< Journey Assistant >
Software Architecture Document
Version 2.0

Group Member

Yiwen Song
Zhihui Xie
Weizhe Wang
Huangfei Jiang
Haoping Chen

Modification History

Date	Version	Description	Author
2019-04-16	1.0	The first version of this document.	Zhihui Xie & Yiwen Song
2019-06-18	2.0	Modify some diagrams.	Zhihui Xie & Yiwen Song & Huangfei Jiang

Contents

1 Introduction

1.1 Purpose

The purpose of this software architecture document is to make clear the architecture of our software, and lead the development team to develop the software according to our architecture.

The intended readers of this document is the developers of the software.

1.2 Scope

This document will be applied in our software development, particularly Journey Assistant. We don't intend to apply this document on any other software under development.

1.3 Definition

Here we list some terms or abbreviations we will use in this file.

Abbreviation	Term	Implication
JA	Journey Assistant	The Name of our Project
	Android	An OS Developed by Google
IM	Instant Messaging	A type of online chat that offers real-time text transmission over the Internet

Table 1: Terms or Abbreviations

2 Architecture of the Current System

The architecture of the current software system is shown below (Figure 1). We can see that the current system don't have a multi-functional architecture and it needs human service to complete the task.

Therefore, what we will do to improve the system is to change the human service into automative service, and make the system more complete.

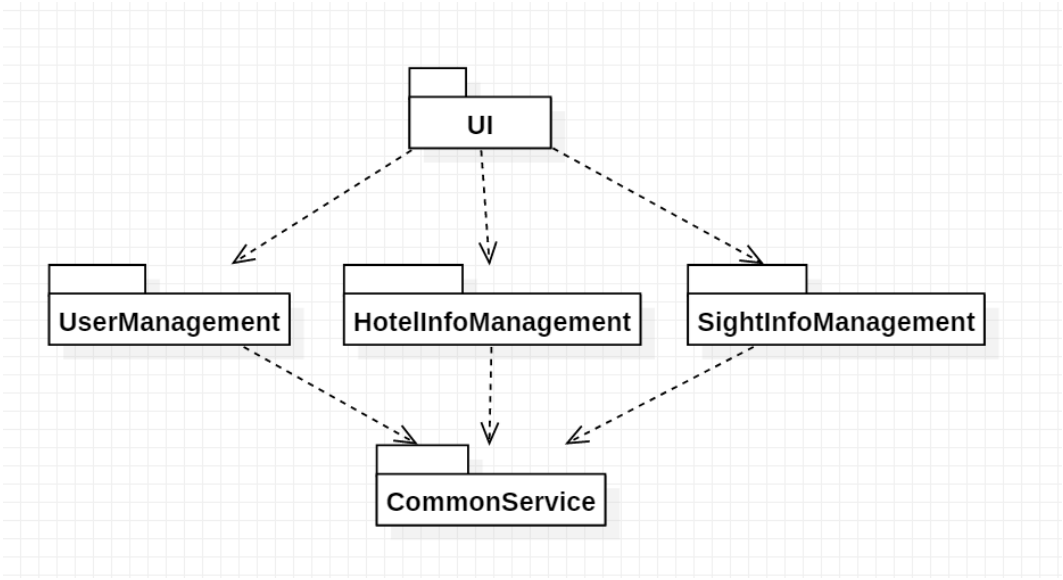

Figure 1: Architecture of the Current System

3 Design Objective of System Architecture

The following requirements and goals of our software has great influence on our software architecture:

(1) The security of our software. We need to make sure that all the information in our software system are kept securely, especially the personal information of our users. Therefore, we must add components in our architecture to ensure the security of our software.

(2) The synchronization of information. The user-provided information and the information we provided to the users should be synchronized on our server. Given the user's account, the server should be able to recover all information provided by the user and recent information we have provided to the user, in case the user re-install the software.

(3) The extension-support. At the beginning of our software, we do not consider using high-level hardware. However, when the user crowd has grown to a certain level, our system would be able to be transferred to a higher-level harware server easily.

(4) High-proficiency level. Our system should be able to react quickly and accurately to the users' various queries.

(5) High capatibility. Our system should be able to run on different versions of Andriod operating systems, or at least run on Android version 6.0, 7.0 and 8.0, which is widely used currently.

4 Architecture of our Proposed System

4.1 Introduction

(1) We use client/server architecture as our main system architecture.

The system is devided into the frontend client and backend server. The frontend is the application of users' cellphones, and the backend is the server that achieves system functionality and saves data.

The reason why we choose client/server architecture is that the architecture is mature and has been widely used in this kind of service application developments. It also helps the centered management of data, while our system is data-centered.

(2) We use 3-layered architecture as our subsystem architecture.

We use 3 layers to organize our subsystems. The first layer is user-interface layer, which provides the port for user interface. The second layer is application logic layer, which is in charge of controling and realizing system functionality. The third layer is web-service layer, which is in charge of the web service of data tranmission.

The 3-layered architecture is mature. It devides the port and the logic layer apart, leading to a low-coupling system, making our system easier to extend and maintain.

(3) We use MVC as our model architecture.

The MVC model fits well with our interaction system. We have devided the system into three layers, so it's easy for us to do MVC separation.

The system includes the following subsystems:

UI In charge of the interaction between the system and users.

UserManagement In charge of management of user information, including "login", "sign in", and "change information" instructions.

JourneyManagement In charge of making plans about itinerary and making customizations given itinerary details.

FeedbackManagement In charge of handling feedback from users.

CommonService In charge of data I/O, the communication between the client and the server.

4.2 Use Case View

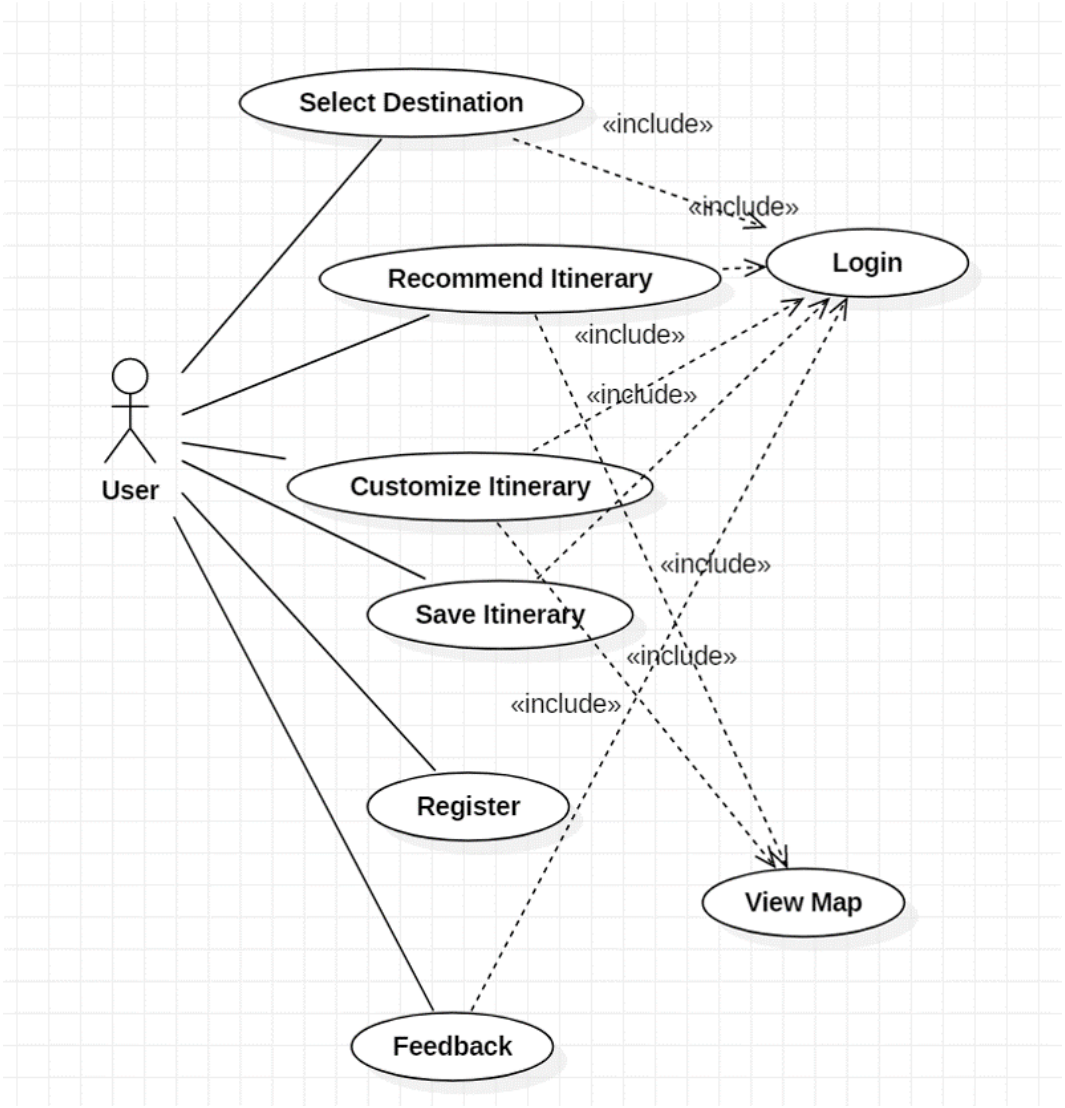

Figure 2: Use Case View

4.3 Logic View

4.3.1 System Architecture

The system architecture diagram is shown in the following figure. The whole system in devided into 7 sub-systems, respectively UI, UserManagement, DataManagement, DialogManagement, customizationService, RecommendationService, and CommonService. Each subsystem will be introduced later.

