CS 457/CS 557 – Database Software Design

SherEats

A restaurant recommendation system based on Android, PostgreSQL and Firebase

---

Table of Contents

- 1. Contribution of each member
- 2. Project description
- 3. Modeling Scheme
- 4. Implementation
- 5. Screenshots of the user interface
- 6. Conclusions and future works

Dependencies

- Android
- Kotlin
- Java
- PostgreSQL
- Tomcat
- Firebase

1. Contribution of each member of your group

Full name	Contribution
Yi Ren (002269013)	Android Development and HTTP data transfer
Wentao Lu (002276355)	Web server and JDBC integration
Junjia Lin (002268506)	Firebase integration and documentation
Luyun Nie (002268087)	PostgreSQL database and documentation

2. Project description

2.1 Give a description of your project

Assigned by a database final project combing maps, our team spent four weeks creating an android application based on our backgrounds and by researching other food recommendation APPs. In this whole project, we adapted modern and common techniques like Android, JavaScript, PostgreSQL, JDBC and RESTful API to implement a client-server system. So far we've completed the features below:

- User **register** and **log in**.
- **Restaurant recommendation**, which include basic **information**(avg price, description), **location**(Google map), and **menu**.
- **Dish recommendation**, which includes price, location, and rating.
- **Shopping cart**(add dishes into cart and checkout).
- **Order history**
- **Online chat**(Firebase).

2.2 Explain the different stages to realize your project.

Analysis

Restaurant recommendation services offer users to search restaurants by locates, ratings, pricing, and so on. Recently, not only foodies but also the public rely on those kinds of websites, especially on phone applications, seeking satisfactory restaurants or dishes. By researching two similar food recommendation applications: 'Yelp' (introduced by Adam (2021)) and 'Dianping' (a Chinese domestic platform), our group decided to build an Android 'mashup' application combining the Google maps and a back-end database as our final project. Also inspired by our Chinese international students' background and peaceful lives in the city of Sherbrooke, we named this project as Sherbrooke tastes, memorizing all things we've been through during the COVID-19 pandemic. This project was accomplished by four members and each of us has contributed responsible work.

Assigned a final project, our group took one week to discuss what would be done among the given project suggestions. Considering two features: two of us had experience building a restaurant database from a course named Software Engineering; another two members were intended to engage in Android development, we decided to build a restaurant-seeking database integrated Google maps, running on an Android system.

This application is aimed at three types of uses: the public, the restaurant, and a platform. For users, they can log in to the APP, search for the dishes they want, and go to designated restaurants or order online based on the scores of the dishes. Sometimes, they can also choose a restaurant that matches most based on the location and rating of the restaurant. The functions of order review and online chatting are additionally provided. For business owners, we plan to allow them to log in to the APP as well. They are authorized to edit restaurant's information, add dishes(items) and fix discounts. However, manipulating ratings and comments are not eligible on the platform avoiding malicious competition. The platform administrator's role is to maintain the platform usage environment. Privacy, security policies and contact information will be published by the administrator. Besides, he/she has the right to intervene in disputes between users and businesses. All assumptions come from online research by ourselves from two famous platforms: Yelp and Dianping.

The whole project will be uploaded to a cloud service that all users including the public, the business owner, and the platform administration can access the application. In this way, we have no needs to apply an encryption algorithm to protect our customers' privacy. Moreover, if time allows, we may learn some concurrency control methods to ensure every operation by our customers would be correctly generated and stored by the database. To realize further data marketing analysis, we are going to add some data buried points to record customers' records.

As final projects assigned before, we only have one month to achieve our goals. Although we assume that three types of users could log in to the APP, one month time may not allow us to complete the whole

construction. In terms of the demands of the final project, the most essential part of our work is to finish the public user building.

Design

Inspired by two platforms: Yelp and the Dianping, we spent three days discussing the designing details of this project. Firstly, we identified the use case as the customer, the restaurant owner and the platform administration based on the assumptions made one week ago. Then, following the software developing process, a class diagram had been made by confirming all functions and connections between each item. At last, our group did additionally process traversal to ensure our customers could use our APP fluently and explicitly.

For the database design, we did much more work than the other two steps to build our entire system. To have higher efficiency, we divided the construction task into three parts. Three team members were responsible for their respective tasks and completed the preliminary tasks for around one week.

The tasks that have been completed in the early stages had given us actual programming guidance. The actual operations were to obey the Use-case Diagram and the Class Diagram. Not surprisingly, we would create nine tables, they were users, stores, platform, items, user orders, user events, user account records, store events, and platform events. The tables of users, stores, items, user events, store events, platform events would have their primary keys such as user_id, store_id, item_id, userevent_id, storeevent_id, platformevent_id respectively. Other tables like user orders, platform, user-account record reference the primary keys. To avoid redundancy, all tables would be listed at the Appendix and the codes were appended in our submitted source code file. After creating all tables, we imported some data from Google for later testing. At the same time, we had also written some query statements to facilitate the construction of other system modules, such as calculating ratings, checking orders, and calculating total consumptions.

Development

In this semester, we had been introduced to PostgreSQL which was an open-source relational database. After several homework exercises, we were already familiar with the software and its language and can operate the interface smoothly. So, we naturally chose this software as the database building platform.

A Technical writer and editor named Alexander (2020) introduced a famous architecture style, Rest API, utilizing HTTP requests to approach and retrieve data. He also mentioned that the REST occupied less bandwidth that it had more efficiency with written by JavaScript and Python. We chose it as server accepting requests from all types of users with looping methods. To connect the database and the server, Oracle (2021) supplied JDBC: The Java Database Connectivity connecting any virtual database. It was a common tool that many instructions were easily found on the internet.

Testing

Three weeks had witnessed our hard work. We finally finished the whole construction of the application and were able to run it on the service. One member not involved in the the building process was responsible to test the application and gave the advice to do some manipulations within one week.

