Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: LodeStar

Final Report

Barış Poyraz, Berk Evren Abbasoğlu, Çelik Köseoğlu, Efe Ulaş Akay Seyitoğlu, Hüseyin Beyan

Supervisor: Halil Bülent Özgüç

Jury Members: Çiğdem Gündüz Demir and Uğur Güdükbay

Progress Report

May 3, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

List of Figures

Definitions, Acronyms and Abbreviations

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

VR: Virtual Reality

API: Application Programming Interface

HTTP: Hypertext Transfer Protocol

SSL: Secure Sockets Layer

UI: User Interface

Client: User End of the Application. Generally installed by the user on the smartphone.

Server: The part of the application which responds to Client's requests. Responsible for data management and API interactions.

Activity: In android an activity is a entry point for a user's interaction with the application[1].

3D: Three-Dimensional

1. Introduction

In today's world, what is the most valuable currency? American Dollars? Gold? Airline or credit card points? Even though they all offer value in their own ways, unarguably the most important currency for everyone is time. So, saving time should offer the greatest profit. Especially where people are forced to waste time due to unavoidable circumstances. One such circumstance is waiting at the airport, possibly for hours at a time.

Imagine booking a flight to a foreign country. The day of the exciting getaway comes knocking on the door. As the punctual person you are, you go to the airport hours before the departure time. You check-in, hand over your luggage, go through passport control and sit next to your gate, waiting for the flight. You check your watch, and then monitor the expected departure time of your flight. Which is two hours away... To make matters worse, it has also been delayed for another hour and a half. The thrill of the holiday hides behind the curtains of a dreaded airport wait. Sounds familiar?

LodeStar aims to help such passengers to utilize this otherwise wasted time efficiently. Thanks to the development of technology, it is now possible to obtain information about most countries. Favorite places to visit, must-see events, historic restaurants and so forth. While such information is already presented in many popular applications already, LodeStar diverges from such applications with one unique feature. With the help of Google Street View, LodeStar will offer users the 360° views of the airport they will be traveling to. Hours before reaching their destinations, the users will be able to see where they can buy a SIM card, rent a car, get on the train, or claim their luggage. LodeStar will show them how to go from the landing site to the mentioned places with directions, and the 360° pictures will etch the path into their minds. If the user possesses virtual reality glasses, these images will also be displayed in virtual reality for an immersive experience.

This report's departure point is an overview of LodeStar's system. In the next section, the algorithmic design of LodeStar will be explained. What follows will be the final architecture and design, consisting of subsystem decomposition and services, client and server. After that, the impact of engineering solutions will be presented. Engineering solutions and contemporary issues will be discussed. Finally, the tools and technologies used in the development of LodeStar will conclude the report.

1. Engineering and Report Writing Standards

For the descriptions of the class interfaces, diagrams, scenarios, use cases, subsystem compositions and hardware depictions, this report follows the UML guidelines[1]. UML is a commonly used way to generate these diagrams, easy to use and since it is the method taught at Bilkent University, we chose to utilize it in the following pages. For the citations, the report follows IEEE's standards[2]. Again, this is a commonly used method and the one preferred in Bilkent University.

1.2. Use of New Tools and Technologies

Firebase: Firebase is a mobile and web application development platform. This platform offers users to rapidly develop applications that require user logins, basic databases, encryption facilities and media sharing services[3].
NodeJS: Allows rapid development and deployment for web micro-services. Allows the developer to save precious development time by providing many simple libraries for building applications[4].
Docker: Allows deployment of NodeJS applications with a simple click. Stores all program code in containers and automatically manages them across a wide variety of platforms[5].
Google VR: Provides virtual reality rendering facilities for VR glasses[6].
Swift Lang: Swift is the programming language use to code applications for iOS. Swift is fast, safe and interactive and gives the ability to program for phones, desktops, servers and anything else that runs code[7]. Because LodeStar will have an iPhone application, it was necessary to use Swift for implementation.
Unity: Game Engine. Allows game developers to build sophisticated games without getting involved into shader programming[8].

1.3. Life Long Learning

The users of LodeStar will be subject to formal and informal learning opportunities throughout their lives. Using LodeStar while traveling across countries will help users to discover new places that they could've missed without the application. The Places to See page under Trip Page will provide the user with plentiful exploration options.

As the developers of LodeStar, our team will be updating the application to support what modern technology brings. VR technology is the pinnacle of our decade and LodeStar is not absent in using this wonder of technology.

1.4. Implementing Creative and Impactful Solutions

This application consists of many internal structures that other applications lack. One of them is the way we set up the server. Since this LodeStar is both for iOS and Android, we have to implement a central node which works with both platforms. The server part is implemented in such a way that both platforms are able to communicate with the server using the same protocols. Using JSONObjects and NodeJS[4], we were able create a server that suits our needs. As could be seen in the next section, most of our applications logic is implemented in the server. The client only sends web requests to LodeStar's server to gather specific data.

Furthermore, our application uses Virtual Reality as one of it's unique features. Users of LodeStar will be able to navigate through the destination airport and tourist attractions of the destination city in a virtual environment.

Last but not least, our application uses the Firebase Database[3] (a database established by Google). Firebase DB is a very strong and easy to use database. It stores the information on the cloud so that the users of it can easily monitor their database.

2. System Overview

When traveling by plane, especially internationally, customers must go to the airports early in order to check-in, register their luggage, go through customs etc. Since any or all of these things could potentially demand a lot of time due to queues or poor management, most people go to the airport very early, and have to wait a long time before boarding the plane. With LodeStar, we aim to help people value their time which would otherwise be wasted at the gates of airports by giving them a glance of the destination airport and city. This should also help alleviate some stress as the passengers will have a clearer idea of what to expect upon arrival.

Some of LodeStar's features are available in other apps. In many, many other apps... One must use Foursquare to check the places of interest, TripAdvisor to mark a route, Google Maps to follow, a weather app to check the forecast, a currency app to see the current transaction rates, and search through many webpages or blogs to find whether public transport or taxi should be their preferred choice.

LodeStar uniquely combines all of these features, and then adds some. With the API's of the mentioned applications, which are free and ready for use, LodeStar merges their capabilities. To build upon that, LodeStar shows how to navigate through the destination airport, where to get a SIM card, rent a car, buy a train ticket or find a taxi and even suggests which one to pick depending on your time and money constraints. All of these functionalities are offered in one app, all from the comfort (or discomfort) of your seat at the gate.

In addition to airports, these features can also be used in favor of tourism companies. When customers who want to make an international trip and go to the offices of such companies, they can be shown, in virtual reality, the cities of their interest. The places to visit or major attractions can be showcased. This will allow the customers to make a better decision for themselves, and increase the likelihood of them booking a ticket.

The dreaded long waits at the airport will be history. Journey into the chaotic unknown of a foreign city will be unknown to the users of LodeStar. A picture is worth a thousand words, and the 360° guides of LodeStar will outmatch any written or verbal description.

3. LodeStar's Algorithmic and Application Design

In this section, we will be discussing the technologies we have incorporated into LodeStar. These technologies have been specifically selected to have their own section since that are the most distinctive elements of the project.

3.1. Virtual Reality Algorithm and Implementation

As a key concept of LodeStar, we have used Virtual Reality to assist people to discover their travel locations. This is accomplished by rendering 360° images using the gyroscope of modern day smartphones. An instance of a SIM Carrier shop in IST Atatürk Airport is shown below.

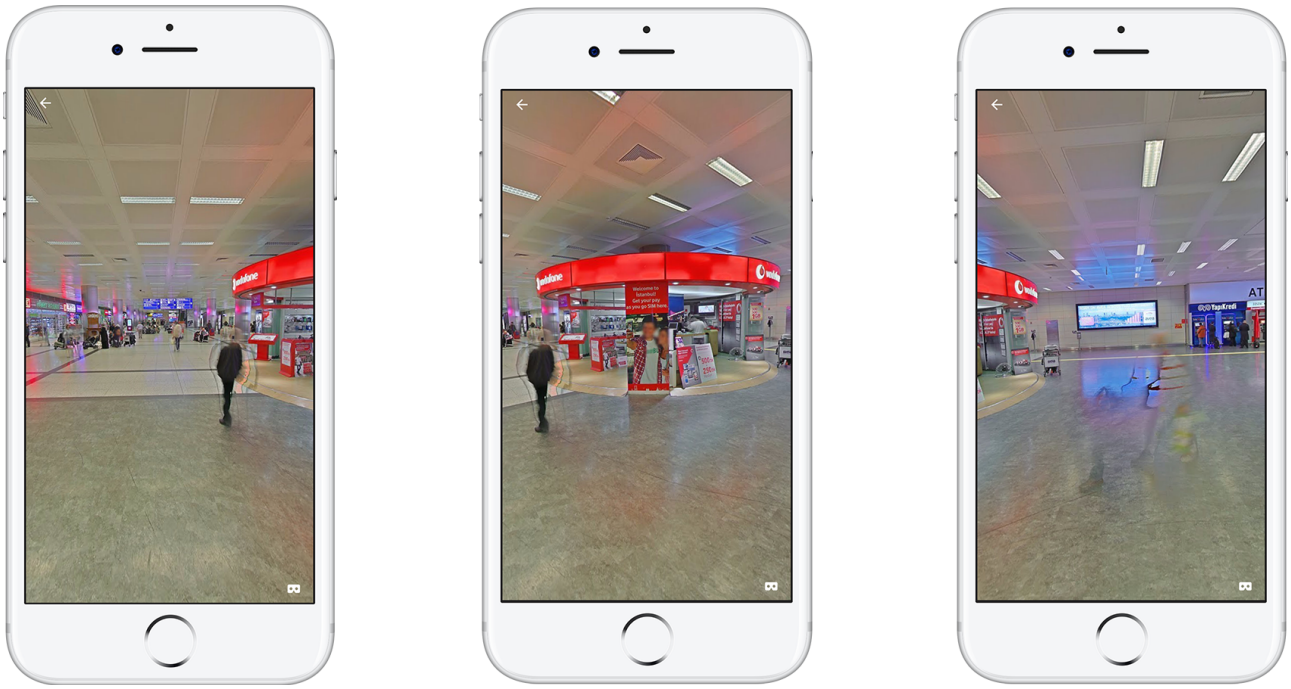


Figure 1 - Virtual Reality Panorama Mode Showcase

VR interface of LodeStar shows to user Google StreetView[9] photospheres of the Venues in the app and provide a navigation mechanism to explore in them in VR mode as well as in plain panorama display. In addition to this, we have constructed Venue and places pages with a map and an image gallery of the place so that the user may see the place beforehand and decide to visit them or not. Our focus and purpose in all these to provide user with a lot of visual and

informative material so that they will benefit from them during their discovery of their destination points. With this introduction in mind, we can discuss the implementation of virtual reality.

The implementation of virtual reality first began with the choosing the platform. Since we were determined to make the LodeStar a mobile application, our selection of VR the platform had to be Google Cardboard because of its ease of use and adaptability to different mobile devices. Google Cardboard is a VR headset designed for smart phones by Google. People can build them from low cost components and if they like it can be built for specific devices for exact fitting to different phones[10]. Furthermore, the VR SDK of the Google Cardboard supports a wide range of mobile devices which improves flexibility. After deciding on the platform, we started implementing the software that will run the VR interface.

Building a virtual reality user interface consists of two fundamental parts, one is rendering the graphics that we shows on the screen and the other is gathering the position data from the device's sensors so that when the user makes a move with VR headset on, rendered graphics will be updated. The VR library of our implementation successfully performs the second task while leaving us to only implement the rendering of the VR graphics.

The Cardboard SDK that we implemented in our project relies on Android's OpenGL ES library for rendering. OpenGL ES provides developers with an interface class called GLSurfaceView that is a child of Android class, View, that is basically used by Android to show anything on the screen. When implementing GLSurfaceView.Renderer in a class, the developer implements three functions: onSurfaceCreated(), onSurfaceChanged() and onDrawFrame()[11]. The operations to be done on every frame call of the rendering is implemented in the scope of onDrawFrame() function. Similar to this class, Cardboard SDK provides us with a CardboardActivity class that delegates similar functions to be implemented but differently, when the GLSurfaceView creates a View object to be placed inside application, CardboardActivity creates a CardBoard screen that displays the graphics in two sides on the screen for two eyes, to be placed in the Cardboard. (Figure 2)

Figure 2 - Virtual Reality Stereoscopic Mode Showcase

Graphic calculations in the Cardboard activity draws the screen with two separate points of views that have a certain distance between them to simulate 3D (real vision) of a place. Because of the pupillary distance, our eyes don't exactly see an object exactly the same but a little different. However this difference in the two eyes' vision enables us to perceive our environment in 3D. To simulate the same effect on the digital screen, we should not only draw the same screen from two points that have a pupillary distance, but also choose virtual distance carefully. If we again look at the above images, we will notice that not only the view angles, but also view distances are changed by the library to render VR screen. If we were to put these two images in the right convergence distance, the brain will perceive them as one 3D vision, called stereoscopic vision. In the right figure, we see that by adjusting the places for images of right and left eye, we can find the right convergence distance and simulate the natural vision as stereoscopic vision. Therefore, the choosing of right and left image's render positions are really important for the performance of the graphics as well as the visional health of the user. In this project, the pupillary distance is chosen as lens distance of the Cardboard.

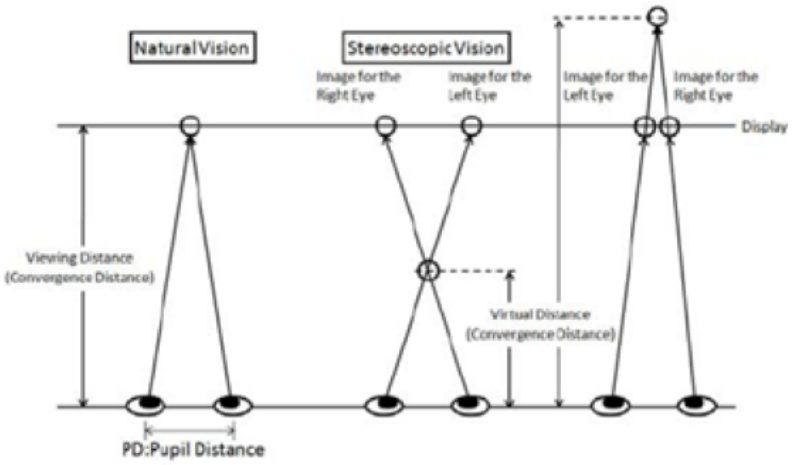


Figure 3 - Natural vs Stereoscopic Vision[53]

The places of right and left image as well as pupillary distance are calculated for different resolutions for different devices, this way we obtain optimal VR graphics. The interpretation of user's head movement are recorded by Cardboard library itself. The library needs an accelerometer and gyroscope to track and update the graphics inside the VR view. For example, user rotates their head around 40 degrees to left, the camera position of renderer are updated by 40 degrees to left, etc. Since the whole interpretation of sensor inputs are done by the library, no additional calculation was necessary for VR interface.

Our discussion up to here was to elaborate how we accomplish this task of drawing the VR interface and show why it is important to use screen parameters and sensors effectively, which was our second task. The first task, the rendering of panoramic images in the screen was done by using OpenGL ES library. This required practical graphics knowledge in contrast to theoretical VR calculations in previous part. Our main purpose for this task is to map a panoramic image that's provided to us by Google StreetView to map on a spherical object as a texture. For this purpose, we used triangle strips to draw a spherical object and used a fragment shader to map the texture points to the texture coordinates given us by the program. This way we could show a triangular object that is used to show direction in the displayed panoramic image. Since we used one main graphical object, we didn't employ additional illumination or shadowing techniques, our aim was to show the panorama as clearly as possible.

Finally, the last aspect to discuss for the Virtual Reality is navigation inside VR mode. When rendering a panoramic view, we put arrows towards the ways that user can go forward, the connections to neighbor panoramas. In the plain panoramic view, we have used ray casting to pick up the user's screen taps. If the user taps inside the view, we try to intersect the ray and triangular arrow area and if we obtain a point, the user will go forward to that direction. However, this cannot be used for the VR mode, since user cannot tap on the arrows with the Cardboard[12] headset on. We solved this problem by calculating the user's head rotation around y axis, if the user is looking at the direction of the arrow and if it presses the cardboard's button, we are navigating the user to next panorama. When cardboard's button is pressed, it simulates a tap on the screen with an extension inside to create a tap (or moves its magnet to alert phone's magnetic sensor, depending on the cardboard's type). By using this input inside the application, we navigate to the next panorama.

In conclusion, development of VR technologies in LodeStar consists of the graphical development of StreetView screens, integration of sensor inputs to these screens, and providing the user with a functional navigation tool. Therefore, although our aim was to achieve a satisfying VR interface, the effort to present them to the user required combination of several engineering skills. From interpreting the device sensors appropriately to retrieving and combining panorama images from Google StreetView, the virtual reality user interface is idealized for offering users a satisfactory VR experience during their travels.

3.2. User Experience and Interface Design

Designing an application for both major mobile platforms require the use of advanced design tools such as Sketch[13], Adobe Indesign[14] and Adobe Xd[15]. Using these tools posed a great challenge at first enabled us to achieve a consistent user experience across both platforms. To support Virtual Reality and several other features of LodeStar.

Moreover, thinking about the design of the user interface before starting an algorithmic implementation allowed us to see the shortcomings of our several solution approaches. One shortcoming of our VR implementation is, Google StreetView is not available at every location, and even if it is available for a location near a touristic place, it doesn't provide interior view of the place most of the time or it doesn't explain the route from the user's location to that place. Since we knew about these possible issues, we tried to compensate these issues by including place images and a map object that shows the location of the place. We have carefully placed other helpful elements throughout the user interface when a panoramic image is not available.

During the design phase of the Landmarks page, we thought that putting a map while displaying text based information would be really helpful for this particular screen. A potential user might want to visit several Landmarks within a limited amount of time. Having an interactive map view allows the user to plan the trip accordingly.

In the pages of our application such as Landmarks, we put a map object so that when we list the Landmarks in a city, the user can also see the them on a map along with their own location so that they could find see the route to that location. By tapping on a Landmark for instance, the user opens the Venue page of that location and sees the place details such as address and reviews, alongside the panorama of the place, if available. Alongside those, since it seemed more convenient to use readily available resources, we get the place images from Google or FourSquare and show them in an image gallery in every Venue Page. In the possibility that place doesn't have a street view nearby, user can use these images to have an idea about the place or gain travel experience in case they want to visit it.

Another example of good user experience design could be the Flight Information page (see Appendix A for a screenshot). This page shows

4. Final System Architecture Design

LodeStar's architecture is formed by several distinct subsystems. These subsystems are vital to the main system's success and must work in harmony among each other. In this section, LodeStar's subsystems will be showcased. The interactions between these subsystems, the classes they consist of and their functions will be discussed. The full class diagram depicting the full architecture will be presented in the last pages of this report.

4.1. Subsystem Overview

Before proceeding with the Class Diagrams in Appendix B, it is best to showcase a high-level architecture design of LodeStar. The following sections give a brief about the architecture.

4.2. Client

The client side of LodeStar consists of the mobile applications that will be running on iOS and Android devices. The client side of the system consists of two packages, View and Controller packages. The client side will enable user to authenticate to the system via establishing a Firebase connection. After getting the user information, the client will navigate to the Home page where user will submit their travel information. When sending or retrieving information from LodeStar's server, the Controller package comes into play so that trip information such as flight details or places to see in the travelled location will be requested by the client side. The View package will consist of classes that will display these information to users by creating the interfaces.

Basically, the client side is the user end of the application program. Using the mobile application that will be developed for iOS and Android, the user will interact with the system. When user enters their Login or Sign Up information in the application, the client will send the login request to the server and the user will be logged in to the system. When travel information is obtained by client, another request will be sent to server and the client will load the information about transport options, weather, flight information, shopping, lounge services, restaurants, places to see, living expenses and accommodation.

Client subsystem will be implemented using Model-View-Controller[16] design pattern. Controller subsystem will be responsible for the connection between client and server, when data is sent, controller collects it and when responses are taken for requests, controller collects it. View subsystem will be responsible for interface operations. Displaying pages or taken data on the

screen will be done by the view subsystem. Since all the data of the client will be taken from the server, implementing a separate Model subsystem will be trivial, therefore we will only use Controller and View subsystems in the client side.

4.2.1. View

The view side of our system will consist of user interfaces that the user will encounter while using our application. The purpose of the view is to present the user with a friendly UI so that he/she will be better able to communicate with the application. For each view, there exists a controller to provide an interaction between client and server.

