Upon an enter user filling in his personal information, the system should save the user's information in the database in a secure way.

4.3 States

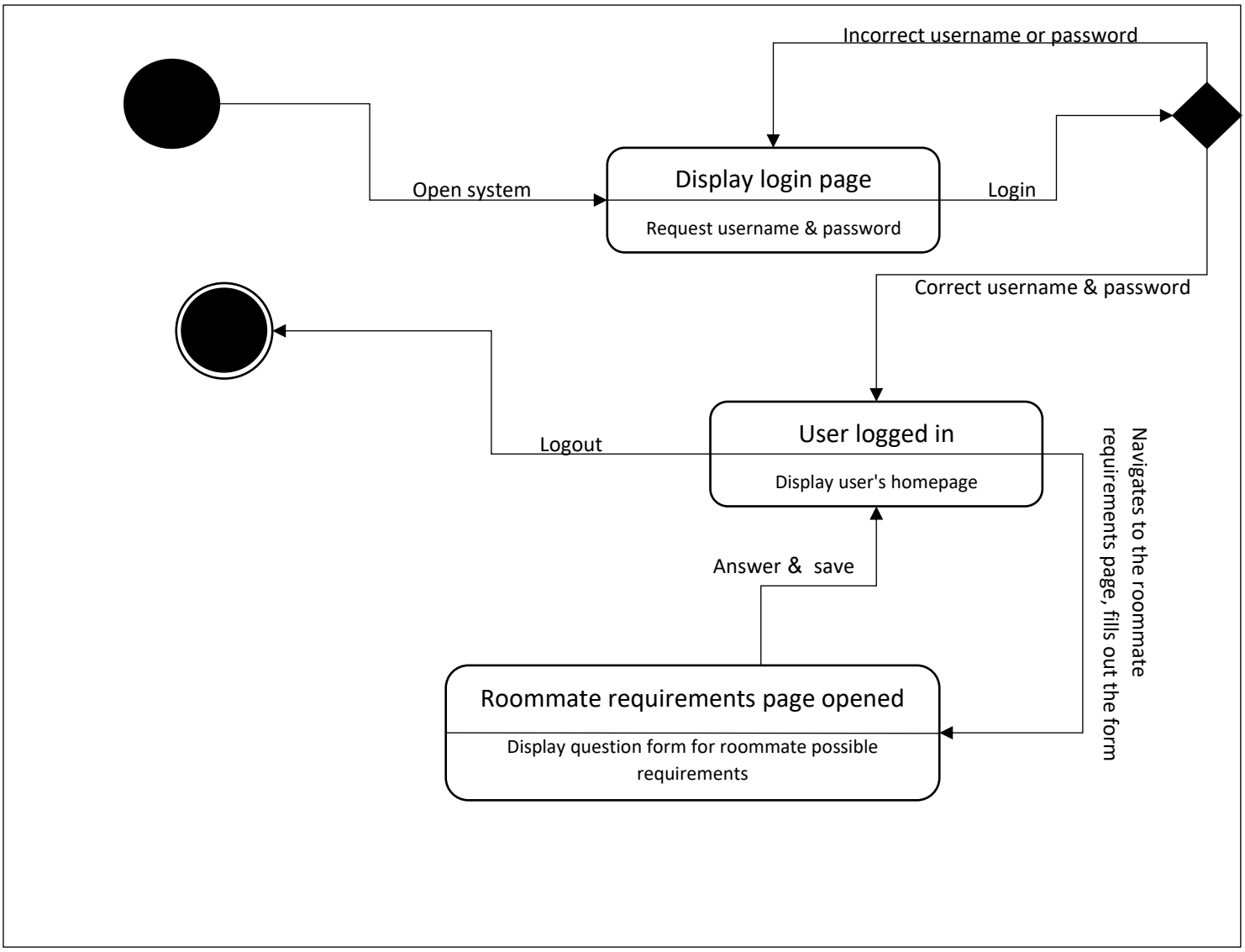
User logins and update property information



