Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: LodeStar

Low Level Design 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

Feb 12, 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 CS491/1.

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].

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 the low-level architecture and design of LodeStar. In the next section, the trade-offs of our design and engineering standards will be explained. What follows will be the documentation guidelines. After that, information about the packages and interfaces of LodeStar's systems will be presented. Finally, the class diagrams and a detailed look into each of LodeStar's software components will conclude the report.

1.1. Design Trade-Offs

In software development, when choosing to enhance a certain feature of a program, usually another one has to be sacrificed. Almost all decisions with respect to design comes with certain trade-offs and and implications[2]. For this reason, we spent a lot of time identifying LodeStar's trade-offs during the design process to create the most optimized system for the project's needs. In the following sections, trade-offs we considered will be presented.

1.1.1. Functionality vs Usability

Two of the most important aspects to consider about design trade-offs are LodeStar's functionality and usability. Functionality refers to whether LodeStar's functionalities are working as intended whereas the usability refers to the ease of use and intuitiveness of these presented functions[3]. Even if LodeStar offers top-notch functionalities and services, they will offer no value if the user cannot interact with them. For this reason, we spent a great amount of effort to make the user interface of LodeStar as intuitive and easy to use as possible. All the functions will be easily accessible by the user. Therefore, it is possible to say LodeStar's design favors usability over functionality.

1.1.2. Security vs. Cost

Security is of paramount importance for an application which guides people through international airports. We recognized that in the near past, many security issues resulted in global problems ranging from fraud to terrorism[4]. On top of sharing the pictures of some of the world's most crowded airports, LodeStar also collects a lot of user data. This information is highly valuable and must be kept confidential at all costs. Therefore, LodeStar's design favors security to cost. No amount of time, effort or money is too great if it can prevent unspeakable horrors.

1.1.3. Space vs. Time

Efficiency is what all software programs strive to achieve, and this efficiency can be considered as in efficiency in space and efficiency in time[5]. LodeStar stores all data in the server side rather than within the application itself. This way, LodeStar will offer high speed within the application as the server will be carrying the bulk of the information. Since we consider server space to be cheaper than the valuable time of the user, LodeStar's application side will operate at top speed. The extra space requirements of the server is a price we are willing to pay for a fluent user experience.

1.1.4. Compatibility vs Extensibility

LodeStar will have iOS and Android versions. Therefore, it is important LodeStar is compatible with all these systems. Extensibility is also important as LodeStar will have to evolve with updates in the future. There may even be a web browser version of LodeStar Although the ability to extend the project is crucial, we feel creating an application compatible with multiple platforms is much more desired.

1.2. Engineering Standards

For the descriptions of the class interfaces, diagrams, scenarios, use cases, subsystem compositions and hardware depictions, this report follows the UML guidelines[6]. 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[7]. Again, this is a commonly used method and the one preferred in Bilkent University.

1.3. 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[8].
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[9].

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.
Google VR: Provides virtual reality rendering facilities for VR glasses[10].
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[11]. 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[12].

1.4. Life Long Learning

The users of LodeStar 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.

1.5. Interface Documentation Guidelines

In this report, all the class names are named in the standard 'ClassName' format, where all of these names are singular. The variable and method names follow a similar rule as in 'variableName' and 'methodName()'. In the class description hierarchy, the class name comes first, seconded by the attributes of the class, and finally concluded with the methods. The detailed outline looks similar to the one presented below:

Class Name	
Description of Class	
Attributes	
Type of Attribute : Name of Attribute	
Methods	
NameOfMethod(ParametersOfMethod):	Description of Method

2. Packages

LodeStar's Low Level system is composed of four parts. First two parts represent the client side. These parts are called the View and the Controller. Last two parts are located inside the Server subsystem are are called Logic Tier and Data Tier. A high level description of all these subsystems can be found in LodeStar's High Level Design Report[13].

In our project, we used a novel approach to connect the client and the server. Client presents the system information to users. It also receives the interaction of users and sends them to server to keep the UI (user interface) running. On the other hand, sever side is responsible for all non-local data processing and API usage.

2.1. 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[14] 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.

2.1.1. 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.

