Airbnb App

M165 course E-Commerce technologies

Team 10: Kalopisis Giannis 7115112200011
Michas Georgios cs2190024

Introduction

This work aimed to develop an Airbnb application for renting rooms/houses for devices with the Android operating system. The development was done using the Android operating system for the client side and the Spring Boot framework for the backend, both in Java. The PostgreSQL relational database was used to store and retrieve the data.

In the Technologies chapter, we describe the technologies chosen and why. We also mention which version of Java, Android SDK, Spring Boot, etc. were used. Before analyzing what was done in the project, we should analyze how we used the data given to us, which of them were used, and how. In the Project Description chapter, we describe exactly what has been implemented from the job requirements and what has been added that we thought was necessary as well as optimizations that we made and consider worthy of reference. In the Visualization chapter, we present some screenshots from the application for a quick overview and familiarization with its environment. In the Recommendations chapter, we analyze how the bonus part of the Matrix Factorization algorithm requested by the speech was implemented and the design decisions we made on it. Finally, in the Conclusion chapter, there is a recap of the work carried out, the difficulties faced, as well as how these were overcome.

Technologies

Client-side (Android)

Android 13 (API level 33) Tiramisu was used for the Android development environment, as had been suggested in the lab. This was chosen as it is the newest API level and quite stable. It should also be mentioned that an Android virtual device was used to develop the application and it is the Pixel 6. Its characteristics are as follows:

- Resolution: 1080 x 2400
- Dp: 412 x 915
- Available storage: 6 GB
- Cores: 4
- RAM: 1500 MB
- Heap size: 1000

Server-side (Spring Boot)

Java Version: 20.0.2
Spring Boot: 3.1
Postgres: 14.9
Build tool: Gradle

Database

Application data are stored in PostgreSQL relational database and more specifically PostgreSQL version 14.9. This database system was chosen for its performance, scaling to the large volume of data and also its ease of use as several queries in the backend were written by us to optimize their performance and to do several things inside the database without pulling data, as I/O would delay the application.

Compilation and Execution

Before running the application you should create a Google API key and insert it in the AndroidManifest.xml file for the maps to run correctly. The key does not need special permissions. In case you set special permissions, it should definitely be able to be used for the Android app and access to all map APIs with this key should also be selected.

The android:value="YOUR-GOOGLE-API-KEY" field within the <meta-data/> section should be populated.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools">

   <application
       .
       .
       .
       android:theme="@style/Theme.FakeBnB">
       <meta-data
           android:name="com.google.android.geo.API_KEY"
           android:value="YOUR-GOOGLE-API-KEY" />