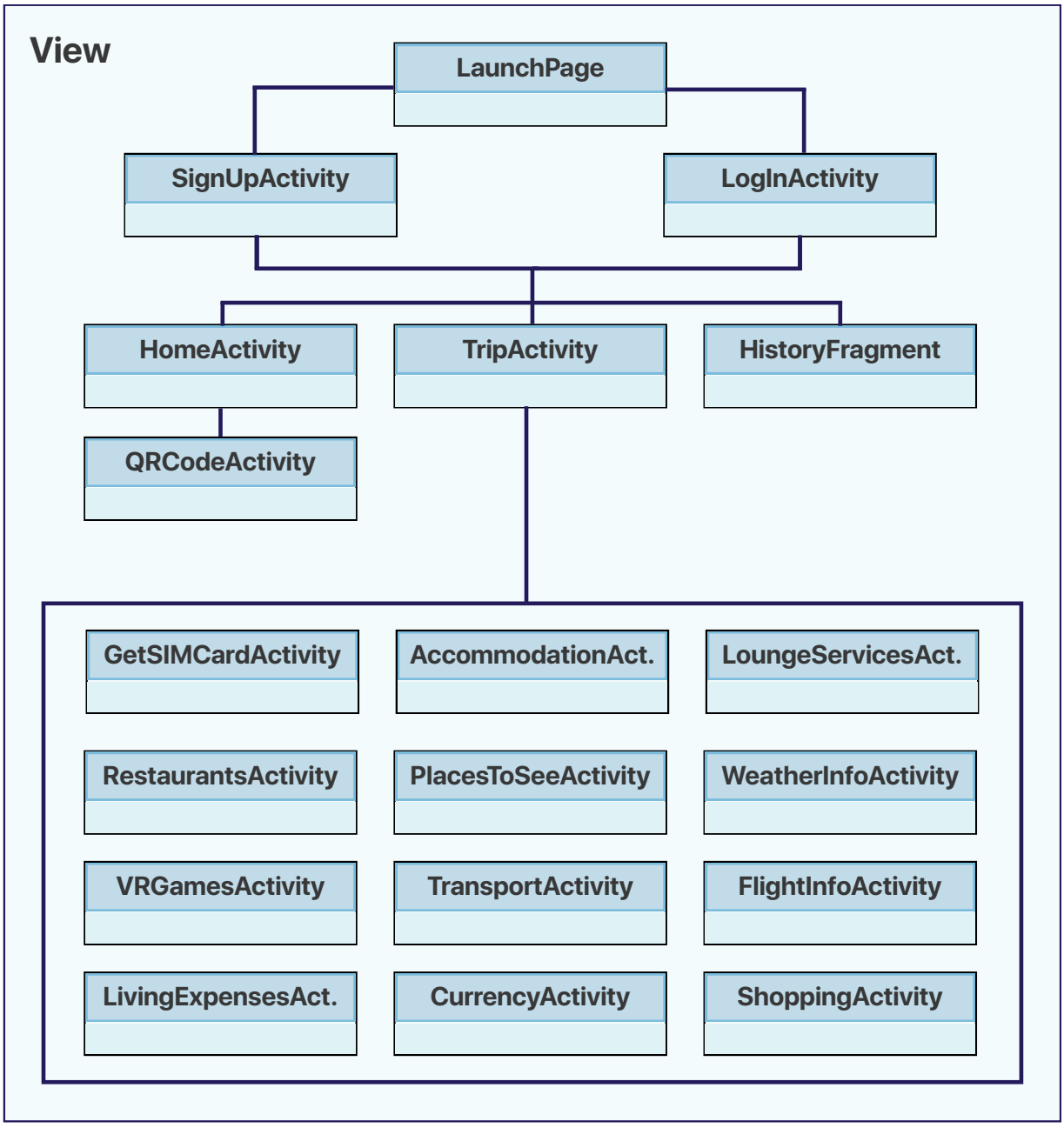


Figure 4 - View Subsystem

LoginActivity: This class accomplishes the Login related duties of the application. If the user has an account he/she will be able to use the methods of this class for authentication.

SignUpActivity: This class will accomplish the Signing-up related activities of the user. This will be the first authentication step the user will need to pass in order to be able able to sign-up for the account.

HomeActivity: This class is the view for main page which will open after user authenticates to the application. It will display interfaces for entering flight number so that user may initialize their trip page.

QRCodeActivity: This class is responsible for detecting the Barcode objects, specifically for the type in boarding passes, then parses the information and sends to the TripActivity.

TripActivity: This class is responsible for displaying the Trip page that will show cities in the travel and interfaces to related pages.

TransportActivity: This class is responsible for showing the transportation options with an estimated cost with respect to the transportation costs, such as taxi fare rates, for that country

WeatherInformationActivity: This class is responsible for creating WeatherInformation objects for a given city by fetching 5-day forecast of that city from the server. This class is also responsible for notifying the adapters so that view can be changed.

FlightInfoActivity: This class is responsible for displaying flight details taken from server side of the system according to flight number.

ShoppingActivity: This class displays the shopping areas in that location

CurrencyActivity: This class displays the currency rates between travelled countries.

RestaurantsActivity: This class displays the top quality restaurants in that location

LivingExpensesActivity: This class is responsible for displaying living expenses by giving most common used things as examples.

PlacesToSeeActivity: This class is responsible for fetching the top places to visit in the specified location

AccommodationActivity: This class is for displaying accommodation information taken from the server.

HistoryFragment: This class is responsible for displaying user history of user's trips.

FavoritesFragment: This class is responsible for displaying the user favorites.

VRGamesActivity: This class displays the VR Game/s and allow users to choose a game

GetSIMCardActivity: This class displays instructions on how to get a SIM card on the arrival airport.

4.2.2. Controller

Controller package is responsible for the information transfer between server and client side. These classes are associated with their respective view (activity) class to get information for them from server side. This is why the diagram looks highly similar.

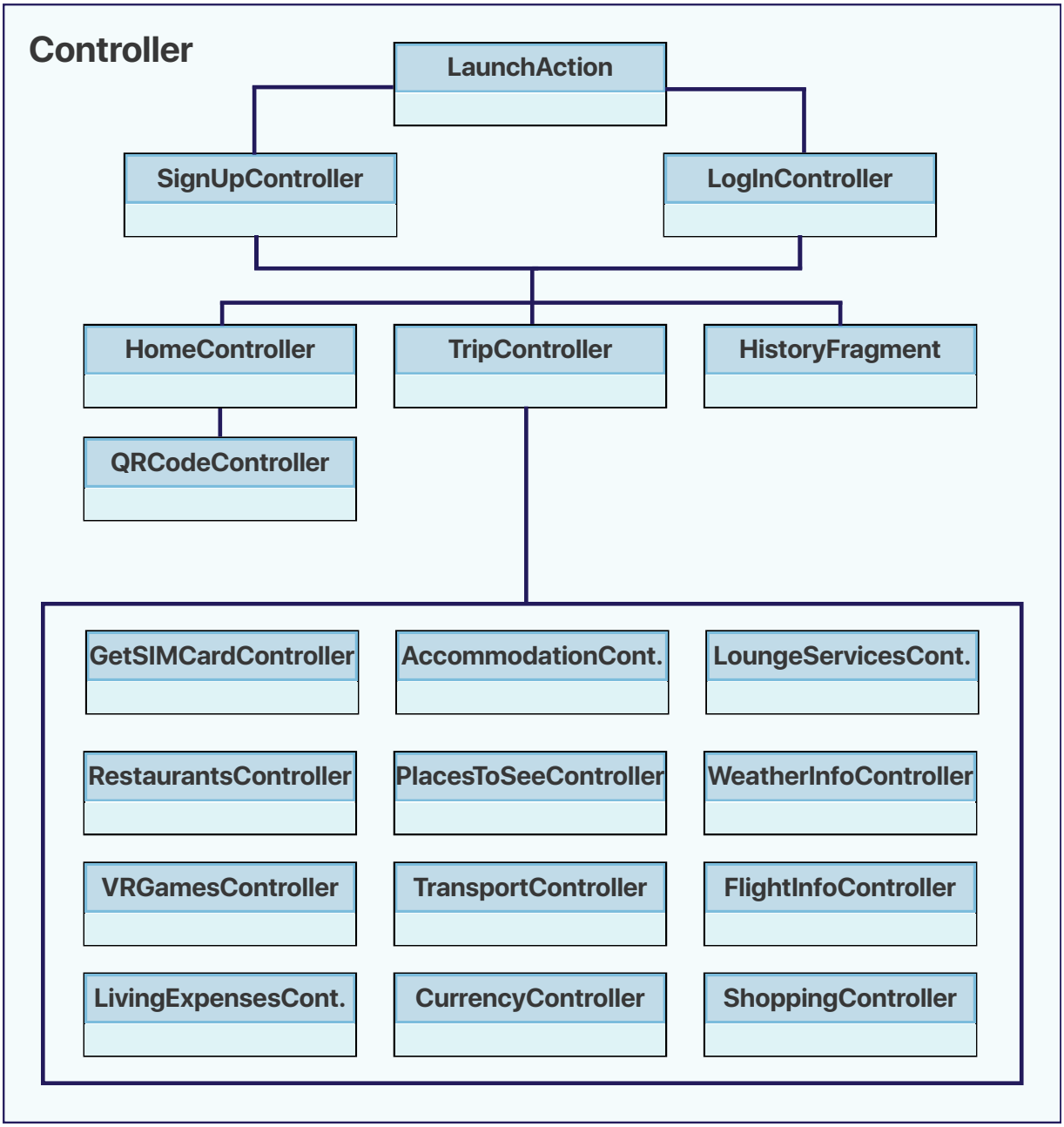


Figure 5 - Controller Subsystem

HomeController: This class is responsible for the controller function related with HomeActivity class, mainly for checking the validity of entered flight information.

HistoryController: This class is for getting the user history from server for logged in user in the client.

FavoritesController: This class is for getting the user favorites information from server for user.

UserController: This class is responsible of the controller activity elated with UserActivity class, like user preferred settings for the client side.

TransportController: This class is the controller associated with TransportActivity class, for getting transport information from server side.

FlightDetailsController: This class is the controller associated with FlightInfoActivity and TripActivity, for obtaining flight details from server.

PlacesController: This class is the controller for PlacesActivity class, for getting the information of nearby places to see from server side.

CurrencyController: This class the controller class for getting currency rates from the server, related with CurrencyActivity page.

ShoppingController: This class is the controller associated with ShoppingActivity class, for getting shopping information from server.

RestaurantController: This class is the controller associated with RestaurantActivity class, for getting shopping information from server.

AccommodationController: This class is the controller associated with AccommodationActivity class, for getting accommodation information from server.

LivingExpensesController: This class is the controller associated with LivingExpensesActivity class, for getting living expenses data from server.

GetSIMCardController: This class is the controller associated with GetSIMCardActivity, for displaying instructions on how to get a SIM card on the arrival airport.

4.3. Server

This part of the application is where all non-local data is processed. Examples of this data would be flight information, weather, currency exchange information and areas of interests. Moreover, the server part will store and process user specific data such as Trip Logs, History and Favorites. Adding on, the server is responsible for API interactions to collect and process data from different platforms such as FlightAware[17], Google, Foursquare[18], Numbeo[19] and OpenWeather[20]. The server gathers this data on demand.

The interaction with the server starts when a user logs in. However, the major part starts when the user wants to scan a boarding pass. In this case, the server will start by fetching all the flight details. Then, it will analyze the flight to see which services are available. According to this data, the user will be presented with LodeStar's available services in the Trip Page.

Server has two layers. Logic Tier and Data Tier. Logic Tier is where all user interaction is handled. Logic Tier interacts with the client in a request/response manner. Every time a user wants to access a service of LodeStar, Logic Tier will handle the request and generate the appropriate ate response for the user. Data Tier includes a Database Management Subsystem. This subsystem handles user data, such as a user's preferences, favorites and trip logs. Basically, this database is where all the persistent objects are stored.

Since LodeStar will be relying on web micro-services, the server is a very crucial part of the project. The server will play a major role starting with just a simple login from a user. Almost every action taken by the user will require a communication with the server. The server will contain usernames and corresponding preferences, reviews, trip logs and favorites. Furthermore, the server is responsible for fetching required data such as flight information, weather, shopping, nearby attractions and transportation details.

4.3.1. Logic Tier

This layer is responsible for all major operations of the system. The part of the server will communicate with many different APIs to service LodeStar's trip page. When the client sends a request, the DockerEngine will parse the request and transmit the request to the appropriate micro-service.

DockerEngine: this class is provided by Docker libraries. It encapsulates other classes and manages them during runtime. If one of the micro-services classes fail, it restarts the service and does some error logging.

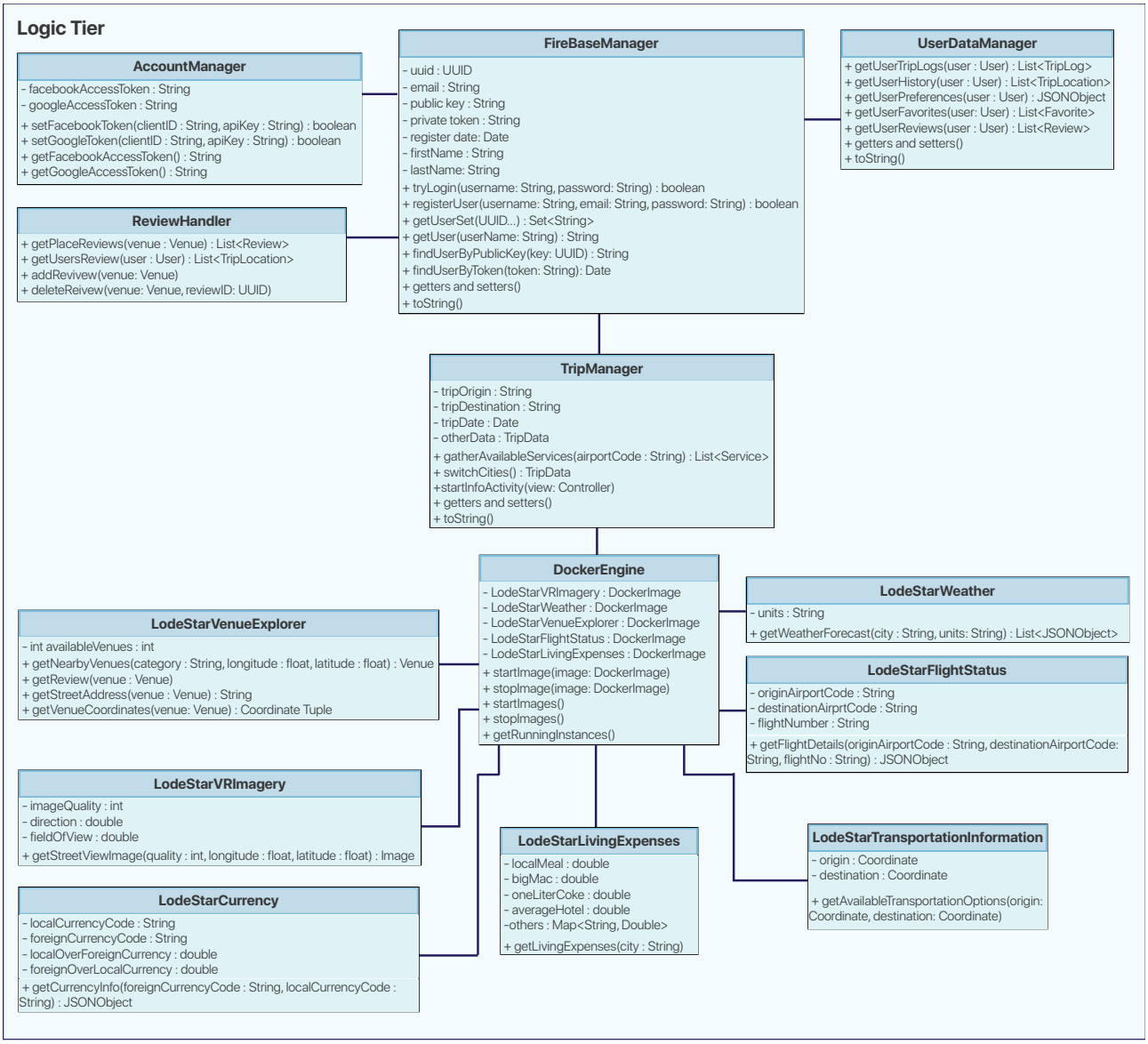


Figure 6 - Logic Tier Subsystem

Vectorized Version Available on Our Site at http://lodestarapp.com/files/ciri.pdf

UserDataManager: is responsible for managing user preferences, trip logs, favorites etc. Communicates with the Data Tier to save the user's preferences. The client will send a request to this class, and the class will call the necessary functions to give the appropriate response to the client.

ReviewHandler: is responsible for handing review requests. When a user wants to write a review, the client will communicate with this part of the server.

AccountManager: LodeStar will keep user preferences in the server. This class will communicate with the client when the user triggers an operation that requires account authentication.

FireBaseManager: This class will be used to communicate with the FireBase database. This database will hold user information and store them in a JSONObject format.

TripManager: is responsible for responding the user's requests while the user is on the Trip Page of LodeStar. This class will call all necessary functions of the following classes.

LodeStarVRImagery for the VR functionality, we will be requiring Google's Street View APIs. This class will be communicating with Google's APIs to retrieve 360° images. When the user wants to see the nearby locations in VR, 360° images for the place will be needed. This class will send requests to Google 360° StreetView to get available images for the location.

LodeStarVenueExplorer: to get available attractions in the city, LodeStar will be relying in FourSquare APIs. The API will retrieve information about nearby places, their addresses and reviews. This class will recognize and sort this data before sending it to the client.

LodeStarFlightStatus When the user scans a boarding pass to get into the Trip Page, this class will send a request to the FlightAwareAPIs to get any available flight data. This data will include almost everything that a user would want to see. Even information about if the flight will have wi-fi or not.

LodeStarTransportationInformation: This class will help the user to create a route given the budget options. This class will take the budget and the location to find possible ways to get from point A to point B. This data will be gathered from Google's APIs.

LodeStarLivingExpenses: This class will gather living expense costs. This data will be provided by Numbeo[19]

LodeStarCurrency: when the user wants to see exchange rates, this will send requests to OpenExchangeRate[21] API and retrieve currency exchange information

LodeStarWeather: this class will send requests to OpenWeather APIs to get weather data for specified city. [20]

4.3.2. Data Tier

This layer manages interactions with the database. It will communicate with the Logic Tier to service requested data.

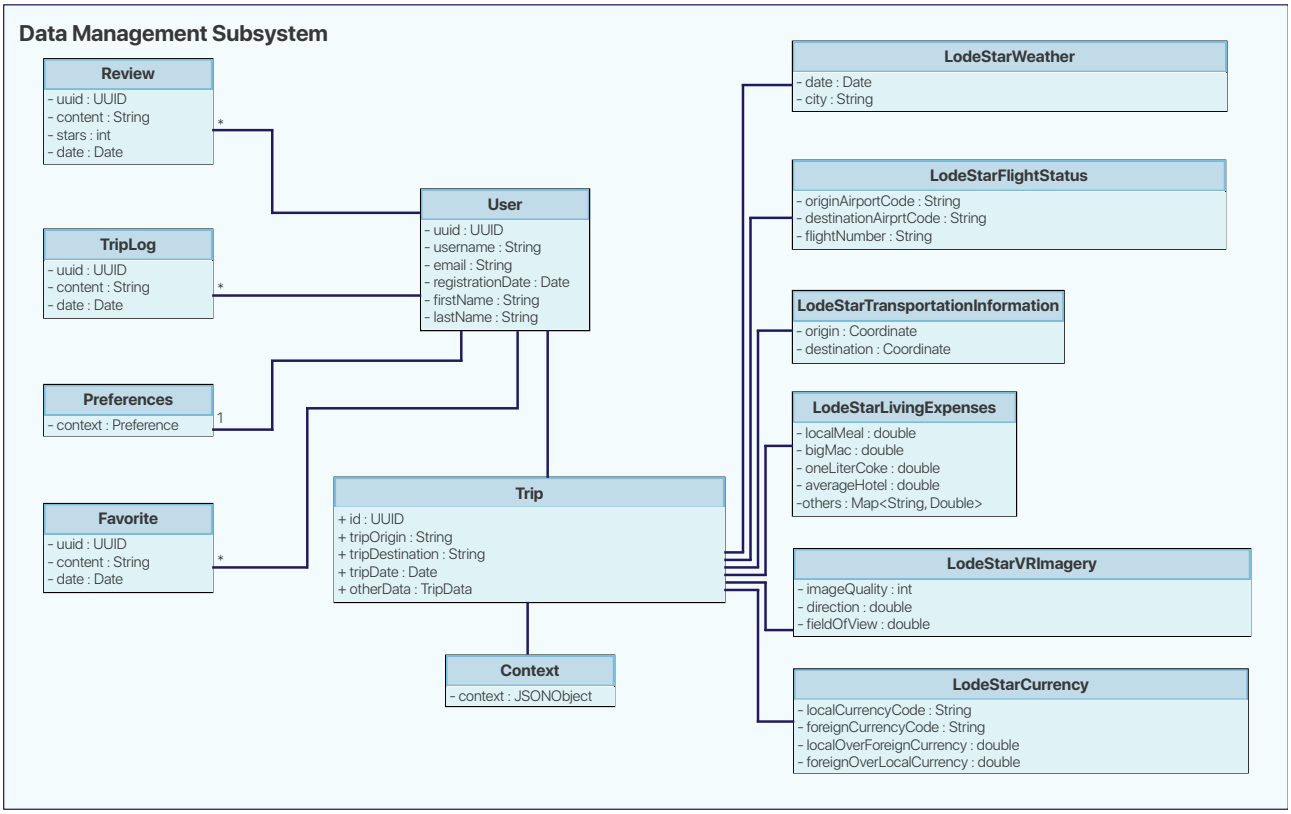


Figure 7 - Data Tier Subsystem

Trip: this class is responsible for caching data for trips. For example, LodeStar will not send many many requests for getting the weather information for the same day. Additionally, every airport will have different services. LodeStar will store available services for the

LodeStarVRImagery: This class will manage cached 360° images for airports.

LodeStarWeather: this class will manage cached weather information for previously requested cities.

Context: this class will manage airport services. It will store which airports provide which services. We will use this data to display the services available on the Trip Page.

LodeStarFlightStatus: Since we need to reduce the number of requests to FlightAware API, we need to cache flight information responses. This class will be responsible for storing flight informations.