- **Verification**

After several rounds of testing and modification, each part of this system was designed according to the original sets of languages and modules. The tester runs each module of the system and they perfectly proceeded. No obvious bugs were suspending the test. All codes would be appended in the submitted

source file. However, certain tables in the database, such as store events and platform events, were not used and referenced in the architecture. Therefore, the tester inferred that some of the originally anticipated functions were not implemented.

- **Validation**

The tester firstly logged in to the application after registering an account. The personal information was successfully stored in the back-end database. Then he clicked the dishes button opening the dish page. On that page, he could browse, search for items and add items from several restaurants. All data stored in the database was retrieved and presented on the detailed page of each item such as location, ratings, and discounts. After choosing the dishes, the user could check out with one click and the order would be inquired on the personal page. If had any questions about the order, the user was also able to chat online with the merchant by clicking the contact button. Also, when the tester was interested in a restaurant, he may click the heart button to save that store in case forgetting. To the surprise of the tester, the function of online chatting derived a module for making friends on the platform. All customers could add contacts and chat on our platform that they may gain more helpful information from others. The user interface screenshots would be attached at the end of the article.

- **Maintenance**

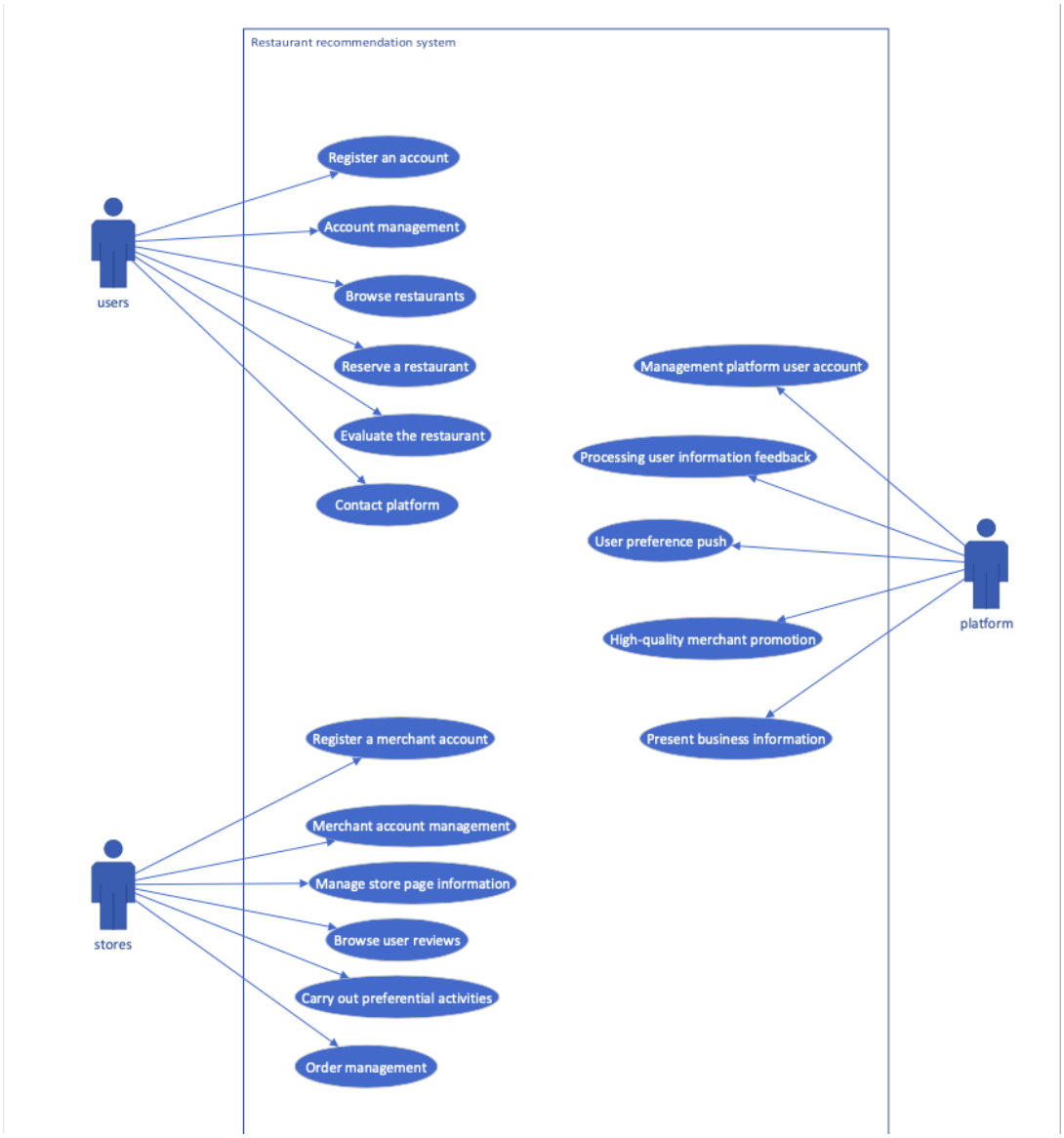
However, two functions were not applied in the whole system. The first one was the login by the stores or restaurants. The business owners had only to contact the platform or application staff to add dishes or fix information. The other one was the in-completed platform policies. There was no place to display all security or privacy principles that supported and informed all real users. In general, this software can be used by users normally, but it is still inconvenient for businesses. At the same time, work-related laws and regulations had not yet been completed. Therefore, there were still a lot of efforts that need to be made in the future.