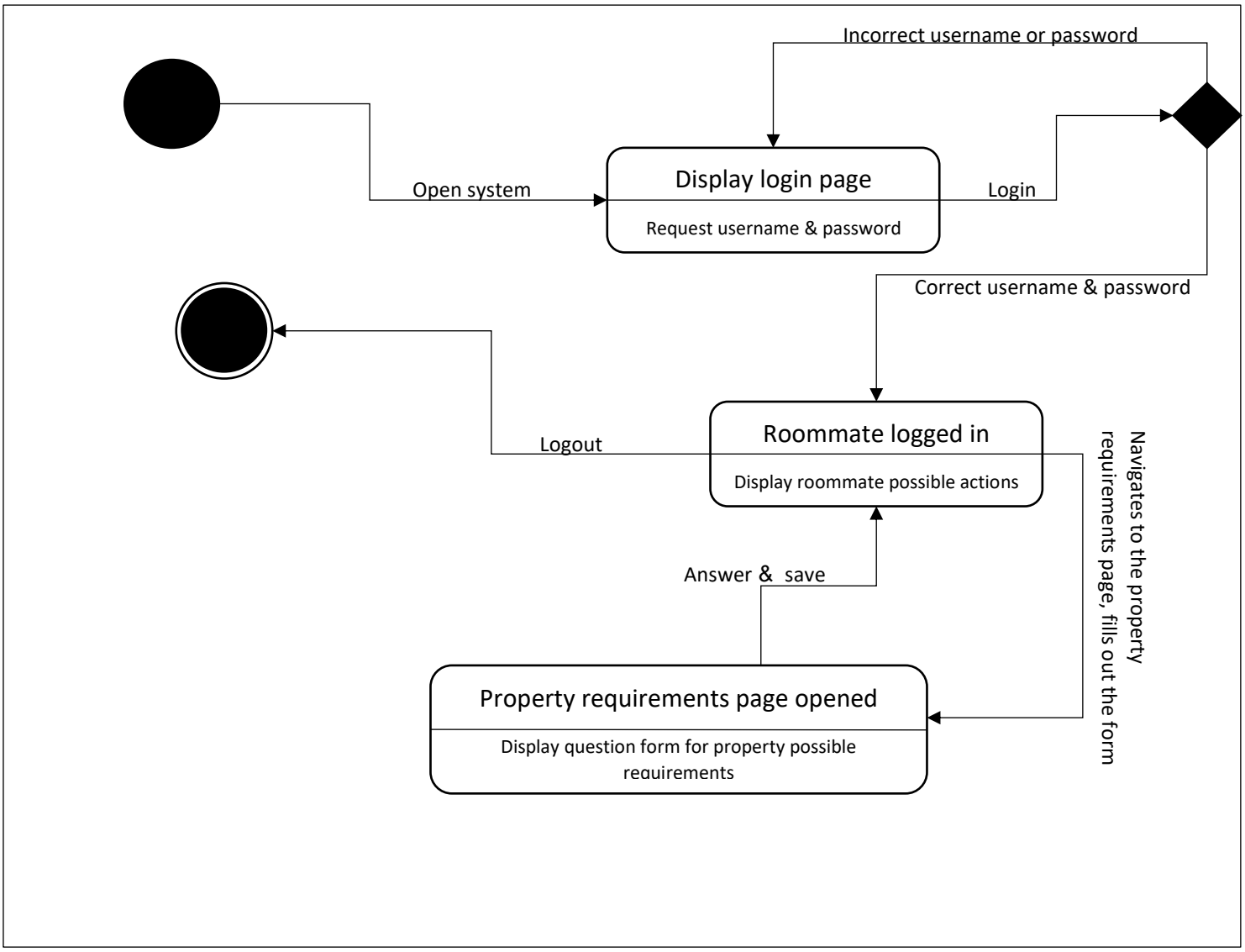
Roommate logins and generates roommate list for home page