User: this class manages user specific data. It is responsible for handling Trip Logs, Preferences, Favorites and Reviews

TripLog: the user will be able to share their trip experiences in a short paragraph. They will also have the ability to showcase these short texts on their profile pages. This class will manage the storage of Trip Logs

Preferences: this class will be responsible for the storage of user preferences in the database

Favorite: the user will be able to add other trip logs and places to their favorites so they can see them later. This class will be responsible for the storage of user favorites in the database

Review: the user will be able to review places they have visited. This class will be responsible for the storage of user reviews in the database

5. Impact of Engineering Solutions

While developing LodeStar, we thought the solution we have developed might have several impacts on today's engineering solutions. The implications we've found are as follows:

5.1.1. Economical Impacts

- Virtual Reality Glasses will be needed for the virtual reality function of the application. The cheapest one (Google Cardboard™ VR platform[12]) costs around 1-2$ on various websites on the internet.
- For Android™ platform , a fee of 25$[57] needs to be paid so that our application can be published on Google Play™. This payment will be done only once.
- For iOS, a fee of 100$[56] will need to be paid. This payment has to be done every year so our application can remain published on App Store®[58].
- All the APIs, frameworks and libraries used will be free. (For example, Google Street View™ mapping service API will be used for virtual reality functionality)
- There will be no in-app purchases. App will be free to download. Business model is described in the 1.2.8 Sustainability Constraints.
- Flight Aware API[17] will be used. This API is free for use for 500 times a day. For the scope of this project, this number is more than enough.
- Domain of the website of LodeStar will be purchased. This costs 8.88$[22] per year.
- A server is required to contain necessary data. Examples of these data include: personal profile information and guides.

5.1.2. Environmental Impacts

- Our application may use virtual reality glasses (if the user wishes to use Virtual Reality functionality). Google Cardboard's environmental impact[23] is almost nonexistent.
- A smartphone will be required. Which contains a Li-ion battery which may need to be properly disposed[24] once the smartphone is out of order.

5.1.3. Social Impacts

- The users of LodeStar would not have to spend their time on less important or seemingly boring activities while waiting for their plane to arrive on the gate. LodeStar will present them information so they can be more educated about the country they are about to visit.
- Our application will enable the users to interact with each other via Facebook integration. For example, while user A is virtually traveling the Atatürk Airport, he will be able to see the comments made by his friend user B about certain places in the virtual environment. This could be exploited where a Facebook user may follow advertisement company accounts.
- The application will not have any sort of mechanism that enables negativism while posting comments or likes.

5.1.4. Political Impacts

- Although the application will be able to show the surroundings of almost all airports around the world in virtual reality, there had been political conflicts with Google and some governments[25]. Thus, some parts inside the airport cannot be photographed due to security concerns.

5.1.5. Impacts on Security

LodeStar uses the information of their users by requesting permissions. Without requesting any permission, it does not access and uses the informations . In order to provide a secure system to the users, LodeStar uses the users existing accounts on Facebook or Google, or it provides registration by using Firebase system of Google.

5.1.6. Impacts on Health & Safety

Since the users of LodeStar will be able to learn about some of the services in the destination airport, they will have an idea about what to expect in terms of food, transportation services and perhaps, pharmacy shops. This way, LodeStar may help people with disabilities or other health issues since they know what services are available in the arrival airport.

6. Tools and Technologies Used

This section lists the tools, frameworks, libraries and APIs used during the development of LodeStar.

6.1. Development Tools and Frameworks

With the help of IDEs and other development tools, we have sped up our development cycle. Some of these tools provide syntax checking and debuting abilities while others enable us to manage dependencies automatically.

6.1.1. Android Studio

Figure 8 - Android Logo[26]

Is used to develop the Android platform of our mobile application. It is one of the most well-known IDE used to develop mobile applications in Android. In order to be able to develop the application, we ran our code on both emulators and actual Android phones. The IDE has a very friendly user interface and some in-built functionalities to create custom activities. It also has code completion and allows source control via GitHub[27].

6.1.2. Xcode

Figure 9 - Xcode Logo[28]

Is the application development IDE that LodeStar uses for iOS development. Xcode is one of the most common IDEs to build iOS mobile applications[28]. It's friendly user interface creates convenience to us developers to be able to efficiently code. It comes with code completion features and is very fast compared to other ides in the market. It also allows source code control via integration to Github[28].

6.1.3. CocoaPods

Figure 10 - CocoaPods Logo[29]

LodeStar uses CocoaPods to manage the dependencies of our API's and libraries for Swift[7]. It's main duty is to manage dependencies and API's in our iOS application. Since LodeStar integrates many API's and works with several libraries, CocoaPods was a very important tool for our application[29].

6.1.4. Github

Figure 11 - Github Logo[27]

GitHub is a very well-known git platform. We used GitHub in our project to manage the source code of our application[27]. As a team of five developers, we needed a platform where we can collaborate and work on applications together. To manage our code, we integrated both the Android Studio and Xcode with our Github repository. This allowed us to work together independently.

6.1.5. NodeJS

Figure 12 - NodeJS Logo[4]

LodeStar uses NodeJs -an open-source (also free to use) web service development framework- to build our server in an efficient manner[4]. NodeJS runs Javascript on the background. To use our API's and server functionalities both iOS and Android applications use NodeJs.

6.1.6. Firebase

Figure 13 - Firebase Logo[3]

Firebase is Google's database platform that enables developers to quickly build their applications[3]. In both of our applications we used Firebase database to efficiently manage user and API data. All the data in our application is stored securely in Google's own cloud. Furthermore, Sign in/up features are implemented using the inbuilt authentication functionalities of Firebase.

6.1.7. Docker

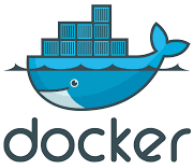


Figure 14 - Docker Logo[30]

Docker is a software containerization platform[30]. Our applications use Docker for continuous integration & deployment and utilization of the server. Furthermore a Docker engine is built on top of the server.

6.1.8. ImageMagick

Figure 15 - ImageMagick Logo[31]

ImageMagick is a software suite that allows to display and edit image files and animations[31]. The team used ImageMagick to create custom images and animations.

6.1.9. Paw and Postman

Figure 16 - Paw Logo[32]

Figure 17 - Postman Logo[33]

Paw[32] and Postman[33] is a tool that we used for debugging server response/requests. Many activities in our application interact with the server. Debugging the server by traditional methods would have been a great hassle. That is why the Paw and Postman tool was very beneficial to save our time.

6.1.10.Sketch

Figure 18 - Sketch Logo[13]

Sketch is a tool built for designers and developers[13]. Before building the applications, the team used sketch to create user-friendly and interactive mockups. While developing the application we took these mockups as reference and created the identical user-interfaces. This tool helped us visualize what our application will look like before building it.

6.1.11.Adobe Indesign

Figure 19 - Adobe InDesign Logo[14]

Android Indesign is a design application used my many people[34]. It is used to create custom images and logos. Our application uses many custom made images and logos. We have used InDesign for images that contain writings. Therefore, Adobe Indesign was a very useful tool for our application.

6.1.12. Adobe Illustrator

Figure 20 - Adobe Illustrator Logo[35]

Adobe Illustrator is a software used by designers and developers to create impressive artwork[35]. This tool is used to assist the design of our application which are generally used for images. The purpose of the illustrator is very similar to the purpose of the Indesign.

6.1.13. Google Cardboard

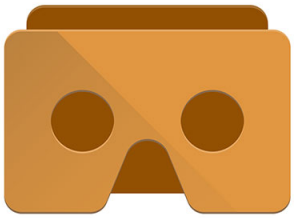


Figure 21 - Google Cardboard Illustration[12]

Google Cardboard may be used with LodeStar for the convenience of the users. Google Cardboard takes the VR and street view image experience to another level. Additionally, the cardboard is very cheap. This enables many people to experience the VR functionality of our application[12].

6.2. Library Resources

In the present section, the library resources LodeStar uses will be discussed and elaborated in detail.

6.2.1. Google Play Services

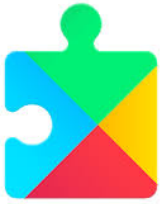


Figure 22 - Google Play Services Logo[37]

Google Play Services Library is used in Android platform of our application to make sure the user can utilize the Google Play functionalities (e.g. geolocation, device sensors, app statistics)[36].

6.2.2. Android Libraries

Figure 23 - Android Libraries Logo[26]

LodeStar uses many Android Libraries for both front-end and back-end UI operations (e.g. Widget, Animation, OS, Graphics etc.)[38].

6.2.3. Google VR Toolkit

Figure 24 - Google VR Logo[6]

LodeStar uses VRToolkit library of Google for VR related operations[6]. This toolkit contains many useful features such as panorama view and VR retrieval.

6.3. APIs Used

The following APIs were used to improve and enhance the capabilities of LodeStar. Moreover, they shaved off a lot of development time for the team.

6.3.1. OpenGL

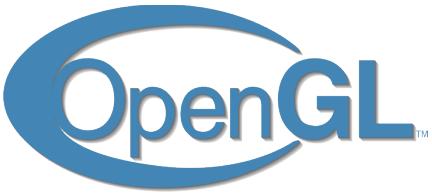


Figure 25 - OpenGL Logos[39]

OpenGL stands for Open Graphics Library used for rendering 2-D and/or 3-D images[54]. It is used for VR functionality of our application.

6.3.2. OpenWeatherMap

Figure 26 - OpenWeatherMap Logo[20]

LodeStar uses OpenWeatherMap API to get weather info[20]. This is later used in Weather Information and Flight Information pages.

6.3.3. Google Maps

Figure 27 - GoogleMaps Logo[41]

LodeStar uses GoogleMaps API to present the user the transportation options and landmarks[40].

6.3.4. Google StreetView

Figure 28 - Google StreetView Logo[42]

Google Street View Image API is used to fetch street view images in LodeStar[42]. These images are used to display 360 venue pictures in Virtual Reality.

6.3.5. Numbeo

Figure 29 - Numbeo Logo[19]

LodeStar uses Numbeo API to fetch the cost of living for an individual[55]. This information is used in the Living Expenses page in LodeStar.

6.3.6. Foursquare

Figure 30 - Foursquare Logo[43]

LodeStar uses Foursquare API to fetch information from venues of user's choice[44]. Foursquare is used to give venue recommendations and gather information such as reviews, ratings, pictures, coordinates, address and phone number.

6.3.7. FlightAware

Figure 31 - FlightAwareLogo Logo[17]

LodeStar uses Flightaware API to fetch detatiled data about a given flight[59]. This information is later displayed in the Flight Information page.

6.3.8. OpenExchangeRates

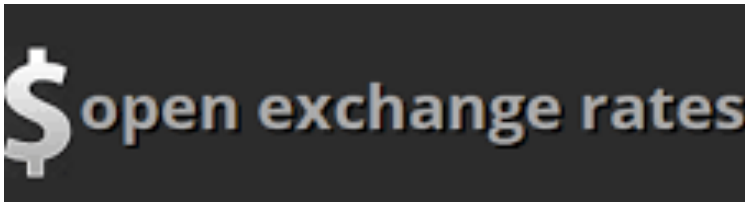


Figure 32 - OpenExchangeRates Logo[21]

LodeStar uses Open Exchange Rates API to fetch the current exchange values[21]. The exchange rate information is very accurate.

7. Engineering Solutions and Contemporary Issues

This section will explain some of the issues we've faced during development. These are not exactly problematic issues but rather hard to make design choices. With every decision made, comes a trade-off. Some of the important topics raised during the development of LodeStar are explained below.

7.1.1. Language and Communication

LodeStar is implemented in English because it is one of the most commonly used languages in the world[45]. This should allow many users worldwide to be able to use LodeStar with acceptable fluency. Some of LodeStar's functionalities like directions to landmarks or comments about restaurants are vital to be understood clearly by the user. Therefore, English is thought to be the overall best choice.

7.1.2. Internet Access and Speed Issues

While continuous internet connection will not be essential for the functioning of the application, it is certainly a plus. To show the user real time landmarks, directions and comments, LodeStar requires internet connection. Even though some predefined queries can be saved while the user has internet access to later be used when the user does *not* have internet access, continuous internet support is fundamental for the application to work as intended. Because cafes, almost all Starbucks and McDonalds, and many bus stops are offering free Wi-Fi , users should not have too much trouble finding an internet connection while traveling[46, 47, 48, 49]. Most airports also offer free Wi-Fi, which is where LodeStar expects most of its users to be at during their prime usage hours[50].

7.1.3. Security Issues

Security is also a concern never too far from any mobile application's tail[51]. LodeStar takes the issue of security, safety and privacy seriously, not as a minor issue. Because of security's utmost importance in all things internet, LodeStar encrypts all user data and has two factor authentication for login.

An obstacle LodeStar faced during development was the tremendous amount of data that needed processing. Since many different applications feed LodeStar's data, sometimes things

can get complicated. Some examples are directions from one place to another, venue information, recent and top rated comments about a landmark, weather information and currency rates. Fetching and processing all of this data challenged LodeStar to a great extent. To overcome this challenge, LodeStar stores most of its data in the database rather than within the running application. This allows speed, efficiency and reliability.

7.1.4. Platform Development Issues

Another challenge during LodeStar's development was to generate two similar, if not exactly parallel, applications that run on iOS and Android. In order to minimize differences among the different versions, developers relied on user interface mock-ups. The mock-ups were drawn at the start by an interior artist with extreme skill, vigilance and discipline; not left to chance. Also, weekly meetings ensured both projects moved swiftly.

Since LodeStar demands data from many applications like FourSquare and Weather.com, it can be as reliable as the least reliable of these applications. This is a limitation which LodeStar attempts to solve by storing the latest data to its server at regular intervals. Therefore, if one API fails, there should be sufficiently relevant data that is stored in the LodeStar server.

These challenges tested LodeStar and made it stronger, like all the hardships it faced. Overcoming each obstacle added to the strength, rigidity and durability of LodeStar and manifested an application that would otherwise be lacking.

7.1.5. Data Privacy Issues

In terms of our application, we can divide the professional and ethical issues into six pieces. These are using the user's location data, using third-party sources to provide suggestions, using login API's, encrypting and protecting private data, and agreements with other companies.

Firstly, our application is going to access user's location if only the user is willing to give the data. An option will be given to the user for that purpose. This data will be used to generate routes from that specific location to a destination location. In case that the user does not want to give their data, this data will be asked as an input. In a scenario, where a route begins in an airport and ends in a destination, the source location can be either taken from the user input or from the information in QR code which user might have scanned.

Based on the reviews and ratings, Foursquare™ generates some top places in order to eat, stay, visit, etc. In our application, these lists will be used to give suggestions to our users. Therefore, we will have to assume objectiveness and integrity of this content. For future work, extra features can be added to enable users to give their feedback on the places they visit.

Additionally, our application will be using login API's of Google and Facebook. In this way, we will be able to manage user information easier and safer, but if user does not want to login by their Google or Facebook accounts, our application will allow users to register using an email and password. For those users, as a future work, a two way authentication system can be implemented.

User's private data will not be shared among third-parties or otherwise. Moreover, all user data on the server side will be encrypted. In case that we come to an agreement with a third-party in the future, we need to ask for a permission from the user to use and share their data.

On top of that, the codes included in the application will either be our development or free source. If the application includes free source code, this will be indicated. Any additional source relevant to the application development will be properly referenced.

7.1.6. Gender Issues

LodeStar does not discriminate any gender. In nowhere in the application is the user is expected to enter his/her gender. All the users are identified by their usernames. LodeStar is an application that respects every gender, race or background that any person may have. All the features built-into the application can be used by anyone without any gender obstacles.

7.1.7. Future Work

As future work, if an agreement can be made with an airline such as Turkish Airlines, they may provide some tailored special offers. If we have an agreement for such a case, to provide the data, we need to ask the user for permission.

Moreover, with an increasing non-English speaking user base, other language support can be implemented. Though, these are as said, future work and are not included in the final version of the project.

8. Appendix A - User Manual

In this appendix, a user manual will be presented. Each sketch is provided with a guide to better specify how LodeStar would interact with the user.

8.1.1. Splash Screen

Figure 33 - Splash Screen Page

Splash screen consists of LodeStar's logo and is displayed when the application is first opened. Then it automatically proceeds to the "Log In / Sign Up Page."

8.1.2. Log In / Sign Up Page

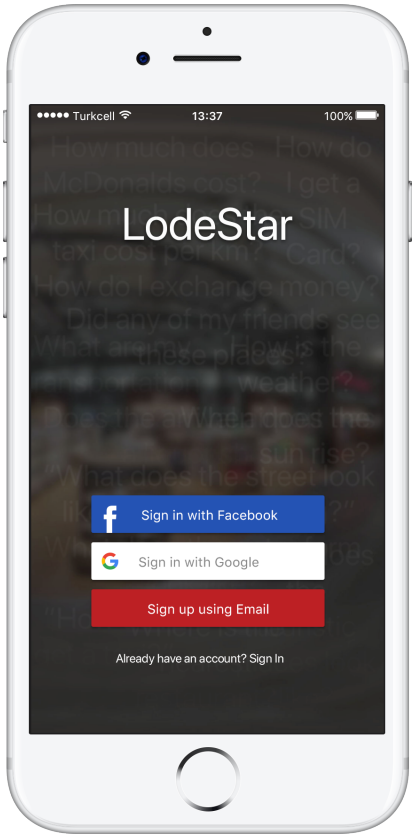


Figure 34 - Welcome Page

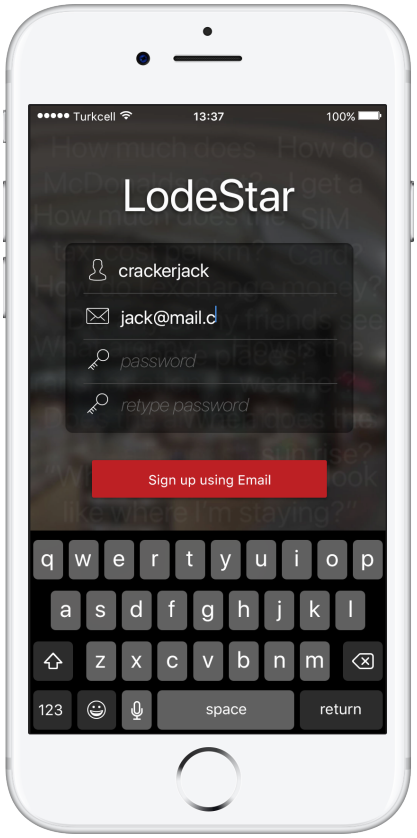


Figure 35 - Sign Up Page

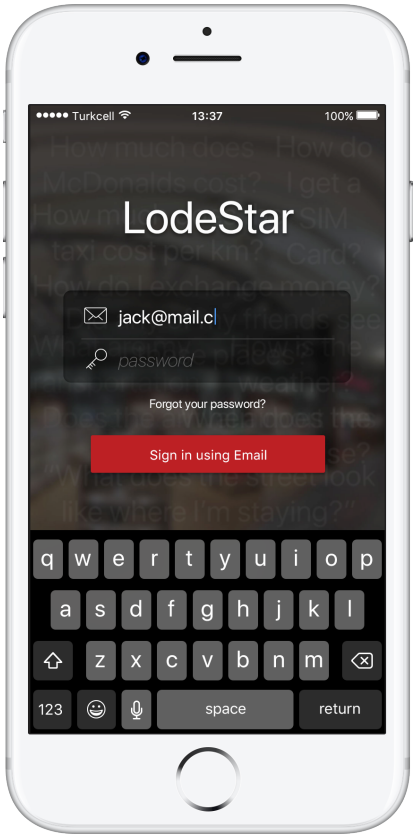


Figure 36 - Log In Page

The user is greeted with three different methods to sign in. They can sign in via Facebook by clicking the top (blue), via Google by clicking the second (white), and via their email address by clicking the bottom "Already have an account? Sign In" button. Alternatively, they can sign up by clicking the red "Sign up using Email" button. This takes them to the screen displayed in Figure-35 and asks the user for a username, valid email and password. After signing up, the user can log in with their email and password as shown in figure Figure-36.

8.1.3. Home Page

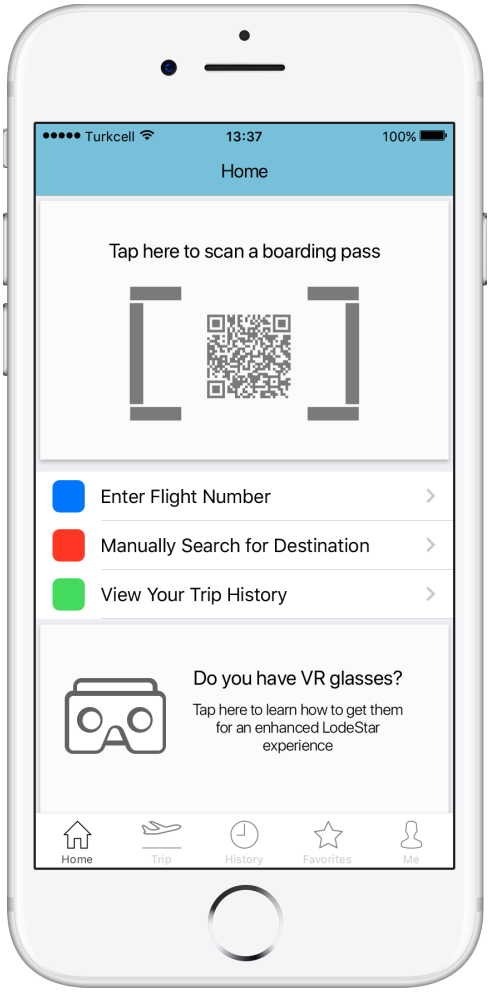


Figure 37 - Home Page

The user is navigated to this page once they pass the log-in page. The page has a big button at the top instructing the user to scan a flight card using their phone camera. Additionally, the user can enter a flight number manually if they do not have a flight card yet. (If the user did online check-in or the user wishes to view another flight). The user can also manually search for cities to learn more about them. Users can learn about different cities without traveling there. Towards the bottom of the page, a simple card will be shown to inform the user about the face that LodeStar experience is enhanced with VR glasses. LodeStar also educates the user about VR glasses and tells them where to get one.

