Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: LodeStar

Low Level Design Report - KRAKEN

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 intercation with the application [7].

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 360o 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 360o 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 optimize a certain feature of a program, usually another has to be sacrificed. Almost all decisions with respect to design come with certain trade-offs and and implications [1]. 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 [2]. 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 to functionality.

1.1.2. Security vs. Cost

Security is of paramount importance for an application which guides people through international airports. We recognised that in the near past, many security issues resulted in global problems ranging from fraud to terrorism [3]. 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 [4]. 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 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 [5]. 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[6]. Again, this is a commonly used method and the one preferred in Bilkent University.

1.3. Use of New Tools and Technologies

Firebase:
NodeJS:
Docker:
Google VR:
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:

1.4. 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
Name of attribute
Type of attribute

Methods
Name of method
Parameters of method
Return value of method

2. Packages

In our project, we used a novel approach to connect both the client and the server. Client presents the system information to users. It also recieves 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 proceesing and API usage.
G2)client ve server'ın sonunda ikisini bağlarken nasıl novel bir solution kullandığımızı göstericez.

2.1. Client

The client side of our system consists of the mobile applications that will be running on iOS and Android devices. The purpose of client side of the system is to present the system information to the users and send user interactions to server to keep the user interface running. The client side of the system consist of two packages, View and Controller packages.
The client side will enable user to authenticate to the system via establishing Firebase connection. After getting user information, client will navigate to the Home page where user will enter their travel information. When sending or retrieving information from server, the Controller package comes into function so that trip information such as flight details or places to see in the travelled location will be taken by the client side. The View package will consist of classes that will display these information to users by creating the interfaces.

What we mean by client side is the user end of the application program. By the mobile application that will be developed for iOS and Android, the user will interact with the system. When user enters 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 design pattern. Controller subsystem will be responsible for the connection between client and server, when data will be 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.