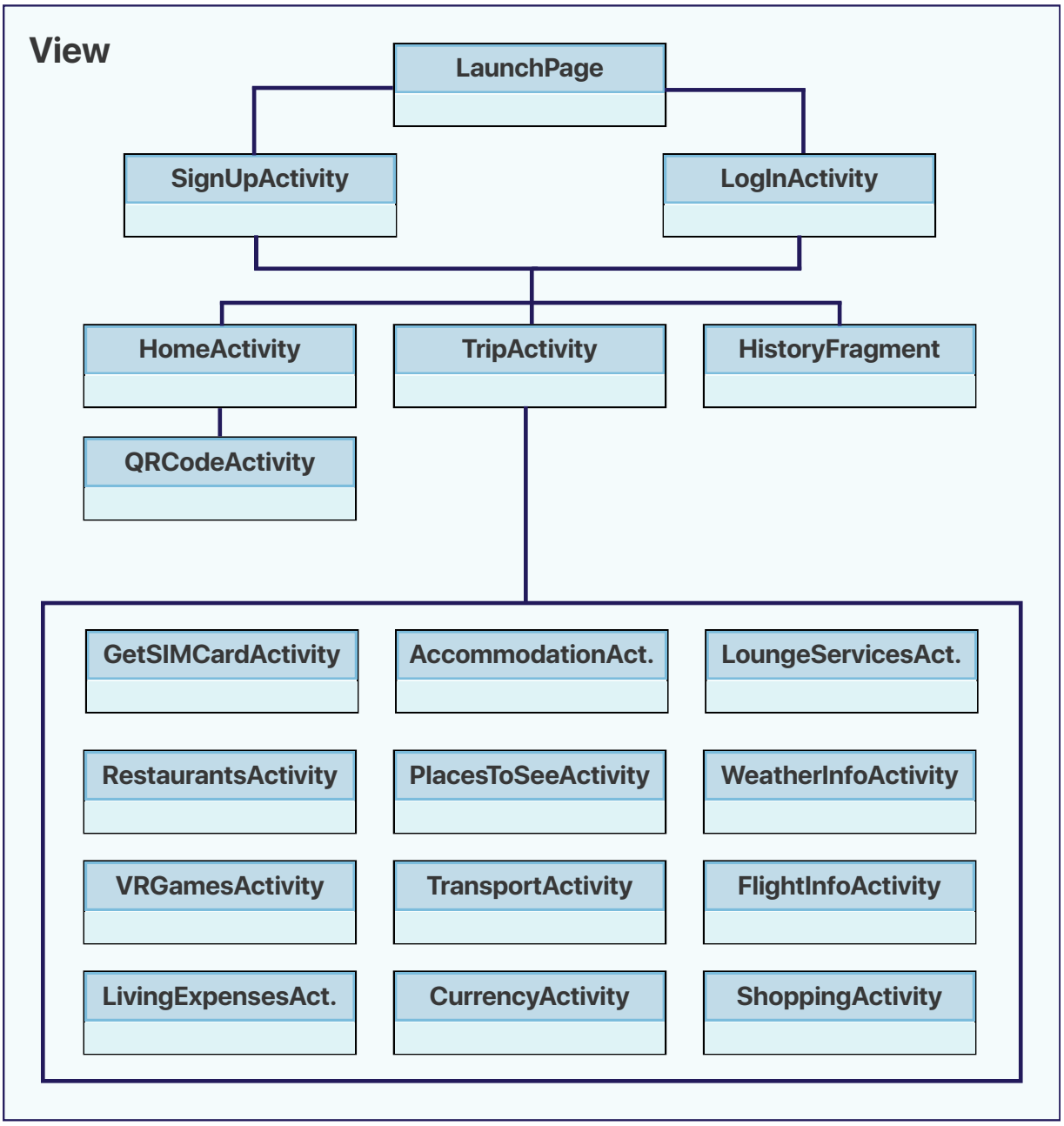

Figure 1 - View Subsystem

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.

2.1.2. 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.