8.1.4. Manually Enter Flight Number Page

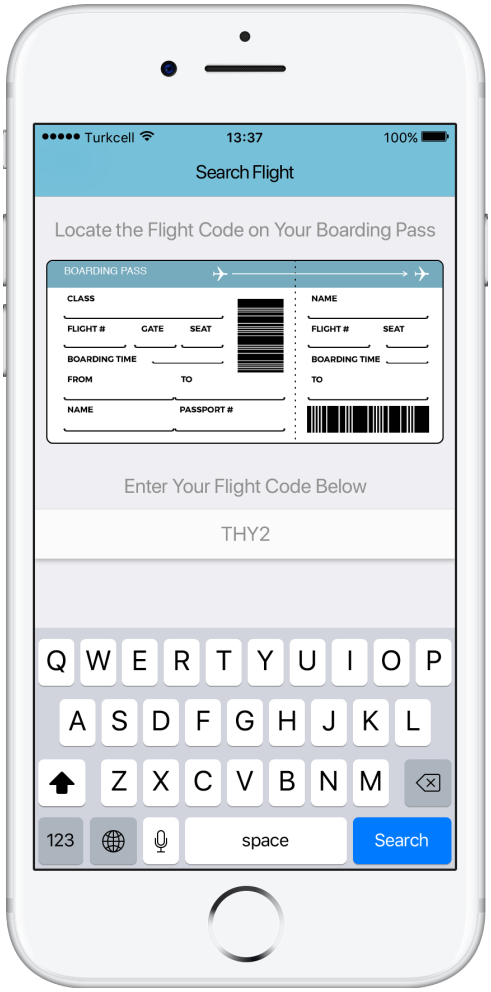


Figure 38 - Manually Enter Flight Number

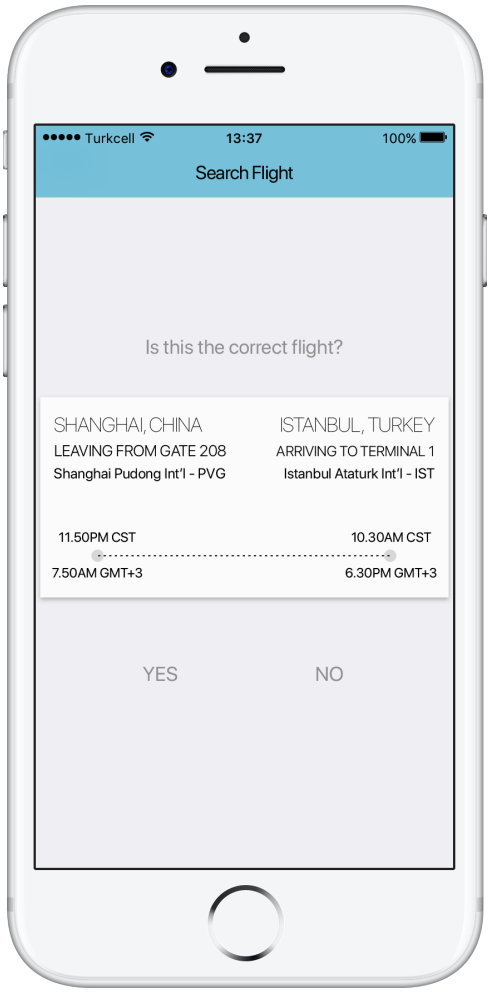


Figure 39 - Confirm Flight Number Page

In this screen, the user can enter their flight code to view information about their flight. Once the user types the code and clicks the "Search" button, if the code is valid, they will be presented with the flight information. The flight card shows the destination and arrival airports, the associated gates or terminals where applicable, expected departure and arrival hours. The user is asked whether the shown flight is the correct flight and if it is, LodeStar proceeds with the given information. If not, the user is taken back where they can reenter the code or choose a different method.

8.1.5. Trip Overview Page

Figure 40 - Trip Departure Page

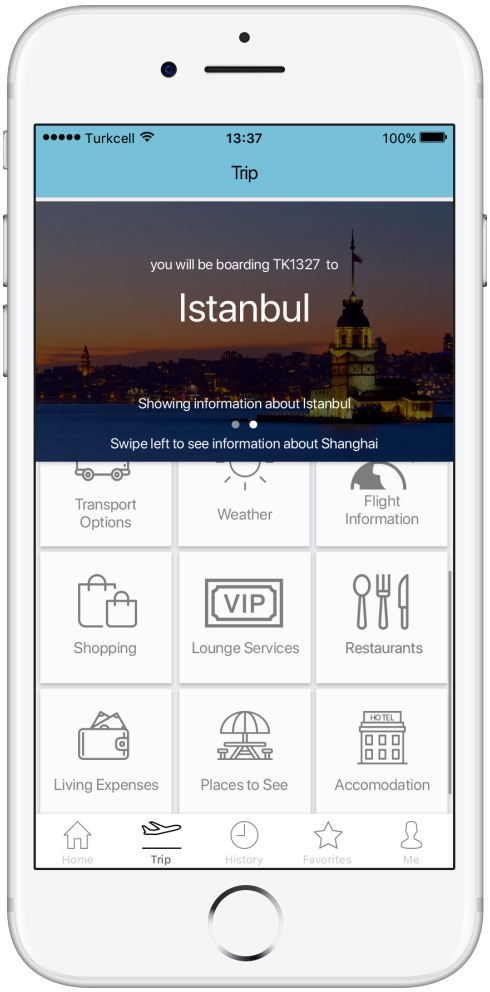


Figure 41 - Trip Arrival Page

This page is navigated to upon a successful flight card scan. Using the QR code located on the flight card, the departure and arrival cities of the corresponding flight can be learnt. This way, LodeStar fetches details about these cities and displays them to the user. Many options are provided to the user like transport options, weather, flight information, shopping, lounge services, restaurants, living expenses, places to see and accommodation.

The user will start seeing cards from the departure city. After swiping right from anywhere on the page, the user will start seeing cards about the destination city.

8.1.6. Flight Information Page

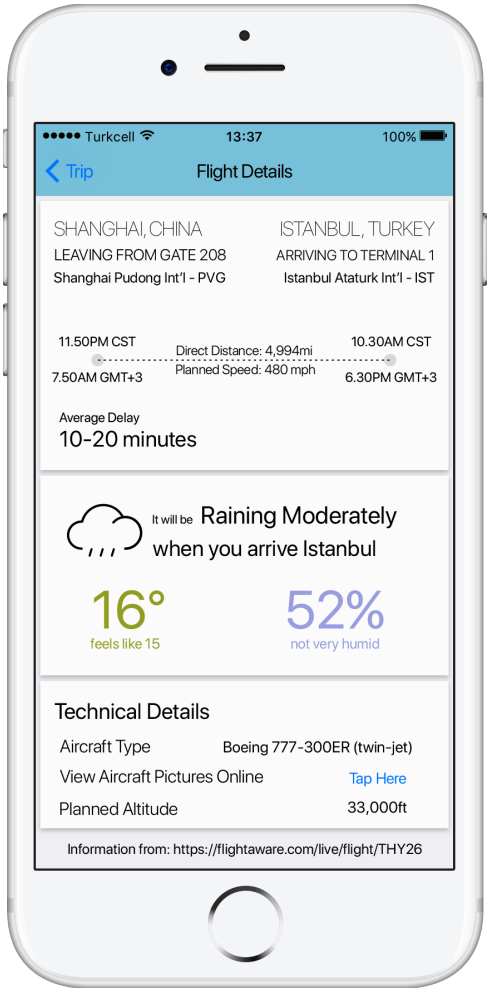


Figure 42 - Flight Information Page

This page is navigated to when the user taps on the "Flight Details" card from the "Trip" page. This screen gathers all available data that can be gathered by scanning a flight card. The flight card shows the destination and arrival airports, the associated gates or terminals where applicable, the duration of flight, expected departure and arrival hours and expected delay. Also, it displays a simple weather summary. The summary consists of the current weather condition like rain or snow, local temperature and humidity. Depending on the value of temperature and humidity, the text color changes. Tech savvy users which are interested in the technical details of a flight can learn the model of the plane and view the pictures of the aircraft.

8.1.7. Landmarks Page

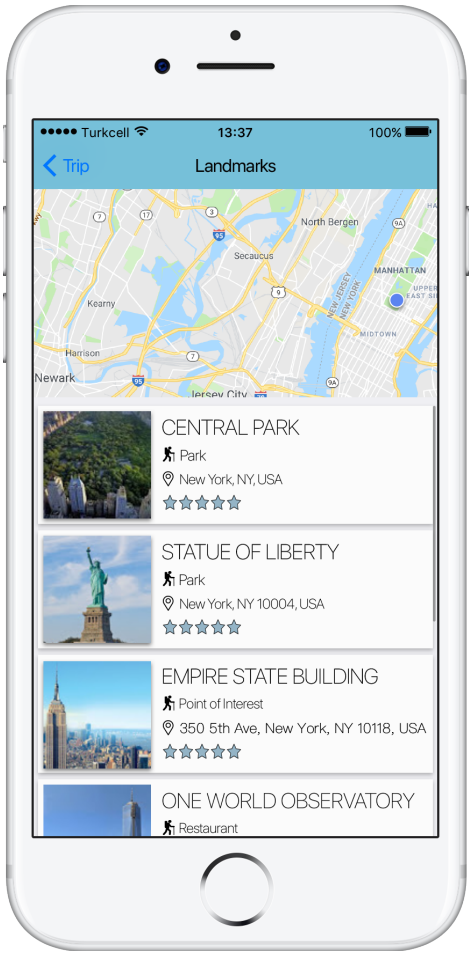


Figure 43 - Landmarks Page

This page is navigated to when the user taps on the "Landmarks" card from the "Trip" page. This screen gathers information about points of interest in the given location. It lists ten of them in a list and shows them on the map on the top. Once clicked, each landmarks image navigates to further information about that specific landmark. Each landmark's name, type, address and rating is listed next to its picture. The map on the top can be clicked to view their exact location on the smart phone's native map application and get guidance. The top landmarks are chosen from FourSquare using their APIs which means they were rated by a lot of people before claiming their places. Therefore, the landmarks page covers most points of interest in an area.

8.1.8. Near Me Page

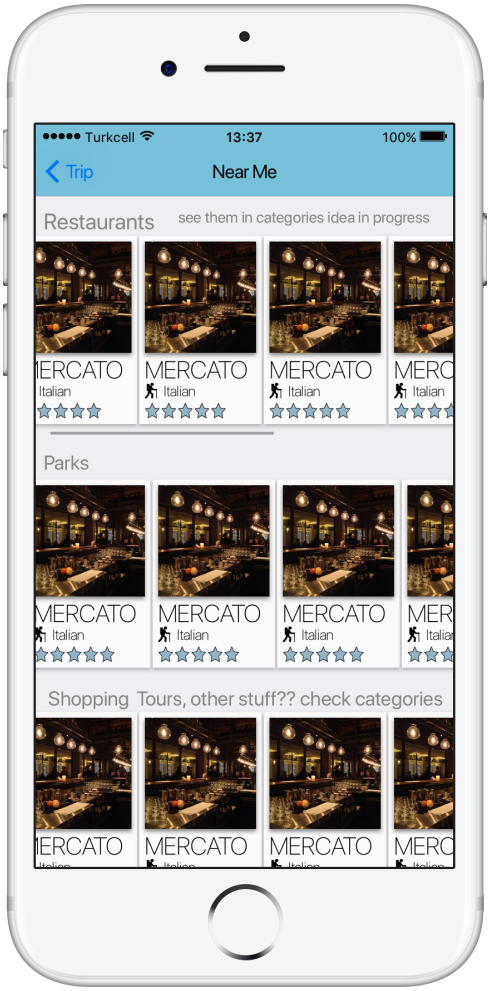


Figure 44 - Near Me Page

This page is navigated to when the user taps on the "Near Me" card from the "Trip" page. This page offers information for those who wish to view the landmarks in their area without having to search for it. It sorts items into different categories like restaurants and cafes. Their names, ratings and types are listed. The user can click on any item to view more information about it.

8.1.9. Venue Details Page

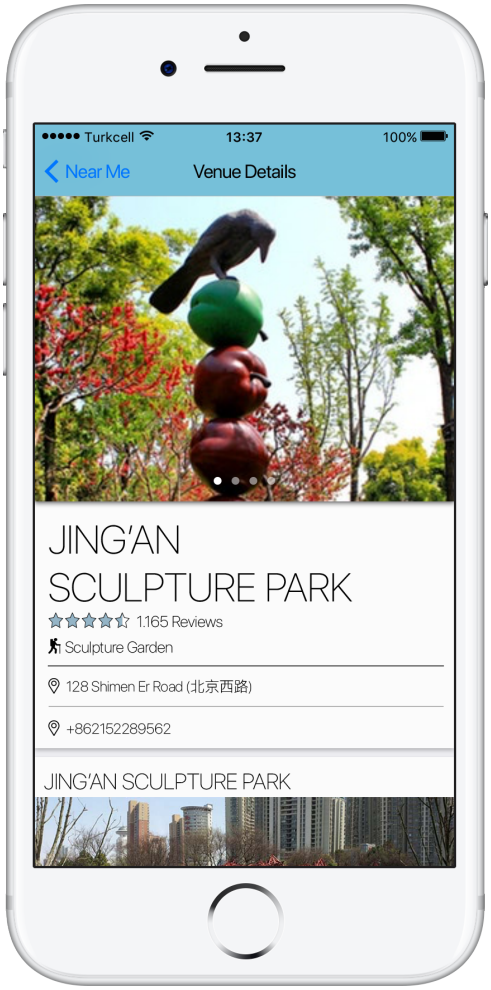


Figure 45 - Venue Details Page

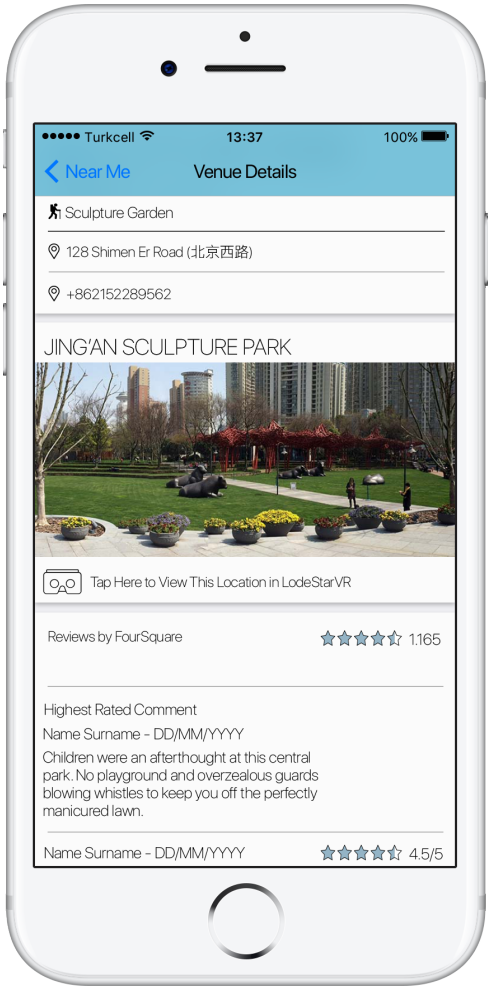


Figure 46 - Venue Details Page (Scrolled Down)

This page is navigated to when the user taps on a landmark from the "Landmarks" or "Near Me" pages. It displays the name of the venue, gives information like phone number, address, ratings and displays pictures of the place. Top rated comments from FourSquare are displayed as well. When the user clicks on the phone number, a call is automatically generated to the venue. If VR photos of the venue are available, the user can click on the cardboard icon below the image (as seen in figure 46) to embrace the experience in LodeStar's defining VR format. While the full experience requires a cardboard or something similar, the phone screen itself provides information as well.

8.1.10.Currency Rates Page

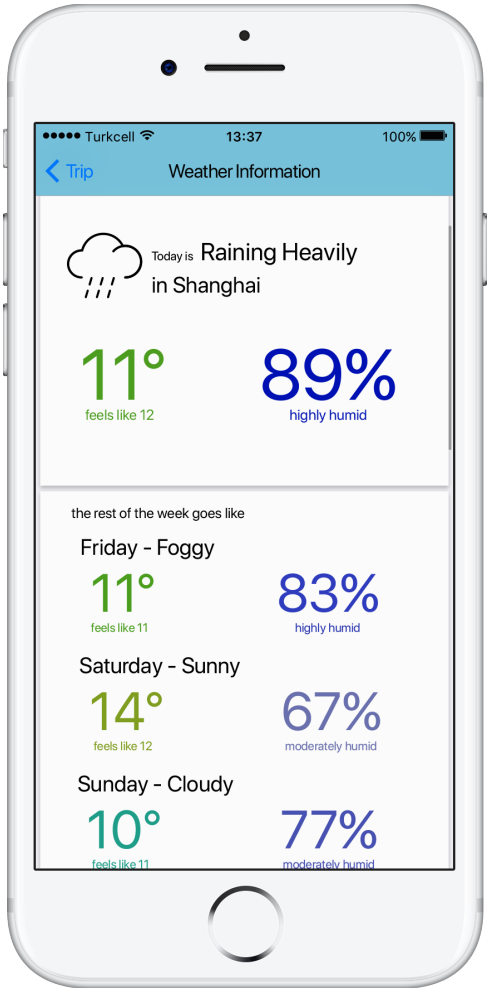


Figure 47 - Weather Information Page

If the user taps on the "Weather" card from the Trip page, the user will see the weather information about the corresponding city. Weather information is shown in a minimalistic and simple way, without any clutter. The user can see whether they need a coat or an umbrella in the city they are traveling. As the temperature and humidity value changes, the color of the text changes as well to reflect this change. For example, the more humid the weather is, the darker the blue of the text. Via scrolling, the user can see the next five days' forecast with the current day's being highlighted.

8.1.11.Currency Rates Page

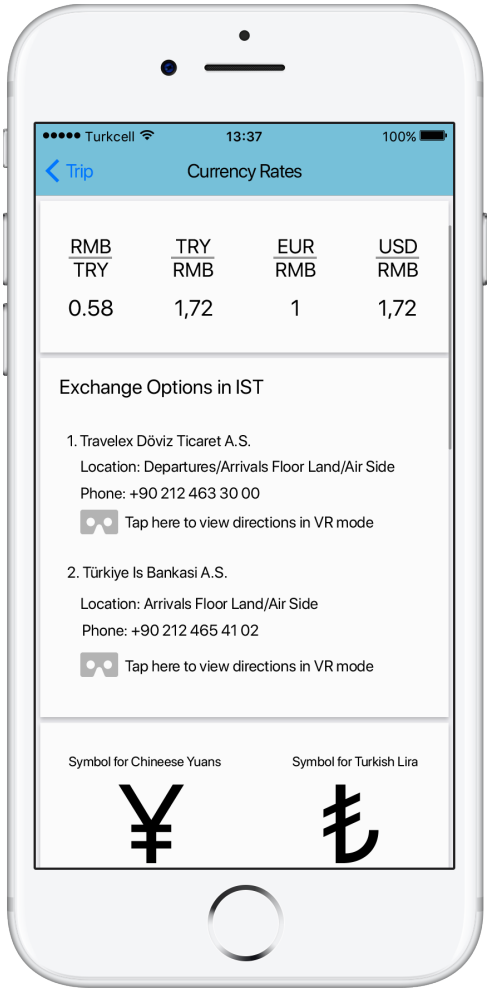
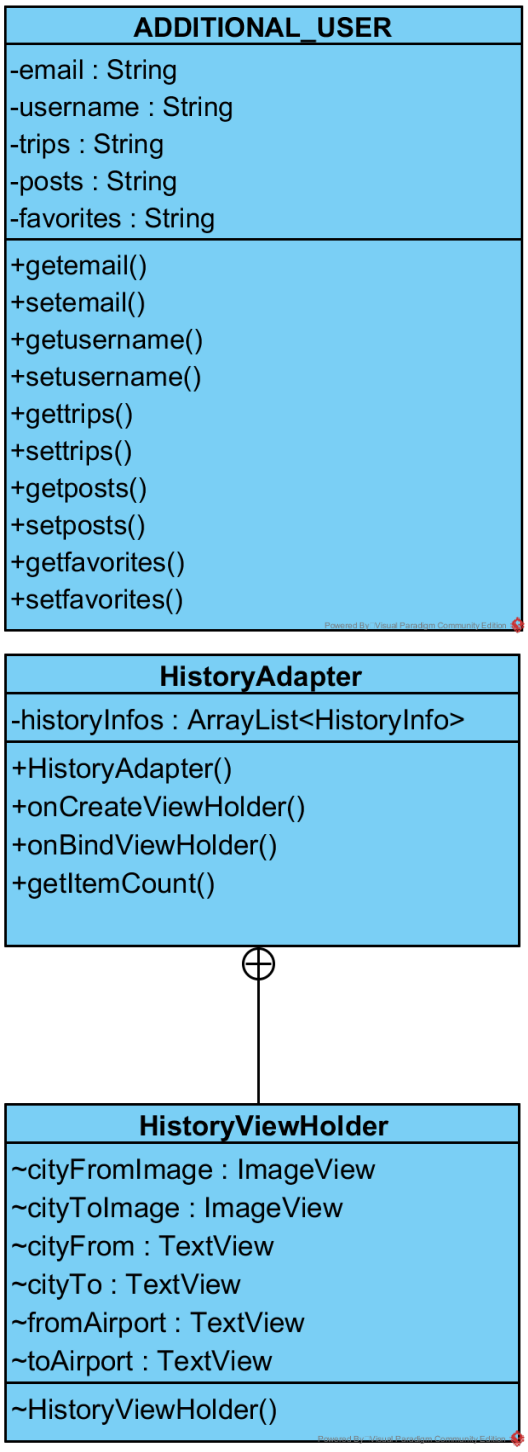
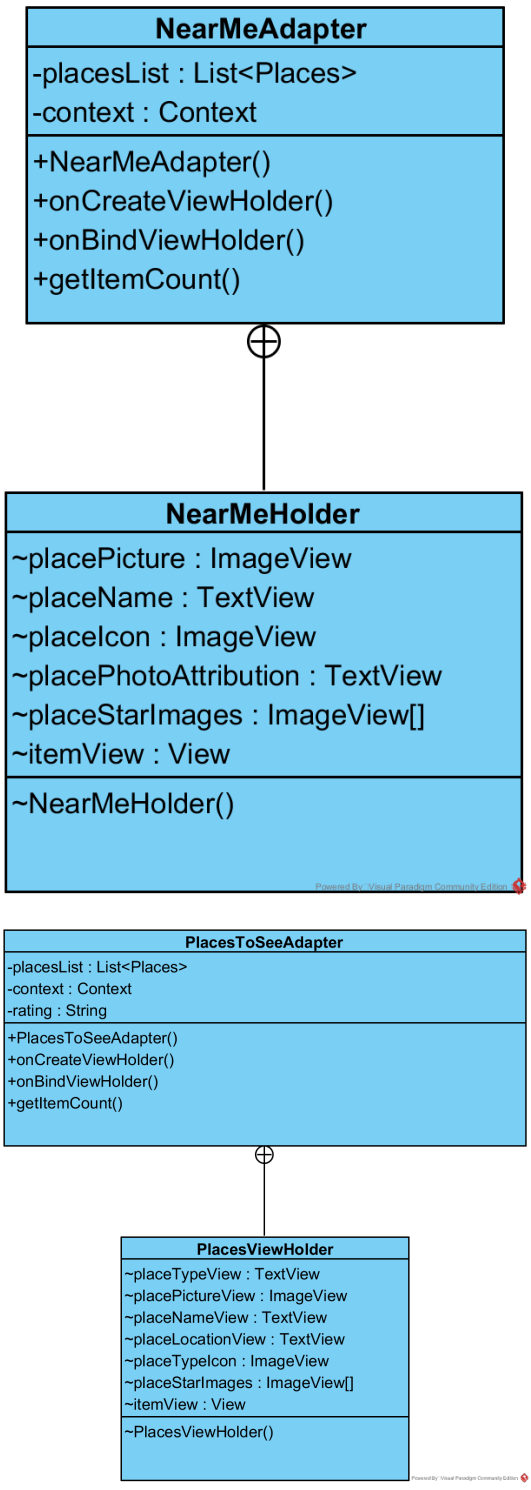