3. Modeling Scheme

3.1 Use cases diagrams

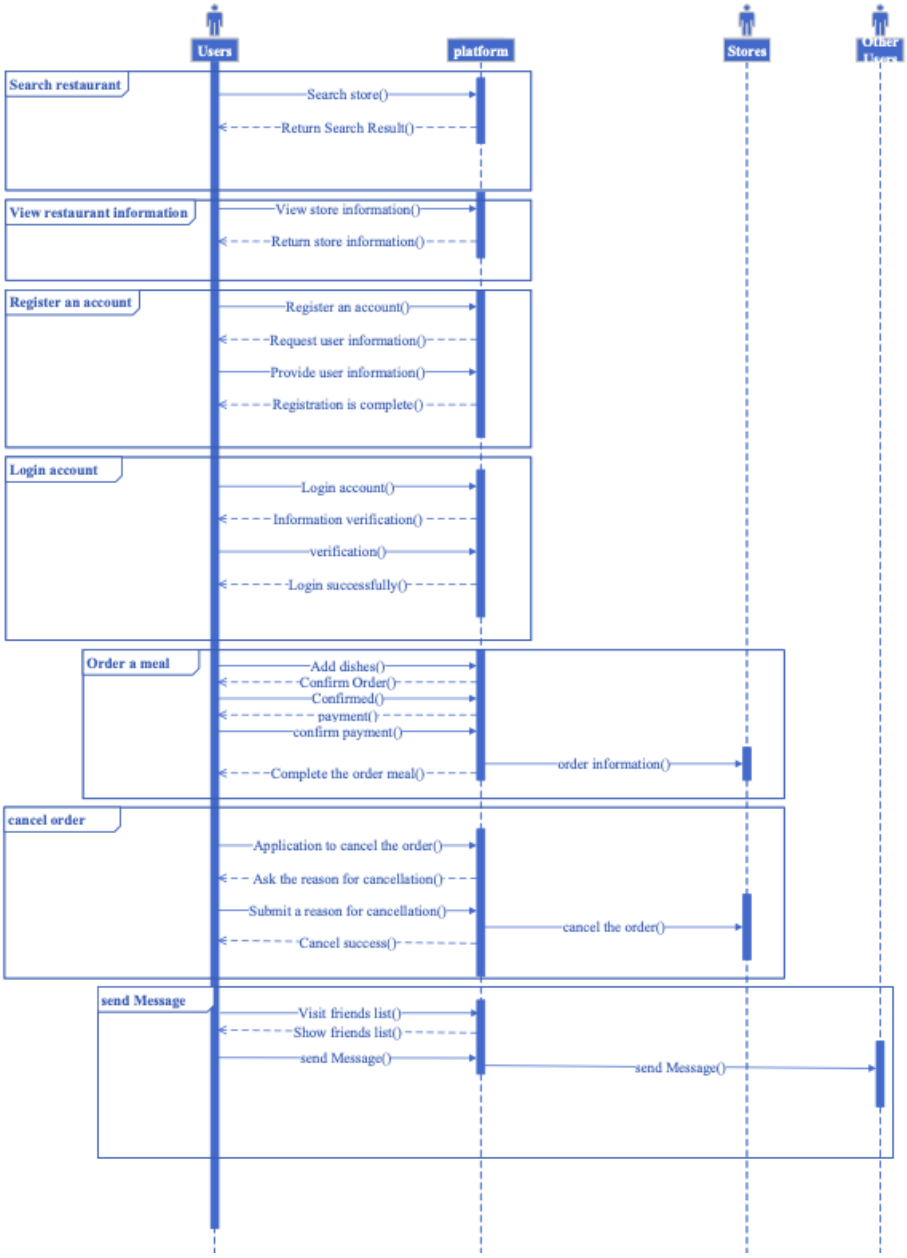
We considered three types of users: the public, the restaurant/the store, and a platform administration. The public users would have six main behaviors. When they first opened the APP, a login interface would be pushed on the screen. If the public user was a new to the system, a register page would be presented instead of the login windows. Also, personal information manipulation was allowed to be manipulated like other recommendation applications. The most important function of browsing dishes and restaurants shouldn't be absent in the main features. As well as making orders, some other ensuring user rights methods were included such as reversing orders and evaluating the restaurants. Each review and rating would affect the overall rating of a store or dish. For some unconventional situations, such as user complaints, incorrect information, etc., users can complain or modify information about businesses or dishes from the portal of contacting platform.

Same as the user side, any stores would have a register and log-in windows. After signing in, business administrators were able to upload their restaurant's information, such as the store name, location, and contacts, and so on. Displaying dishes was also a function of the restaurant owner, by adding dishes details like dish name, price, and genre. When a store wanted to push some discounts for promotion and analyze customer data, modules of deals and order management supported all the need. From the platform perspective, the platform administrator acts as a bridge and establishes the connection between users and merchants. Not only need to solve user problems, but also play a role in supporting business operations.



3.2 Sequence diagrams

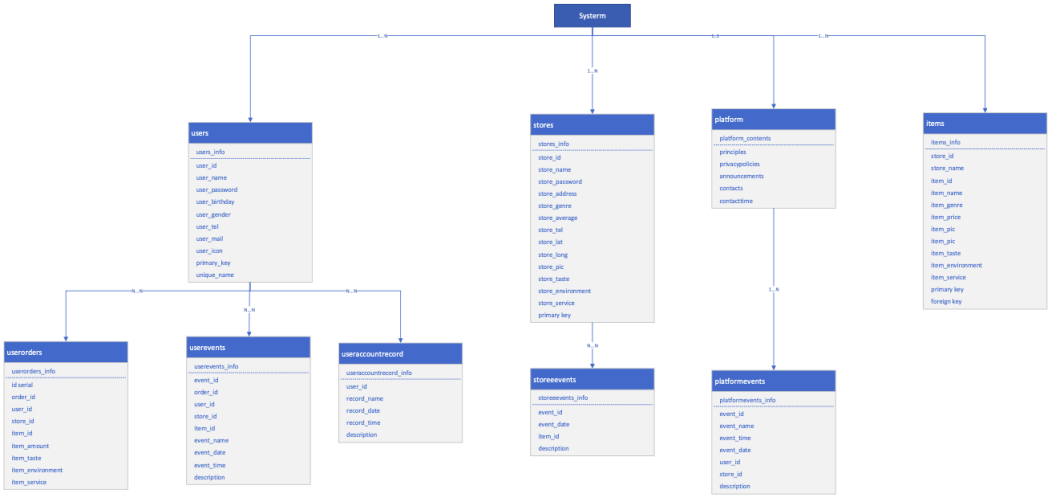
Considering the issue of time execution, we chose to make the user-side process first. Seven scenarios were deriving from our assumptions: searching restaurants, viewing restaurant information, registering accounts, logging accounts, ordering meals, order cancellations, and sending messages. Unlike the first four sequences interacting with the platform, the last three operations would be sent to the stores and other users.