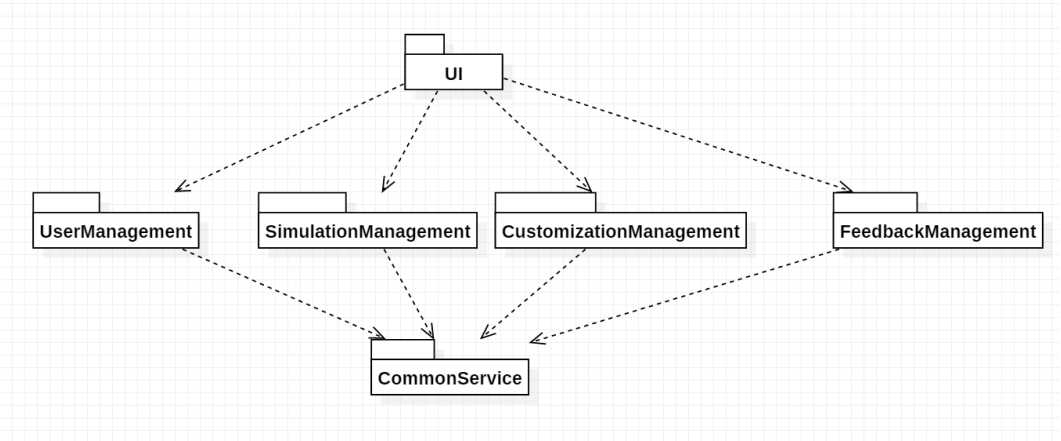

Figure 3: System Architecture

The following class diagrams (Figure 4 – 7) show inner logic for every package.

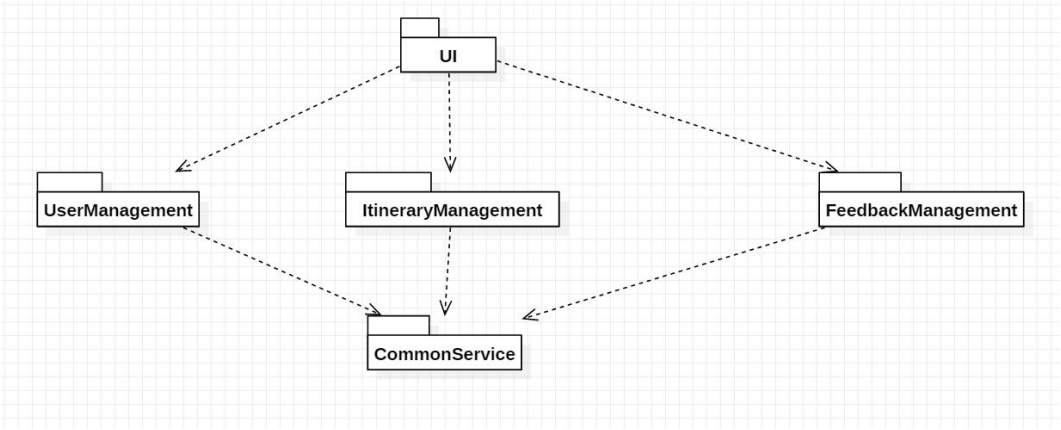

Figure 4: UI Subsystem

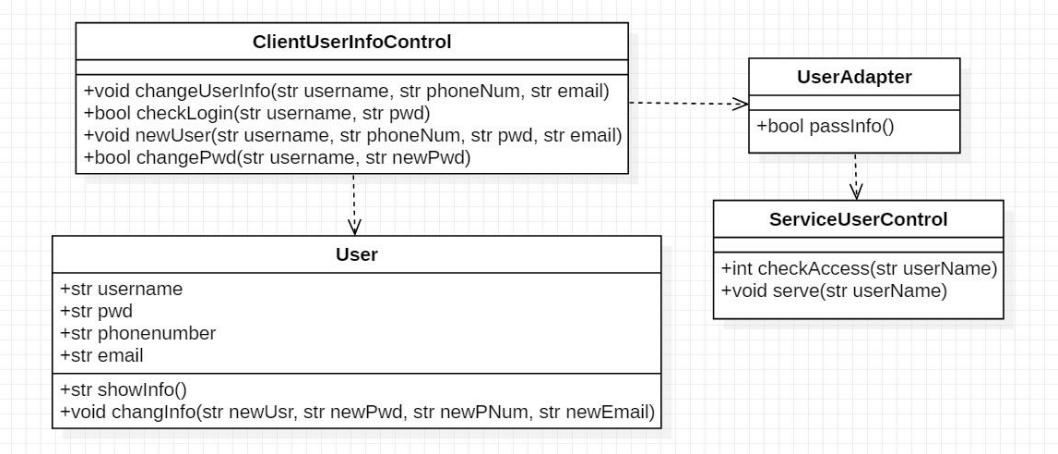

Figure 5: User Management Subsystem

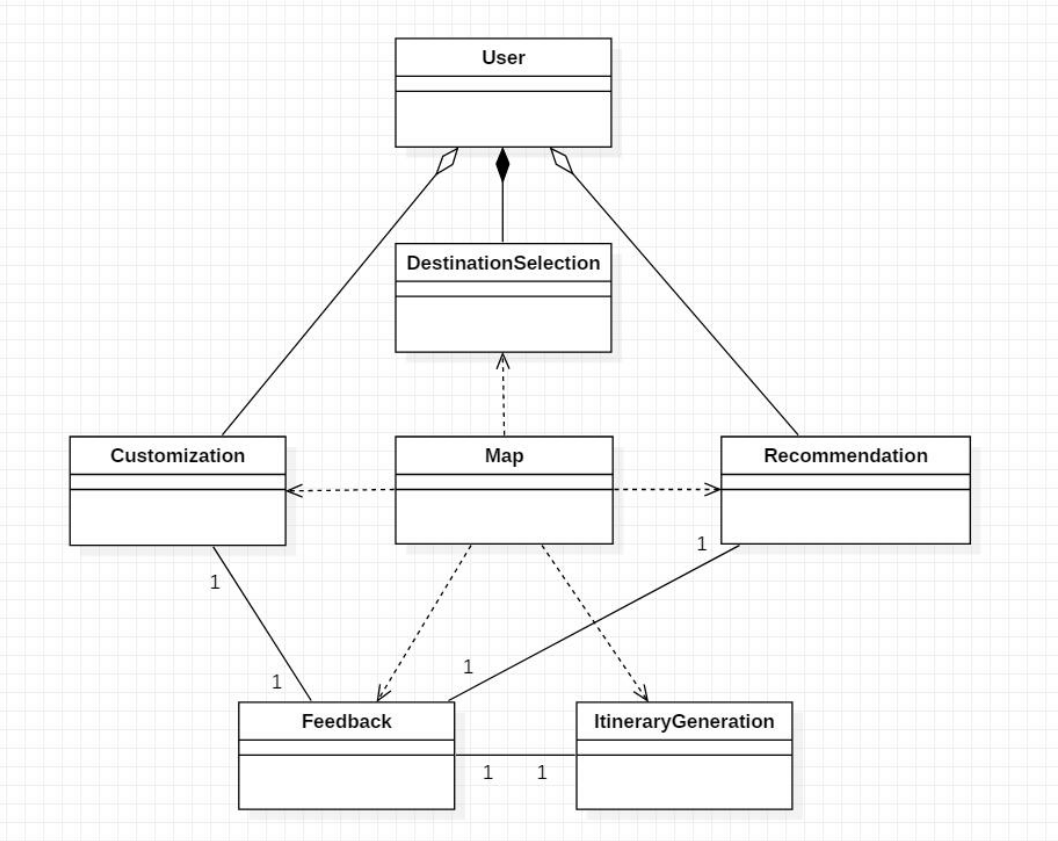

Figure 6: Itinearary Management Subsystem

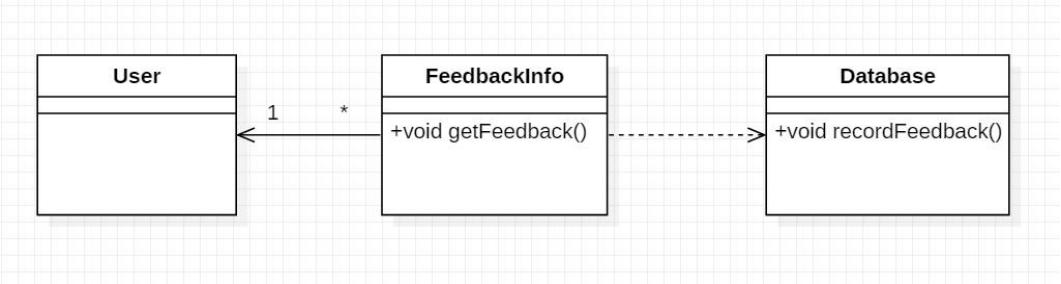

Figure 7: Feedback Management Subsystem

4.3.2 Subsystem

Our whole system is composed of 5 subsystems, including the UI subsystem, the UserManagement subsystem, the JourneyManagement susbsystem, the FeedbackManagement subsystem, and the CommonService subsystem. The detailed ports and the components of the subsystems will be introduced in the following part.