   </application>

</manifest>
```

Data and Parsing

In order to make our application “real”. We have implemented CSV parsers in Java for all the objects in our back end. So, when the application starts we populate the entire database with our custom input data. Moreover, We have carefully loaded all the recommended data by the instructor in order to test our Matrix Factorization algorithm.

Project Description

In the application, everything that was requested in the project has been implemented. A user can either take on the role of a renter and search for rooms/houses for rent or the role of a host to add new houses for rent and modify older buildings they have added. Each user

can also modify their account information, change their profile picture, and get the other role they don't have (user/host). Also, users can exchange messages with the owners of the rooms/houses they have rented and the owners can reply to the messages that come to them respectively. In other words, there is the possibility of live chat with many users of the application at the same time. We should also mention that when the user has to fill in a field and has not filled it in, or has filled it out incorrectly, an appropriate error message is displayed either in red letters above the field or with a Toast message in the lower part of the screen. If the user has to choose a date, either to search and rent an apartment or because he is a host and wants to enter or modify the details of an apartment, the dates start from the local date and he is obliged to select the initial date and then the final date. Otherwise, the system does not let him choose the final date and displays an error message. Also once he selects the start date the end date field is limited from the day of the start date onwards to avoid mistakes and malicious actions. Below we further analyze the capabilities of each user according to their role in the application. We also mention choices made either in the front end or in the back end, problems that existed, and how they were solved.

All communication between the Android application and the server is done with REST API calls and using SSL/TSL for security.

Android app

In this chapter, we will analyze the capabilities of each user according to their role.

Tenant-User

The possibilities that a user using the application as a tenant has are:

- To see on his home page room suggestions according to what he has rated or searched for.
- To search for rooms/apartments by region, type, and date.
- Click on a rental and see more details about it.
- To communicate with the owner of the rental via chat.
- On the page with all his conversations, he can see, in addition to the name of each interlocutor, his photo, and the last message of the conversation. Also if the message has not been read it is in bold black letters, while if it has been read it is in gray letters.
- To see reviews of the rental and the owner in general.
- To make a reservation, only if the room appeared after a search.