3.3 Class diagram

The whole database system would have four main classes. They are the user class, the store class, the platform class, and the item class. As the figure showed below, the users_info, the store_info, the platform_contents, and the item_info were associations of each class respectively. Users had an order service relationship with user orders and a behave recording relationship with user events. To back up the user's information record, users also had a relationship with user-account records.

Simultaneously, stores and the platform also had relationships with store events and platform events to back up restaurant administration and platform manipulation records. To make searching for dishes more convenient, we have separated the items/dishes of the restaurant. Each item had its item_id.



4. Implementation

Tool	Description
Android Studio 4.1.1	Android APP development
IDEA 2019.3.4	Web service development
PG Admin 4	Database management
Sublime Text 3.2.2	SQL development
SharedPreference 1.1.1	Local Persistent storage
Google Gson 2.8.6	Convert Json to object
Google Firebase 19.7.0	NoSQL database for message synchronization
Google Map 17.0.0	Show location of restaurant
Maven 3.6.1	Web service package management
Gradle 7.0	Android package management
JDBC(PostgreSQL) 42.2.19	Connect with database
Java Servlet 4.0	Receive and handle HTTP request
Tomcat 8.5.53	A simple servlet container
Kotlin 1.4.32	Android implementation
Java 14.0.1	Web service implementation

4.1 Android App

For the front end, we implemented an application based on Android and MVP architecture. We briefly divided the project into three parts for decoupling.

- **Model:** is an interface defining the data to be displayed or otherwise the way it acts in the user interface.
- **View:** is a passive interface that displays data (the model) and routes user commands (events) to the presenter to retrieve that data.
- **Presenter:** acts upon the model and the view. It retrieves data from repositories (the model) and formats it for display in the view.

Then we integrated Google Map API, Gson, and Firebase into our project, for the location display, data conversion, and online chat module.

4.2 Web Service

So far we implemented our Web Server based on RESTful API, we constrained the way that clients access web resources on the server(basically with HTTP GET/POST). The web server is composed of three basic components:

- **Servlet:** classes that extend HttpServlet, to handle the request from client incessantly.
- **DAO:** Database access object, for the connection management between server and database.
- **JDBC:** Java database connection driver, which provides the interface to access the database from the server.

Then we deployed our server on Apache Tomcat at localhost.

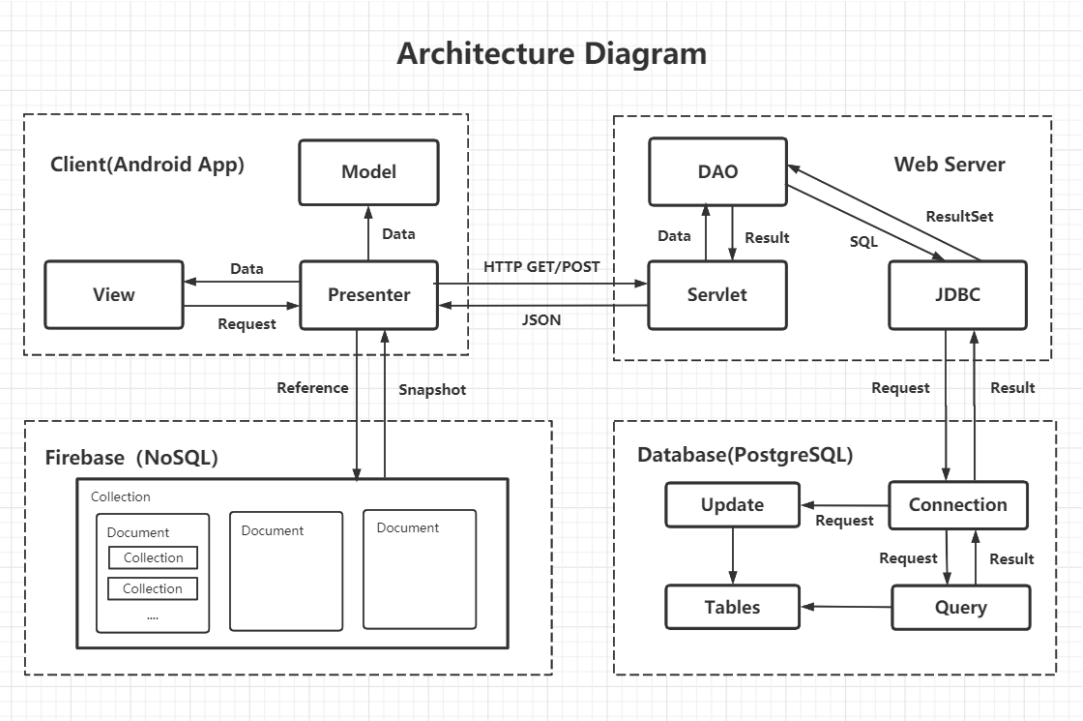
4.3 PostgreSQL

For the database, we designed some table and their relations, also some constraints like foreign key were implemented accordingly. We mainly access our database with `executeQuery` and `executeUpdate` for the basic CRUD. Then we implemented some basic scripts like 'createtable', 'populatedata'and 'retrievedata'.

4.4 Firebase

Since we can easily update schemas and fields in the NoSQL database, we implemented our online chat module with Google Firebase. The data could be stored in nested documents, in this way, we could access and maintain the chat session with ease. All we need to do is to add a listener in the presenter of our client and keep the connection alive.