UI Subsystem

Functionality The interaction between the system and users.

Service UI will provide the input/output box, buttons, frames to support I/O of the messages.

Class Diagram Figure 8

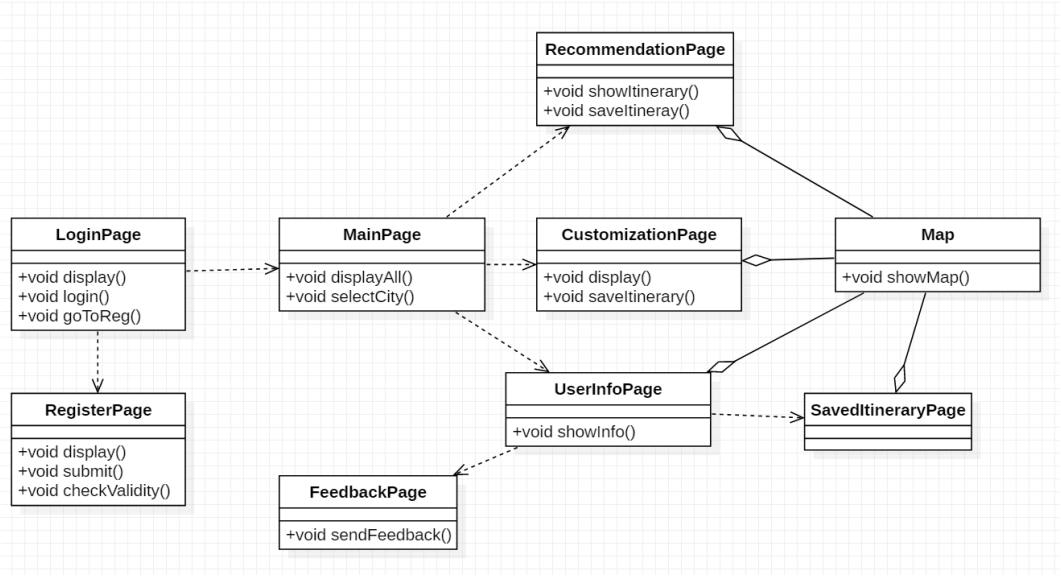

Figure 8: UI Subsystem Class Diagram

UserManagement Subsystem

Functionality Management of user information, including “login”, “sign in”, and “change information” instructions.

Service registration, submitUserInfo, login, showPersonalInfo, updatePersonalInfo

Class Diagram Figure 9

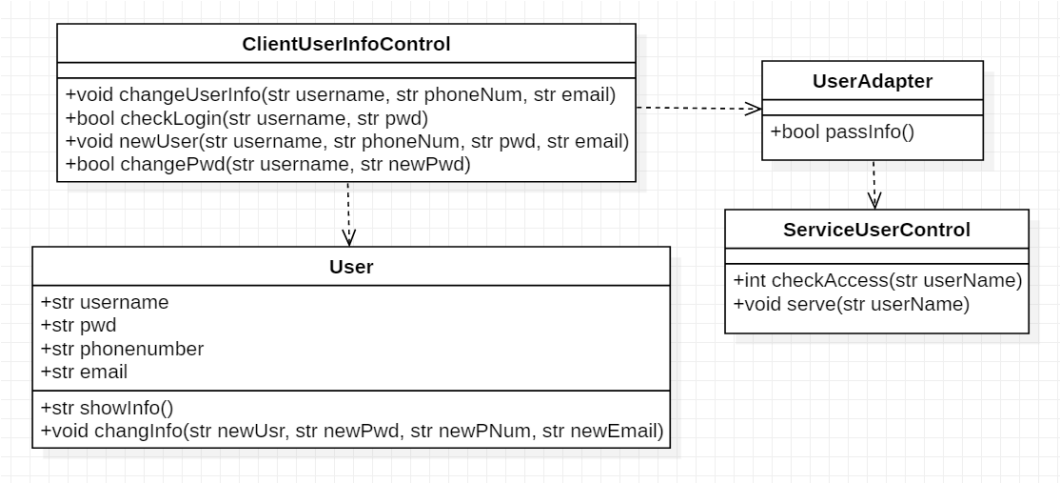

Figure 9: UserManagement Subsystem Class Diagram

Component Diagram Figure 10

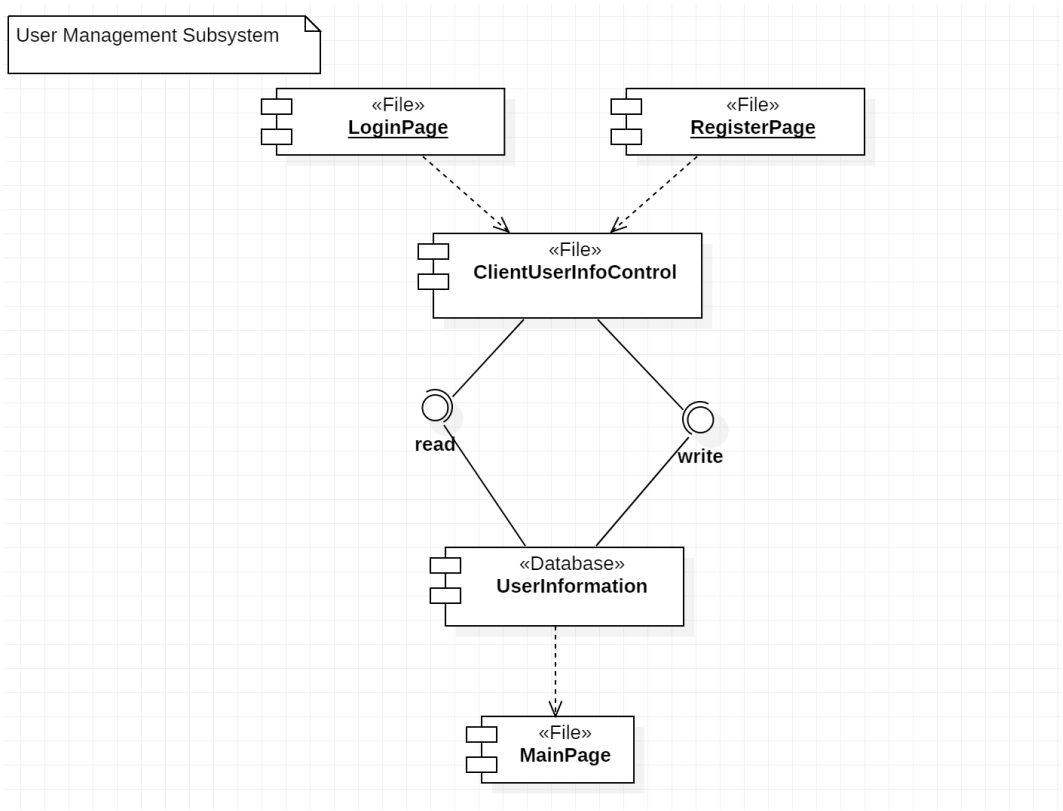

Figure 10: UserManagement Subsystem Component Diagram

ItineraryManagement Subsystem

Functionality Making plans about itinerary and making customizations given itnerary details.

Service designItinerary, changeItinerary, simulateItinerary, selectDestination, showResult, generateResult

