Roommate Recommender

Student 1: Georgina Scanlon ID Number: 19392373
Student 2: Rhea Varkey ID Number: 19452962

4/3/2022

1. Overview

This project is targeted at young renters who are looking for shared accommodation and homeowners needing a platform to advertise their property. It is similar to a matching based social/dating up, which gives you results based on the preferences/ needs you input on your website which works as a filter and provides you with the best property/roommate and these preferences can be changed/updated anytime in your profile.

When you first visit our page you must sign up/ register an account, this will allow you to personalize your profile by uploading your own profile picture, writing your own bio and having your own filter of what you want in a roomate/property (eg. wheelchair access, smoking etc…) and as mentioned above this can be changed anytime. Some of these filters are basic yes or no status while others are based on a scale of 1-10 like preferences such as cleaniness or nosiness.

If you are a homeowner, once logged in you have the option to add a property in the site's listing. By clicking on your profile dropdown you'll see an "Add Property" button which takes you to a form where you must fill out the information needed (eg. rent, wheelchair accessible etc..) and a lot of these information are mandatory. Once you have finished writing up the form you can just click the submit button and it'll be part of the listing.

When you're looking for your new roommate/house the site will give you the closest match based on your preferences, for example if you need wheelchair access you would receive recommendations of property which have wheelchair access. You then have the option to like or dislike the property/roommate. If you have disliked the recommendation the option is removed and if you like the option you'll receive the contact details for the house/roommate and can then call or email the homeowner and or roommate from there on

2. Motivation

Finding affordable accommodation has been a growing problem in Dublin city and for many people who are willing to share accommodation with others, there are few easy ways for them to find suitable roommates.

We got this idea by looking at different places to rent in Dublin. As you can imagine renting in Dublin is not cheap and often requires two or more occupants AKA roommates to split the rent between in order to have a roof over your head without scrambling for cash or opting for a loan.

Most of the time your roommate is someone you know or your assigned one in student accommodations, but this is not the case if you are outside student accommodations or you are a graduate looking for a place to stay. When looking for a roommate people often write up ads or hang up posters which is pretty time consuming and requires a lot of energy or you might not have many engagements with your ad. And most apps or websites that set you up with a potential roommate often have vague information, not many recent users or even the fact that the app or website is generally not the best to navigate around.

That where the idea for our project came along, it helps students like us or any recent graduate who has the paycheck to afford rent find roommates who have the similar goal and interests as us in a quick and easy way and you can take your time going through each profile to see if they are a right fit for you and all you need to do this sign up and say what type of roommate you want as well as the amount of rent you and your roommates need to provide.

This could be sold to universities which will have a large number of students looking to rent a room and don't mind sharing the living space as well as find out and see what type of roommate they could live with on the property. This can be useful for any student accommodation organization, as they can hand over the roommate assignment responsibility to the renters so that there is less work and the renters are happy with who they share their living space with, this can apply to any renting business (e.g Rent.ie) who can accommodate to students/young renters as well as new landlords to get them started.

3. Research

3.1 Database

For this component we used SQL to store the data we had retrieved from the user through their profile and registration information, as well as any property information they shared if they wanted to advertise their home. The reason we used SQL for this component was because we were more familiar with it through previous modules. It is also very easy to store and organize a large amount of data in SQL, this allowed less time in working with data and made it easier to manipulate or fix the data as we went.

3.2 Framework

Through our experience with our Full Stack module and advice from our supervisor, we decided to use Django as our framework. Some of the reasons we picked this as our framework was because, it contained its own web server which proved to be very useful in running and testing our web pages, contained a HTTP libraries which provided a large number of imports, middleware support and a Python unit test framework which was the most important as we used Python as one of our main coding languages for the backend of this project.