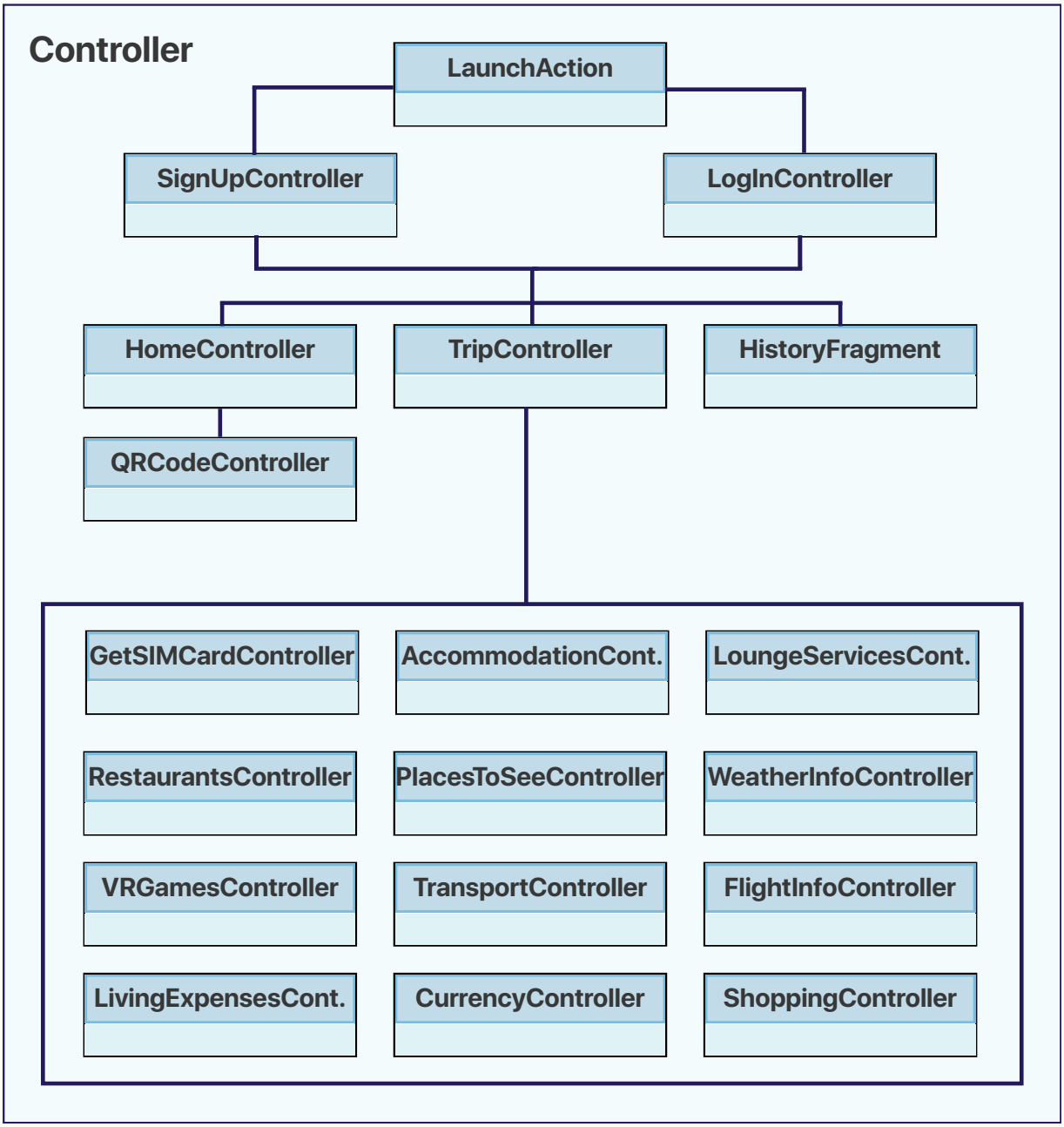

Figure 2 - Controller Subsystem

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.

2.2. 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[15], Google, Foursquare[16], Numbeo[17] and OpenWeather[18]. 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.

2.2.1. 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.

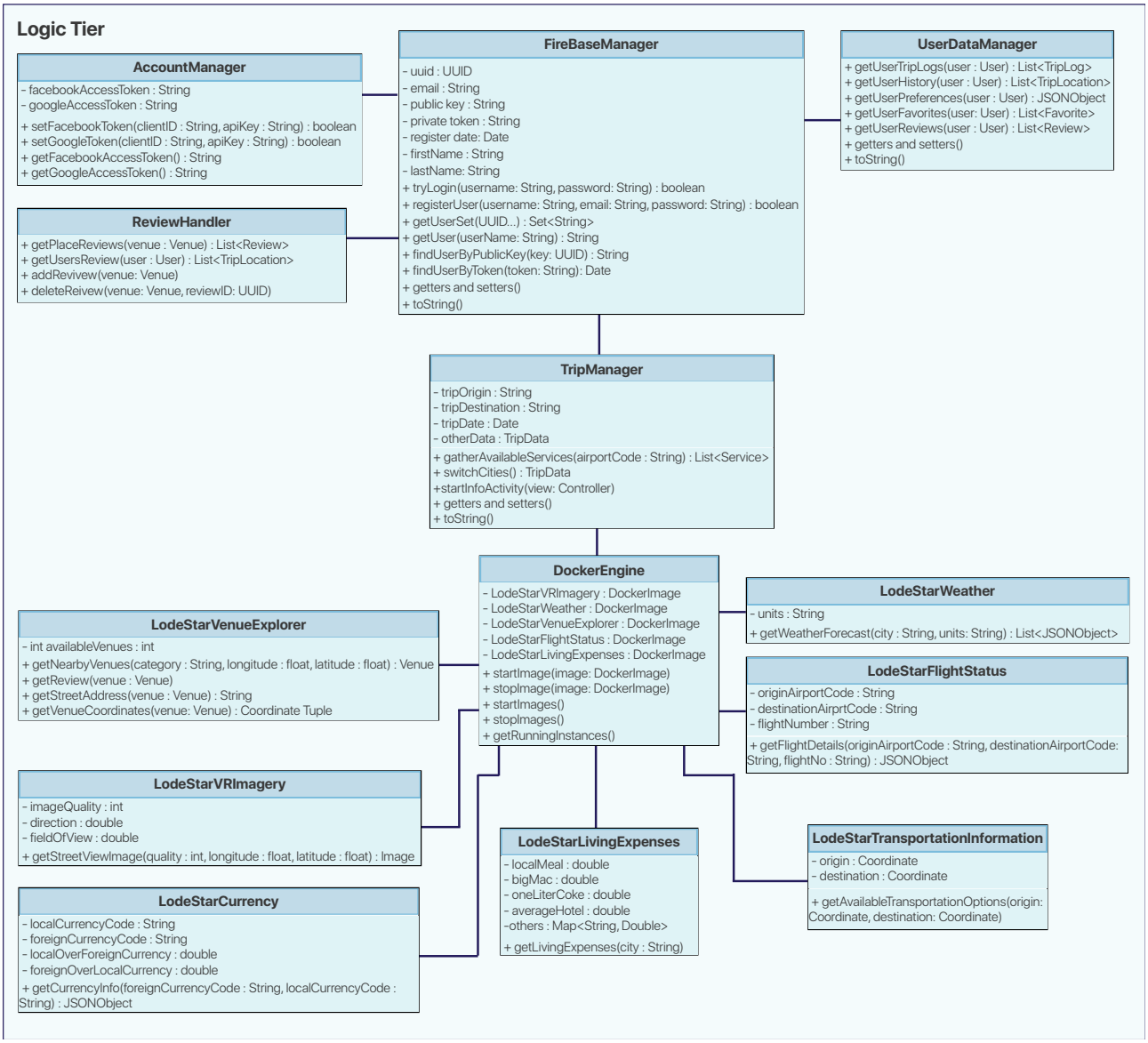

Figure 3 - Logic Tier Subsystem

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

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.