Roommate logins and answers roommate requirements

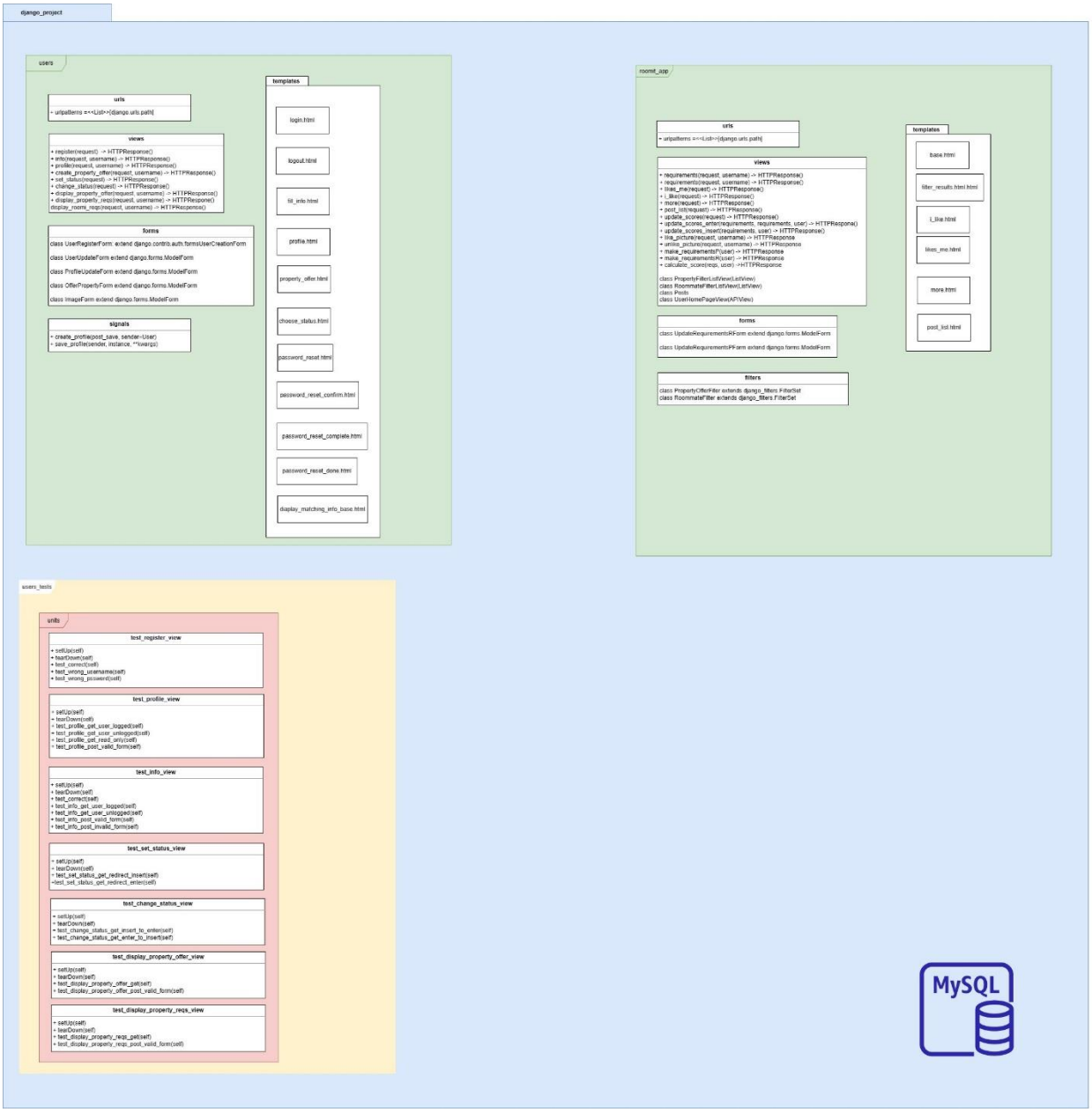


Roommate logins and answers property requirements



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:

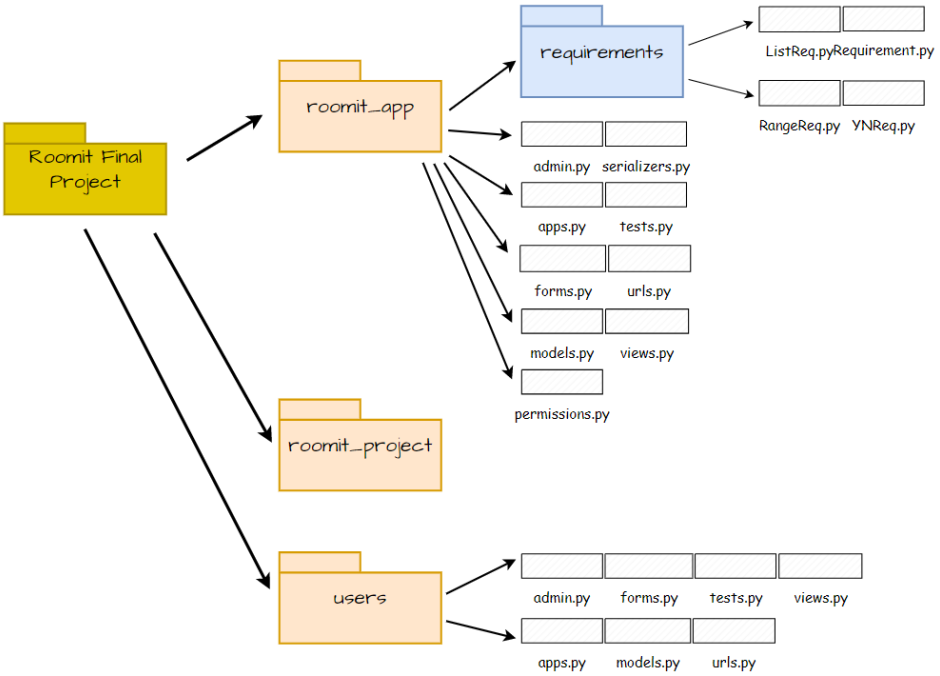
RequirementsR module is responsible for all roommate requirements required by the different users. The requirements are filled by the user (either at registration or later when the user is online), saved, and used to calculate the matching score between the different users.

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.

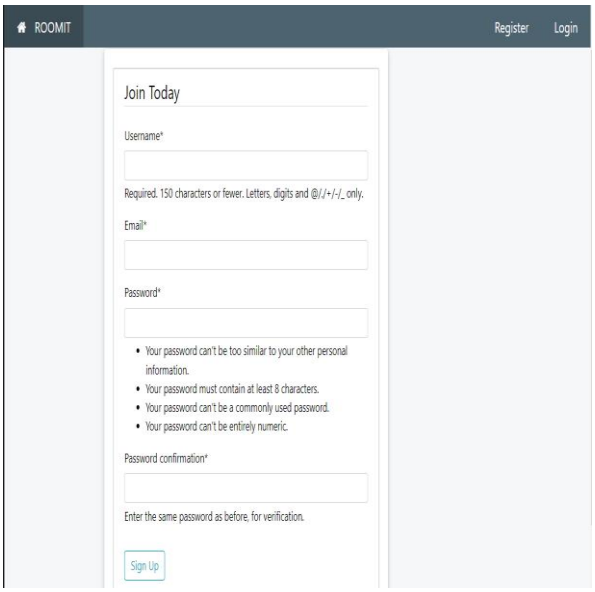
The selected tasks for performance testing were intentionally chosen as they represent the heaviest operations in terms of database interactions and computational complexity. By focusing on these tasks, we aim to evaluate the system's performance under demanding scenarios that involve substantial database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