- **Collection** The root reference of the Firebase store database.
- **Document** Store data as key-value pair or other collections.



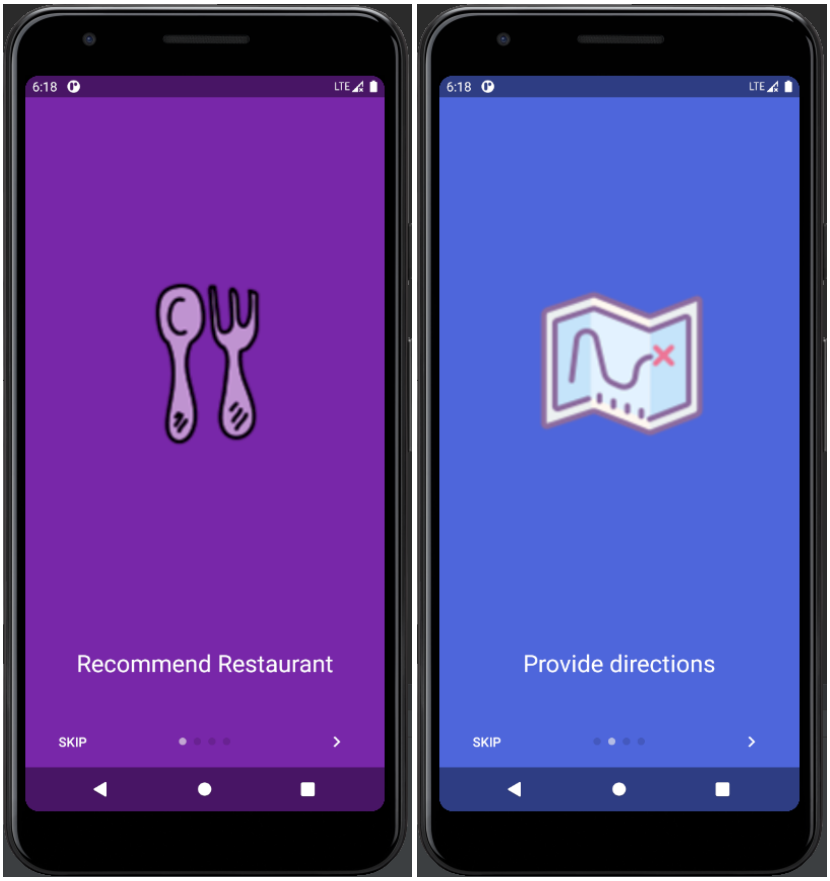
5. Screenshots of the user interface of your final project

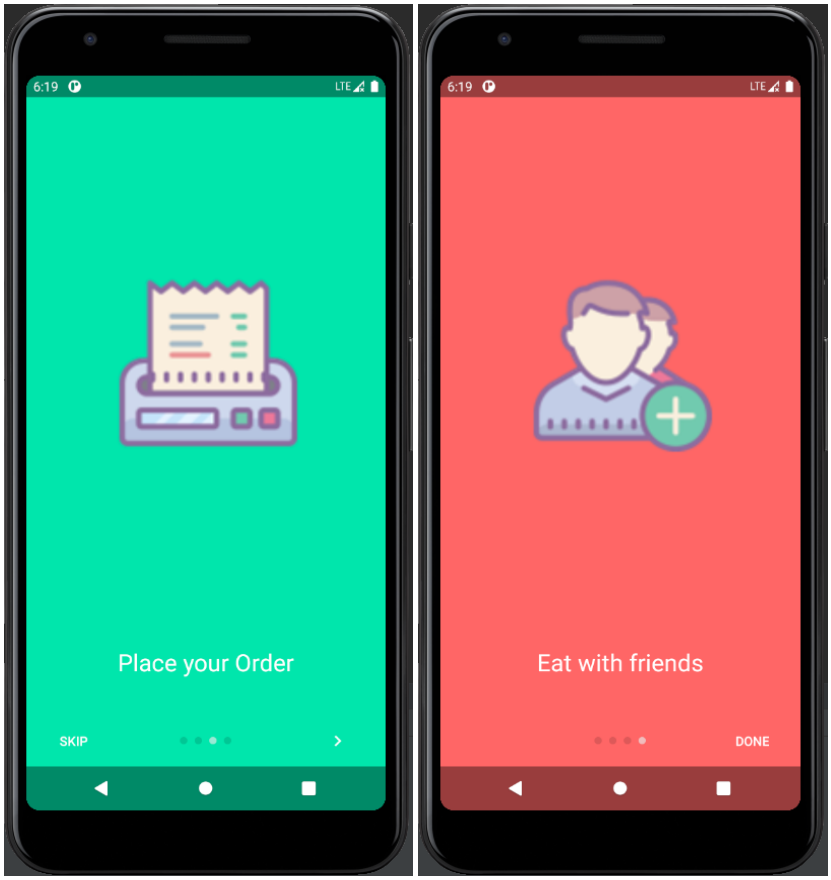
Video Demo

SherEats video demo

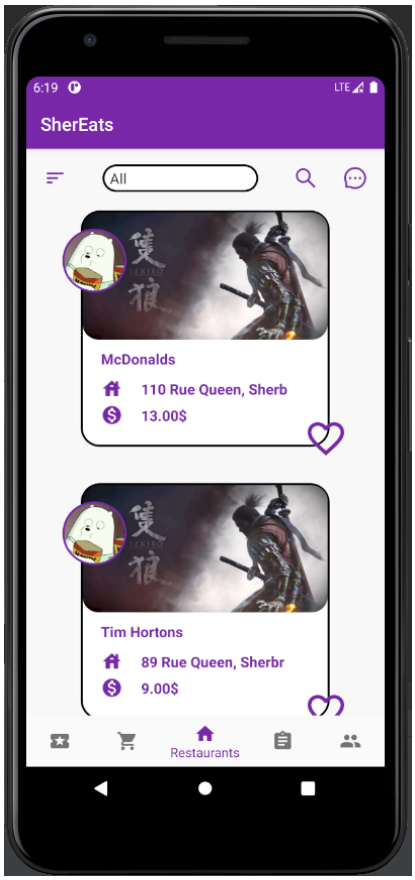
(https://www.youtube.com/watch?v=5nV9D6fmRhk&t=1s)