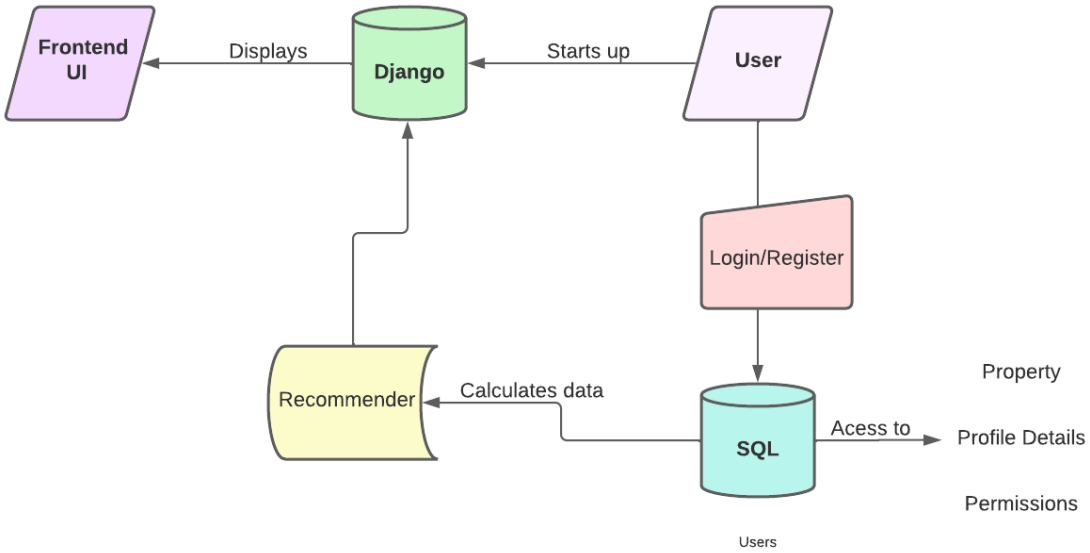
3.3 Recommender System

The recommendation function was the main part of our project so it was important to ensure that it worked accurately and that we could implement it well. We researched a few different ways that we could implement it. We found that it would be possible to work it out similar to how a Divide and Conquer algorithm works and that doing it this way would be accurate while being quite easy to implement. The way this worked was to simply break it down into finding the difference between each individual attribute for the user and property and then adding those values together to find the overall euclidean difference between the property's attributes and the user's attributes. During our research we also found that it would be important to take into account the weight of different values. For us, the price/budget values would usually be a lot bigger than all of the other values and therefore the difference between them might also be bigger. For example, a difference of 5 for the loudness attribute would be quite big, but a difference of €5 would be negligible for the price/budget attributes.

4. Design

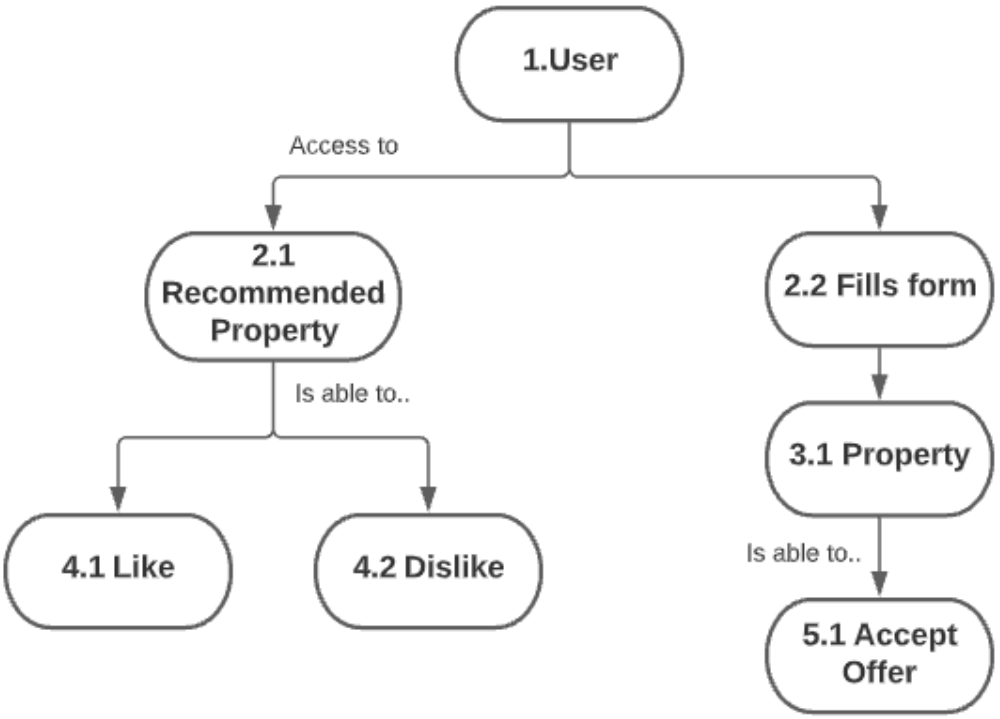
The design for our website has the basics of any social app, with the registration and login pages and data. The only difference is that we have merged the aspects of a house listing features (list of properties). Below you can see our systems architecture and our main component for our site