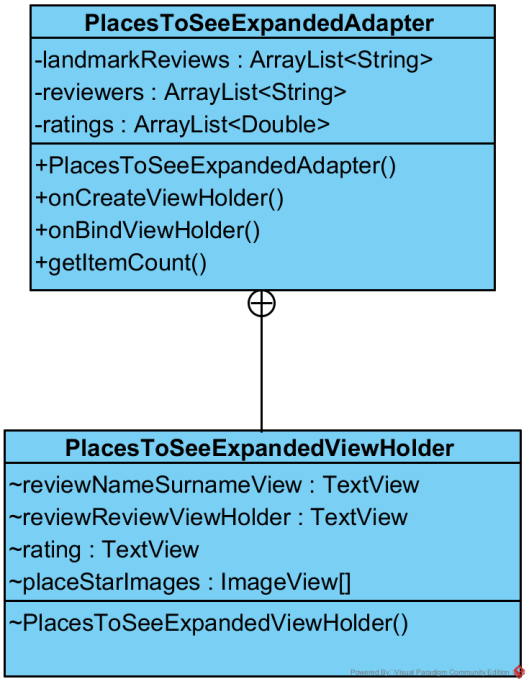
Figure 48 - Currency Rates Page

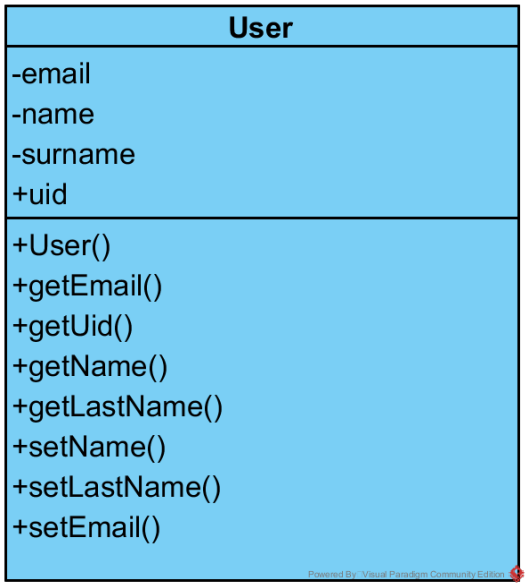
This page is navigated to when the user taps on the "Currency Rates" card from the "Trip" page. This screen shows the user cross currency rates between the cities they are traveling. Also, it displays the currency exchange rates for internationally known currencies such as USD and EUR[52]. Towards the bottom of the page, symbols for the currencies will be shown to inform the user about their symbols.

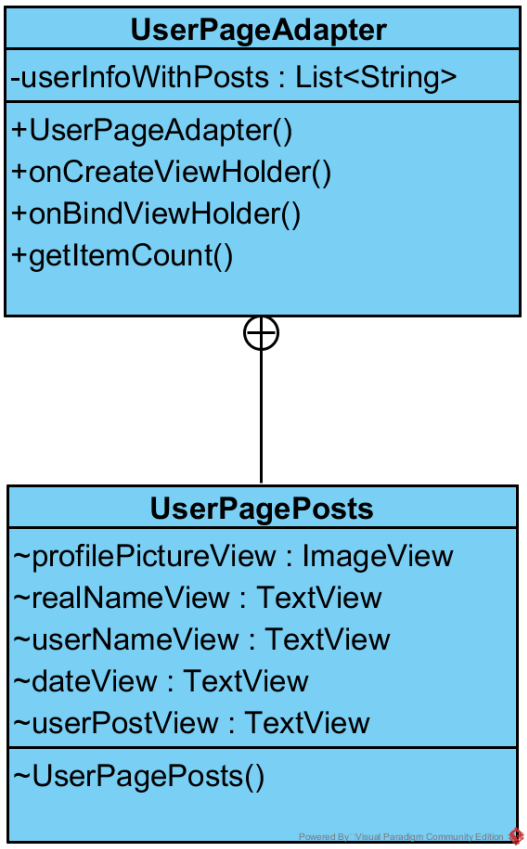
9. Appendix B - Class Diagrams

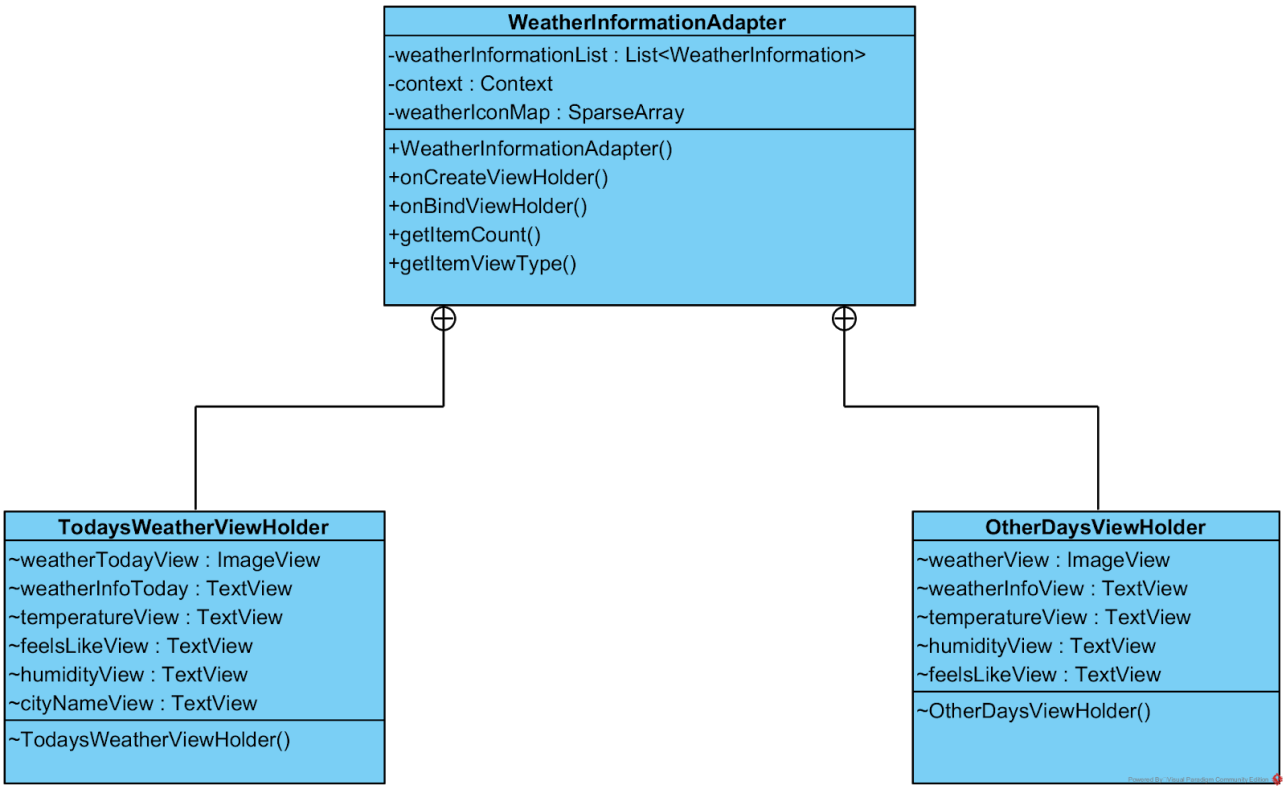



NearMeAdapter
-placesList : List<Places>
-context : Context
+NearMeAdapter()
+onCreateViewHolder()
+onBindViewHolder()
+getItemCount()
NearMeHolder
~placePicture : ImageView
~placeName : TextView
~placeIcon : ImageView
~placePhotoAttribution : TextView
~placeStarImages : ImageView[]
~itemView : View
~NearMeHolder()
PlacesToSeeAdapter
-placesList : List<Places>
-context : Context
-rating : String
+PlacesToSeeAdapter()
+onCreateViewHolder()
+onBindViewHolder()
+getItemCount()
PlacesViewHolder
~placeTypeView : TextView
~placePictureView : ImageView
~placeNameView : TextView
~placeLocationView : TextView
~placeTypeIcon : ImageView
~placeStarImages : ImageView[]
~itemView : View
~PlacesViewHolder()

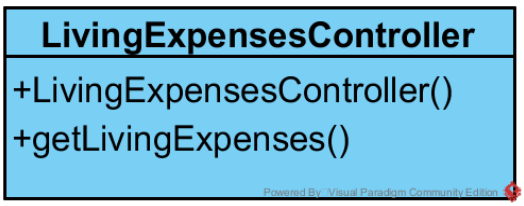

PlacesToSeeExpandedAdapter
-landmarkReviews : ArrayList<String>
-reviewers : ArrayList<String>
-ratings : ArrayList<Double>
+PlacesToSeeExpandedAdapter()
+onCreateViewHolder()
+onBindViewHolder()
+getItemCount()
PlacesToSeeExpandedViewHolder
~reviewNameSurnameView : TextView
~reviewReviewViewHolder : TextView
~rating : TextView
~placeStarImages : ImageView[]
~PlacesToSeeExpandedViewHolder()

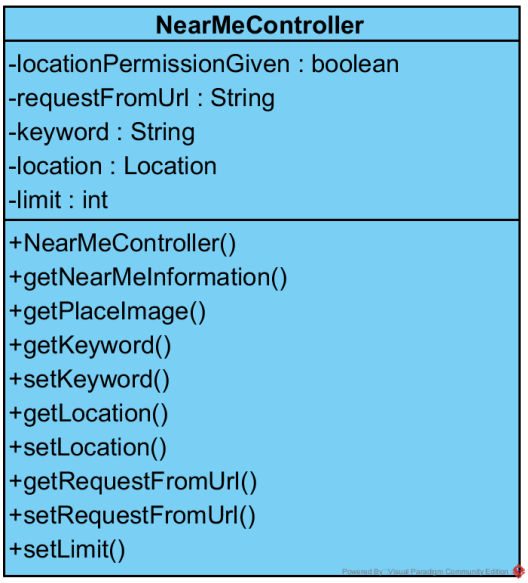

User
-email
-name
-surname
+uid
+User()
+getEmail()
+getUid()
+getName()
+getLastName()
+setName()
+setLastName()
+setEmail()

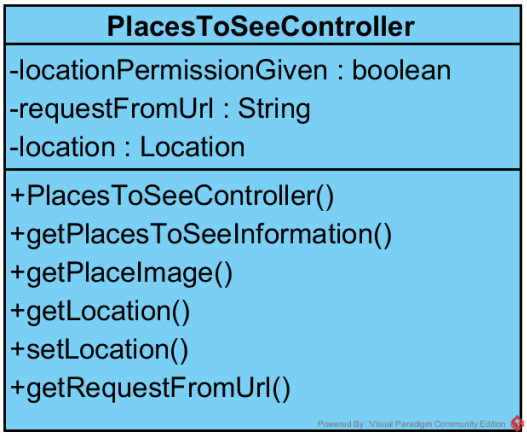

UserPageAdapter
-userInfoWithPosts : List<String>
+UserPageAdapter()
+onCreateViewHolder()
+onBindViewHolder()
+getItemCount()
UserPagePosts
~profilePictureView : ImageView
~realNameView : TextView
~userNameView : TextView
~dateView : TextView
~userPostView : TextView
~UserPagePosts()

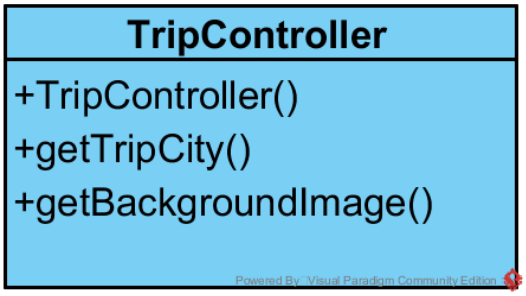

WeatherInformationAdapter
-weatherInformationList : List<WeatherInformation>
-context : Context
-weatherIconMap : SparseArray
+WeatherInformationAdapter()
+onCreateViewHolder()
+onBindViewHolder()
+getItemCount()
+getItemViewType()
TodaysWeatherViewHolder
~weatherTodayView : ImageView
~weatherInfoToday : TextView
~temperatureView : TextView
~feelsLikeView : TextView
~humidityView : TextView
~cityNameView : TextView
~TodaysWeatherViewHolder()
OtherDaysViewHolder
~weatherView : ImageView
~weatherInfoView : TextView
~temperatureView : TextView
~humidityView : TextView
~feelsLikeView : TextView
~OtherDaysViewHolder()

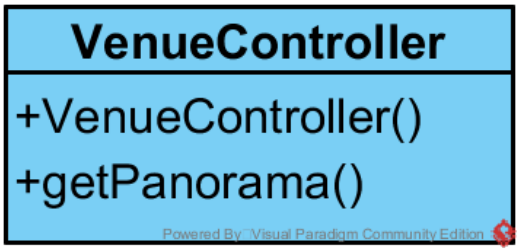
9.1.1. Controllers

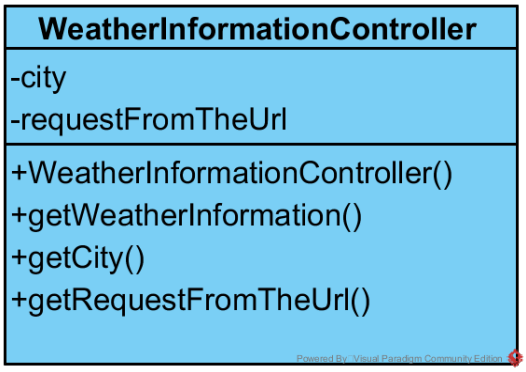




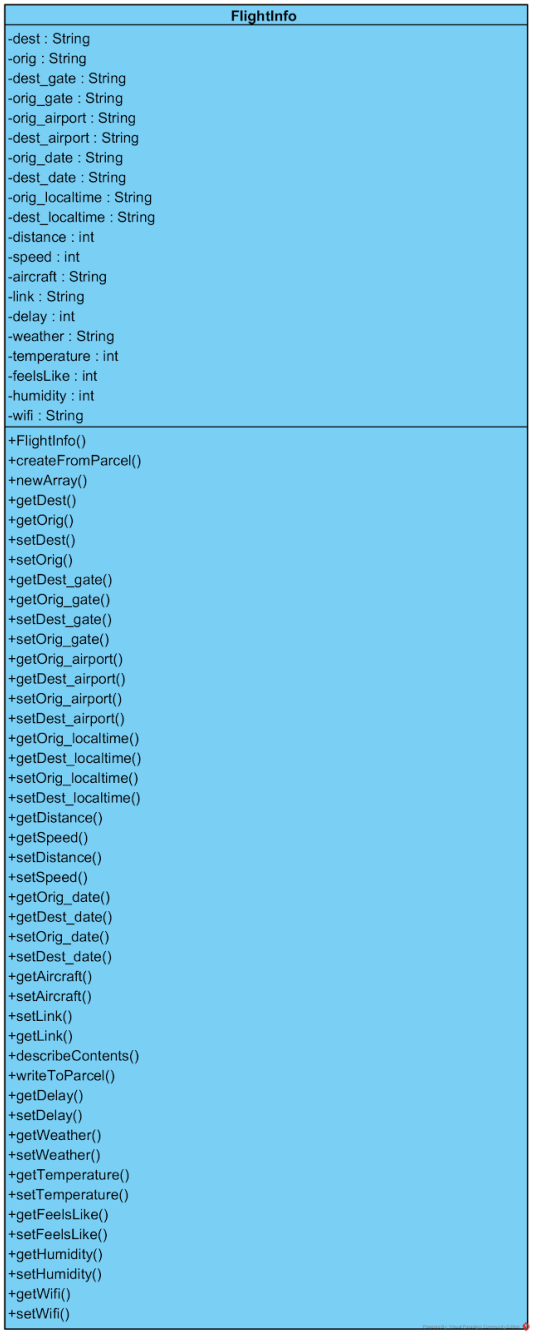

PlacesToSeeController
-locationPermissionGiven : boolean
-requestFromUrl : String
-location : Location
+PlacesToSeeController()
+getPlacesToSeeInformation()
+getPlaceImage()
+getLocation()
+setLocation()
+getRequestFromUrl()
Powered By Visual Paradigm Community Edition

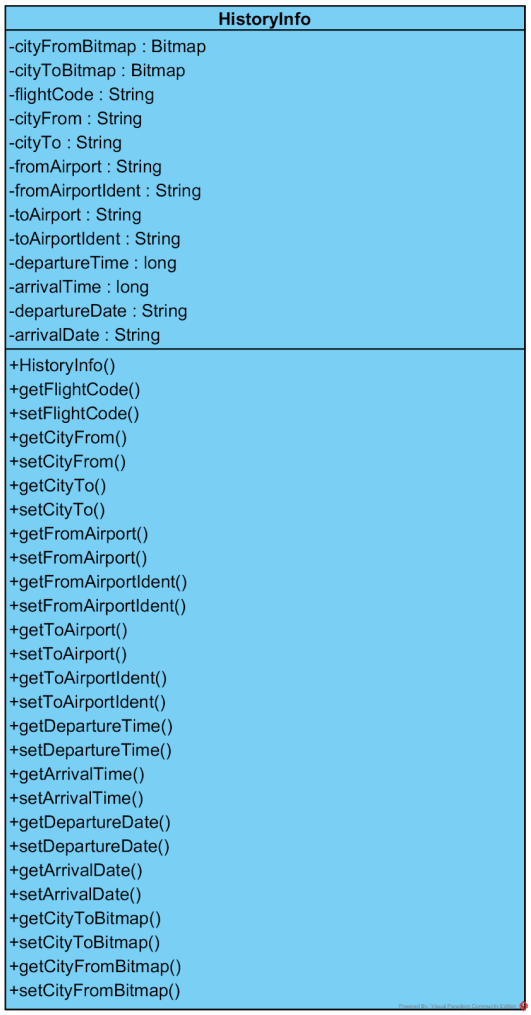

TripController
+TripController()
+getTripCity()
+getBackgroundImage()
Powered By Visual Paradigm Community Edition

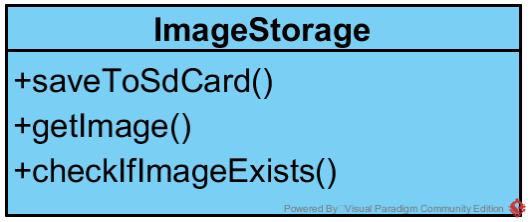

VenueController
+VenueController()
+getPanorama()
Powered By Visual Paradigm Community Edition

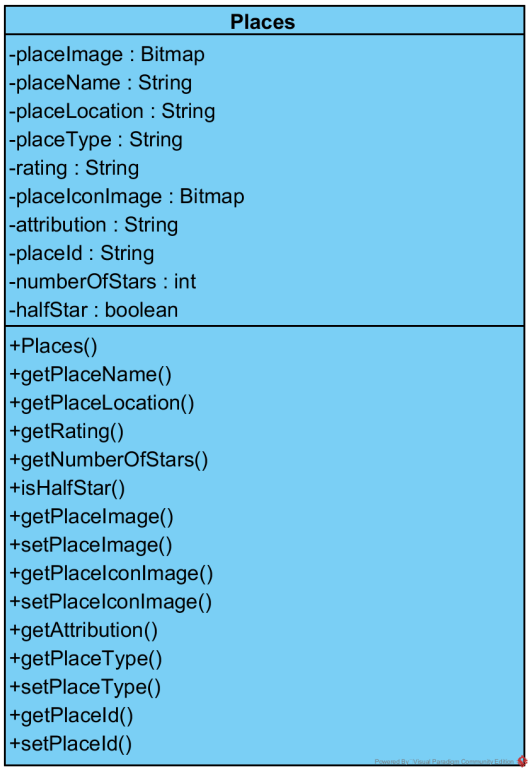

WeatherInformationController
-city
-requestFromTheUrl
+WeatherInformationController()
+getWeatherInformation()
+getCity()
+getRequestFromTheUrl()
Powered By Visual Paradigm Community Edition