- To write a review about the room, the reservation, and the host, only after the reservation date has passed.
- To change his profile information and image.
- To take over the role of host if he/she does not already have it.
- Choose from the list of his conversations who he/she wants to chat with now.
- Register and enter the application.
- Finally, on all pages deemed necessary, there is a bar at the bottom of the page, from which the user can:
    - Navigate to the chat page
    - Navigate to the profile page

    - Transfer to the host's main page (without logging out), if he/she has this role of course.

Host-User

The possibilities that a user has when entering the application as a host are:

- To see on his homepage all his properties with some basic information about each one.
- By clicking on the properties he/she can see and modify their information, the images he/she has added, or even delete them completely.
- On the chat page, he/she can see all messages from tenants and reply to them by selecting a chat room.
- On the page with all his conversations, he can see, in addition to the name of each interlocutor, his photo, and the last message of the conversation. Also if the message has not been read it is in bold black letters, while if it has been read it is in gray letters.
- On the profile page, he/she can, like the ordinary user, change the information, and the profile picture and assume the role of an ordinary user if do not already have it.
- Register and enter the application.
- Finally, on all pages deemed necessary, there is a bar at the bottom of the page, from which the user can:
    - Navigate to the chat page
    - Navigate to the profile page
    - Transferred to the user's main page (without logging out), if he/she has this role of course.

Optimizations

In this section, we will mention some of the optimizations made to improve performance either on the Android side (frontend) or on the Spring Boot server side (backend).

Android Optimizations

The optimizations made on the Android side are:

1. When we bring the rentals, either on the home page through the recommendation system, or through the search, we bring the image of each building through API calls (asynchronously) so that the application and the display of the buildings are not delayed. Thus the rental may be seen until its image comes from the backend.
2. Another reason they come one by one in the above case is so that we don't have large requests that can more easily fail.
3. The same happens with the images of the rental when the user wants to see all its details (after clicking on it), as well as with the photo of the owner.
4. Also, all the information comes asynchronously so that there is no problem with the network and the whole home page is not waiting for the user's name or photo for example.

5. Even when the owner changes the images he has uploaded for the respective rental, only the new images and the list IDs of the old ones that were deleted (if any) are sent back to the server, so that the request is not loaded. Two requests are made, one for the changed information, which also contains the list of deleted images, and one with the new images if any.