4.1 System Architecture

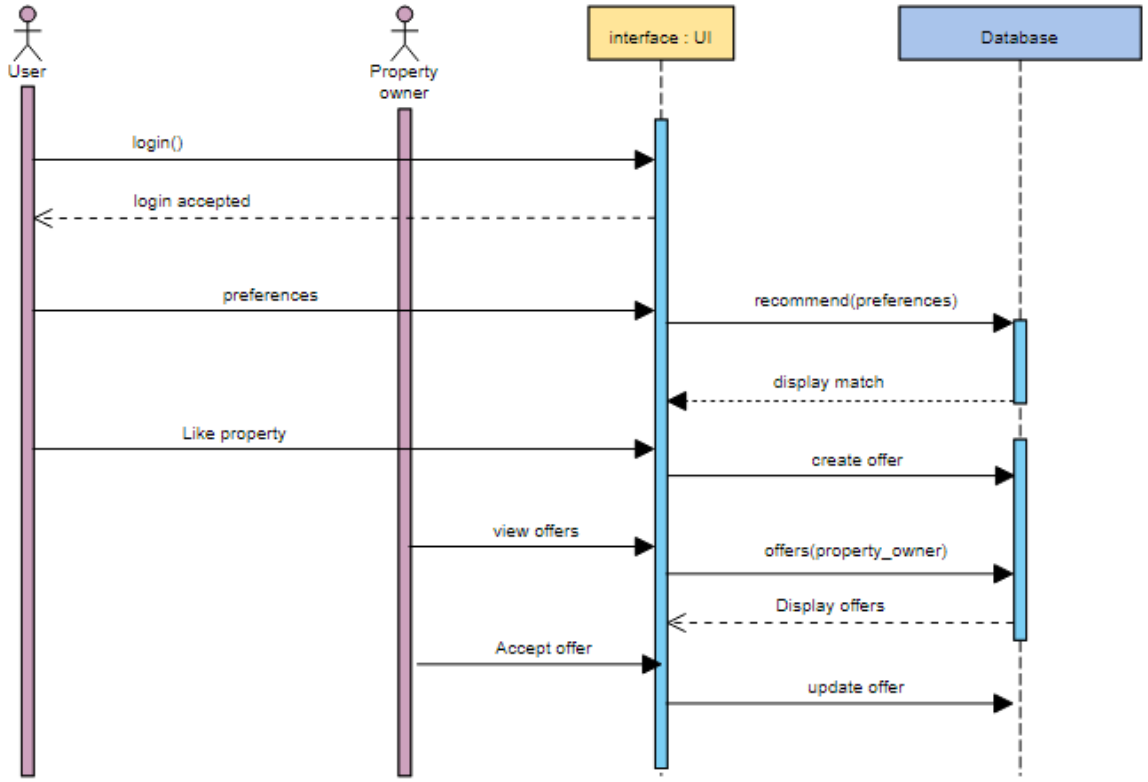


Our website starts off with Django starting up the frontend, from there the user can either register and log in to the site. The user's data is stored in the SQL database along with their preference and property details. From there the data calculates the best property for the user based on their preferences and is sent to the recommender page which Django displays in the frontend.

4.2 Recommendation system



Data flow diagram showing how a user is able to get recommended a property or add a property themselves. If they get recommended a property they are able to Like or Dislike it. Liking it will send an offer to the property owner, who is then able to accept the offer.



Sequence diagram showing how users would interact with the recommendation system and how they would create an offer if they liked the property.

5. Implementation

5.1 Recommendation function

We tried to keep the implementation of the recommendation system quite simple while also ensuring that it provided accurate matches. First, as there are some things that a user might require such as Wheelchair Access we filtered out any properties that didn't match these boolean attributes:

```
# Filter properties depending on hard requirements the user has set
if user.smoking is True:
    all_p = all_p.filter(allows_smoking=True)
if user.has_pet is True:
    all_p = all_p.filter(allows_pets=True)
if user.wc_access is True:
    all_p = all_p.filter(wc_access=True)
if user.re_gender != "4": # If the user has a gender preference
    all_p = all_p.filter(gender=user.re_gender)
```

We also filtered out any properties that the user had already disliked. Then we used these filtered properties to find a property that closely matches the user's attributes. We did this by making a list of the user's attribute values and then for each property making a list with the property's attribute values. Then for each attribute, we found the difference between the values and added that result to a total. Once it goes through all the attributes, we add the total difference to a dictionary with the value mapped to the related property object. Then when we go through all the properties, we get the property with the smallest value, as that would be the one with the most similar attributes.

```
for p in all_p:
    # Check if the user has disliked the property, if they have ignore it
    if Dislikes.objects.filter(user=user, property=p).exists():
        continue
    else:
        l = [p, p.cleanliness, p.noise_level, p.age, (p.price/5)]
        sum_l = [sum(l[1::]), sum(userl)]
        max_v = max(sum_l)
        total = 0
        for a in range(0, len(userl)):
            total += abs(l[a+1] - userl[a])

        d[l[0]] = total

most_similar = min(d, key=d.get)
```

In order to even out the attribute values, we changed the weight of the budget/price attributes. This was because all of the other attributes were more limited, whereas a bigger difference between price and budget would likely be accepted. We didn't change the weight too much though, as we did believe that many users would place more importance on price than the other attributes.