Welcome Page

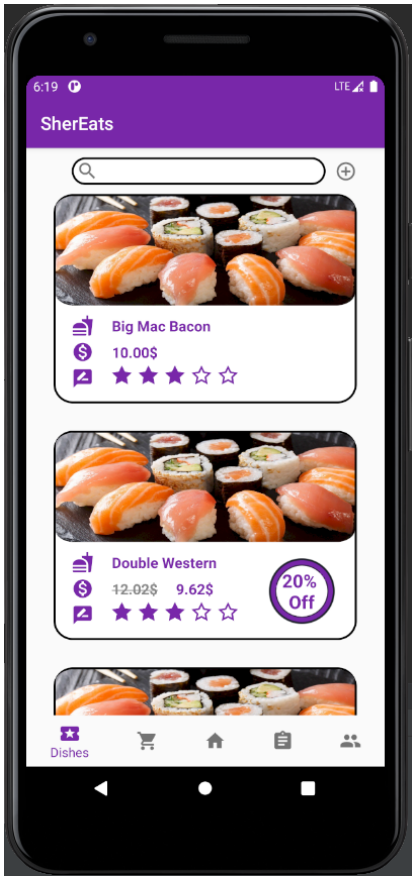




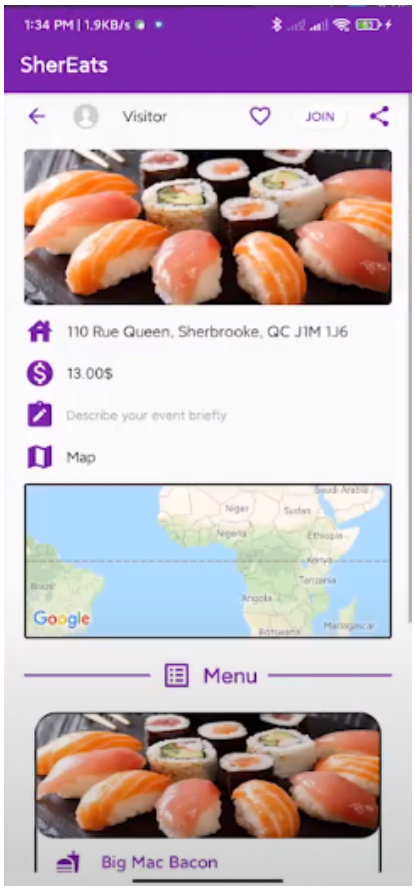
Main Page(restaurant recommendation)