Class Diagram Figure 11

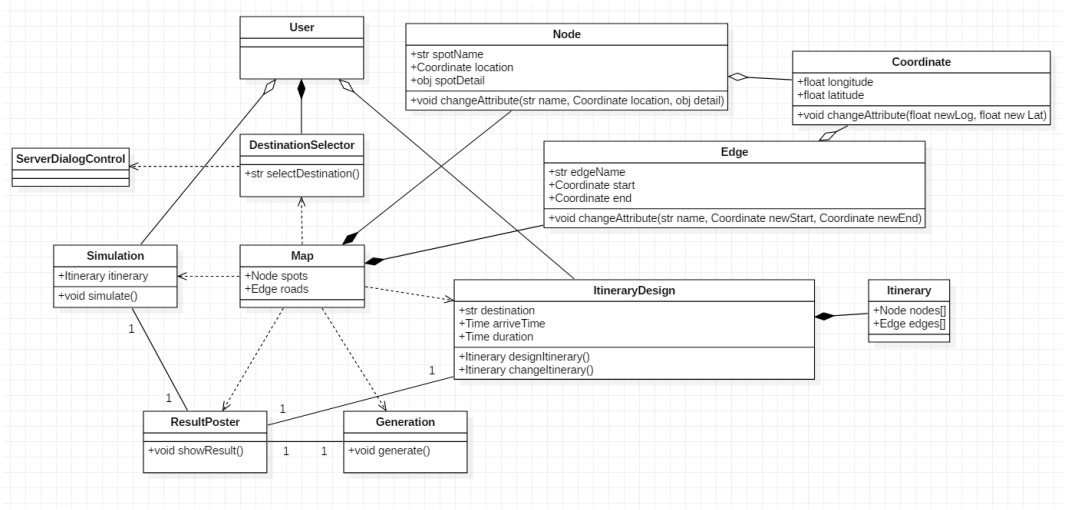

Figure 11: ItineraryManagement Subsystem Class Diagram

Component Diagram Figure 12

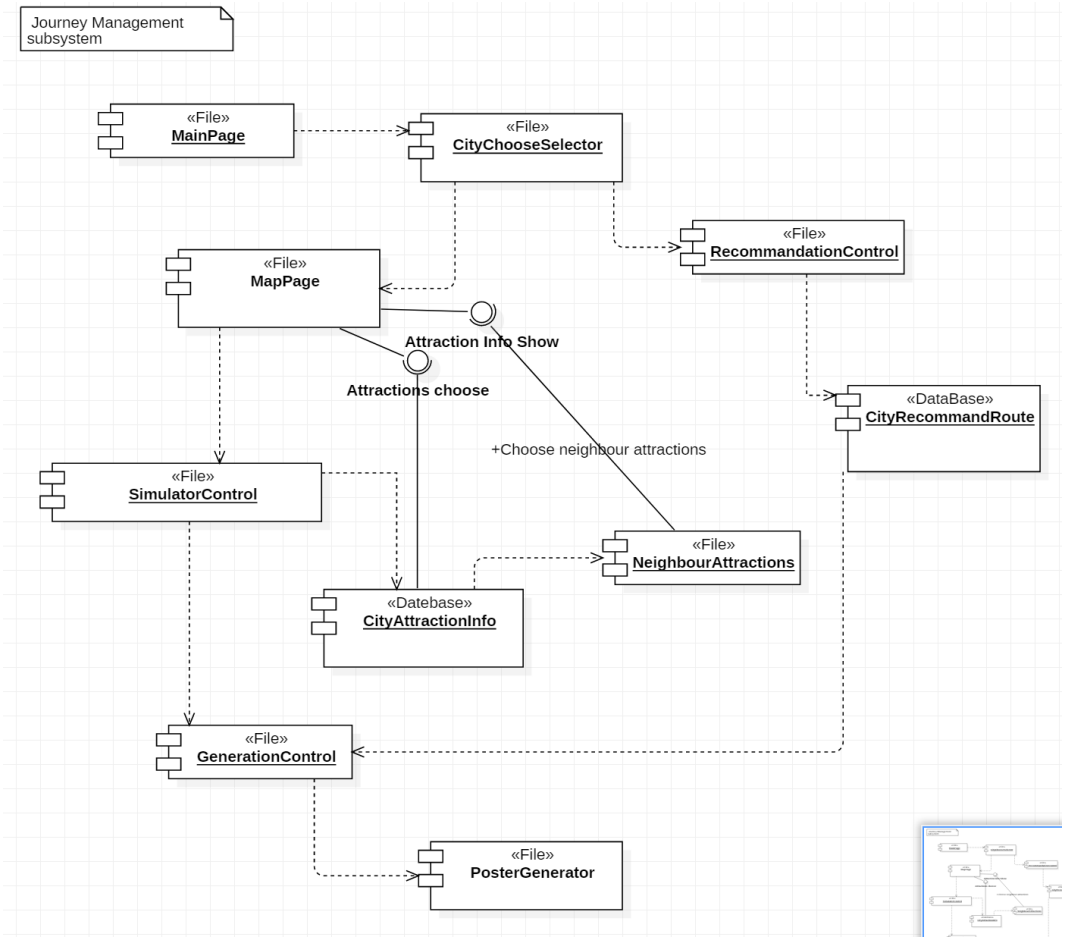

Figure 12: ItineraryManagement Subsystem Component Diagram

FeedbackManagement Subsystem

Functionality Handle feedback from users.

Service feedbackIssue

Class Diagram Figure 13

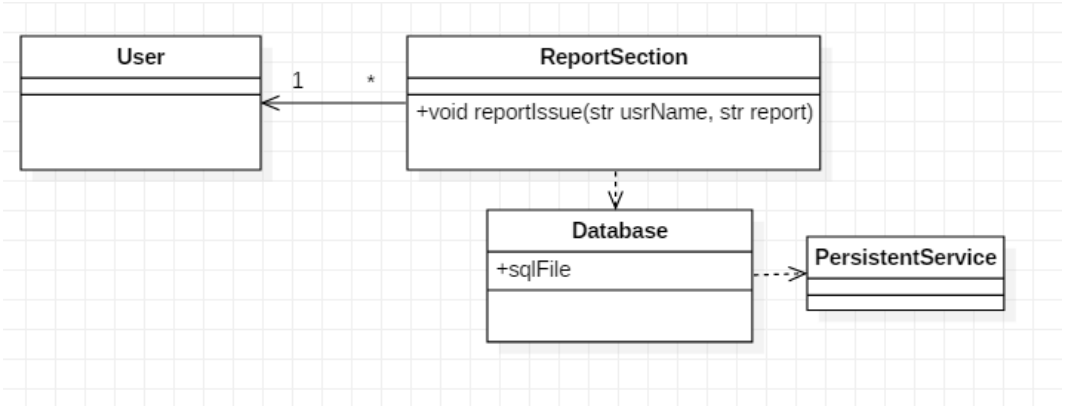

Figure 13: FeedbackManagement Subsystem Class Diagram

Component Diagram Figure 14

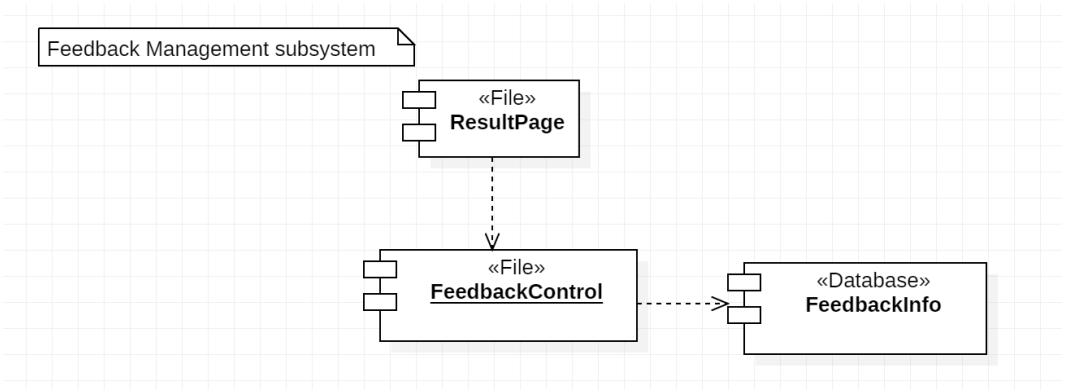

Figure 14: FeedbackManagement Subsystem Component Diagram

CommonService Subsystem

Functionality Data I/O, the communication between the client and the server

Service request, startCommunication, sendMsg.

Class Diagram Figure 15

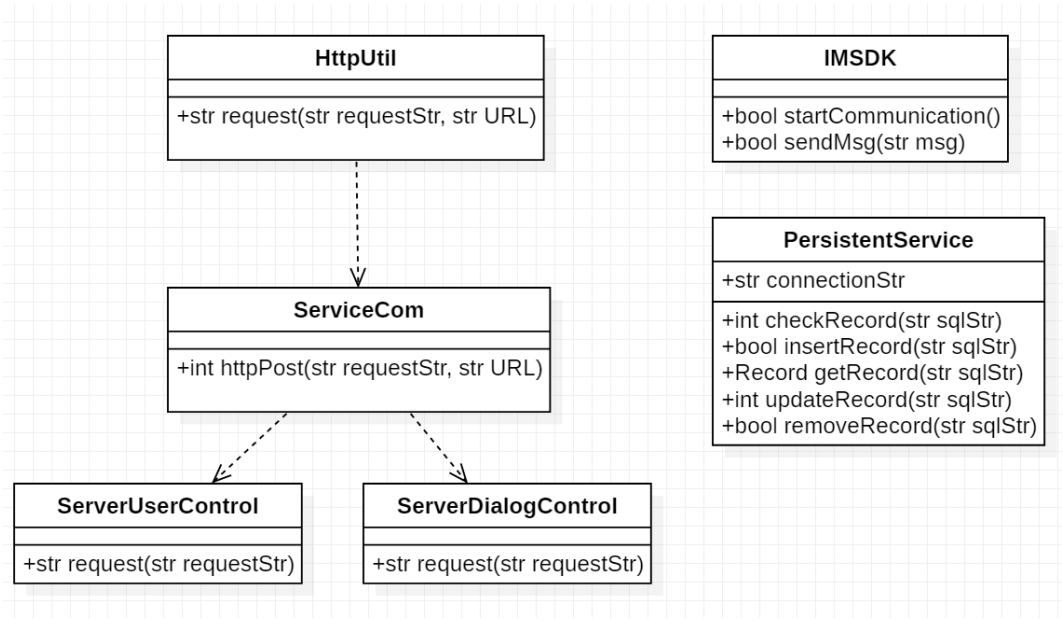

Figure 15: CommonService Subsystem Class Diagram

4.3.3 Use Case Realization

For each type of use case, we use an interaction diagram to display our corresponding inner design.