6. Problems and Solutions

6.1 Creating Static files for CSS files

When we were first testing our HTML and CSS files we had noticed that our CSS wasn't working for our webpages. At first we were slightly confused as to why it wasn't working as we made sure the href was correct and after a few minutes of looking through the Django documentation and checking previous solutions on Stackoverflow we realized that we were lacking a static directory where our images and CSS would be stored and we were not correctly linking to it. Fixing this solved our problem and allowed the CSS to come through on the webpage and made it easier to store the CSS and images to be stored and managed in a separate directory.

6.2 Migration Errors

Rhea had some errors involving Python migrations which lead to some of the webpages producing an error when clicked, she tried to fix it by updating her PyCharm from Git but when that failed she ran "python manage.py makemigrations" and "python manage.py migrate". However when that didn't work she noticed from the error message that she was missing a column in her SQL database and had to manually add the column she was missing as updating using Git wasn't doing it for her. This allowed the webpages to be accessed normally without any errors.

6.3 Coming up with the Recommender system

As the recommender system was the main part of our site, we had to ensure it was implemented well. Coming up with how we wanted to compare the properties' attributes to the user's took a while and we researched a lot of different options before settling on the way we chose to do it. Overall, we picked our way because it was relatively simple to implement and it allows us to easily change the weights of different attributes if we wanted some to be considered more important.

6.3 Handling Offers

It was difficult to know how to design the offer/liking system for when a user likes a property and wants to get in touch with the owner. It was important to only give contact information for each user to the other when the offer was accepted by the owner. We struggled with finding a way to achieve this but eventually found with advice from our supervisor that we could create an offer model with the attribute 'is_accepted' and within that model store both the user objects and the property object in order to access information about those objects. Then on the frontend we could display the contact information to both users only when the owner of the property also accepted the offer.

7. Testing

For all kinds of tests include a text paragraph explaining the tests run (how were they run) + any screenshots / results

7.1 Unit Testing

The most important part of the project to test was the Recommendation system. We tested users and properties with a large variety of different attribute values to ensure that the correct property was matched each time. We would create test cases where a property should be a 100% match (i.e all attributes would be the same) as well as cases where all the properties' attributes were quite different from the user's to ensure that the correct match was found in each scenario. We also tested cases where multiple properties were exactly the same, to ensure that a match was still found. In the case that that happened, the property that was submitted first would be returned.

We also tested the Offer system in order to make sure that changes to an offer (i.e it being accepted/declined) was displayed for both users involved. We made sure that we could correctly link to the related property and to either user's profile and that we could display either user's contact details. At first, we had some issues with this and found that the best way of saving the users' and property's information to the offer model was by using the save function to store the information when the offer is made.

7.2 Integration Testing

We ran tests to ensure that all of the components of the website worked together. In order to do this, we tried using the site from the point of view of a completely new user.

We first ran through a scenario where a user looking for a property joins. We went through the account creation process, checked that they got correctly recommended a property based on the inputted attributes, we checked that disliking properties ensured that they weren't recommended again and then sent an offer by liking a property and checked that the offer was displayed correctly on the offer page. We made sure that the property that was liked no longer displayed the like and dislike buttons, instead showing a message to confirm the offer was sent. We also checked that the offer appeared for the user that owned the property and that accepting the offer was reflected correctly for both users.

We then ran through a scenario from the point of view of someone adding a property. We ensured that the form for adding a property worked correctly and that the user could edit the property they owned but not a property they did not own. We also checked that the property added showed up correctly on the 'my properties' page.

7.3 UI Testing

To test that everything on the UI was working correctly. This mostly involved just checking that links worked correctly and that each page could be easily navigated to. We also tested the UI on multiple browsers and devices to ensure that the styling showed up correctly on each one.

7.4 User Testing

We asked a friend who was not involved in the project to test the website in order to hopefully get some realistic user feedback and find out if there were any improvements we could make, particularly to the UI design. In order to get the best

feedback possible, we asked him to simply use the website as he would use any new site and to keep in mind any issues he ran into or anything that he thought could be made more clear in the UI. As well as that, we asked his opinion on the recommendation system, to identify if there might be a problem that we hadn’t noticed. We asked him if he thought the recommendations would be accurate for him