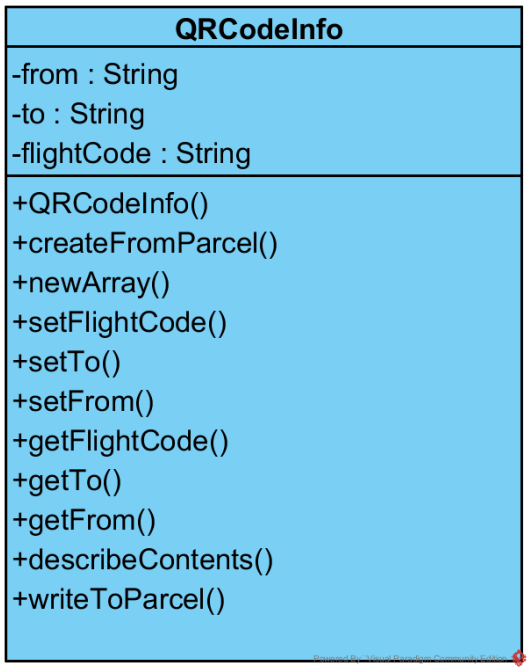
9.1.2. Models

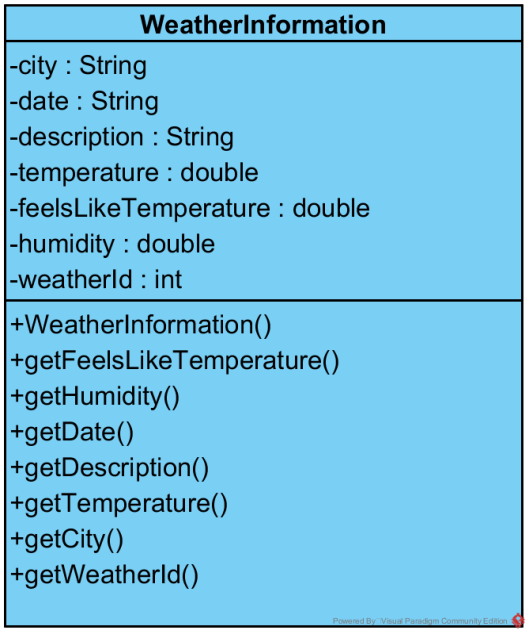

FlightInfo
-dest : String
-orig : String
-dest_gate : String
-orig_gate : String
-orig_airport : String
-dest_airport : String
-orig_date : String
-dest_date : String
-orig_localtime : String
-dest_localtime : String
-distance : int
-speed : int
-aircraft : String
-link : String
-delay : int
-weather : String
-temperature : int
-feelsLike : int
-humidity : int
-wifi : String
+FlightInfo()
+createFromParcel()
+newArray()
+getDest()
+getOrig()
+setDest()
+setOrig()
+getDest_gate()
+getOrig_gate()
+setDest_gate()
+setOrig_gate()
+getOrig_airport()
+getDest_airport()
+setOrig_airport()
+setDest_airport()
+getOrig_localtime()
+getDest_localtime()
+setOrig_localtime()
+setDest_localtime()
+getDistance()
+getSpeed()
+setDistance()
+setSpeed()
+getOrig_date()
+getDest_date()
+setOrig_date()
+setDest_date()
+getAircraft()
+setAircraft()
+setLink()
+getLink()
+describeContents()
+writeToParcel()
+getDelay()
+setDelay()
+getWeather()
+setWeather()
+getTemperature()
+setTemperature()
+getFeelsLike()
+setFeelsLike()
+getHumidity()
+setHumidity()
+getWifi()
+setWifi()


HistoryInfo
-cityFromBitmap : Bitmap
-cityToBitmap : Bitmap
-flightCode : String
-cityFrom : String
-cityTo : String
-fromAirport : String
-fromAirportIdent : String
-toAirport : String
-toAirportIdent : String
-departureTime : long
-arrivalTime : long
-departureDate : String
-arrivalDate : String
+HistoryInfo()
+getFlightCode()
+setFlightCode()
+getCityFrom()
+setCityFrom()
+getCityTo()
+setCityTo()
+getFromAirport()
+setFromAirport()
+getFromAirportIdent()
+setFromAirportIdent()
+getToAirport()
+setToAirport()
+getToAirportIdent()
+setToAirportIdent()
+getDepartureTime()
+setDepartureTime()
+getArrivalTime()
+setArrivalTime()
+getDepartureDate()
+setDepartureDate()
+getArrivalDate()
+setArrivalDate()
+getCityToBitmap()
+setCityToBitmap()
+getCityFromBitmap()
+setCityFromBitmap()

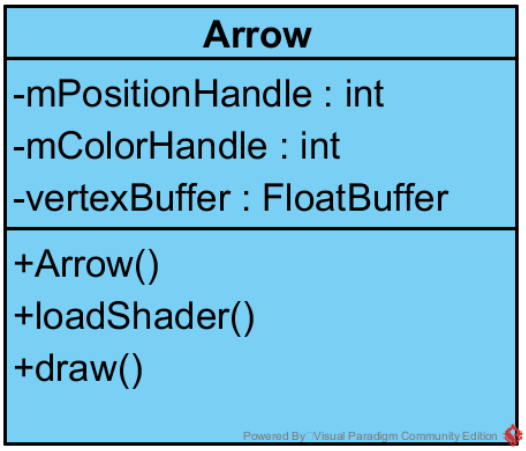

ImageStorage
+saveToSdCard()
+getImage()
+checkIfImageExists()
Powered By Visual Paradigm Community Edition

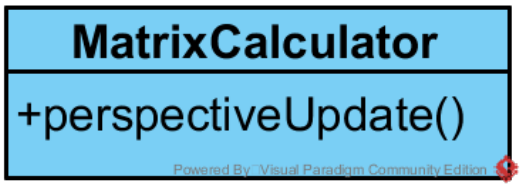

Places
-placeImage : Bitmap
-placeName : String
-placeLocation : String
-placeType : String
-rating : String
-placeIconImage : Bitmap
-attribution : String
-placeId : String
-numberOfStars : int
-halfStar : boolean
+Places()
+getPlaceName()
+getPlaceLocation()
+getRating()
+getNumberOfStars()
+isHalfStar()
+getPlaceImage()
+setPlaceImage()
+getPlaceIconImage()
+setPlaceIconImage()
+getAttribution()
+getPlaceType()
+setPlaceType()
+getPlaceId()
+setPlaceId()

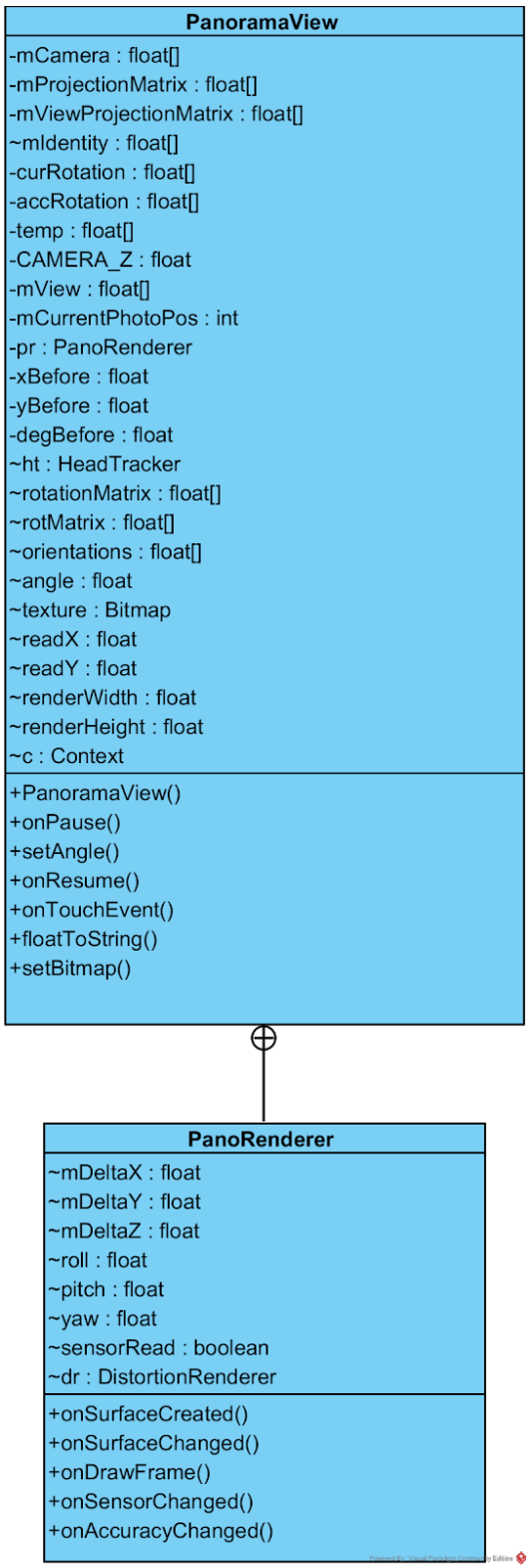

QRCodeInfo
-from : String
-to : String
-flightCode : String
+QRCodeInfo()
+createFromParcel()
+newArray()
+setFlightCode()
+setTo()
+setFrom()
+getFlightCode()
+getTo()
+getFrom()
+describeContents()
+writeToParcel()

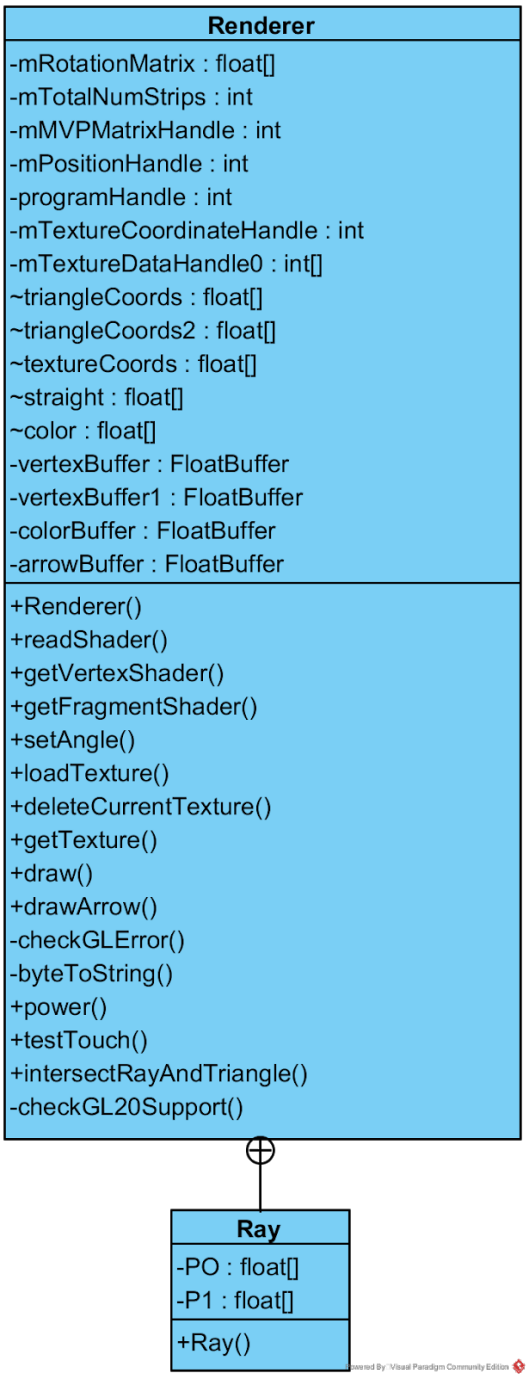


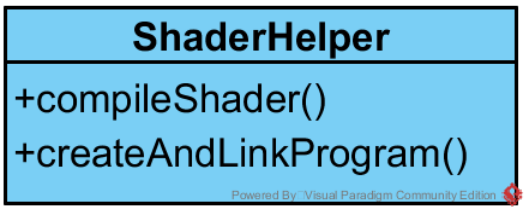
9.1.3. Virtual Reality Related Classes

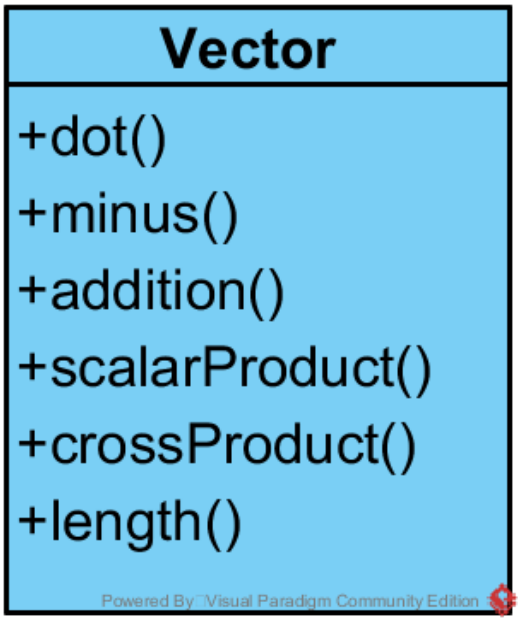




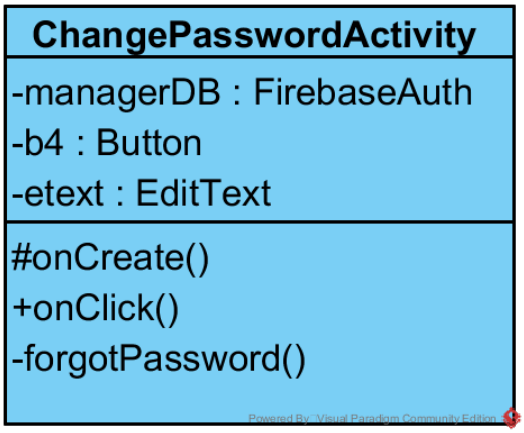

PanoramaView
-mCamera : float[]
-mProjectionMatrix : float[]
-mViewProjectionMatrix : float[]
~mIdentity : float[]
-curRotation : float[]
-accRotation : float[]
-temp : float[]
-CAMERA_Z : float
-mView : float[]
-mCurrentPhotoPos : int
-pr : PanoRenderer
-xBefore : float
-yBefore : float
-degBefore : float
~ht : HeadTracker
~rotationMatrix : float[]
~rotMatrix : float[]
~orientations : float[]
~angle : float
~texture : Bitmap
~readX : float
~readY : float
~renderWidth : float
~renderHeight : float
~c : Context
+PanoramaView()
+onPause()
+setAngle()
+onResume()
+onTouchEvent()
+floatToString()
+setBitmap()
PanoRenderer
~mDeltaX : float
~mDeltaY : float
~mDeltaZ : float
~roll : float
~pitch : float
~yaw : float
~sensorRead : boolean
~dr : DistortionRenderer
+onSurfaceCreated()
+onSurfaceChanged()
+onDrawFrame()
+onSensorChanged()
+onAccuracyChanged()

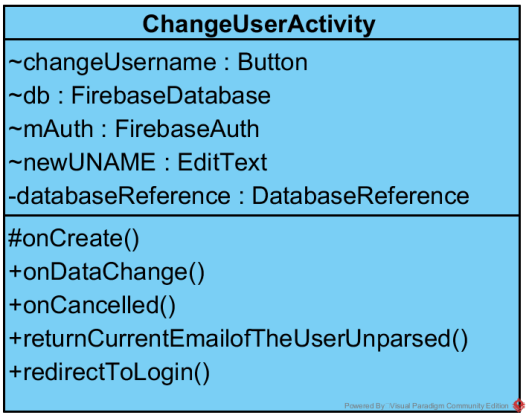

Renderer
-mRotationMatrix : float[]
-mTotalNumStrips : int
-mMVPMatrixHandle : int
-mPositionHandle : int
-programHandle : int
-mTextureCoordinateHandle : int
-mTextureDataHandle0 : int[]
~triangleCoords : float[]
~triangleCoords2 : float[]
~textureCoords : float[]
~straight : float[]
~color : float[]
-vertexBuffer : FloatBuffer
-vertexBuffer1 : FloatBuffer
-colorBuffer : FloatBuffer
-arrowBuffer : FloatBuffer
+Renderer()
+readShader()
+getVertexShader()
+getFragmentShader()
+setAngle()
+loadTexture()
+deleteCurrentTexture()
+getTexture()
+draw()
+drawArrow()
-checkGLError()
-byteToString()
+power()
+testTouch()
+intersectRayAndTriangle()
-checkGL20Support()
Ray
-PO : float[]
-P1 : float[]
+Ray()
Powered By Visual Paradigm Community Edition

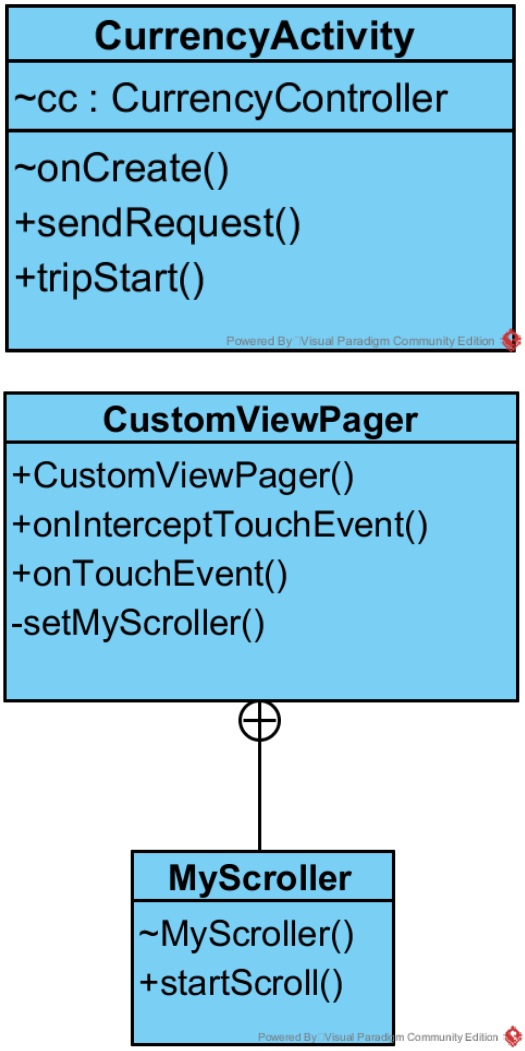

ShaderHelper
+compileShader()
+createAndLinkProgram()
Powered By Visual Paradigm Community Edition

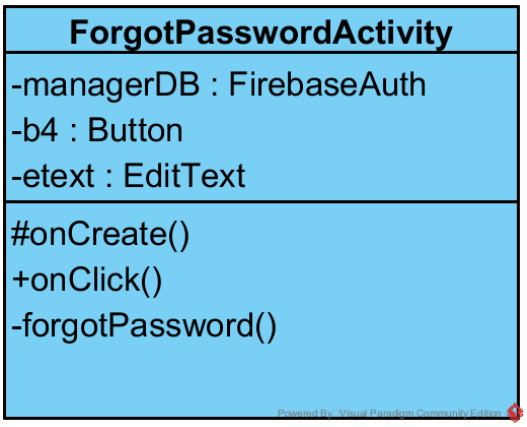


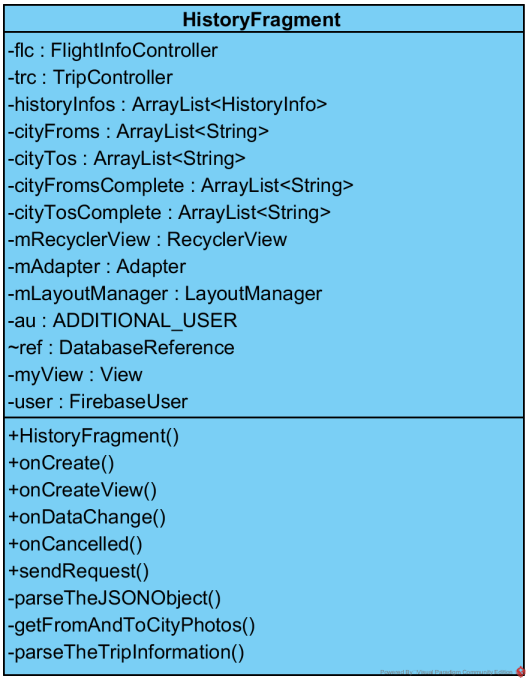
9.1.4. Activities

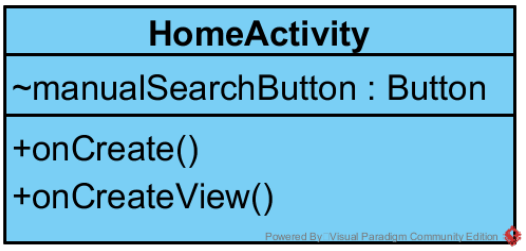


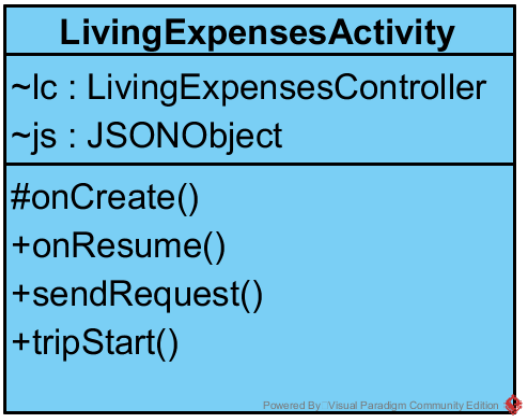


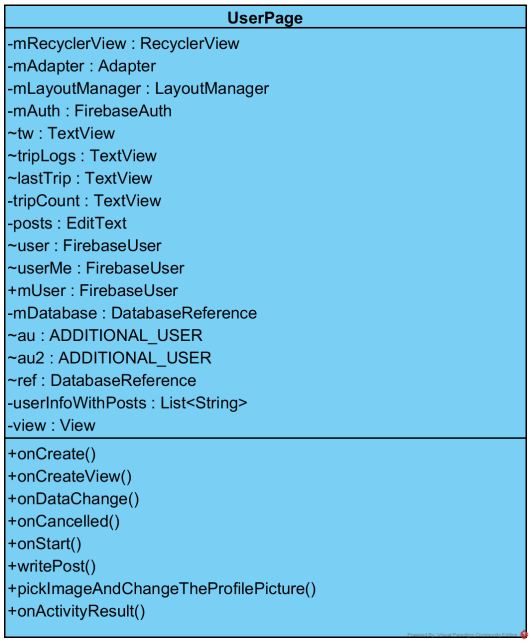

CurrencyActivity
~cc : CurrencyController
~onCreate()
+sendRequest()
+tripStart()
Powered By Visual Paradigm Community Edition
CustomViewPager
+CustomViewPager()
+onInterceptTouchEvent()
+onTouchEvent()
-setMyScroller()
MyScroller
~MyScroller()
+startScroll()
Powered By Visual Paradigm Community Edition
FlightInfoActivity
~flightInfo : FlightInfo
~onCreate()
+tripStart()
+openLink()
Powered By Visual Paradigm Community Edition