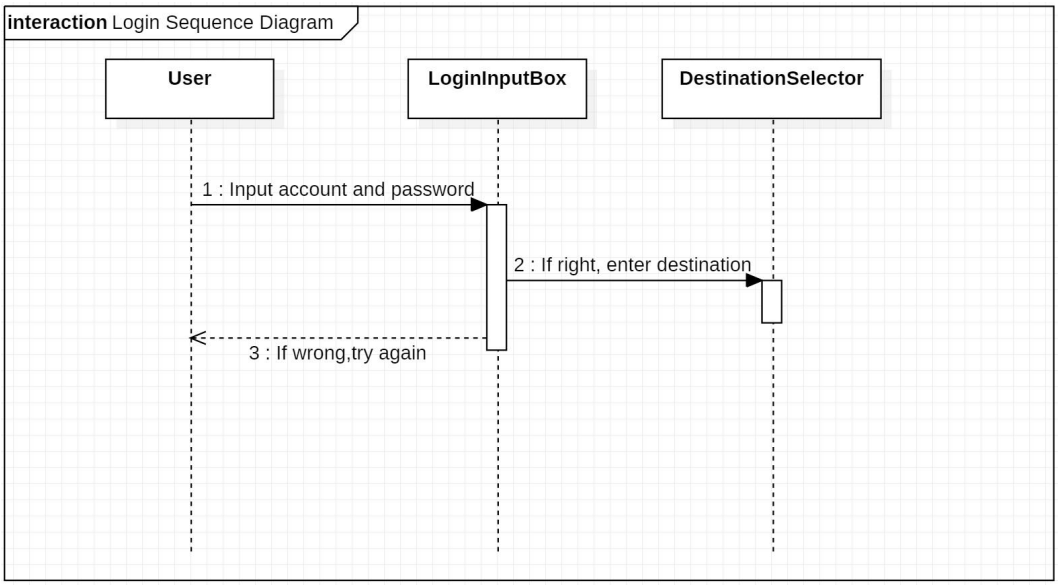

Figure 16: Log-in Sequence Diagram

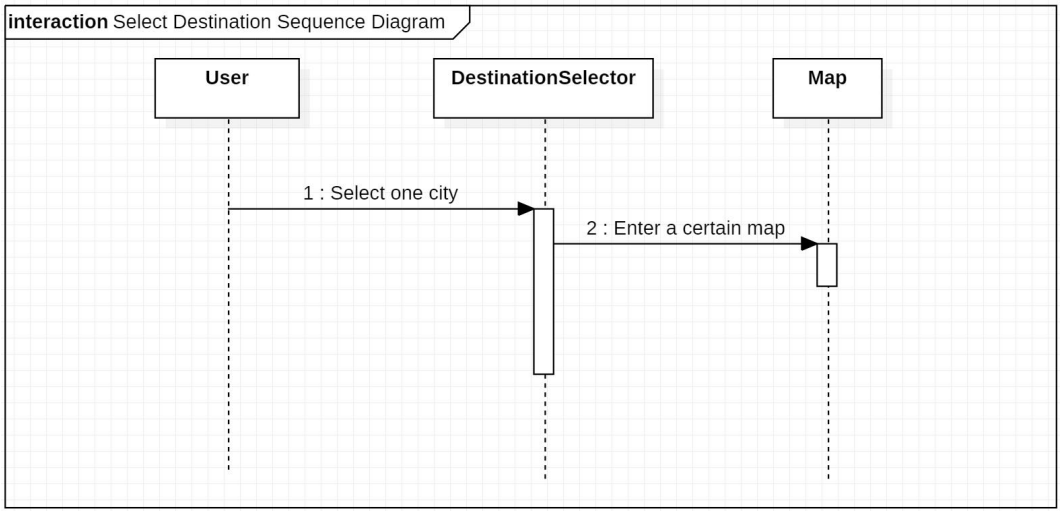

Figure 17: Select Destination Sequence Diagram

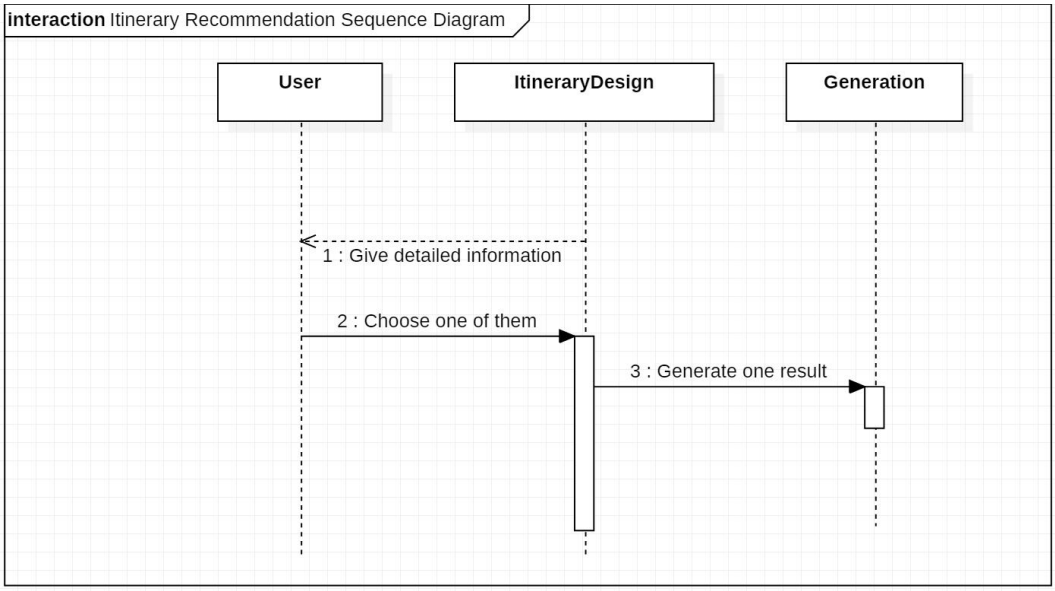

Figure 18: Itinerary Recommendation Sequence Diagram

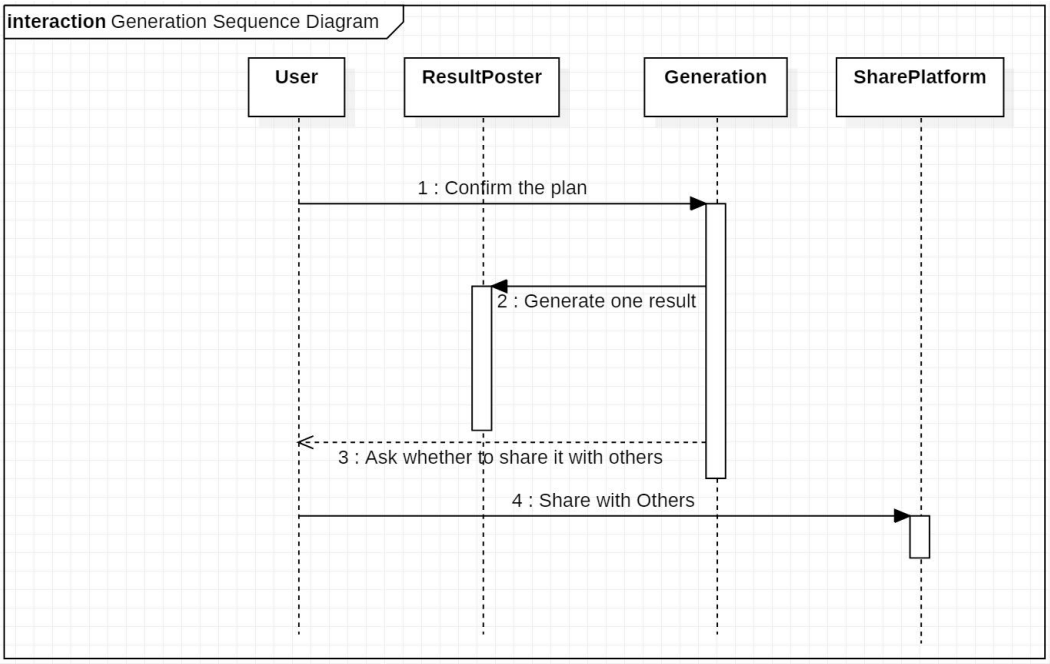

Figure 19: Generation Sequence Diagram

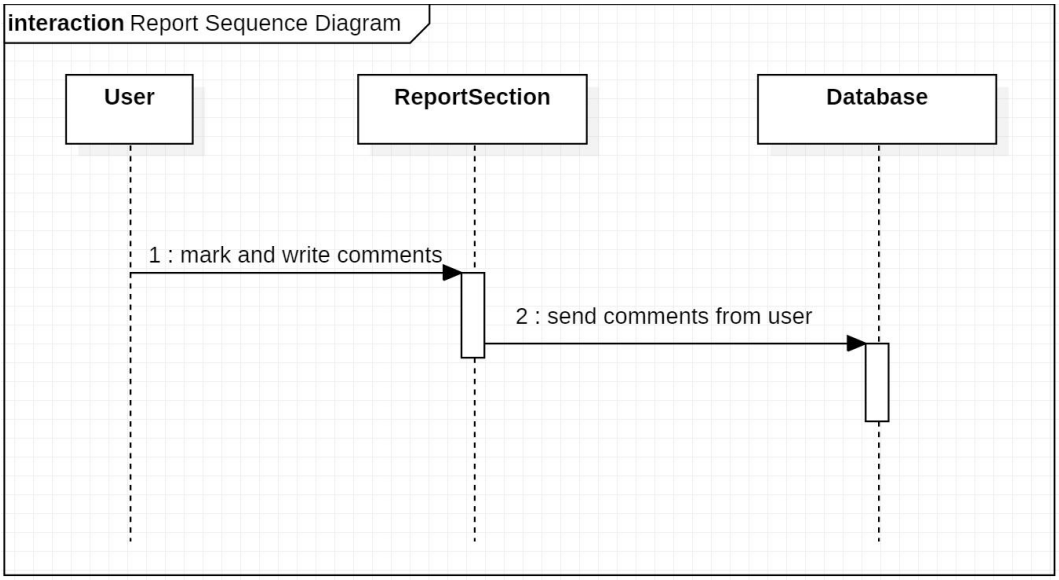

Figure 20: Feedback Sequence Diagram

4.3.4 Subsystem Cooporation

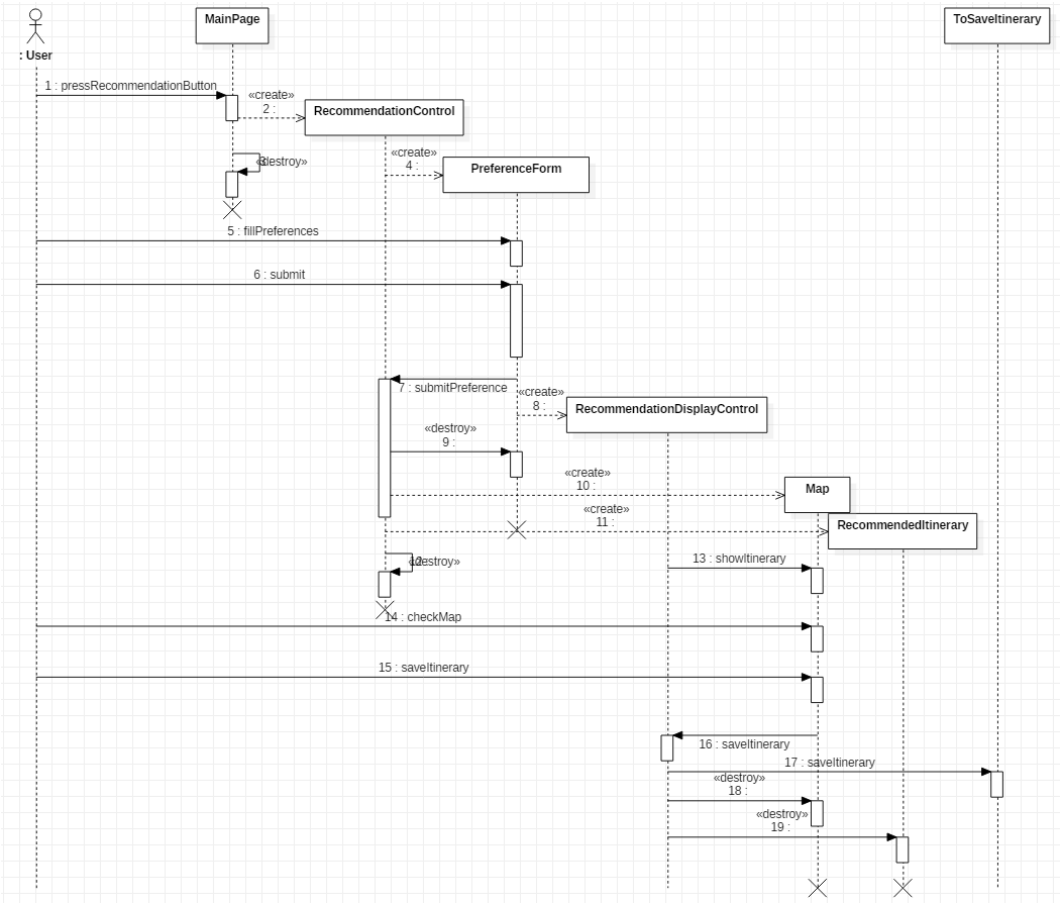

Figure 21: Recommendation Sequence Diagram

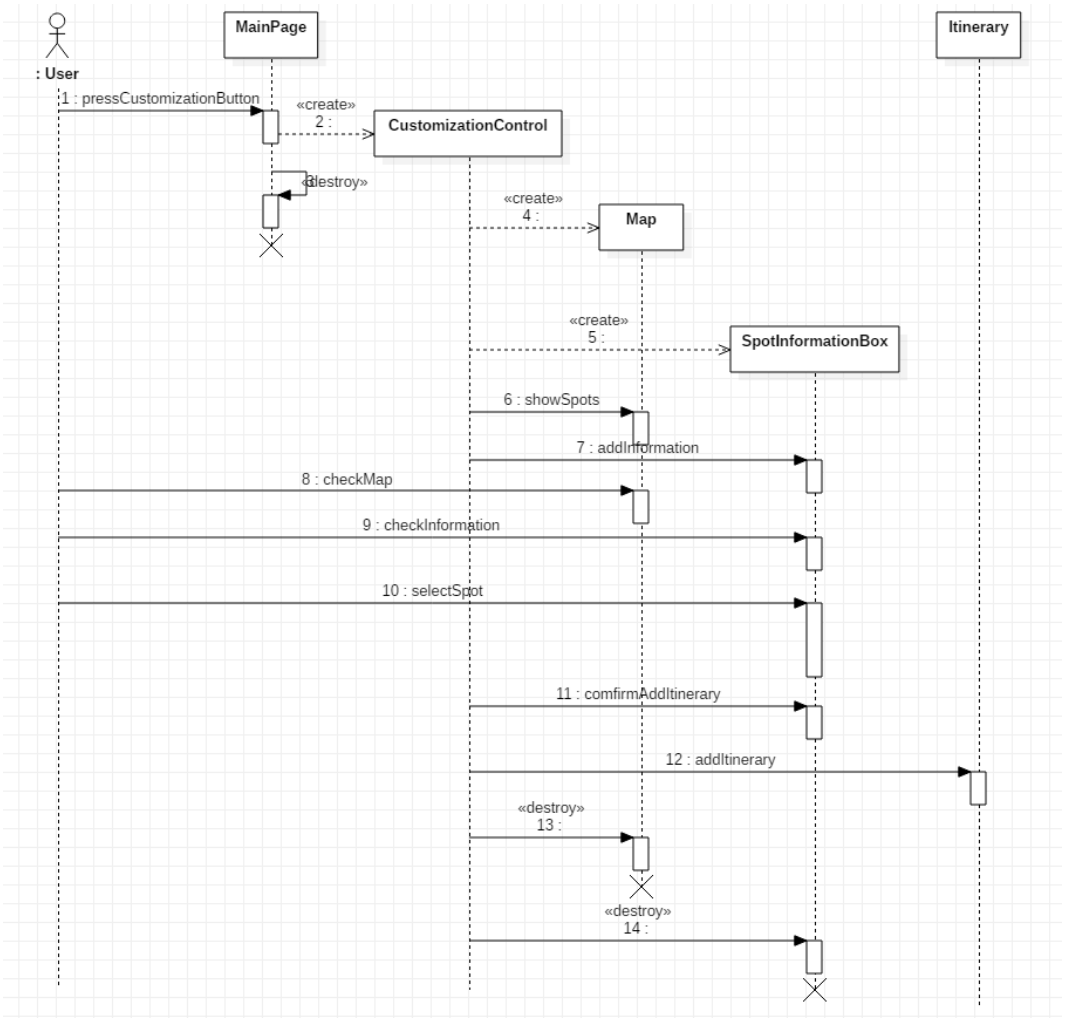

Figure 22: Customization Sequence Diagram

4.4 System Operation Diagram

We use thread control as our control flow of our system. We split our operation diagram as Client Operation Diagram and Server Operation Diagram.

4.4.1 Client Operation Diagram

At the client, the UI thread is only in charge of the interaction between the system and the user. At the same time, we start a single thread for each controller and each dialog module. This kind of design can assure that the frontend, the realization of functions, and the dialog won't collide with each other to crash our whole system.

4.4.2 Server Operation Diagram

At the server, for the dialog module, we will use multi-thread control to respond each client's query. At the same time, each controller has its own thread to ensure that they won't influence each other. For consistent service, we also use multi-thread to make the system stable.