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[17]

LodeStarCurrency: when the user wants to see exchange rates, this will send requests to OpenExchangeRate[19] API and retrieve currency exchange information

LodeStarWeather: this class will send requests to OpenWeather APIs to get weather data for specified city. [18]

2.2.2. Data Tier

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

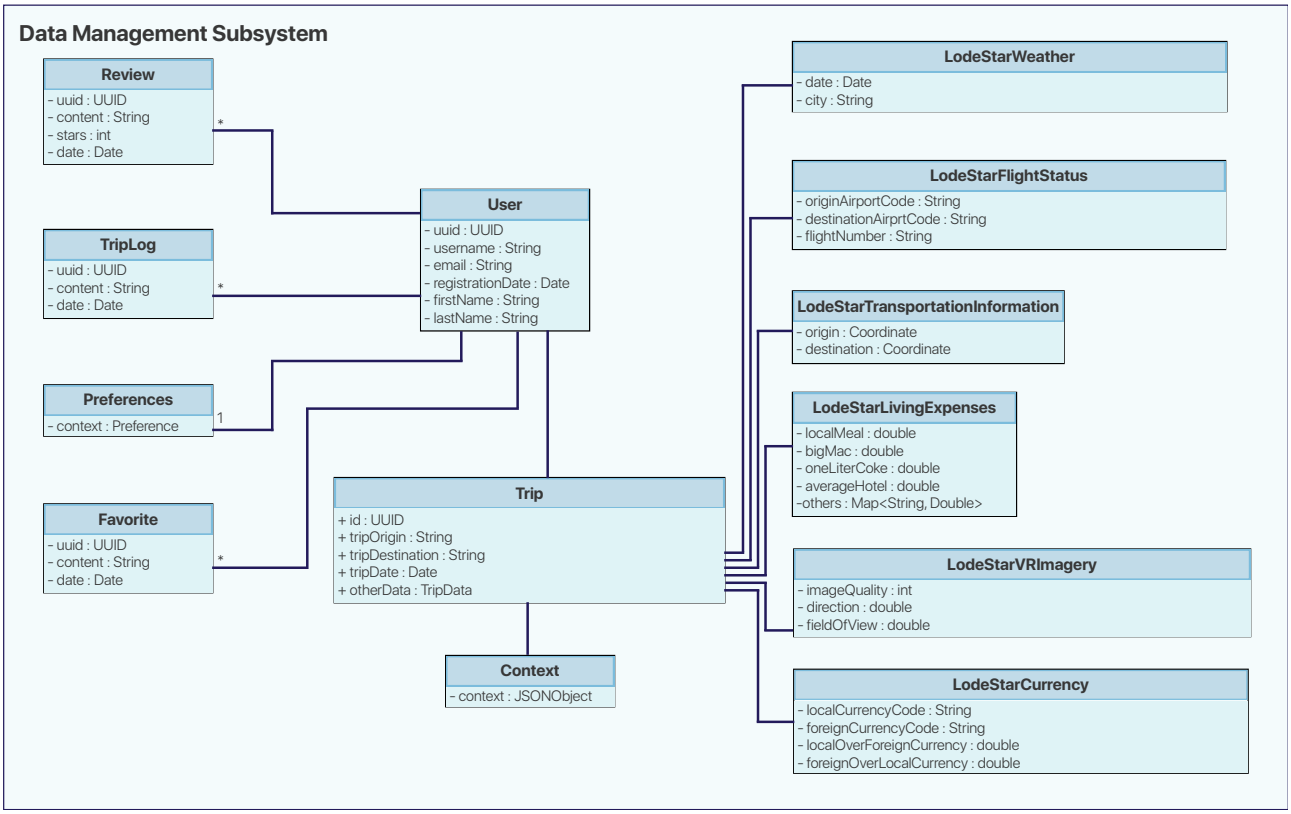

Figure 4 - Data Tier Subsystem

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

3. Class Interfaces

In this section of the report, signatures, properties and methods of the classes will be provided. Furthermore, their specific duties will be discussed in detail.

3.1. Client

The subsystems in this section are dedicated for the mobile applications. Listed class names and function names are subject to change throughout the development lifecycle of the project.

3.1.1. View

class SignUpActivity extends AppCompatActivity	
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.	
Attributes	
private EditText emailField	
private EditText usernameField	
private EditText passwordField	
private EditText reTypeField	
Methods	
public int tryToRegister()	This function tries to register the user. If the user has provided wrong credentials (e.g. if a proper e-mail is not entered), it displays an error message saying that the user could not be registered and returns -1. If the user entered the credentials correctly, the functions returns 1.
public void txtSignInXML(View v)	This function is called whenever the text “Already Signed Up?” is clicked. It redirects the user to LoginActivity so that the user can login.
public void registerButtonXML(View v)	This function is called whenever the register button is clicked. It takes the values in user credentials and puts them into our firebase database.
public boolean reTypeCheck()	Checks if the initial entered password matches the reTypeField to avoid any ambuguities in password creation.
public void onCreate()	Initializes the variables and sets up the page

class LogOutActivity extends AppCompatActivity	
This class is responsible from ensuring a secure logout from the system for a user. After a successful logout, the account of the user becomes inaccessible until after a new login.	
Attributes	
private static final String TAG	
private static final int RC_SIGN_IN	
private Button signInWithEmail	
private SignInButton signOutButton	
Methods	
public void onCreate():	This function initializes the variables and sets up the page for use.
public void onClick(View v):	Whenever there is an onclick event, this method is called to redirect user to the appropiate place.
public void signIn():	This function ensures the account with the provided credentials safely logs out of the system.
public void onConnectionFailed(ConnectionResult connectionResult):	Checks if the connection to the database is established or not. Since the application is using the Firebase database, it is important that the user has a reliable internet connection in order to be able to safely sign out.