6. Also where necessary we bring the data using pagination. That is, they are loaded gradually as the user scrolls up or down. When the project was announced, it was only requested for the rentals that were the results of the search. We saw fit to apply this to the owner's rentals that appear on the host page, as well as chats on the chat page and chat messages as well.
7. Regarding the maps, every time the host either puts the address in a new rental or modifies it when changing the details of the rental, the application understands what is written and automatically resolves the address in Google Maps. In order not to call the Google API for each letter, we made a Runnable function, which resolves the address every two seconds. But this timer resets every time the address changes so that it gives the user enough time to type and thus actually do the resolution and render on the map when he has actually finished typing. We did this because when we initially had to call the API for each letter the application would hang and close many times as all these API calls were made in the main thread.

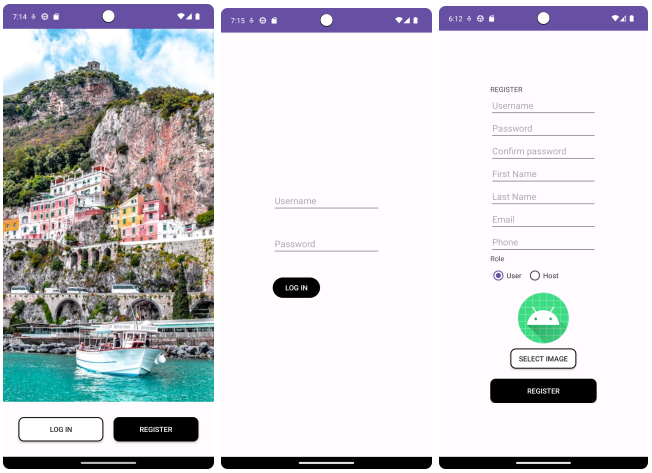
Spring Boot (Back End) Optimizations

1. We have carefully selected the minimum field types for all our database objects. For example, for costs, we use the BigDecimal for not losing precision, for the number of bedrooms we use the short type.
2. For all endpoints that return a big amount of data loaded (not only for the search), such as the listing of items, we use paging.
3. We don’t preload images when they are unnecessary. We have an endpoint that you can hit in order to get each image each time. (less loading)
4. We store the images directly in the file system, and in the database, we only store the path for this image/name in order to load it. (blob approaches etc. are too slow and messy).
5. We have implemented a controllers/services/repository approach for writing clean and “easy” expanding code.
6. We have implemented exception handling for all the possible cases that result in errors.
7. We used JWT authentication to make our lives easier and have an effective and optimal authentication/authorization mechanism. Moreover, all the user passwords are encrypted first and then stored in the database.
8. For the search we have used the Specification mechanism of the spring boot and we use district, city, country, availableStartDate, availableEndDate, rentalType, and maxVisitors parameters. This enables the ability to a fast/elegant way of creating dynamic queries that can be applied with all the combinations of the input parameters.
9. We have also implemented logging. This means that when a user of examples does a search we store all the parameters and also all the apartments we see in order to use them for recommendation and also to get insights into his interests on our application.

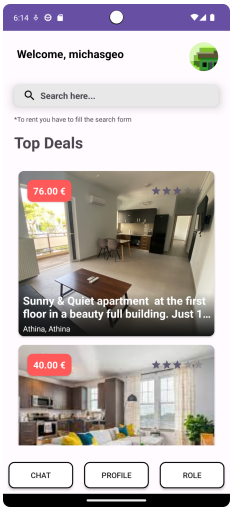
Visualization

In this chapter we present some images from the application:

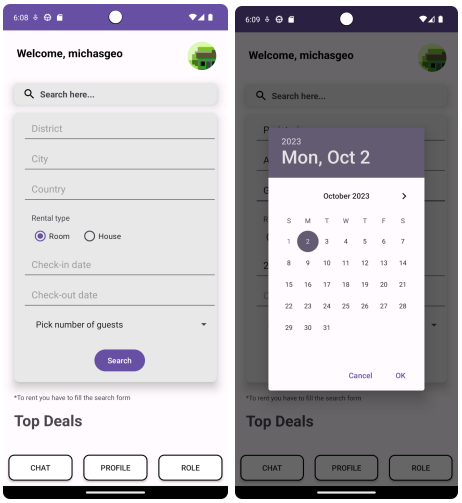