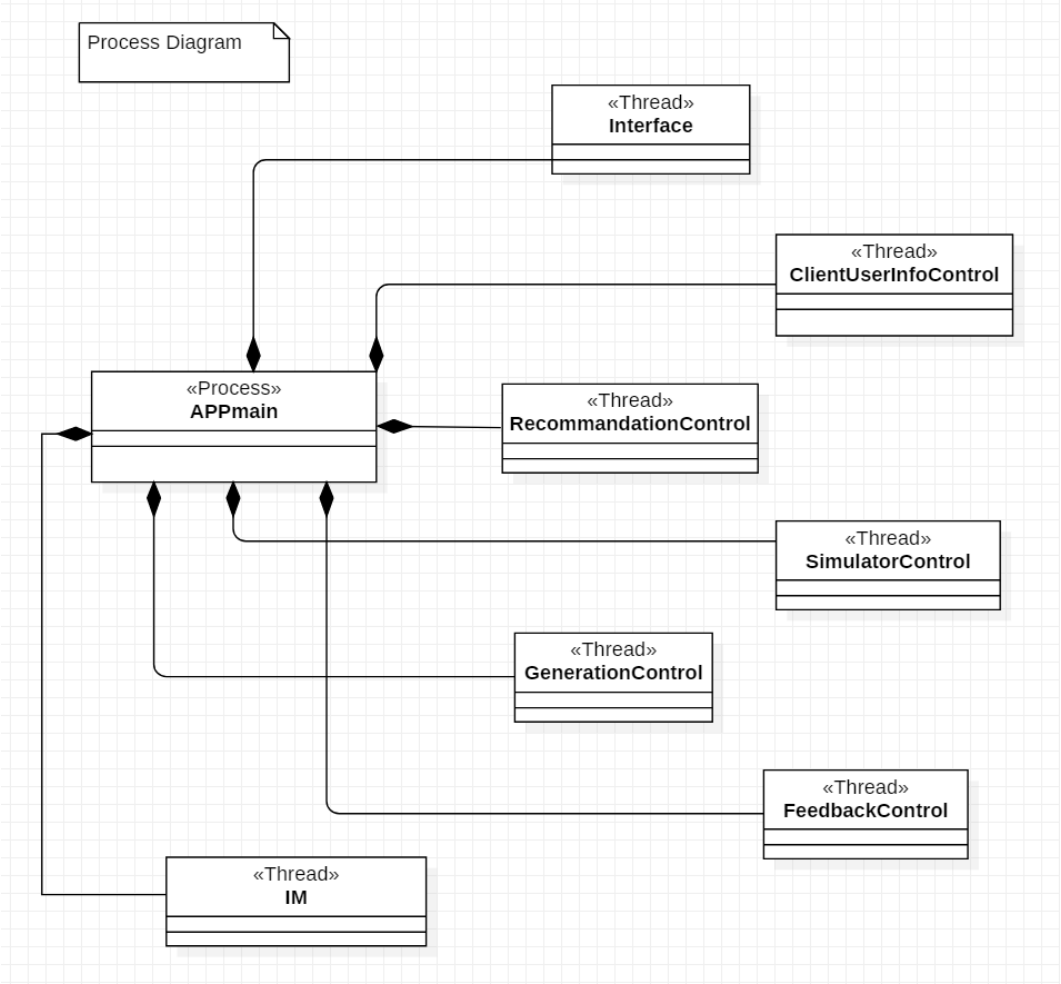

Figure 23: Process Diagram

4.5 Physical View of the System

The hardware disposition requests are as follows:

User Machine An Android device with 2GB (or more) RAM and a mainstream CPU such as Qualcomm Snapdragon and HUAWEI Kirin. Low-end phones will not face any trouble using our application.

Application Server To meet the requirement of high-density computing and data flow, we need at least 2 servers with Intel Core CPU (7-Gen or higher), 8GB (or more) memory, and 512GB (or more) storage. To be

more specific, we may deploy two different kinds of hard disk: For our 'Management' server, we will use SSD to speed up the service. For the 'Database' server, we will use HDD to make sure the data is safe. Besides, the 'Management' server will be deployed with a NVIDIA GPU which supposts CUDA. It will help with machine-learning-related tasks.

Network Take the maximum number of concurrent events as 1000. To support our system, we need 20MB exclusive bandwidth to deal with the estimated 5kb/s data flow. The exclusive bandwidth guarantee the performance of data exchange.

4.6 Design of Boundary Condition

We will here discuss three kinds of use cases: startup, shutdown and error handling. Detailed interaction diagrams are also present to explain how these use cases are realized. However, due to the fact that the coding job has not been finished yet, we will not discuss it in detailed. Only rough realizations will be shown here.

4.6.1 Server Startup

To start up the server, we need to launch a series of service that ensure our system runs persistently and all components works smoothly.

(1) Click the startup icon.

(2) Initialize the object of server site 'ServerSite'.

(3) Create the object of persistent service 'PersistentService'.

(4) Create the objects, which will be named as 'ServerDialogControl', 'ServerDataControl' and 'ServerUser-Control' respectively, to control system dialog, data and user information. These objects will refers to 'ServerSite' and 'PersistentService'.

(5) Launch the IM (instant messaging) service, which is packed in 'IMSDK'.

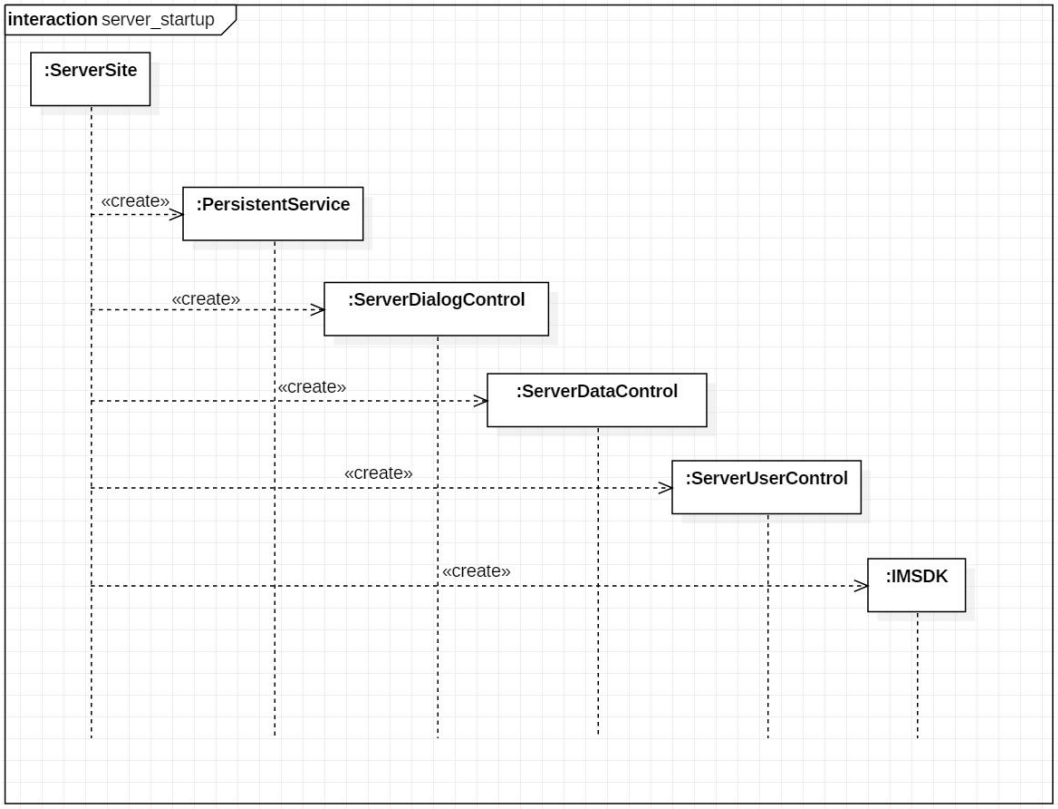

Figure 24: Use Case Realization of 'Server Startup'

4.6.2 Server Shutdown

(1) Click the shutdown icon.

(2) Delete 'ServerUserControl', 'ServerDialogControl', and 'ServerDataControl'. Close the IM service as well.

(3) Disconnect from the database, and delete 'PersistentService'.

(4) Delete 'ServerSite'.

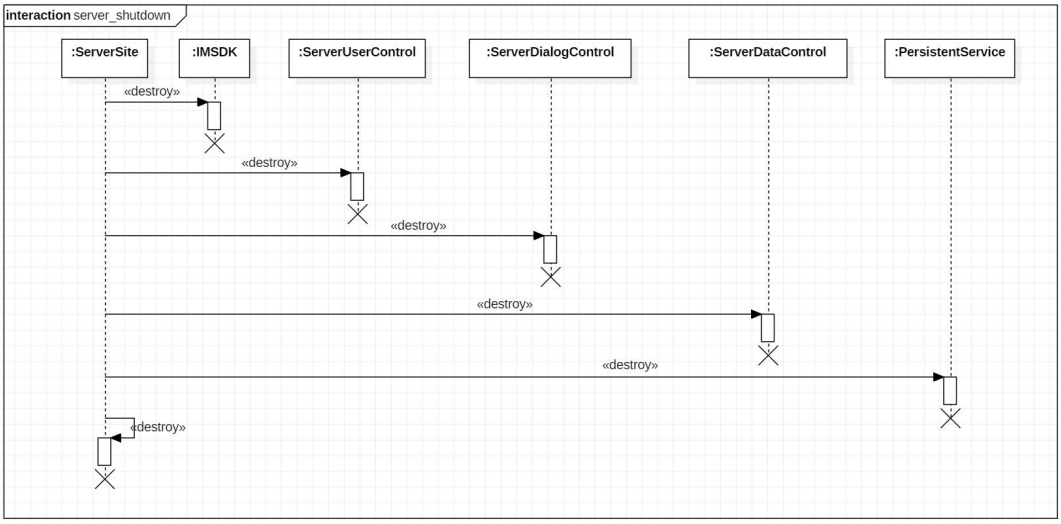

Figure 25: Use Case Realization of 'Server Shutdown'

4.6.3 App Startup

To start up the application, we need to enable our system to manage different kinds of requests.

(1) Click the App icon.

(2) Launch the 'MainPage' object.

(3) The 'MainPage' object launchs 'HttpUtil' object to handle http request.

(4) The 'MainPage' object launchs IM service.

(5) The 'MainPage' object launchs a series of 'Adapter' objects to deal with all kinds of user request. They are 'DialogAdapter', 'DataAdapter', and 'UserAdapter', which will refer to 'HttpUtil'.