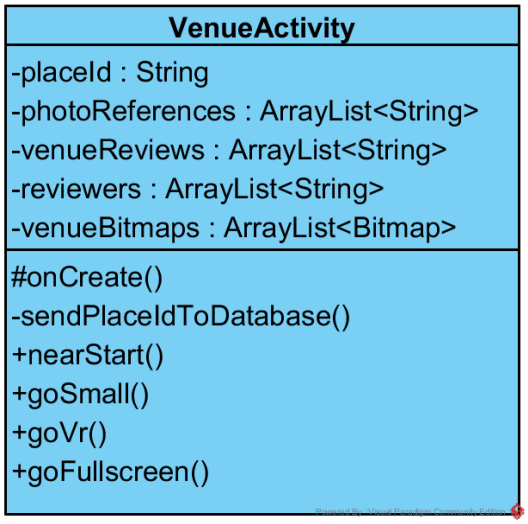

ForgotPasswordActivity
-managerDB : FirebaseAuth
-b4 : Button
-etext : EditText
#onCreate()
+onClick()
-forgotPassword()
Powered By Visual Paradigm Community Edition

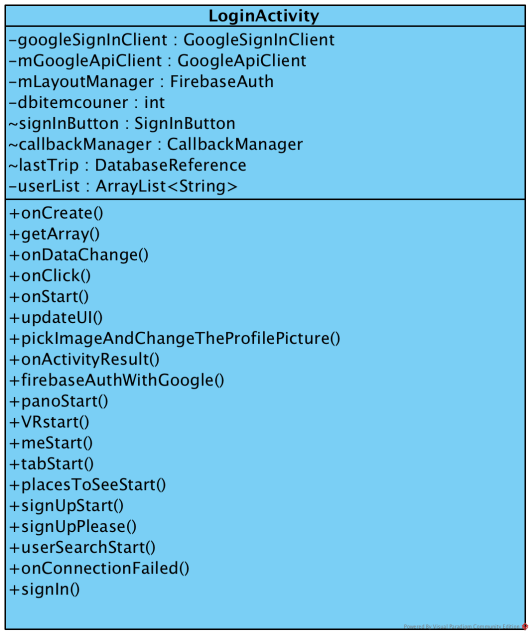

HistoryFragment
-flc : FlightInfoController
-trc : TripController
-historyInfos : ArrayList<HistoryInfo>
-cityFroms : ArrayList<String>
-cityTos : ArrayList<String>
-cityFromsComplete : ArrayList<String>
-cityTosComplete : ArrayList<String>
-mRecyclerView : RecyclerView
-mAdapter : Adapter
-mLayoutManager : LayoutManager
-au : ADDITIONAL_USER
~ref : DatabaseReference
-myView : View
-user : FirebaseUser
+HistoryFragment()
+onCreate()
+onCreateView()
+onDataChange()
+onCancelled()
+sendRequest()
-parseTheJSONObject()
-getFromAndToCityPhotos()
-parseTheTripInformation()

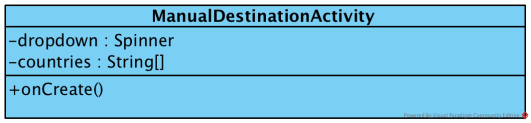

HomeActivity
~manualSearchButton : Button
+onCreate()
+onCreateView()
Powered By Visual Paradigm Community Edition

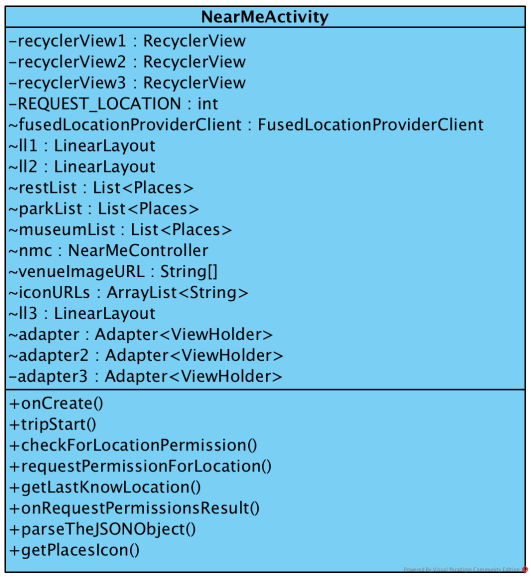

LivingExpensesActivity
~lc : LivingExpensesController
~js : JSONObject
#onCreate()
+onResume()
+sendRequest()
+tripStart()
Powered By Visual Paradigm Community Edition

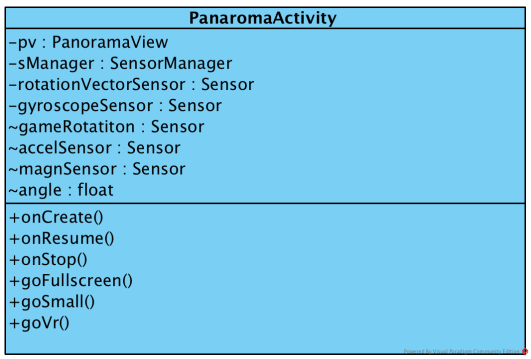

UserPage
-mRecyclerView : RecyclerView
-mAdapter : Adapter
-mLayoutManager : LayoutManager
-mAuth : FirebaseAuth
~tw : TextView
~tripLogs : TextView
~lastTrip : TextView
-tripCount : TextView
-posts : EditText
~user : FirebaseUser
~userMe : FirebaseUser
+mUser : FirebaseUser
-mDatabase : DatabaseReference
~au : ADDITIONAL_USER
~au2 : ADDITIONAL_USER
~ref : DatabaseReference
-userInfoWithPosts : List<String>
-view : View
+onCreate()
+onCreateView()
+onDataChange()
+onCancelled()
+onStart()
+writePost()
+pickImageAndChangeTheProfilePicture()
+onActivityResult()

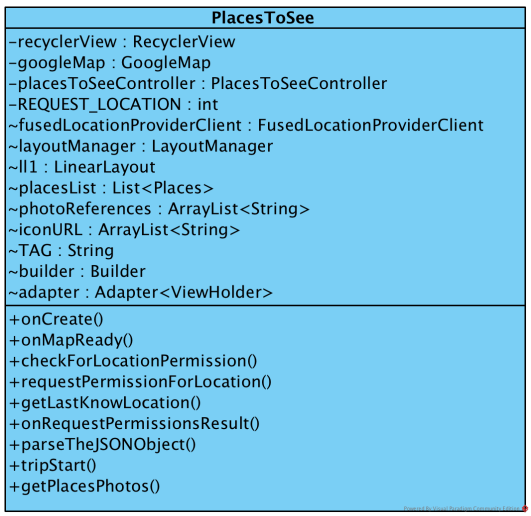

VenueActivity
-placeId : String
-photoReferences : ArrayList<String>
-venueReviews : ArrayList<String>
-reviewers : ArrayList<String>
-venueBitmaps : ArrayList<Bitmap>
#onCreate()
-sendPlaceIdToDatabase()
+nearStart()
+goSmall()
+goVr()
+goFullscreen()

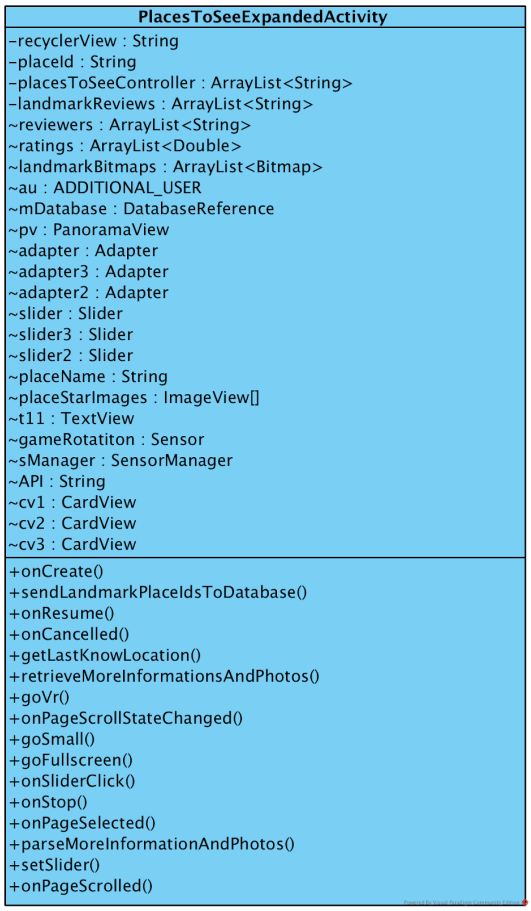

LoginActivity
-googleSignInClient : GoogleSignInClient
-mGoogleApiClient : GoogleApiClient
-mLayoutManager : FirebaseAuth
-dbitemcouner : int
~signInButton : SignInButton
~callbackManager : CallbackManager
~lastTrip : DatabaseReference
-userList : ArrayList<String>
+onCreate()
+getArray()
+onDataChange()
+onClick()
+onStart()
+updateUI()
+pickImageAndChangeTheProfilePicture()
+onActivityResult()
+firebaseAuthWithGoogle()
+panoStart()
+VRstart()
+meStart()
+tabStart()
+placesToSeeStart()
+signUpStart()
+signUpPlease()
+userSearchStart()
+onConnectionFailed()
+signIn()

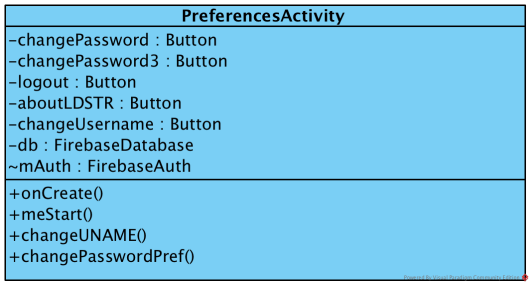

ManualDestinationActivity
-dropdown : Spinner
-countries : String[]
+onCreate()

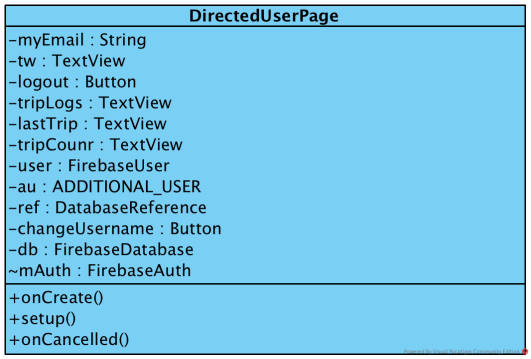

NearMeActivity
-recyclerView1 : RecyclerView
-recyclerView2 : RecyclerView
-recyclerView3 : RecyclerView
-REQUEST_LOCATION : int
~fusedLocationProviderClient : FusedLocationProviderClient
~ll1 : LinearLayout
~ll2 : LinearLayout
~restList : List<Places>
~parkList : List<Places>
~museumList : List<Places>
~nmc : NearMeController
~venueImageURL : String[]
~iconURLs : ArrayList<String>
~ll3 : LinearLayout
~adapter : Adapter<ViewHolder>
~adapter2 : Adapter<ViewHolder>
-adapter3 : Adapter<ViewHolder>
+onCreate()
+tripStart()
+checkForLocationPermission()
+requestPermissionForLocation()
+getLastKnowLocation()
+onRequestPermissionsResult()
+parseTheJSONObject()
+getPlacesIcon()

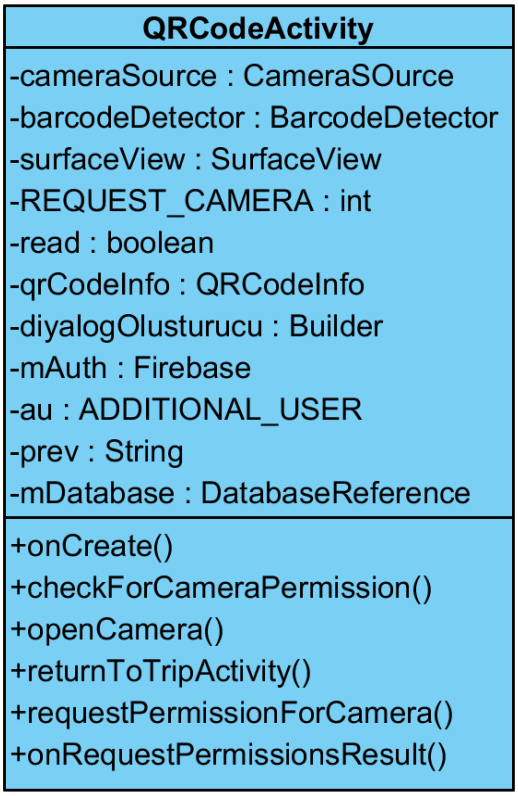

PanaromaActivity
-pv : PanoramaView
-sManager : SensorManager
-rotationVectorSensor : Sensor
-gyroscopeSensor : Sensor
~gameRotatiton : Sensor
~accelSensor : Sensor
~magnSensor : Sensor
~angle : float
+onCreate()
+onResume()
+onStop()
+goFullscreen()
+goSmall()
+goVr()

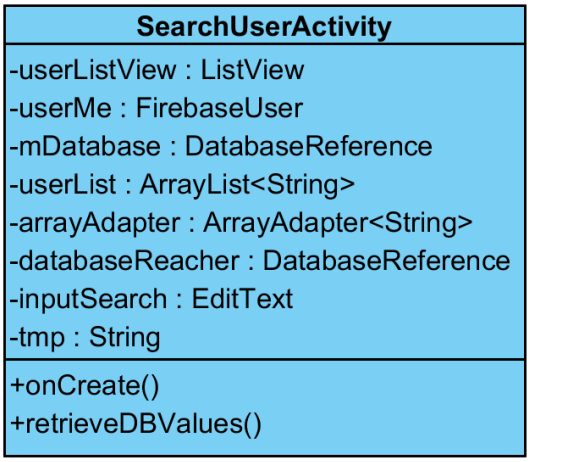

PlacesToSee
-recyclerView : RecyclerView
-googleMap : GoogleMap
-placesToSeeController : PlacesToSeeController
-REQUEST_LOCATION : int
~fusedLocationProviderClient : FusedLocationProviderClient
~layoutManager : LayoutManager
~ll1 : LinearLayout
~placesList : List<Places>
~photoReferences : ArrayList<String>
~iconURL : ArrayList<String>
~TAG : String
~builder : Builder
~adapter : Adapter<ViewHolder>
+onCreate()
+onMapReady()
+checkForLocationPermission()
+requestPermissionForLocation()
+getLastKnowLocation()
+onRequestPermissionsResult()
+parseTheJSONObject()
+tripStart()
+getPlacesPhotos()

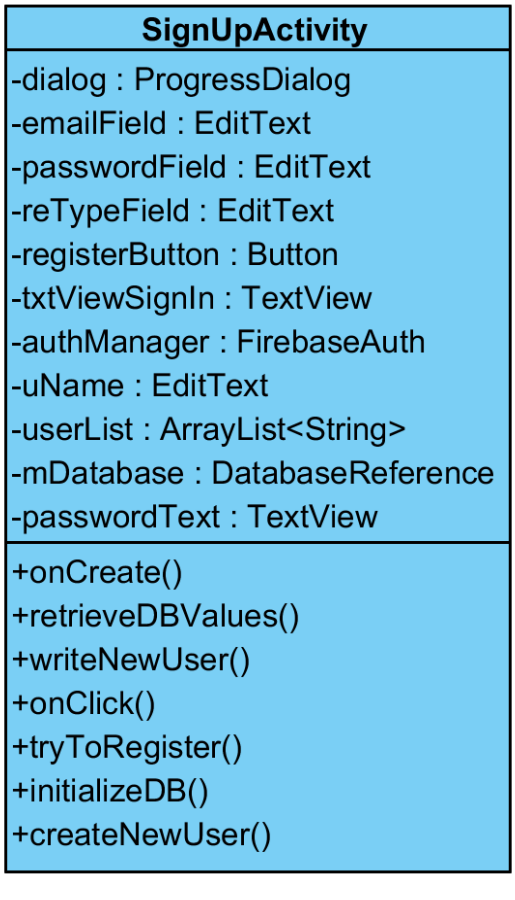

PlacesToSeeExpandedActivity
-recyclerView : String
-placeId : String
-placesToSeeController : ArrayList<String>
-landmarkReviews : ArrayList<String>
~reviewers : ArrayList<String>
~ratings : ArrayList<Double>
~landmarkBitmaps : ArrayList<Bitmap>
~au : ADDITIONAL_USER
~mDatabase : DatabaseReference
~pv : PanoramaView
~adapter : Adapter
~adapter3 : Adapter
~adapter2 : Adapter
~slider : Slider
~slider3 : Slider
~slider2 : Slider
~placeName : String
~placeStarImages : ImageView[]
~t11 : TextView
~gameRotatiton : Sensor
~sManager : SensorManager
~API : String
~cv1 : CardView
~cv2 : CardView
~cv3 : CardView
+onCreate()
+sendLandmarkPlaceIdsToDatabase()
+onResume()
+onCancelled()
+getLastKnowLocation()
+retrieveMoreInformationsAndPhotos()
+goVr()
+onPageScrollStateChanged()
+goSmall()
+goFullscreen()
+onSliderClick()
+onStop()
+onPageSelected()
+parseMoreInformationAndPhotos()
+setSlider()
+onPageScrolled()

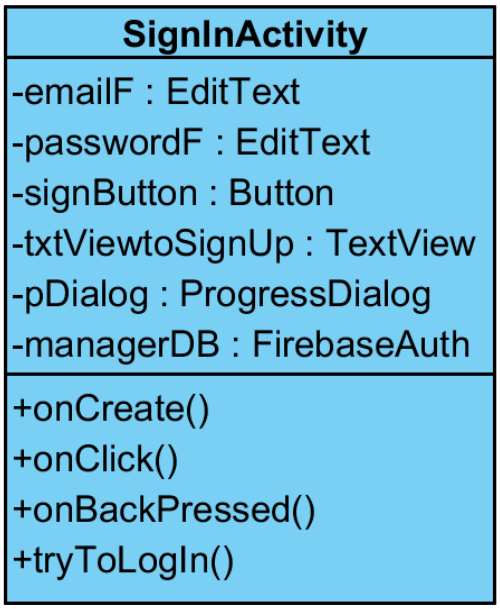

PreferencesActivity
-changePassword : Button
-changePassword3 : Button
-logout : Button
-aboutLDSTR : Button
-changeUsername : Button
-db : FirebaseDatabase
~mAuth : FirebaseAuth
+onCreate()
+meStart()
+changeUNAME()
+changePasswordPref()


DirectedUserPage
-myEmail : String
-tw : TextView
-logout : Button
-tripLogs : TextView
-lastTrip : TextView
-tripCounr : TextView
-user : FirebaseUser
-au : ADDITIONAL_USER
-ref : DatabaseReference
-changeUsername : Button
-db : FirebaseDatabase
~mAuth : FirebaseAuth
+onCreate()
+setup()
+onCancelled()

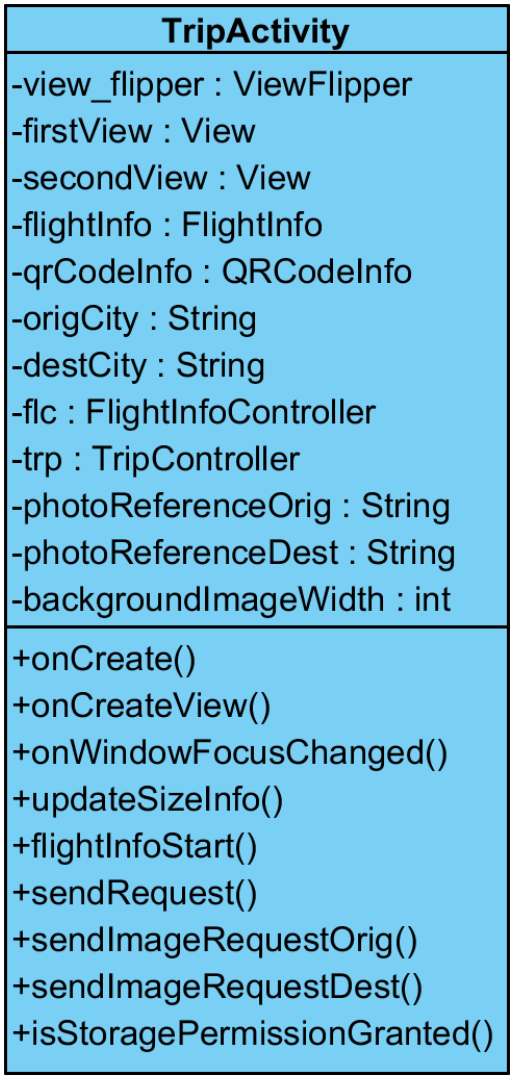

QRCodeActivity
-cameraSource : CameraSOurce
-barcodeDetector : BarcodeDetector
-surfaceView : SurfaceView
-REQUEST_CAMERA : int
-read : boolean
-qrCodeInfo : QRCodeInfo
-diyalogOlusturucu : Builder
-mAuth : Firebase
-au : ADDITIONAL_USER
-prev : String
-mDatabase : DatabaseReference
+onCreate()
+checkForCameraPermission()
+openCamera()
+returnToTripActivity()
+requestPermissionForCamera()
+onRequestPermissionsResult()
Powered By Visual Paradigm Community Edition