class LoginActivity extends AppCompatActivity	
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.	
Attributes	
private static final String TAG	
private static final int RC_SIGN_IN	
private Button signInWithEmail	
private SignInButton logInButton	
Methods	
public void onCreate():	This function initializes the variables and sets up the page for use.
public void onClick(View v):	Whenever there is an onclick event, this method is called to redirect user to the appropiate place.
public void signIn():	This function is called whenever the register button is clicked. It takes the values in user credentials and puts them into our firebase database.
public void onConnectionFailed(ConnectionResult connectionResult):	Checks if the connection to the database is established or not. Since the application is using the Firebase database, it is important that the user has a reliable internet connection in order to be able to successfully sign in.

class RegisterViaEmailActivity extends AppCompatActivity	
This class is responsible for creating a new account for the user granted the email provided is unique and the password is adequately safe.	
Attributes	
private static final String TAG	
private String email	
private String password	
private String accountID	
Methods	
public void onCreate()	This function initializes the variables and sets up the page for use.
public void onClick(View v)	Whenever there is an onclick event, this method is called to redirect user to the appropiate place.
public void checkEmail(String email)	This function checks whether the entered email by the user is already being used.
public void checkPassword(String password)	This function checks whether the entered password by the user contains a number, a capital letter, contains at least 8 characters, and contains nothing other than English characters and numbers.
public void createAccount(String email, String password)	After making sure the email is unique and the password is safe to use, this method creates a new account for the user and issues a unique ID to it.

class ForgotPasswordActivity extends AppCompatActivity	
This class is called when the user forgets his/her password. In that case user is asked to provide the e-mail he/she used during sign up process. An auto generated password is sent to the user's e-mail.	
Attributes	
private EditText emailField	
private Button forgotPasswordButton	
Methods	
public void onCreate()	This function initializes the variables and sets up the page for use.
public void forgotPasswordClick(View v):	Whenever the use presses the forgot password button, this method validates if there exists a user with the given e-mailField in the database. If there exists such a field, sends a auto generated password to the user's email.

class WeatherInformation	
This class is a model class for the 5-day forecast of a given city. This class is used when the 5-day forecast data is fetched from the server.	
Attributes	
private String date	
private String description	
private double feelsLikeTemperature	
private double humidity	
Methods	
Getters for attributes	

class WeatherInformationAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder>	
This class is responsible for taking a list of WeatherInformation objects from the WeatherInformationActivity and showing them in CardView type	
Attributes	
private List<WeatherInformation> weatherInformationList	
private static final int TODAY	
private static final int OTHER	
Methods	
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)	This function binds the view to it's respective CardView
public void onBindViewHolder(RecyclerView.ViewHolder holder, int position)	This function binds the elements in the CardView with their respective information
public int getItemCount()	This function returns the number of WeatherInformation objects that are taken from the server
public int getItemViewType(int position)	This function allows to distinguish today's weather with the other day's weather

class FlightInfo	
This class is responsible for getting flight information using the flight number. Sends back the requested flight information in JSONObject format.	
Attributes	
private String dest	
private String orig	
private String dest_gate	
private String orig_gate	
private String orig_airport	
private String dest_airport	
private String orig_date	
private String dest_date	
private String orig_localtime	
private String dest_localtime	
private int distance	
private int speed	
private String aircraft	
private String link	
Methods	
Get methods for attributes	

class FlightInfoActivity	
This class is responsible for displaying flight details taken from server side of the system according to flight number.	
Attributes	
FlightInfo flightInfo	
Methods	
public void tripStart()	Navigates the view back to Trip page

class 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.	
Methods	
public void readQRCode()	Navigates the view back to Trip page
public void getFlightNo()	Gets the flight no entered by user in the input area
pulic void openCardboard()	Opens the related website when user taps its button

class LivingExpensesActivity	
This view class displays the living expenses information taken from server for the selected city	
Methods	
public void livingExpenses()	Initializes the LivingExpenses page

class QRCodeInfo
This class is used in storing the informations from the detected Barcode
Attributes
private String from
private String to
private String flightCode
Methods
Getters and setters for the attributes

class 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.	
Attributes	
private CameraSource cameraSource	
private BarcodeDetector barcodeDetector	
private SurfaceView surfaceView	
private final int REQUEST_CAMERA	
private QRCodeInfo qrCodeInfo	
Methods	
protected void onCreate(Bundle savedInstanceState)	This function runs on page initialization. Parses UI XML just before rendering the view
private void checkForCameraPermission()	This function dynamically checks permission for camera usage, if the permission is not given, it requests the permission from the user dynamically
private void openCamera()	This function creates and binds the BarcodeDetector and CameraSource objects and activates the camera
public void surfaceCreated(SurfaceHolder surfaceHolder)	This function is one of the callback methods for the SurfaceView object, at runtime it opens the camera
public void surfaceDestroyed(SurfaceHolder surfaceHolder)	This function stops the camera when user goes back from this activity
public void receiveDetections(Detector.Detections<Barcode> detections)	This function detects the Barcode object by using the camera, parses the returned information from the Barcode object and passes it to the TripActivity
public void returnToTripActivity()	This function creates an Intent object to go to the TripActivity while taking the information for the Barcode object
private void requestPermissionForCamera()	This function requests permission from user for camera usage at runtime.
public void onRequestPermissionsResult(int requestCode, @NonNull String permissions[], @NonNull int[] grantResults)	This function gets the result from the permission and either opens the camera to detect the barcode or goes back to the previous activity depending on the result