- Starting, Login, and Register page



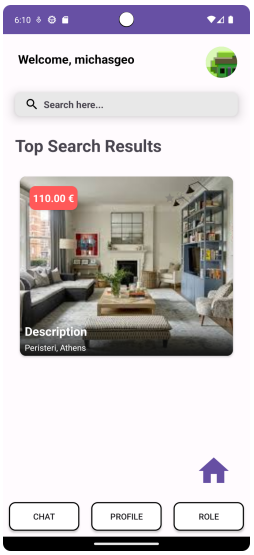
- User page
  - Main page



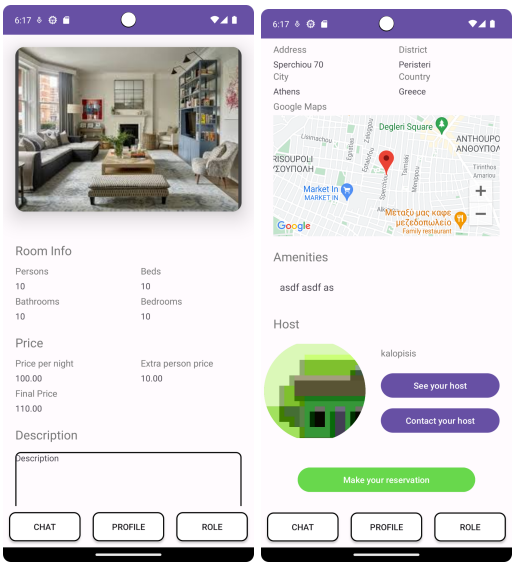
  - Search fields



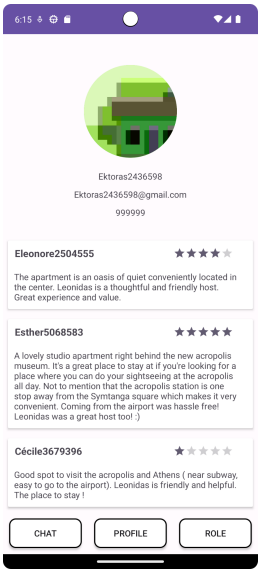
- Search results



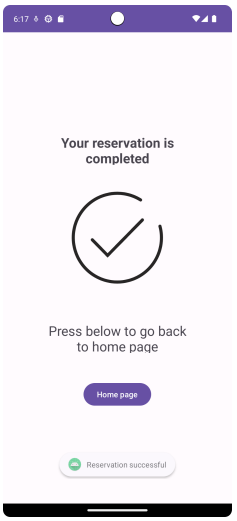
- Rental information and reservation page



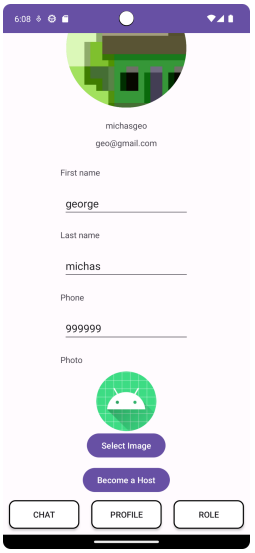
- View host's and apartment's reviews



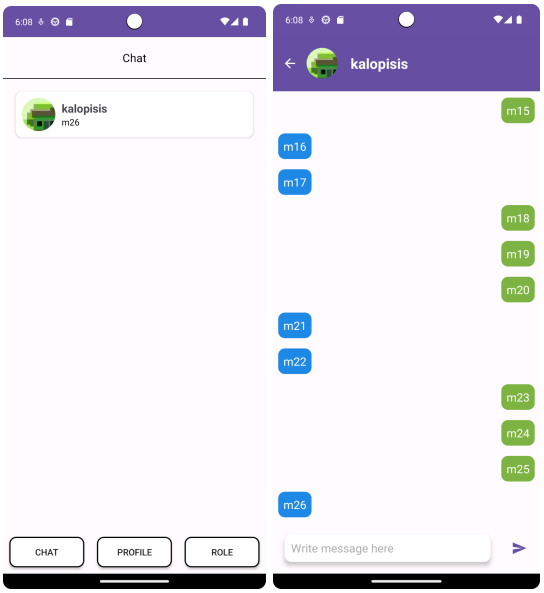
- Reservation Done page



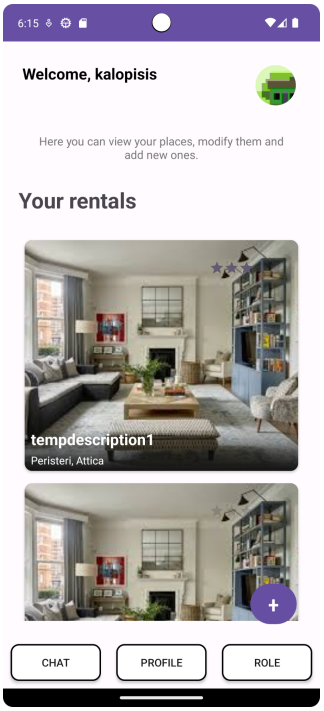
- Profile page



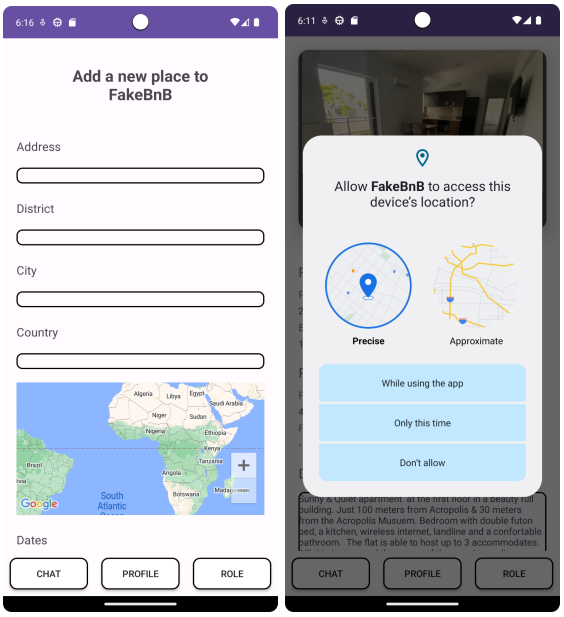
- Chat page and Messages



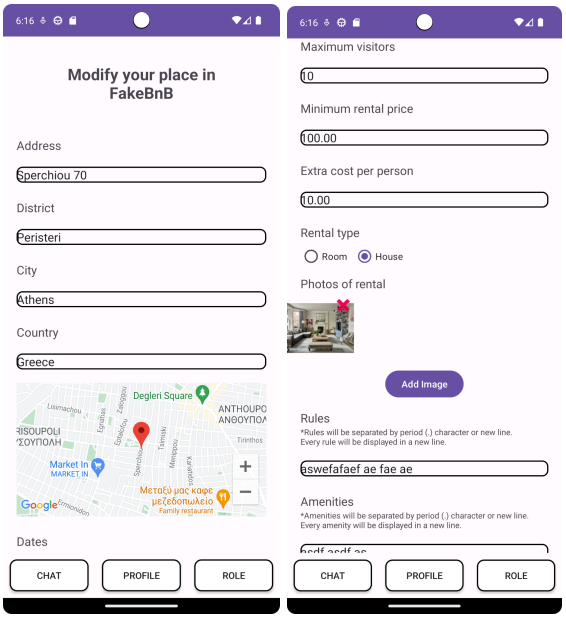
- Host page
  - Main host page



  - Add new rental page



- Modify information of existing rental



Extra Bonus - Recommendation

For the bonus we implemented our own Matrix Factorization algorithm based on the slides of the lectures. For this, we create the array of #users (registered users of the application) X #items (apartments) based on the rating that has been applied to each apartment. Moreover, in order to avoid data sparsity we use the searches (apartments) that he/she has done in combination with apartment specifications and use as a rating the average of them with some specific weights (normalized to [minRating=0, maxRating=5]). Then we execute the algorithm and return the top 5 items of the predicted results. Moreover, we have parameterized the algorithm as a result of the K (latent space), V and F init distribution vectors (normal, uniform), η (learning rate for gradient algorithm). As a result, you can change all the variables from the application properties and run the algorithm for different configurations.

Conclusion

Through this project we saw how a real Android application can be created, which interacts with the backend. We have seen how the communication with the backend and the database should be designed, and the careful design of API calls that do not burden the system, how to design an efficient database to optimize the query time, but also how to represent it through JPA. Regarding Android, we saw how this operating system works, what its life cycle is, and how we interact with it. In general, we faced difficulties about how to view the data, and how to bring it little by little with the scroll, especially in the messages. In the backend, we faced design problems in the database, so that we could make a decision on how to build it, and how to implement the paging technique, the search with dynamic filters was complicated as there were available dates of the rentals that were empty, and finally the recommendation was quite complex and required a combination of many techniques and

knowledge in order to come out correctly and efficiently. In closing, we should mention that parsing the data, cleaning it, and matching it to the requirements of the project was a part of what made it difficult for us. Nevertheless, it was a huge task (and we think it should be a bigger part of the final grade) and we saw a lot of contemporary technologies that are used every day.