(6) The 'MainPage' object launchs other 'ClientControl' objects to deal with further manage requests. They are 'ClientDialogControl', 'ClientDataControl', and 'ClientUserControl', which will refer to those 'Adapter' objects.

(7) The 'MainPage' object launchs 'LoginPage' object, and 'ClientUserControl' will be refered to it.

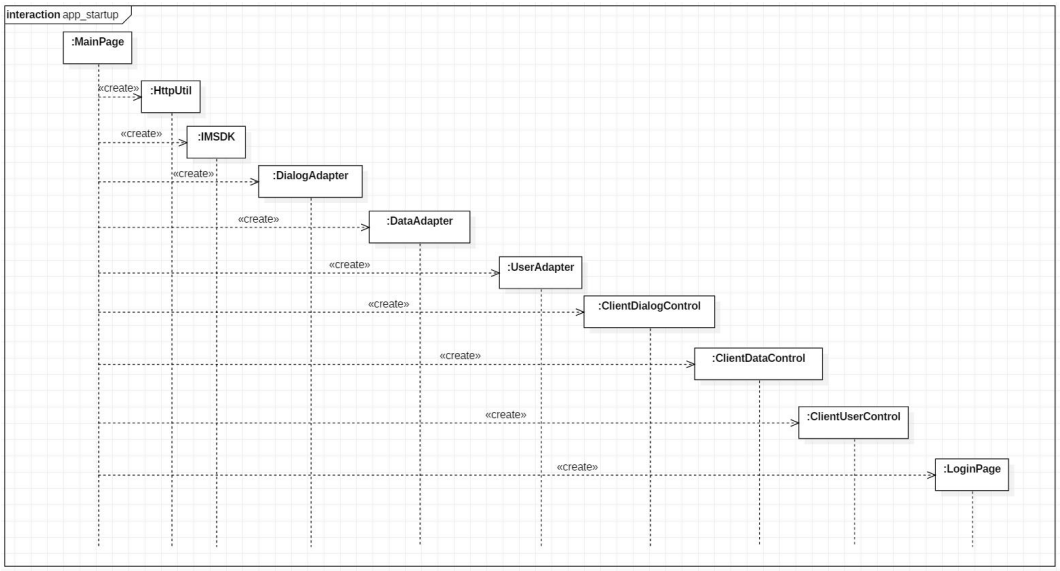

Figure 26: Use Case Realization of ’App Startup’

4.6.4 App Shutdown

(1) Close the App.

(2) Delete all the ’ClientControl’ objects.

(3) Delete all the ’Adapter’ objects.

(4) Close IM service and delete ’HttpUtil’.

(5) Delete ’MainPage’.

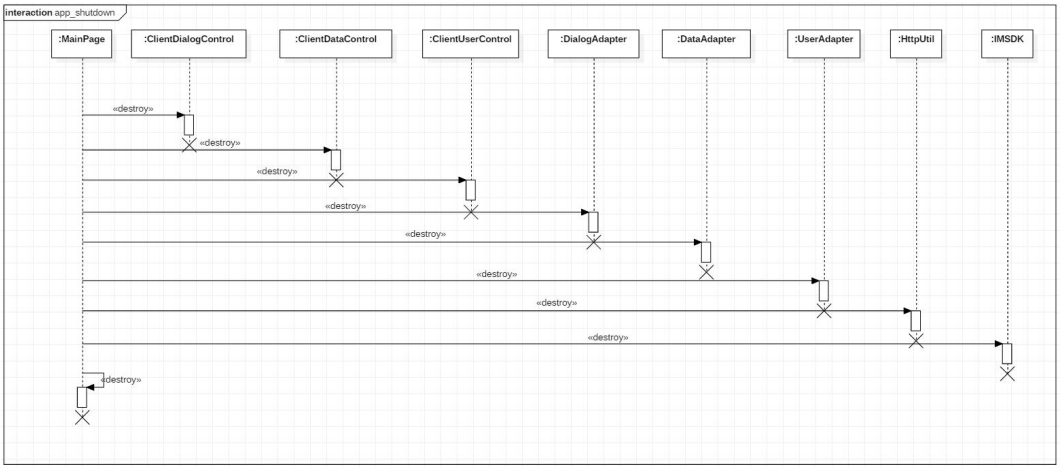

Figure 27: Use Case Realization of ’App Shutdown’

4.6.5 Error Handling

(1) If the application has an error, restart App.

(2) If the server has an error, restart the server.

(3) We can assume that our database is safe since nearly any database has event management system.

4.7 Data Management Design

We will use relational database to manage our data. One of the reasons is that we are familiar with relational databases such as MySQL. But more than that, relational databases are better at handling massive data, comparing with some non-relational databases.

Several typical tables are present as shown here (Table 2 - Table 5):

No	Field	Description	Type	Allow_NULL	Primary Key	Unit	Remark
1	userID	User ID	int	N	Y		
2	userName	User Name	String	N	N		
3	password	User Password	String	N	N		
4	mail	User Email	String	N	N		

Table 2: UserInfo

No	Field	Description	Type	Allow_NULL	Primary Key	Unit	Remark
1	itineraryID	Itinerary ID	int	N	Y		
2	userID	User ID	int	N	N		

Table 3: ItineraryList

No	Field	Description	Type	Allow_NULL	Primary Key	Unit	Remark
1	itineraryID	Itinerary ID	int	N	Y		
2	spotList	Spots on the itinerary	int	N	N		
3	cityID	City ID	int	N	N		

Table 4: Itinerary

No	Field	Description	Type	Allow_NULL	Primary Key	Unit	Remark
1	spotID	Spot ID	int	N	Y		
2	spotType	Spot Type	int	N	N		
3	arriveTime	Time when the user arrives at the spot	time	N	N		
4	leaveTime	Time when the user leaves the spot	time	N	N		
5	rate	Recommendation Rate	int	N	N		
6	description	Basic description of the spot	String	N	N		
7	itineraryID	Itinerary ID	int	N	Y		

Table 5: Spot

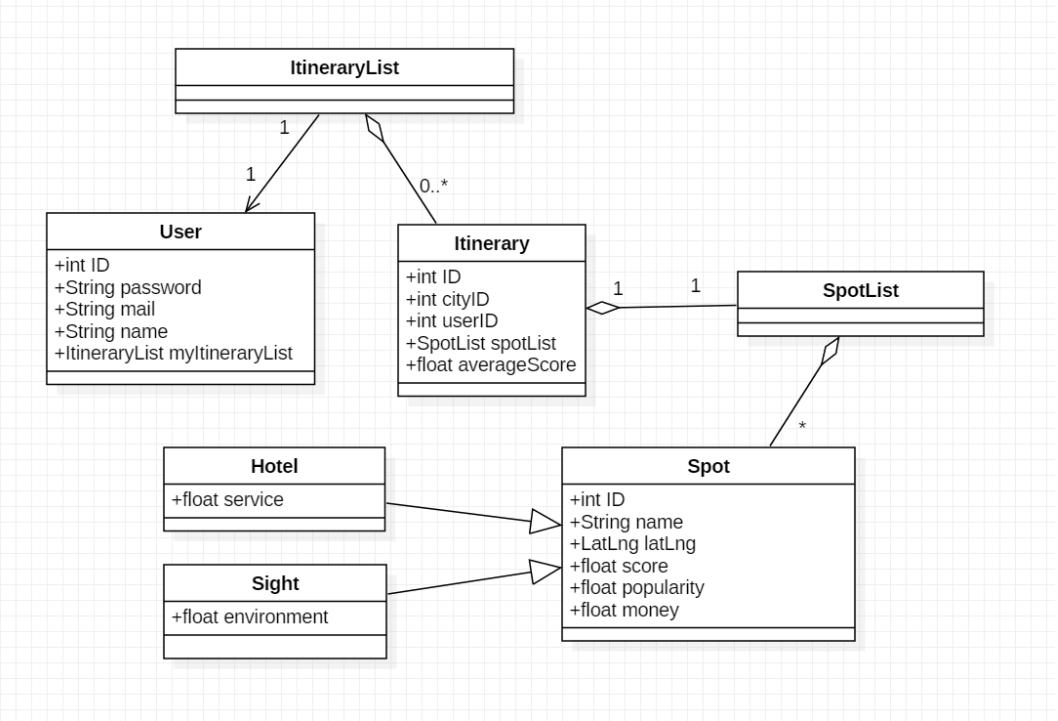

Figure 28: Entity Class Design Diagram

4.8 Other Design

In this part, we are going to talk something about our special designs.

4.8.1 Access Control and Safety Design

Logging in our system needs correct user name and password. We will match the input and our database to affirm whether it is effective. Retrieving password needs to get the verification code by origin phonenumber or e-mail address. A user who does not log in can only have limited access rights. Only the ones who log in

can visit their browsing history and give a feedback to our system.

Table 6 shows detailed access rights after login.

	User Information	Itinerary	Feedback
Log-in User	Modify Information View History	Design Itinerary customization Recommandation	Send User Feedback

Table 6: Access Control and Safety Design

4.9 Reliability Design

To make our service more nice, the feedback part will be updated monthly. That is, related database will be updated. Additionally, we will make a buck-up for the database of user's information, in order to protect data from losing and leaking. Besides, a spare server will be equipped to deal with emergencies.