class TripActivity	
This class is responsible for displaying the Trip page that will show cities in the travel and interfaces to related pages.	
Attributes	
public ViewFlipper view_flipper	
public View firstView	
public View secondView	
private FlightInfo flightInfo	
Methods	
public void weatherStart(View view)	Navigates to weather page
public void flightInfoStart(View view)	Navigates to flight details page
public void currencyStart(View view)	Navigates to flight details page
public void livingStart(View view)	Navigates to living expenses page
public void transportStart(View view)	Navigates to transportation page
public void shoppingStart(View view)	Navigates to shopping page
public void restaurantsStart(View view)	Navigates to restaurants page
public void placesStart(View view)	Navigates to places to see page
public void accomodationStart(View view)	Navigates to accommodation page
private void updateSizeInfo()	Updates button sizes dynamically to fit the size of the phone upon page load

class 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.	
Attributes	
private RecyclerView mRecyclerView	
private RecyclerView.Adapter mAdapter	
private RecyclerView.LayoutManager mLayoutManager	
private List<WeatherInformation> weatherInformationList	
Methods	
protected void onCreate(Bundle savedInstanceState)	This function runs on page initialization. Parses UI XML just before rendering the view
public void sendRequestToServer(String requestFromTheUrl)	This function sends request to server with a city name parameter to fetch the 5-day forecast data of that city from the server
public void onResponse(JSONArray response)	This function stores the servers response in a JSONArray
private void parseTheJSONResponse(JSONArray weatherInformationServer)	This function takes the response from the server, parses the information and creates the respective WeatherInformation objects

class CurrencyActivity	
This class displays the currency rates between travelled countries.	
Methods	
public void tripStart()	Navigates the view back to Trip page

3.1.2. Controller

Class interfaces for controller classes are listed below. For each of the following controller classes, you can find a short description and the related methods listed. All the methods available have their parameters and descriptions available as well.

HomeController	
This class is responsible for the controller functions related with HomeActivity class, basically for checking the validity of entered flight information.	
Methods	
public boolean checkFlight(String)	Checks the validity of the entered flight info of user.

HistoryController	
This class is for getting the user history from server for logged in user in the client.	
Methods	
public List<String> getHistory(int)	Receives user history information details from server side in String format.

FavoritesController	
This class is for getting the user history from server for logged in user in the client, associated with FavoritesFragment class.	
Methods	
public List<String> getFavorites(int)	Gets user favorite details from server side in String format.

PlacesController	
This class is responsible of the controller activity elated with UserActivity class, like user preferred settings for the client side.	
Methods	
public JSONObject getPreferences(String)	Gets user's preferred settings from server side.
public List<String> getComments	Gets user comments from last trip to show on User page.

TransportController	
This class is the controller associated with TransportActivity class, for getting transport information from server side.	
Methods	
public JSONObject getTransportOptions(String)	Gets transport options near the airport in JSON format.

FlightDetailsController	
This class is the controller associated with FlightInfoActivity and TripActivity, for obtaining flight details from server.	
Methods	
public FlightInfo getFlightInfo(String)	Gets flight details from server side for flight no

PlacesController	
This class is the controller for PlacesActivity class, for getting the information of nearby places to see from server side.	
Methods	
public List<String> getPlaces(String)	Gets places to see in the travelled city in string format

CurrencyController	
This class the controller class for getting currency rates from the server, related with CurrencyActivity page.	
Methods	
public String getCurrency(String)	Gets currency information in String format

ShoppingController	
This class is the controller associated with ShoppingActivity class, for getting shopping information from server.	
Methods	
public List<String> getShops(String)	Gets shops near airport for entered criteria or item

RestaurantController	
This class is the controller associated with RestaurantActivity class, for getting shopping information from server.	
Methods	
public List<String> getRestaurants(String)	Gets restaurants in the city for entered criteria

AccommodationController	
This class is the controller associated with AccommodationActivity class, for getting accommodation information from server.	
Methods	
public String getAccomodaiton(String)	Gets accommodation details for entered criteria

LivingExpensesController	
This class is the controller associated with LivingExpensesActivity class, for getting living expenses data from server.	
Methods	
public JSONObject getLivingExpense(String)	Gets living expenses for entered city from server

3.2. Server

The subsystems in this section are dedicated for the Server. Listed class names and function names are subject to change throughout the development lifecycle of the project.