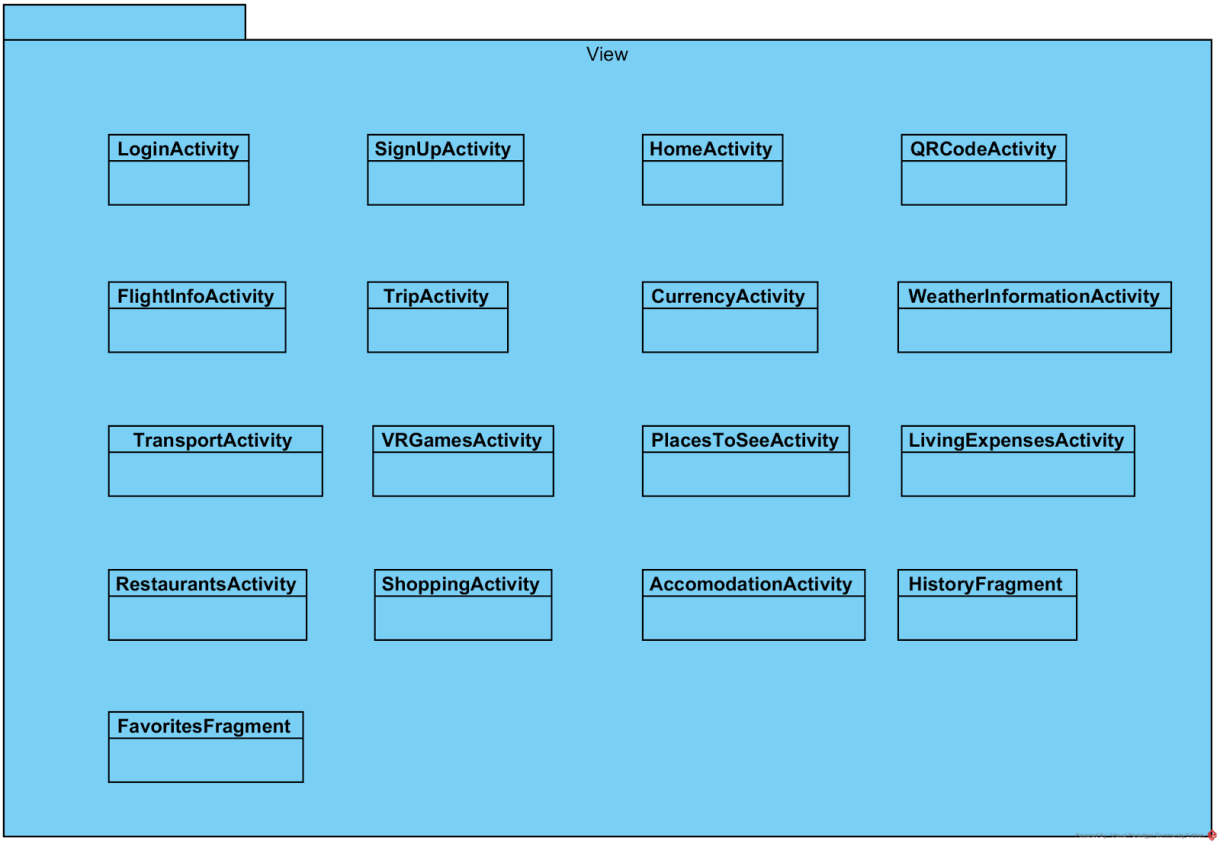


Figure 1 - Subsystem Decomposition

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.

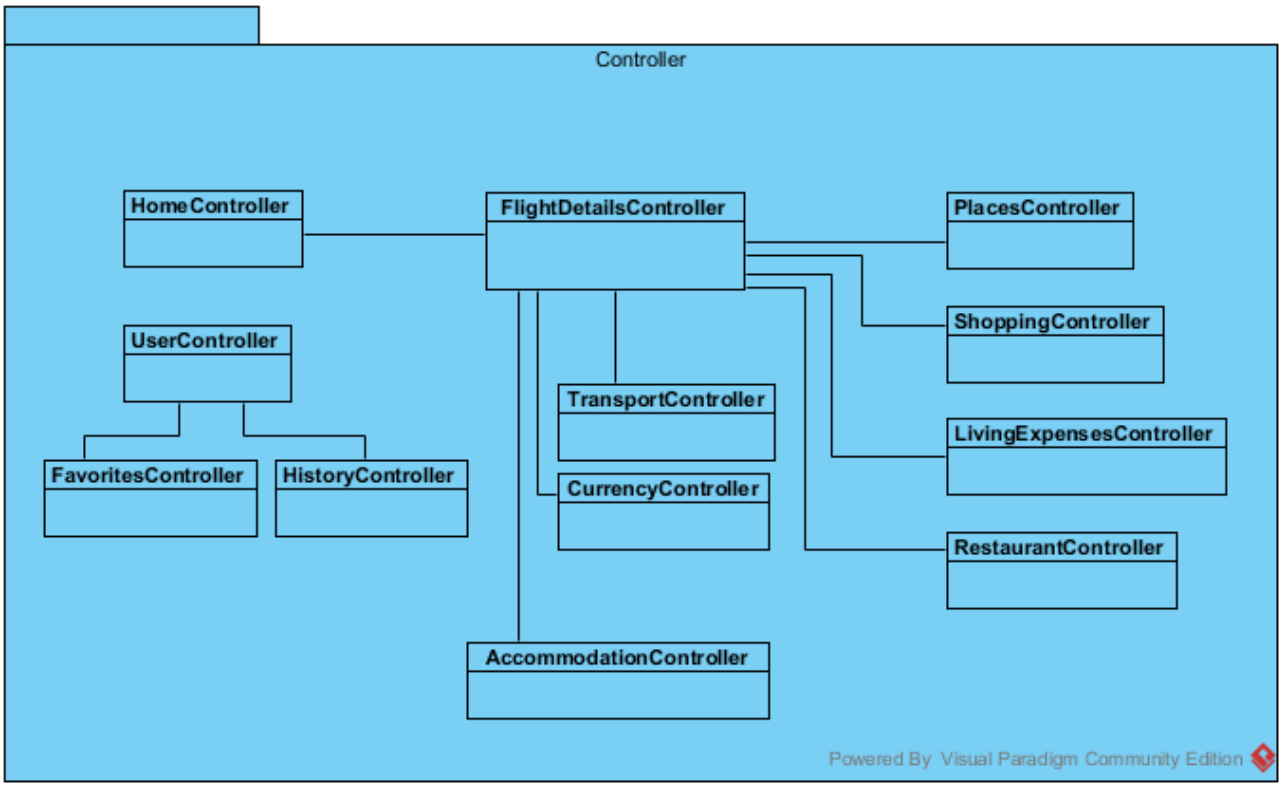
HistoryFragment: This class is responsible for displaying user history of user's trips.

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

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 class to get information for them from server side.



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.