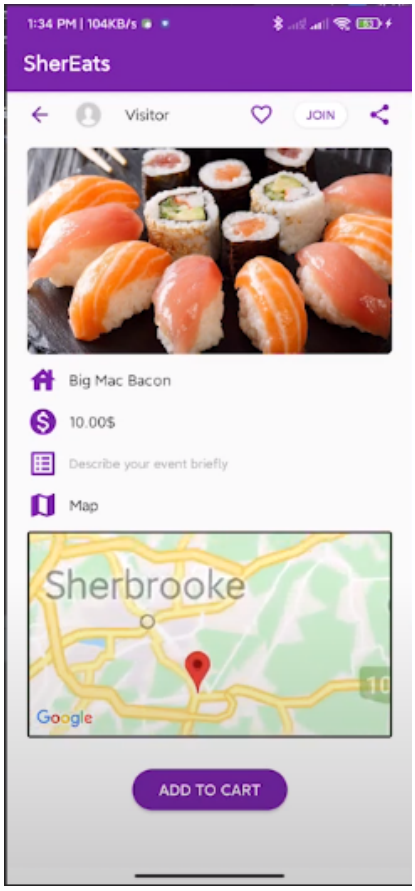
Dish Recommendation



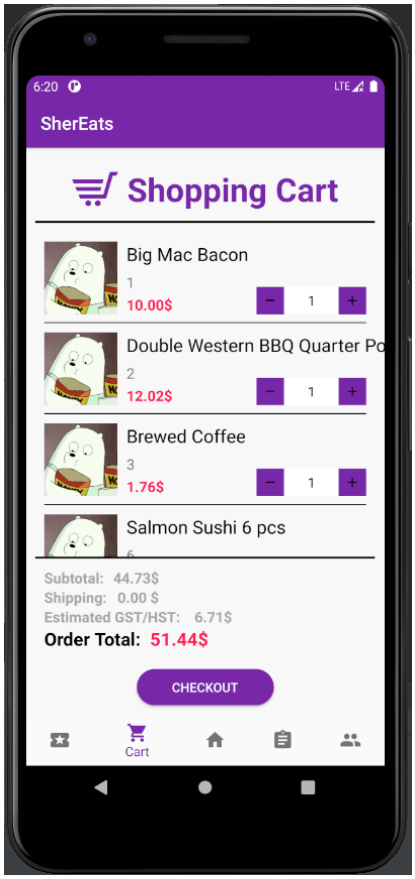
Restaurant Detail Page



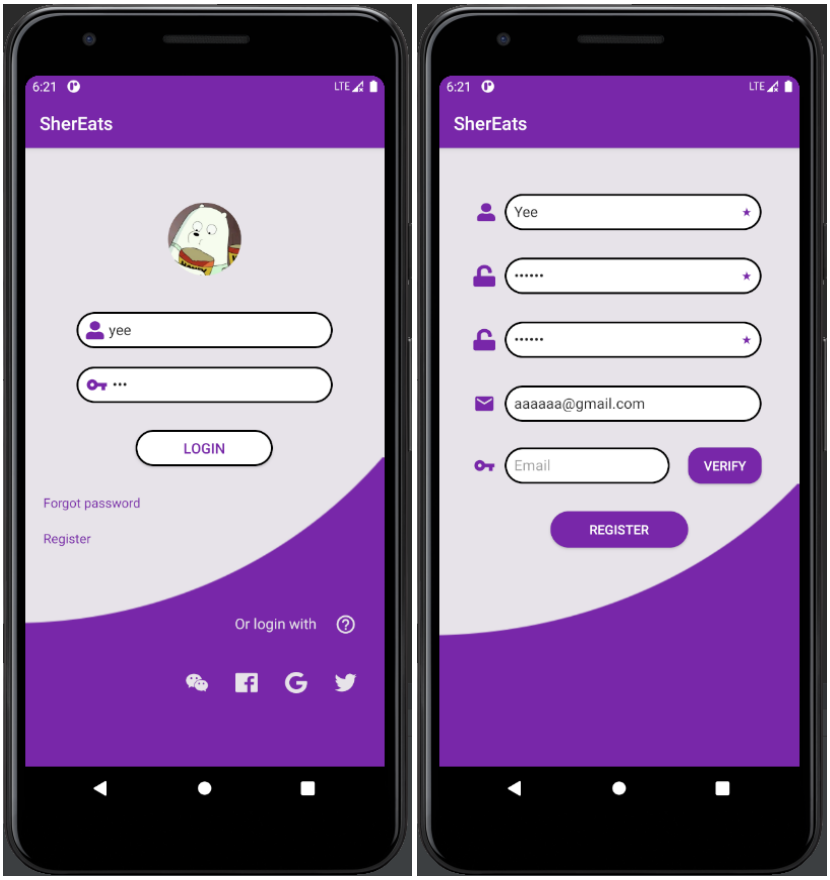
Dish Detail Page



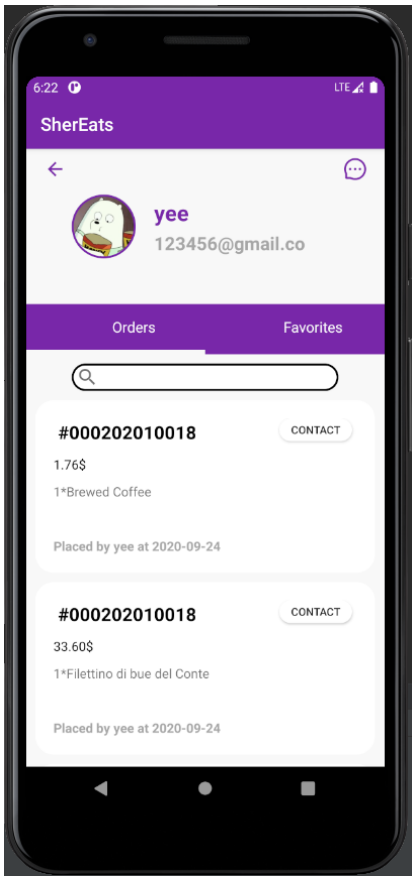
Shopping Cart



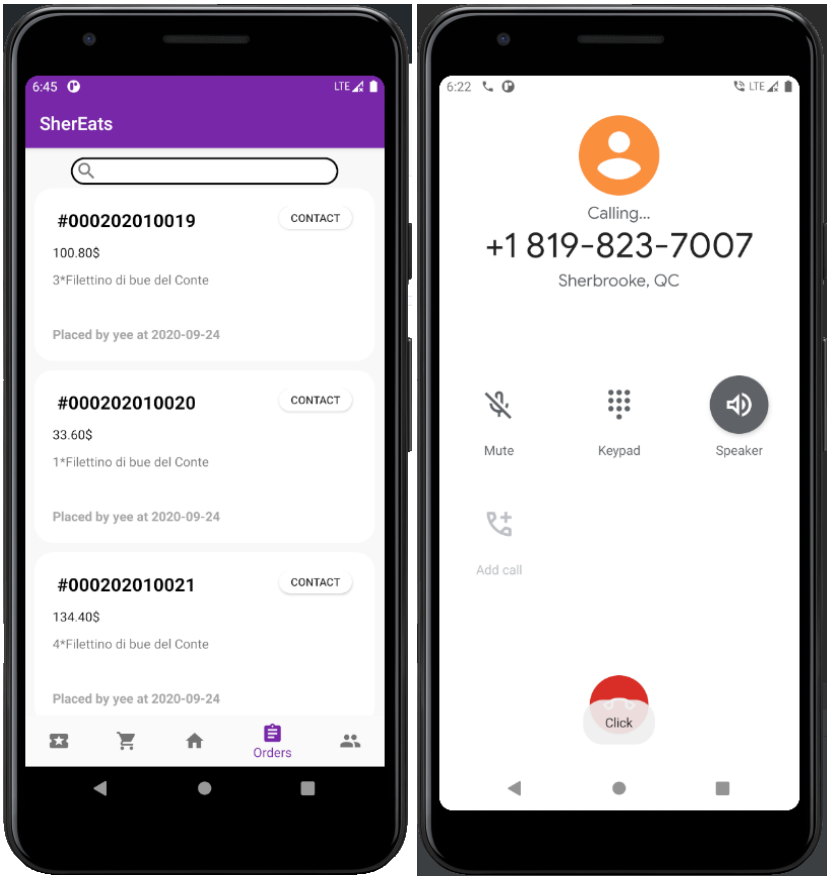
Login & Register



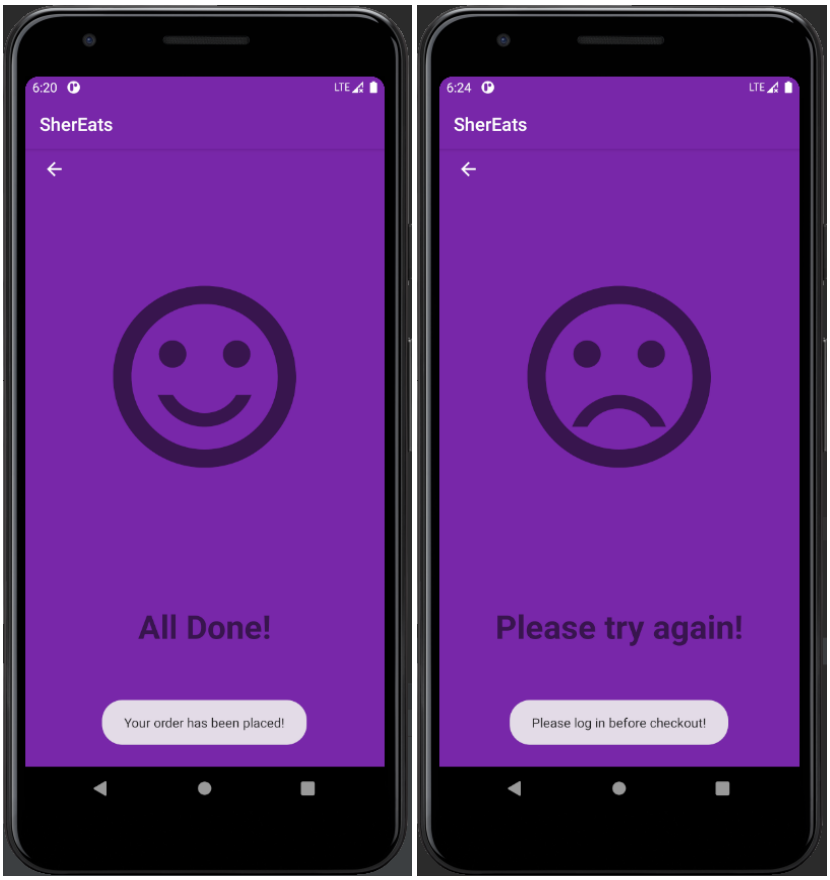
User Information Page



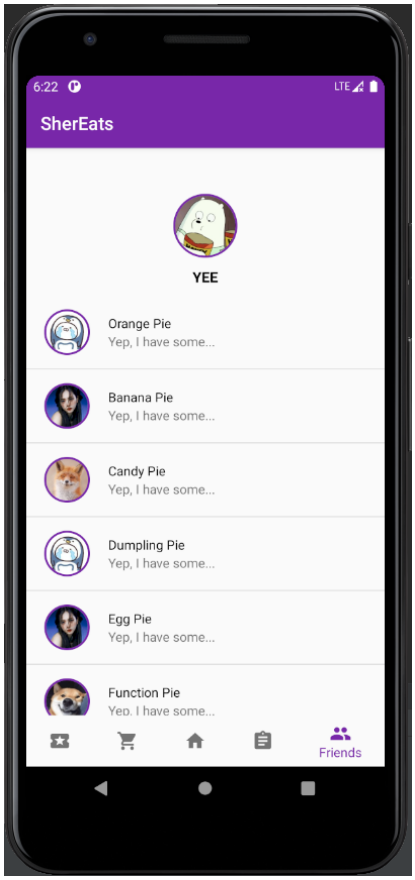
Order History Page



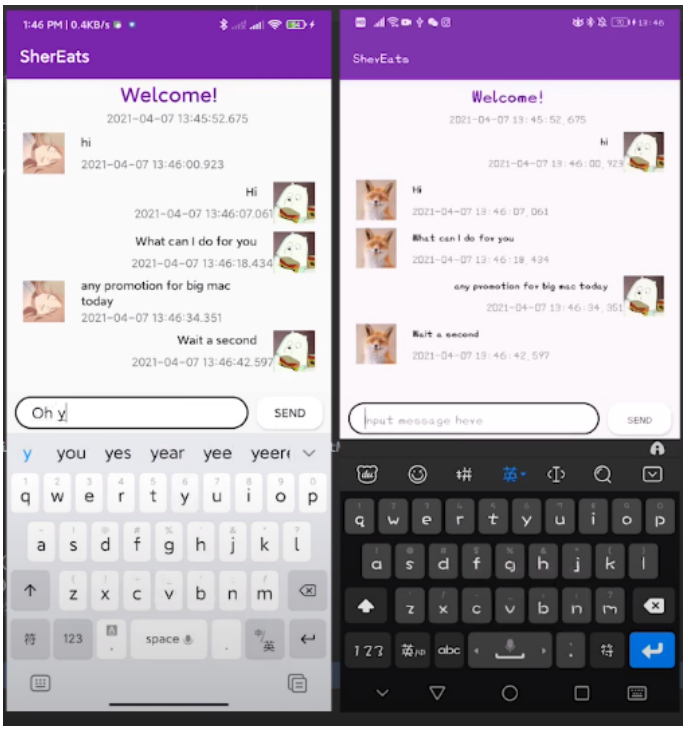
Result Page



Friend List Page



Chat Page



6. Conclusions and future works

After four weeks of hard work, we finally completed the restaurant recommendation system. Basically, we've fully accomplished the features raised in the design stage with agile project management, since we went through several iterations(incremental steps) towards the completion of a project. For now, there remain some issues to be fixed.

- **Cloud deployment:** so far our server is deployed at localhost, which means no guarantee of security and robustness.
- **Data transfer security:** we encapsulated our data transfer method with HTTP protocol, which is a lack of security.
- **Cross-platform:** for the most popular platforms, our project is only available on Android.
- **Image Storage:** the image of restaurants and dishes are demo images integrated into the Android APK, we still need to provide our user with an image upload module.

In the future projects:

- **We'll try to move our server and database to cloud server**
- **Implement data transfer with HTTPS**
- **Build the front end with a cross-platform framework, such as Flutter, React Native, or Weex.**
- **Build an image database with Firebase Storage or other flexible and scalable cloud databases.**