3.2.1. Logic Tier

class FireBaseManager	
This class is responsible for managing request/response tuples with Firebase. Firebase will be required for storing and manipulating user data	
Attributes	
private UUID uuid	
private String email	
private String publicKey	
private String privateToken	
private Date registerDate	
private String firstName	
private String lastName	
Methods	
public tryLogin(String username, String password) : boolean	This function sends a login request to Firebase. If the credentials provided for the login are correct, the login is successful.
public registerUser(String username, String email, String password) : boolean	This function sends user credentials to Firebase for a new user creation. If the credentials have no problems, (matching id, short password) a new user gets created.
public getUserSet(UUID… uuidList) : Set<String>	This function returns a set of users with the specified ids.
public getUser(String userName) : String	This function returns the details for a single user
public findUserByPublicKey(UUID key) : String	This function returns the details for a single user using their public key
public findUserByToken(String token) : String	This function returns the details for a single user using their token
public toString() :String	This function wraps the class into a string and returns it
getters and setters for attributes	Required to change or retrieve information during runtime

class AccountManager	
This class is responsible for managing account credentials such as tokens and access protocols	
Attributes	
private String facebookAccessToken	
private String googleAccessToken	
Methods	
public setFacebookToken(String clientID, String apiKey) : boolean	Sets the token received fro Facebook login to be the current token
public setGoogleToken(String clientID, String apiKey)	Sets the token received fro Google login to be the current token
public getFacebookAccessToken() : String	Gets the Facebook token for operations which require Facebook authorization
public get GoogleAccessToken() : String	Gets the Google token for operations which require Facebook authorization

class ReviewHandler	
This class is responsible for handling user reviews. The users may be able to write reviews to visited places.	
Methods	
public getPlaceReviews(Venue venue) : List<Review>	Gathers the reviews for the specified place from Foursqaure
public getUsersReview(User user) : List<TripLocation>	Gathers the reviews written by the specified LodeStar user
public addReview(Venue venue)	Adds a review written by a LodeStar user to the specified venue
public deleteReview(Venue venue, UUID reviewID)	Deletes a review written by a LodeStar user to the specified venue

class UserDataManager	
This class is responsible for managing and manipulating user data.	
Methods	
public getUserTripLogs(User user) : List<TripLog>	Returns the Trip Logs (kind of a tweet in LodeStar) of a specified LodeStar user
public getUserHistory(User user) : List<TripLocation>	Returns the trip history of a specified LodeStar user
public getUserPreferences(User user) : JSONObject	Returns the preferences of a specified LodeStar user
public getUserFavorites(User user) : List<Favorite>	Returns the favorite places of a specified LodeStar user

class UserDataManager	
public getUserReviews(User user) : List<Review>	Returns the list of reviews of a specified LodeStar user
public toString() : String	This function wraps the class into a string and returns it
getters and setters for attributes	Required to change or retrieve information during runtime

class TripManager	
This class is responsible for getting data for the trip. This service starts by exploring the available services for the specified airport.	
Attributes	
private String tripOrigin	
private String tripDestination	
private Date tripDate	
private TripData otherData	
Methods	
public gatherAvailableServices(String airportCode) : List<Service>	Explores the services offered by the airport. For example, some airports offer premium lounge services while others don't.
public switchCities() : TripData	Switches cities to explore a different airport in a different city
public startInfoActivity(Controller view)	Starts the LodeStar service. (Currency, Weather, FlightInfo)
public toString() : String	This function wraps the class into a string and returns it
getters and setters for attributes	Required to change or retrieve information during runtime

class DockerEngine	
This class is responsible for managing the docker container. It will start and stop Docker Images	
Attributes	
private DockerImage LodeStarVRImagery	
private DockerImage LodeStarWeather	
private DockerImage LodeStarVenueExplorer	
private DockerImage LodeStarFlightStatus	
private DockerImage LodeStarLivingExpenses	
Methods	
public startImage(DockerImage image)	Starts the specified DockerImage
public stopImage(DockerImage image)	Stops the specified DockerImage
public startImages()	Starts all DockerImages in the container
public stopImages()	Stops all DockerImages in the container
public getRunningInstances() : List<DockerImage>	Returns a list of all running instances in the Docker container

class LodeStarVenueExplorer	
This class is responsible for retrieving a list of venues near the specified location	
Attributes	
private int availableVenues	
Methods	
public getNearbyVenues(String category, float longitude, float latitude) : Venue	Retrieves a list of venues near the specified coordinates
public getReview(Venue venue) : Review	Retrieves a list of reviews of the venue
public getStreetAddress(Venue venue) : String	Retrieves the street address of the venue
public getVenueCoordinates(Venue venue) : Coordinate	Retrieves the location of the venue

class LodeStarWeather	
This class is responsible for retrieving weather information for the specified city on the specified date	
Attributes	
private String units	
Methods	
public getWeatherForecast(String city, String units) : List<JSONObject>	Retrieves the weather information from OpenWeatherMapAPI.

class LodeStarVRImagery	
This class is responsible for getting VR images. The direction and field of view is required for viewing 360 images in virtual reality mode	
Attributes	
private int imageQuality	
private double direction	
private double fieldOfView	
Methods	
public getStreetViewImage(int quality, float longitude, float latitude) : Image	Downloads the StreetView image to LodeStar server

class LodeStarCurrency	
This class is responsible for retrieving currency information and returning it to the client side	
Attributes	
private String localCurrencyCode	
private String foreignCurrencyCode	
private double localOverForeignCurrency	
private double foreignOverLocalCurrency	
Methods	
public getCurrencyInfo(String foreignCurrencyCode, String localCurrencyCode) : JSONObject	Retrieves detailed current information for the specified currencies

class LodeStarFlightStatus	
This class is responsible for getting any flight information that could be gathered by only using the flight number	
Attributes	
private String originAirportCode	
private String destinationAirportCode	
private String flightNumber	
Methods	
public getFlightDetails(String originAirportCode, String destinationAirportCode) : JSONObject	Retrieves flight information from FlightAware API.

class LodeStarLivingExpenses	
This class is responsible for getting approximate living expense cost in a city from the Number statistics API.	
Attributes	
private double localMeal	
private double bigMac	
private double oneLiterCoke	
private double averageHotel	
private Map<String, Double> others	
Methods	
public getLivingExpenses(String city) : JSONObject	Returns a list of living expenses. The list consists of widely known items such as a Big Mac Meal from McDonalds.

class LodeStarTransportationInformation	
This class is responsible for getting available transport options for the airport's location	
Attributes	
private Coordinate origin	
private Coordinate destination	
Methods	
public getAvailableTransportOptions(Coordinate origin, Coordinate destination) : JSONObject	Returns all transport options such as taxi, public transport, train etc.