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

Buraya açıklama gerekli sanırım.

3.1.1. View

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

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 resposible from ensuring a secure logout from the system for a user. After the logout is succesful, the account of the user becomes inaccesible 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 from 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 Properties	

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	
AÇIKLAMA NERDE	
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	
Getters for Properties	

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 selects 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 properties

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)	AÇIKLAMA
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)	AÇIKLAMA
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. Daha iyi açıklama yaz. Grr

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.

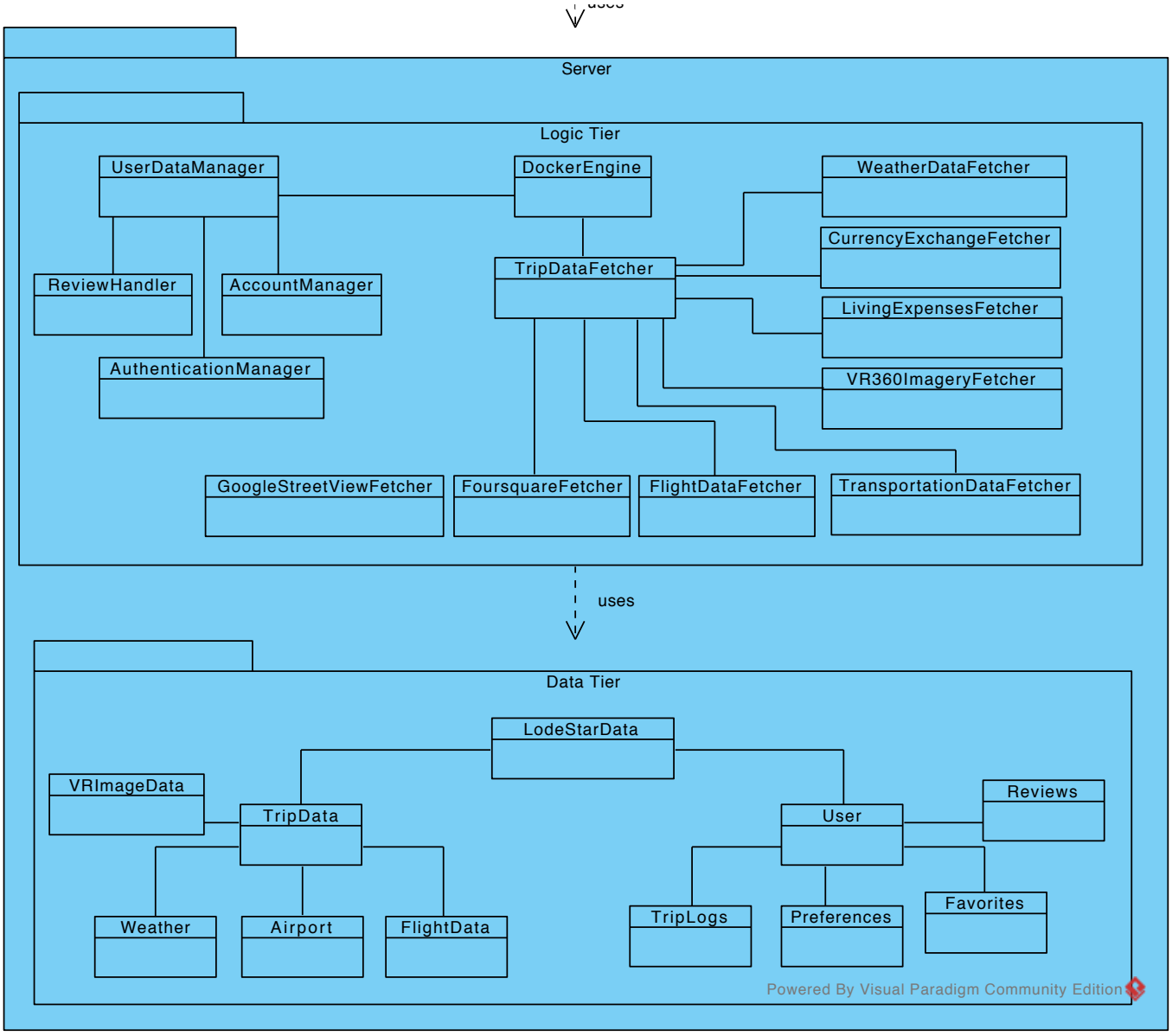
FavoritesController	
This class is for getting the user history from server for logged in user in the client, associated with FavoritesFragment class.	
Methods	
public List<String> getFavorites(int)	Gets user favorite details from server side in String format.