SearchUserActivity
-userListView : ListView
-userMe : FirebaseUser
-mDatabase : DatabaseReference
-userList : ArrayList<String>
-arrayAdapter : ArrayAdapter<String>
-databaseReacher : DatabaseReference
-inputSearch : EditText
-tmp : String
+onCreate()
+retrieveDBValues()
Powered By Visual Paradigm Community Edition

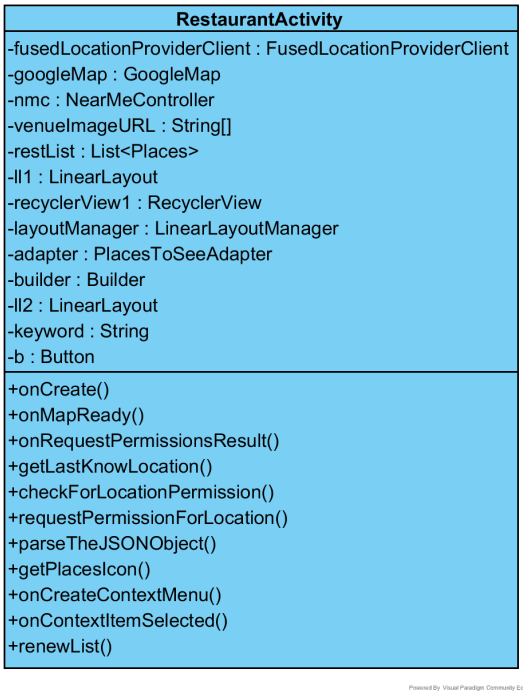

SignUpActivity
-dialog : ProgressDialog
-emailField : EditText
-passwordField : EditText
-reTypeField : EditText
-registerButton : Button
-txtViewSignIn : TextView
-authManager : FirebaseAuth
-uName : EditText
-userList : ArrayList<String>
-mDatabase : DatabaseReference
-passwordText : TextView
+onCreate()
+retrieveDBValues()
+writeNewUser()
+onClick()
+tryToRegister()
+initializeDB()
+createNewUser()
Powered By Visual Paradigm Community Edition

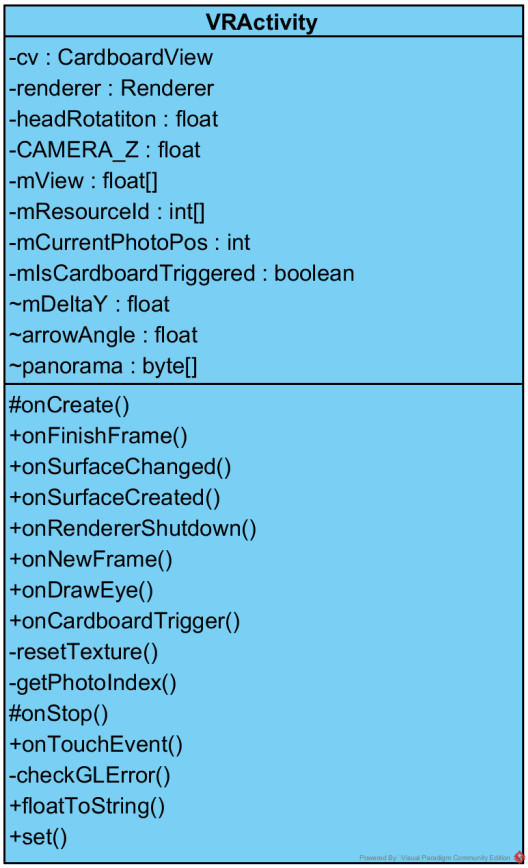


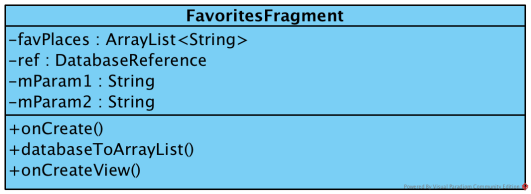
Powered By Visual Paradigm Community Edition

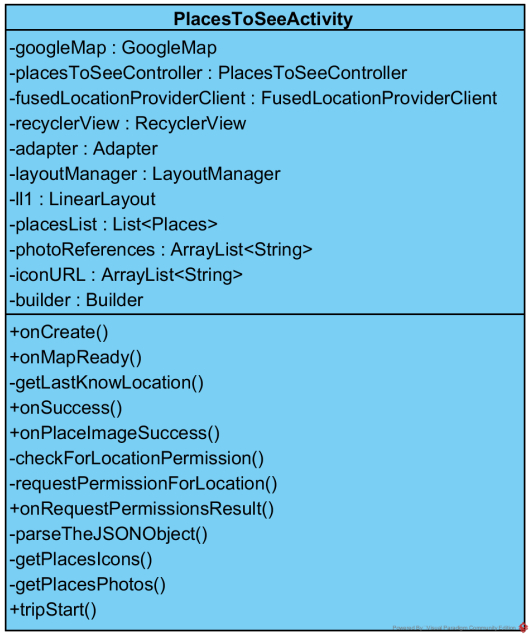


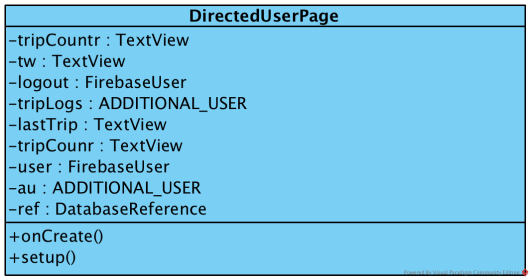
Powered By Visual Paradigm Community Edition


RestaurantActivity
-fusedLocationProviderClient : FusedLocationProviderClient
-googleMap : GoogleMap
-nmc : NearMeController
-venueImageURL : String[]
-restList : List<Places>
-ll1 : LinearLayout
-recyclerView1 : RecyclerView
-layoutManager : LinearLayoutManager
-adapter : PlacesToSeeAdapter
-builder : Builder
-ll2 : LinearLayout
-keyword : String
-b : Button
+onCreate()
+onMapReady()
+onRequestPermissionsResult()
+getLastKnowLocation()
+checkForLocationPermission()
+requestPermissionForLocation()
+parseTheJSONObject()
+getPlacesIcon()
+onCreateContextMenu()
+onContextItemSelected()
+renewList()


VRActivity
-cv : CardboardView
-renderer : Renderer
-headRotatiton : float
-CAMERA_Z : float
-mView : float[]
-mResourceId : int[]
-mCurrentPhotoPos : int
-mIsCardboardTriggered : boolean
~mDeltaY : float
~arrowAngle : float
~panorama : byte[]
#onCreate()
+onFinishFrame()
+onSurfaceChanged()
+onSurfaceCreated()
+onRendererShutdown()
+onNewFrame()
+onDrawEye()
+onCardboardTrigger()
-resetTexture()
-getPhotoIndex()
#onStop()
+onTouchEvent()
-checkGLError()
+floatToString()
+set()


FavoritesFragment
-favPlaces : ArrayList<String>
-ref : DatabaseReference
-mParam1 : String
-mParam2 : String
+onCreate()
+databaseToArrayList()
+onCreateView()


PlacesToSeeActivity
-googleMap : GoogleMap
-placesToSeeController : PlacesToSeeController
-fusedLocationProviderClient : FusedLocationProviderClient
-recyclerView : RecyclerView
-adapter : Adapter
-layoutManager : LayoutManager
-ll1 : LinearLayout
-placesList : List<Places>
-photoReferences : ArrayList<String>
-iconURL : ArrayList<String>
-builder : Builder
+onCreate()
+onMapReady()
-getLastKnowLocation()
+onSuccess()
+onPlaceImageSuccess()
-checkForLocationPermission()
-requestPermissionForLocation()
+onRequestPermissionsResult()
-parseTheJSONObject()
-getPlacesIcons()
-getPlacesPhotos()
+tripStart()


DirectedUserPage
-tripCountr : TextView
-tw : TextView
-logout : FirebaseUser
-tripLogs : ADDITIONAL_USER
-lastTrip : TextView
-tripCounr : TextView
-user : FirebaseUser
-au : ADDITIONAL_USER
-ref : DatabaseReference
+onCreate()
+setup()

10. Glossary

In this part of this report, we will briefly discuss our preferred choice of development methodology. After that, we will explain how the server architecture runs on a cloud deployed service.

10.1.1. Development Strategy and Methodology

While developing LodeStar, we have embraced the agile[60] development methodology. This has accelerated our program development since this development methodology requires frequent meetings with the team. With each meeting, we have assigned ourselves to one of the open issues and worked on that until the next meeting. Every issue had its own difficulty score. If one of choose a difficult task, he was given more time to solve the problem. This has enabled us to create a more or less equal working environment and kept motivation up at all times.

10.1.2. Server and Deployment Technology

Our server runs on a Host OS, which runs on Ubuntu. Above this layer, we have integrated the Docker Engine in order to manage our server side applications. Docker Engine works with container images, which according to the definition in Docker's website "are an abstraction at the app layer that packages code and dependencies together"[10]. There can be multiple containers in a host OS, and for that reason, in our server, we have a container/s depending on the traffic to run our server side applications which are in Node JS.

*The code presented in this report is subject to change until CSFair 2018. We may incorporate small bug fixes and add minor features until the demo day

11. References

[1] IBM, "UML - Basics," June 2003. [Online]. Available: http://www.ibm.com/developerworks/rational/library/769.html.

[2] IEEE, "IEEE Citation Reference," September 2009. [Online]. Available: https://m.ieee.org/documents/ieeecitationref.pdf. [Accessed 9-Feb-2018].

[3] *Firebase*. [Online]. Available: https://firebase.google.com/. [Accessed: 11-Feb-2018].

[4] N. Foundation, *Node.js*. [Online]. Available: https://nodejs.org/en/. [Accessed: 11-Feb-2018].

i[5] "Docker," *Docker*. [Online]. Available: https://www.docker.com/. [Accessed: 11-Feb-2018].

[6] "Google VR", GitHub, 2018. [Online]. Available: https://github.com/googlevr. [Accessed: 02- May- 2018].

[7] Apple Staff. 2006. About Swift. [Online]. Available: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html. [Accessed: 10-Feb-2018]

[8] "Unity," *Unity*. [Online]. Available: https://unity3d.com/. [Accessed: 11-Feb-2018].

[9] "Street View | Google Developers", Google Developers, 2018. [Online]. Available: https://developers.google.com/streetview. [Accessed: 02- May- 2018].

[10] "OpenGL ES | Android Developers," Android Developers. [Online]. Available: https://developer.android.com/guide/topics/graphics/opengl. [Accessed: 02-May-2018].

[11] D. Pierce, "Google Cardboard Is VR's Gateway Drug," *Wired*, 03-Jun-2017. [Online]. Available: https://www.wired.com/2015/05/try-google-cardboard/. [Accessed: 02-May-2018].

[12] "Google Cardboard," *Google Cardboard – Google VR*. [Online]. Available: /. [Accessed: 02-May-2018].

[13] S. today, "The digital design toolkit", Sketch, 2018. [Online]. Available: https://www.sketchapp.com/. [Accessed: 02- May- 2018].

[14] "Desktop publishing software | Download free Adobe InDesign CC trial," *Buy Adobe InDesign CC | Desktop publishing software and online publisher*. [Online]. Available: http://www.adobe.com/InDesign. [Accessed: 02-May-2018].

[15] "Adobe XD CC," *Buy Adobe XD CC | UX/UI design, prototyping & collaboration tool*. [Online]. Available: https://www.adobe.com/products/xd.html. [Accessed: 02-May-2018].

[16] "Coding Horror," *Understanding Model-View-Controller*. [Online]. Available: https://blog.codinghorror.com/understanding-model-view-controller/. [Accessed: 11-Feb-2018].

[17] "Flight Status API / Flight Tracking API / FlightAware API ✈ Commercial Services ✈ FlightAware," FlightAware. [Online]. Available: https://flightaware.com/commercial/flightxml/. [Accessed: 09-Oct-2017].

[18] "Foursquare Developer", Developer.foursquare.com, 2018. [Online]. Available: https://developer.foursquare.com/. [Accessed: 02- May- 2018].
[19]Cost of Living. [Online]. Available: https://www.numbeo.com/cost-of-living/. [Accessed: 22-Dec-2017].
[20] OpenWeatherMap.org. Current weather and forecast. openweathermap. [Online]. Available: https://openweathermap.org/. [Accessed: 22-Dec-2017].
[21] Our currency data API powers the Internet's most dynamic startups, brands and organisations. Exchange Rates API, JSON format, for Developers. [Online]. Available: https://openexchangerates.org/. [Accessed: 22-Dec-2017].
[22] "Domain Name Registration Search Results • Namecheap.com," Namecheap. [Online]. Available: https://www.namecheap.com/domains/registration/results.aspx?domain=lodestarapp. [Accessed: 02-Oct-2017].
[23] "Manufacture Cardboard," Google. [Online]. Available: https://vr.google.com/cardboard/manufacturers/. [Accessed: 05-Oct-2017].
[24] "Lithium Ion Batteries," Lithium Ion Batteries - American Disposal. [Online]. Available: https:// www.americandisposal.com/blog/lithium-ion-batteries. [Accessed: 04-Oct-2017].
[25] Claire Cain Miller and Kevin J. O'Brien, "Germany's Complicated Relationship With Google Street View," The New York Times, 23-Apr-2013. [Online]. Available: https://bits.blogs.nytimes.com/2013/04/23/germanys-complicated-relationship-with-google-street-view/. [Accessed: 08-Oct-2017].
[26] Android Developers. Activities. Android Developers. [Online]. developer.android.com/guide/components/activities/index.html. [Accessed: 10-Feb-2018].
[27] "Build software better, together", GitHub, 2018. [Online]. Available: https://github.com/. [Accessed: 02- May- 2018].
[28] "Xcode - Apple Developer", Developer.apple.com, 2018. [Online]. Available: https://developer.apple.com/xcode/. [Accessed: 02- May- 2018].
[29] "CocoaPods for Development-Time Modularity – Twitch Blog", Twitch Blog, 2018. [Online]. Available: https://blog.twitch.tv/cocoapods-for-development-time-modularity-36f92f63a5d3. [Accessed: 02- May- 2018].
[30] "What is Docker?", Docker, 2018. [Online]. Available: https://www.docker.com/what-docker. [Accessed: 02- May- 2018].
[31] ImageMagick Studio LLC, "Convert, Edit, Or Compose Bitmap Images @ ImageMagick," ImageMagick. [Online]. Available: https://www.imagemagick.org/. [Accessed: 02-May-2018].
[32] Paw Inc, "Paw – The most advanced API tool for Mac," Paw – The most advanced API tool for Mac. [Online]. Available: https://paw.cloud/. [Accessed: 02-May-2018].

[33] "Postman," *Postman*. [Online]. Available: https://www.getpostman.com/. [Accessed: 02-May-2018].
[34] S. Lundquist, "Photoshop vs. Illustrator vs. InDesign. Which Adobe product should you use?," *99designs*, 27-Apr-2018. [Online]. Available: https://99designs.com/blog/tips/photoshop-vs-illustrator-vs-indesign/. [Accessed: 02-May-2018].
[35] "What is Illustrator?", Helpx.adobe.com, 2018. [Online]. Available: https://helpx.adobe.com/illustrator/atv/cs6-tutorials/what-is-illustrator-.html. [Accessed: 02- May- 2018].
[36] "Overview of Google Play Services | Google APIs for Android | Google Developers," *Google*. [Online]. Available: https://developers.google.com/android/guides/overview. [Accessed: 02-May-2018].
[37] "Google Play services - Apps on Google Play," *Google*. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.gms. [Accessed: 02-May-2018].
[38] "Android", Android, 2018. [Online]. Available: https://www.android.com/intl/tr_tr/. [Accessed: 02- May- 2018].
[39] Khronos Group, "The Industry's Foundation for High Performance Graphics," *OpenGL.org*. [Online]. Available: https://www.opengl.org/. [Accessed: 02-May-2018].
[40] "Google Maps APIs | Google Developers", Google Developers, 2018. [Online]. Available: https://developers.google.com/maps/. [Accessed: 02- May- 2018].
[41] "Google Maps | Brands of the World™ | Download vector logos and logotypes", Brandsoftheworld.com, 2018. [Online]. Available: http://www.brandsoftheworld.com/logo/google-maps-0. [Accessed: 02- May- 2018].
[42] "Google Street View Image API | Google Developers", Google Developers, 2018. [Online]. Available: https://developers.google.com/maps/documentation/streetview/. [Accessed: 02-May- 2018].
[43] "Four square, Foursquare", Shareicon.net, 2018. [Online]. Available: https://www.shareicon.net/four-square-foursquare-101980. [Accessed: 02- May- 2018].
[44] "Food, Nightlife, Entertainment," FourSquare. [Online]. Available: https://foursquare.com. [Accessed: 05-Nov-2017].
[45] J. Lane, "The 10 Most Spoken Languages In The World," Babbel Magazine, April 2014. [Online]. Available: https://www.babbel.com/en/magazine/the-10-most-spoken-languages-in-the-world/. [Accessed: Apr 29 2018].
[46] C. Arthur, "Free Wi-Fi with that coffee? Why cafes are embracing an age of connectivity," The Guardian, December 2011. [Online]. Available: https://www.theguardian.com/technology/2011/dec/22/free-wi-fi-access-cafes-uk. [Accessed: Apr 29 2018].

[47] "Free Wi-Fi at Starbucks," Wi-Fi Space. [Online]. Available: https://wifispc.com/articles/free-wi-fi-at-starbucks.html. [Accessed: Apr 29 2018].
[48] Huff Post Tech, "Free McDonald's WiFi Starts TODAY: Find Locations," Huff Post, May 2011. [Online]. Available: https://www.huffingtonpost.com/2010/01/15/free-mcdonalds-wifi-start_n_424425.html. [Accessed: Apr 29 2018].
[49] P. Sisson, "Wi-Fi Bus Stops and Wired Phone Booths: The Future of the Digital City," Curbed, September 2015. [Online]. Available: https://www.curbed.com/2015/9/18/9920076/wifi-bus-stop-and-wired-phone-booths-the-future-of-the-digital-city. [Accessed: Apr 29 2018].
[50] S. Murphy, "Where to Find Free Wi-Fi at U.S. and International Airports," Mashable, August 2013. [Online]. Available: https://mashable.com/2013/08/16/airports-free-wifi/#jpJaFFB2amqf. [Accessed: Apr 29 2018].
[51] H. Thompson, "How Secure Are Mobile Applications," Business Insider, May 2011. [Online]. Available: http://www.businessinsider.com/how-secure-are-mobile-applications-2011-5. [Accessed: Apr 29 2018].
[Online]. Available: [Accessed: Apr 29 2018].
[52] Investopedia Staff, "6 Top-Traded Currencies and Why They're So Popular," Investopedia. [Online]. Available: https://www.investopedia.com/articles/forex/11/popular-currencies-and-why-theyre-traded.asp. [Accessed: 2 May 2018].
[53] "Natural problems for stereoscopic depth perception in virtual environments," *Semantic Scholar*. [Online]. Available: https://www.semanticscholar.org/paper/Natural-problems-for-stereoscopic-depth-perception-Wann-Rushton/3632f5792dcca1a66229ff467d18c7291a00a861. [Accessed: 02-May-2018].
[54] "OpenGL," *Wikipedia*, 30-Apr-2018. [Online]. Available: https://en.wikipedia.org/wiki/OpenGL. [Accessed: 02-May-2018].
[55] "Numbeo Cost of Living", ProgrammableWeb, 2018. [Online]. Available: https://www.programmableweb.com/api/numbeo-cost-living. [Accessed: 02- May- 2018].
[56] Apple Inc, "Purchase and Activation," *Purchase and Activation - Support - Apple Developer*. [Online]. Available: https://developer.apple.com/support/purchase-activation/. [Accessed: 02-May-2018].
[57] T. Mackenzie, "App store fees, percentages, and payouts: What developers need to know," *TechRepublic*. [Online]. Available: https://www.techrepublic.com/blog/software-engineer/app-store-fees-percentages-and-payouts-what-developers-need-to-know/. [Accessed: 02-May-2018].

[58] "Connecting to the iTunes Store.," *Connecting to the iTunes Store*. [Online]. Available: https://www.appstore.com/. [Accessed: 02-May-2018].
[59] "Flight Aware API , ProgrammableWeb, 2018. [Online]. Available: https://www.programmableweb.com/api/flightaware-firehose. [Accessed: 02- May- 2018].
[60] "Agile 101: Back to the Basics," *VersionOne*. [Online]. Available: https://www.versionone.com/agile-101/. [Accessed: 02-May-2018].

[This page is intentionally left blank]

[end of report]