3.2.2. Data Tier

class User
This class is used to store user information
Attributes
private UUID uuid
private String username
private String email
private String registrationDate
private firstName
private lastName

class Review
This class is used to store user reviews. Users may be able to review places they visited in trips.
Attributes
private UUID from
private String content
private int stars
private Date date

class TripLog
This class is used to store trip logs. These are short summaries from user's trips. Just like tweets
Attributes
private UUID uuid
private String content
private Date date

Preferences
This class is used to store user preferences. This class contains many attributes and are subject to change. Assume that context refers to the preferences that the user will be able to change.
Attributes
private Preference context

class Favorite
This class is used to store user favorites. Places, trips, etc.
Attributes
private UUID id
private String content
private Date date

class Trip
This class is used to store all related trip information.
Attributes
private UUID id
private String tripOrigin
private String tripDestination
private Date tripDate
private TripData otherData

class Context
This class is used to store other Trip related data that may be added later during project development cycle
Attributes
private JSONObject context

class LodeStarWeather
This class is used store weather information for the specified city on the specified date
Attributes
private Date date
private String city

class LodeStarFlightStatus
This class is used to store any flight information that could be gathered by only using the flight number
Attributes
private String originAirportCode
private String destinationAirportCode
private String flightNumber

class LodeStarTransportationInformation
This class is used to store transportation information. This information is gathered from the Google Maps API. Can list options for public transport, train, walking and taxi.
Attributes
private Coordinate origin
private Coordinate destination

class LodeStarLivingExpenses
This class is used to store living expenses for the specified city. Data is required for the living expenses page.
Attributes
private double localMeal
private double bigMac
private double oneLiterCoke
private double averageHotel
private Map<String, Double> others

class LodeStarVRImagery
This class is used to store VR images. The direction and field of view is required for viewing 360 images in virtual reality mode
Attributes
private int imageQuality
private double direction
private double fieldOfView

class LodeStarCurrency
This class is used to store currency values for different currencies used in different countries
Attributes
private String localCurrencyCode
private String foreignCurrencyCode
private double localOverForeignCurrency
private double foreignOverLocalCurrency

4. Glossary

Activity: In android an activity is a entry point for a user's interaction with the application.
Recycler View: Recycler View is "a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views"[1]

Our server runs on a Host OS, which runs on Ubuntu. Above this layer, we are planning to integrate 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 are planning to have a container/s depending on the traffic to run our server side applications which are in Node JS. The server will still be similar in terms of design. *

*The code presented in this report is subject to change. As new technologies are being developed, the team may adopt a newer technology which could be beneficial for the future development of this project. Moreover, the team may change the some of the code design if any obstacles are encountered during development.

5. References

[1] Android Developers. Activities. Android Developers. [Online]. developer.android.com/guide/components/activities/index.html. [Accessed: 10-Feb-2018].

[2] Fant J.S., Pettit R.G. (2008) Cost-Performance Tradeoff for Embedded Systems. In: Brinkschulte U., Givargis T., Russo S. (eds) Software Technologies for Embedded and Ubiquitous Systems. SEUS 2008. Lecture Notes in Computer Science, vol 5287. Springer, Berlin, Heidelberg

[3] Parcell J. 2013. Functionality and Usability Testing Resources. [Online]. Available: https://www.digitalgov.gov/2013/05/25/functionality-and-usability-testing-resources. [Accessed: 9-Feb-2018].

[4] Rasmussen, G.T. 2005. Implementing Information Security: Risks vs Cost. [Online]. Available: http://www.gideonrasmussen.com/article-07.html. [Accessed: 9-Feb-2018].

[5] Suh, E. 2017. Space-Time Tradeoffs and Efficiency. [Online]. Available: https://www.cprogramming.com/tutorial/computersciencetheory/space-time-tradeoff.html. [Accessed: 9-Feb-2018].

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

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

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

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

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

[11] 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]

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

[13] CrackerJack, "High Level Design Report," *LodeStar*. [Online]. Available: http://lodestarapp.com/files/evie.pdf. [Accessed: 12-Feb-2018].

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

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

[16] "Food, Nightlife, Entertainment," FourSquare. [Online]. Available: https://foursquare.com. [Accessed: 05-Nov-2017].

[17]Cost of Living. [Online]. Available: https://www.numbeo.com/cost-of-living/. [Accessed: 22-Dec-2017].
[16] 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].
[17] Cost of Living. [Online]. Available: https://www.numbeo.com/cost-of-living/. [Accessed: 22-Dec-2017].
[18] OpenWeatherMap.org. Current weather and forecast. openweathermap. [Online]. Available: https://openweathermap.org/. [Accessed: 22-Dec-2017].
[19]"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].

[This page is intentionally left blank]

[end of report]