PLACES CONTROLLERDA KALDIN!!!!

3.2.Server

Buraya açıklama gerekli sanırım. Aşağıya yazdın oradan al!

3.3.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. The server basically contains all data of the users. Adding on, the server is responsible

for API interactions to collect and process data from different platforms such as FlightAware, Google, Foursquare, Numbeo and OpenWeather. The server gathers this data on demand.

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.

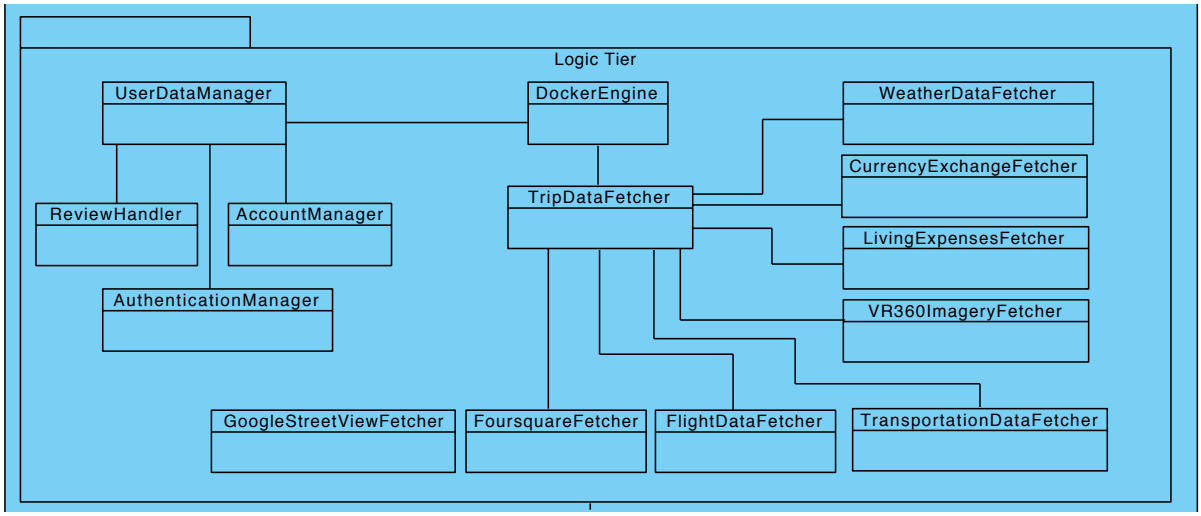
Since LodeStar will be relying on web micro-services, the server is a very crucial part of the project. The server will play a major role starting with just a simple login from a user. Almost every action taken by the user will require a communication with the server. The server will contain usernames 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.

As stated, 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.

3.3.1.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.

UserDataManager: is responsible for managing user preferences. 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 wants to change app preferences.

AuthenticationManager: Almost every operation will require authentication. The Data Tier will be keeping all user data encrypted with a user password. This class will be responsible for carrying off data interactions with the lower layer.

TripDataFetcher: 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.

GoogleStreetViewFetcher: 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.

FoursquareFetcher: 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.

FlightDataFetcher: 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.

TransportationDataFetcher: 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.

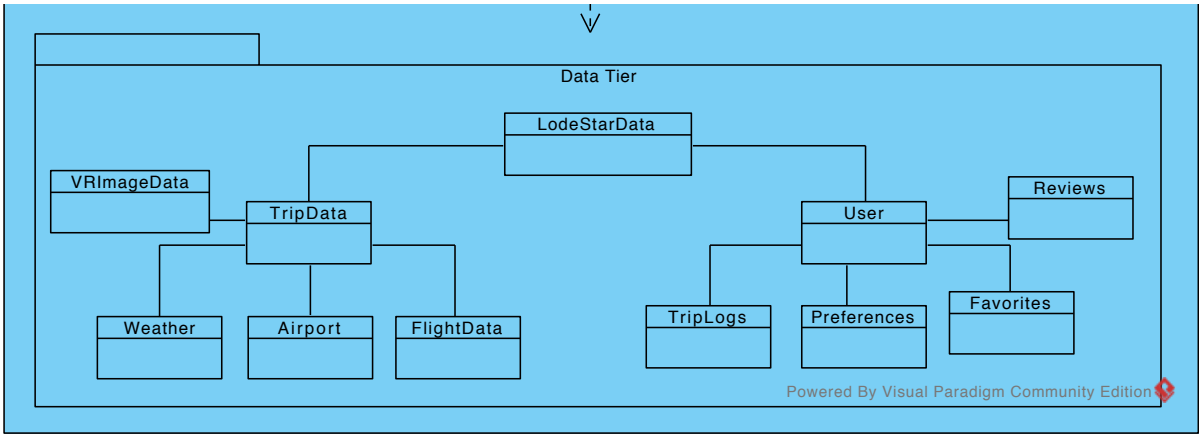
VR360ImageryFetcher: When the user wants to see the nearby locations in VR, 360 images for the place will be needed. This class will send requests to Google 360 StreetView to get available images for the location.

LivingExpensesFetcher: This class will gather living expense costs. This data will be provided by Numbeo[9]

CurrencyExchangeFetcher: when the user wants to see exchange rates, this will send requests to OpenExchangeRate[10] API and retrieve currency exchange information

WeatherDataFetcher: this class will send requests to OpenWeather APIs to get weather data for specified city. [11]

3.3.2.Data Tier



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

LodeStarData: this class manages TripData and User classes
TripData: 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
VRImageData: This class will manage cached 360 images for airports.
Weather: this class will manage cached weather information for previously requested cities.
Airport: 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.
FlightData: 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
TripLogs: 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
Favorites: 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
Reviews: 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

4. Glossary

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"[12]. 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.

Activity: In android an activity is a entry point for a user's intercation with the application [7].
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"[8]

5. References

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

[2] "Instant Street View," Google Street View. [Online]. Available: https://www.instantstreetview.com. [Accessed: 05-Nov-2017].

[3] "Maps and Driving Information," Google Maps. [Online]. Available: https://www.google.com.tr/maps/@39.9033766,32.7627648,11z?hl=en [Accessed: 05-Nov-2017].

[4] "Top Things To Do Near Me," TripAdvisor. [Online]. Available: https://www.tripadvisor.com/Attractions. [Accessed: 05-Nov-2017].

[5] "Yelp," Yelp. [Online]. Available: https://www.yelp.com. [Accessed: 05-Nov-2017].

[6] "National and Local Weather Radar, Daily Forecast," Weather.com. [Online]. Available: https://weather.com/en-GB/. [Accessed: 05-Nov-2017].

[7] "Google Cardboard," Google Cardboard – Google VR. [Online]. Available: https://vr.google.com/cardboard/. [Accessed: 05-Nov-2017].

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

[9]"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].

[10]Cost of Living. [Online]. Available: https://www.numbeo.com/cost-of-living/. [Accessed: 22-Dec-2017].

[11]OpenWeatherMap.org, "Current weather and forecast," openweathermap. [Online]. Available: https://openweathermap.org/. [Accessed: 22-Dec-2017].

[12] "What is a Container," Docker, 11-Dec-2017. [Online]. Available: https://www.docker.com/what-container. [Accessed: 22-Dec-2017].

[13] "What is component? - Definition from WhatIs.com," WhatIs.com. [Online]. Available: http://whatis.techtarget.com/definition/component. [Accessed: 22-Dec-2017].

[14] "LodeStar," *VR View - gallery example*. [Online]. Available: http://lodestarapp.com/. [Accessed: 22-Dec-2017].

[1] 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

[2] 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].
[3] Rasmussen, G.T. 2005. Implementing Information Security: Risks vs Cost. [Online]. Available: http://www.gideonrasmussen.com/article-07.html. [Accessed: 9-Feb-2018].
[4] Suh, E. 2017. Space-Time Tradeoffs and Efficiency. [Online]. Available: https://www.cprogramming.com/tutorial/computersciencetheory/space-time-tradeoff.html. [Accessed: 9-Feb-2018].
[5] IBM, "UML - Basics," June 2003. [Online]. Available: http://www.ibm.com/developerworks/rational/library/769.html.
[6] IEEE, "IEEE Citation Reference," September 2009. [Online]. Available: https://m.ieee.org/documents/ieeecitationref.pdf. [Accessed 9-Feb-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]
[7] Android Developers "Activities." Android Developers, [Acessed 10 Feb 2018] developer.android.com/guide/components/activities/index.html.
[8] Android Developers "Creating List and Cards." Android Developers, [Acessed 10 Feb 2018] https://developer.android.com/training/material/lists-cards.html.