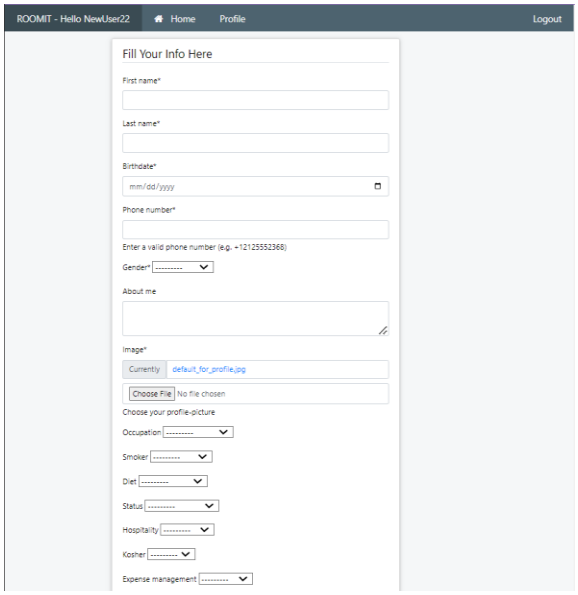
By running this performance test with Locust and scaling up the number of users, it becomes possible to measure the web application's performance under load. This module can help identify potential performance bottlenecks, such as slow response times or resource limitations, allowing developers to optimize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

CHAPTER 6 - USER INTERFACE DRAFT

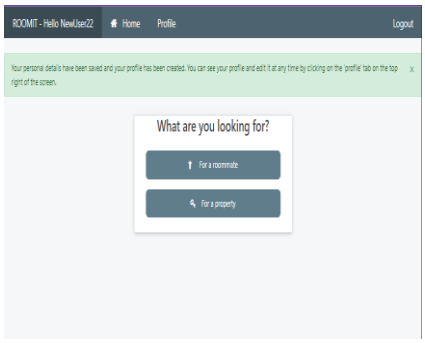
These are the screens from the website we built using HTML & CSS.

Here are the screens which are coming while signing up to the application:

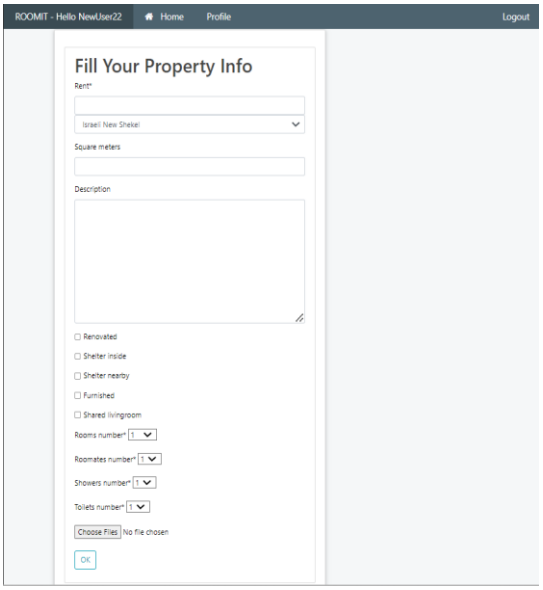


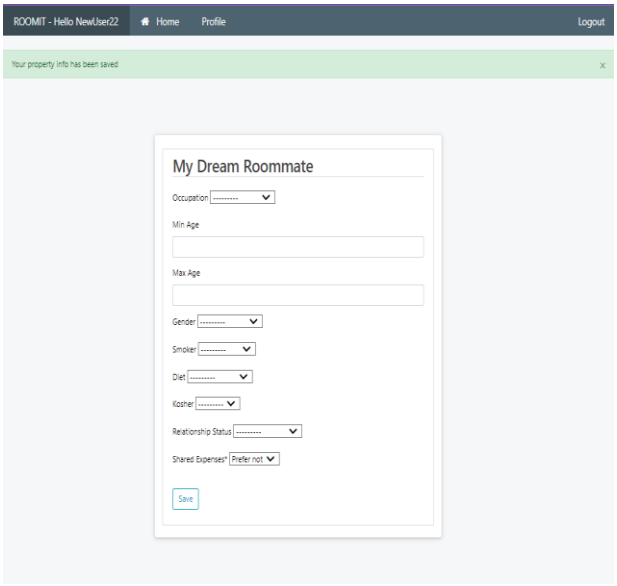


The screen that allows the user to choose his status in the system:

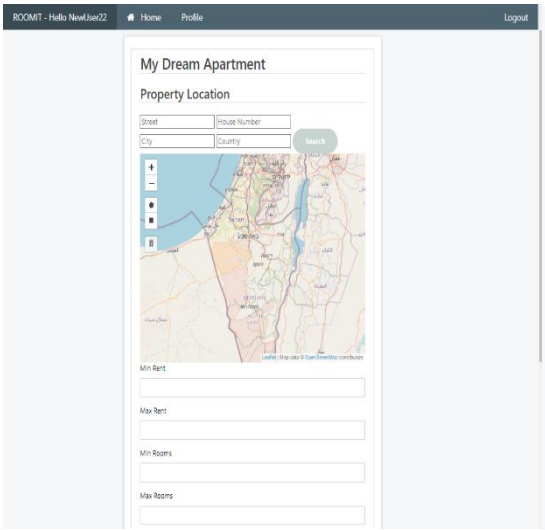


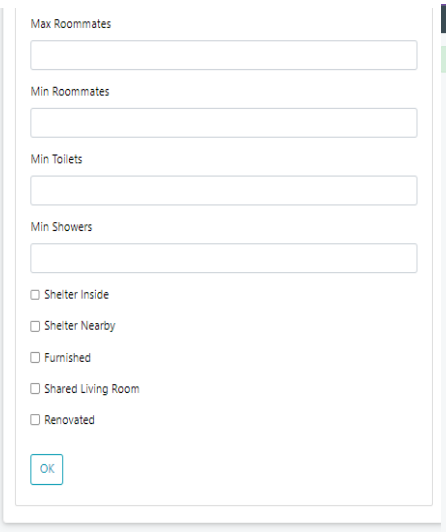
If the user is looking to insert a roommate to his apartment here are the screens for continuing sign-up:

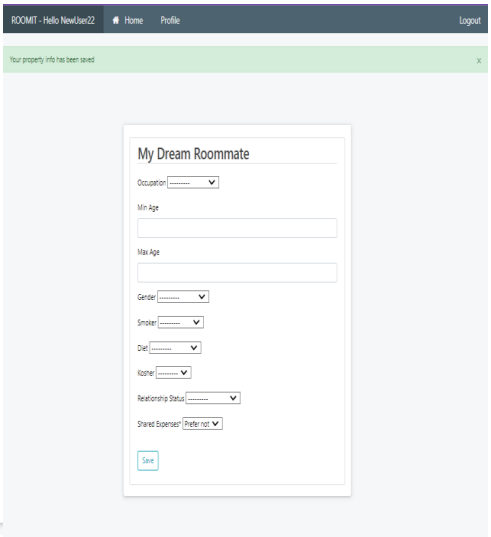




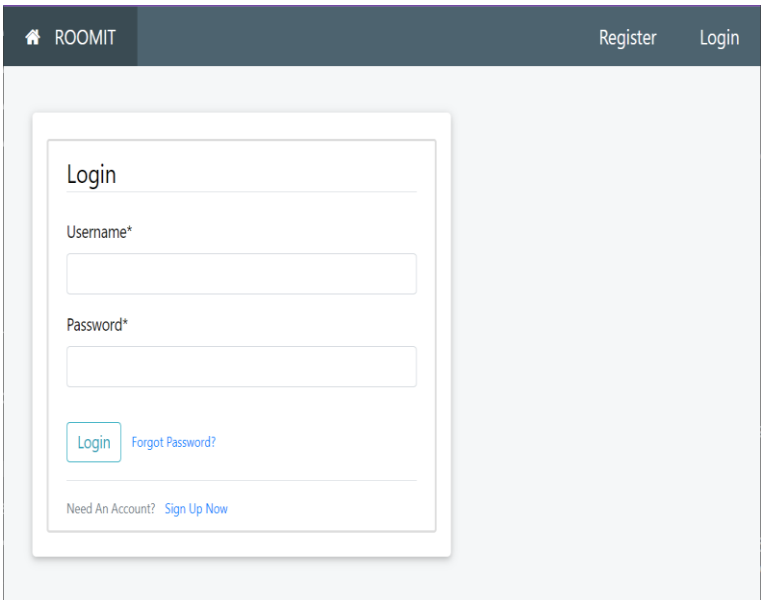
If the user is looking to enter an apartment here are the screens for continuing sign-up:



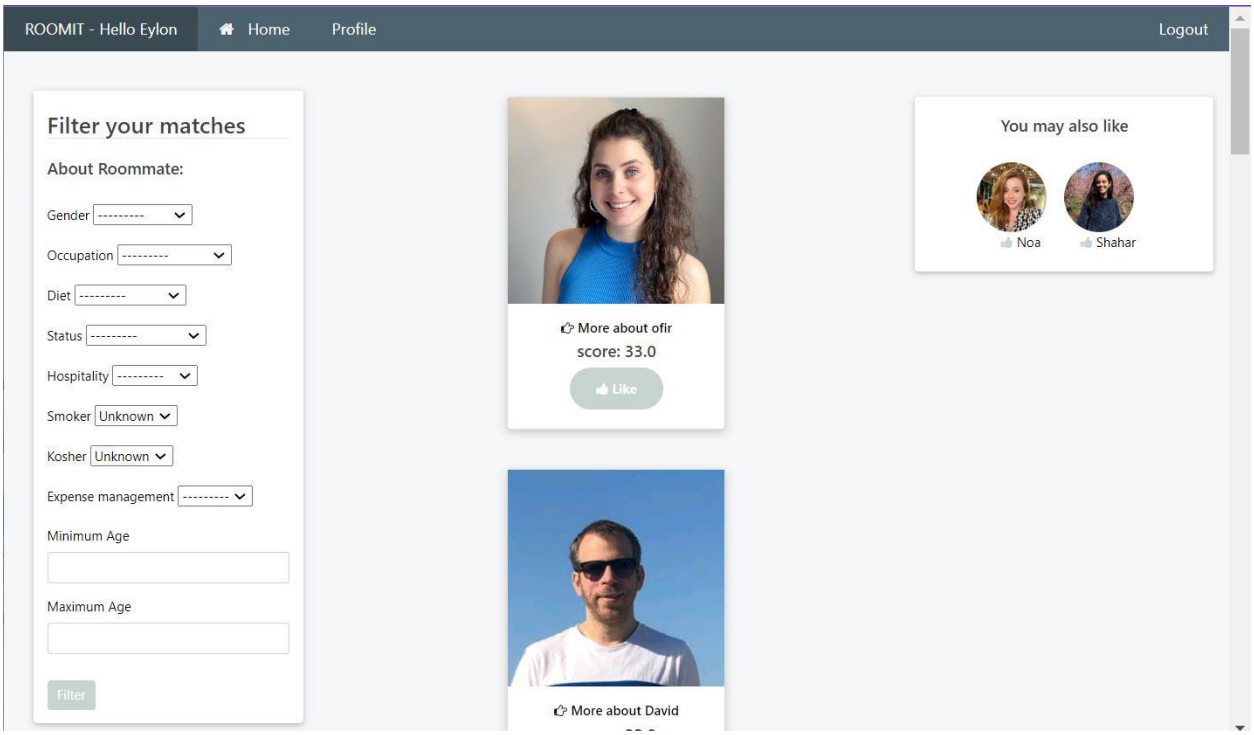




The sign-in screen:



If the user is looking for a roommate to insert to his apartment, here are the screens through which he can see his personal information, scroll for other roommates:



If the roommate is looking to enter an apartment here are the screens through which he can see his personal information, scroll for other roommates: