Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: LodeStar

High 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

Dec 22, 2017

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

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 architecture and design of LodeStar. In the next section, the purpose of the system and the design goals will be explained. Following that, architectural information about existing technologies will be given as well as the difference LodeStar will make in contrast to these ones. Afterwards, a detailed analysis of the structure of LodeStar is presented.

1.1. Purpose of the System

When traveling by plane, especially internationally, customers must go to the airports early in order to check-in, register their luggage, go through customs etc. Since any or all of these things could potentially demand a lot of time due to queues or poor management, most people go to the airport very early, and have to wait a long time before boarding the plane. With LodeStar, we aim to help people value their time which would otherwise be wasted at the gates of airports by giving them a glance of the destination airport and city. This should also help alleviate some stress as the passengers will have a clearer idea of what to expect upon arrival.
Some of LodeStar's features are available in other apps. Many, many other apps... One must use Foursquare to check the places of interest, TripAdvisor to mark a route, Google Maps to follow, a weather app to check the forecast, a currency app to see the current transaction rates, and search through many webpages or blogs to find whether public transport or taxi should be their preferred choice.

LodeStar uniquely combines all of these features, and then adds some. With the API's of the mentioned applications, which are free and ready for use, LodeStar merges their capabilities. To build upon that, LodeStar shows how to navigate through the destination airport, where to get a SIM card, rent a car, buy a train ticket or find a taxi and even suggests which one to pick depending on your time and money constraints. All of these functionalities are offered in one app, all from the comfort (or discomfort) of your seat at the gate.
In addition to airports, these features can also be used in favour of tourism companies. When customers who want to make an international trip and go to the offices of such companies, they can be shown, in virtual reality, the cities of their interest. The places to visit, major attractions, or staff-picked videos can be showcased. This will allow the customers to make a better decision for themselves, and increase the likelihood of them booking a ticket.

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

1.2.Design Goals

In this section, LodeStar's design goals will be presented and important user experience aspects will be discussed.

1.2.1.Reliability

- The application should be updated frequently in order to not show any misinformation about the airports or tourist attractions.
- The application should collect real time data regarding the flights such as delays or cancellations.
- The application should precisely display the routes to targets chosen by the user in images, and if available, in VR view.
- The application should have information about real time traffic to keep travel times accurate via taxi or bus.
- The application should give recommendations specific to the user.
- The application should provide accurate transportation recommendations, considering both the allocated budget and time.

1.2.2.Usability

- The application should be easy to use for all types of users and provide a user-friendly and a easy-to-use interface.
- The application should provide necessary information on how to put on and use Google Cardboard for VR view.
- The application should show as much information as it can by taking minimum information from the user.
- The application should give recommendations according to the user's needs and specifications.
- The application should display the comments and ratings of airport or tourist attractions clearly.

1.2.3.Efficiency

- The application should respond to user and gather travel information quickly while using the least possible amount of system resources and internet bandwidth as possible.
- The app should not take a lot of time while switching to Virtual Reality mode.
- The application should provide navigation through the application with as few steps as possible.
- The application should calculate taxi or bus routes quickly.

1.2.4.Security

- The application will not distribute personal data to third parties.
- A user's credentials will be encrypted in our database. Therefore, nobody will have access to it but the given information will still be able to be utilized.

1.2.5.Maintainability

- To increase maintainability, Lodestar will use a relational database to keep track of venue, time, location and personal data information.
- Every user will be provided with information that is related to their needs. This will increase maintainability since each user will be given information specific to their needs rather than all the information that Lodestar can provide.

1.2.6.Extensibility

- The application should be able to include new features with ease, so it should be developed keeping this in mind.
- The system should be open to future additions, such as integration of bus terminals and train stations.
- The system should be available for web integration in the future.
- The system should be able to be used with VR glasses other than Google Cardboard for a more satisfying experience.

1.2.7.Portability

- The application should be able open to use on different hardware and software platforms, in our case on both Android and iOS devices.

1.2.8.Supportability

- The application will be open to future updates. For instance content creation by users might be considered a possible future update.

1.3.Definitions, Acronyms and Abbreviations

TCP: Transmission Control Protocol
UDP: User Datagram Protocol
VR: Virtual Reality
API: Application Programming Interface
HTTP: Hypertext Transfer Protocol
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.

1.4.Overview

There are many ways one can utilize LodeStar. Think of a scenario where a user has a flight to catch up. Using the features of Google Maps, LodeStar will take the user from his/her home to the airport by showing detailed directions. At the airport, user will be able to scan his/her boarding pass to get information about the flight. While waiting in the terminal or lounge room, the user will be given the option to travel the destination city and airport at the meantime. This travel will optionally be in virtual reality environment. To spend time, the user will be able to play random mini-games in virtual reality environment. During flight, the user will able to gather information about the city s/he is flying over. After the landing, the user will be able to get recommendations from LodeStar about possible ways of transportation. These recommendations will have a budgeting feature in which the user will be given the option to specify a budget. By using this

budget as a parameter, LodeStar Transportation Recommendation Manager will recommend the least cost path to a specified destination by considering the transportation options.

These specified things are handled by server which lies under the Docker Engine. Since we use Docker Engine, each of these requests are handled by separate applications. In the glossary part, how this is done is explained briefly.

2. Current Software Architecture

There are already a few applications available for LodeStar to fetch data from. Below, you can find such applications, their details, and how LodeStar will incorporate their functions to its own.

FourSquare[1]

- shows trending, popular, or interesting places nearby
- allows users to comment on and rate these places
- allows users to check-in to these places

Google Street View[2]

- shows 360º images of many places around the world
- allows for quick travel between nearby places, like walking down a road

Google Maps[3]

- displays roads and streets of many places around world in map form
- calculates distance between two places
- calculates viable routes between two places
- calculates the time required to reach from one place to another depending on means of transport

Trip Advisor[4]

- gives recommendations on travel ideas
- recommends places to eat, spend time at, and visit
- shows major attractions of a city

Yelp[5]

- shows places like cafes and restaurants nearby

- allows users to comment and rate these places
- gives information about the food offered in a specific place

Weather.com[6]

- shows weather conditions in all cities around the world in real time
- shows likelihood of rain, snow etc.
- shows the sunset and sunrise hours

Google CardBoard[7]

- allows a simple VR experience via a smart phone
- unlocks the full potential of VR based applications

FlightAware[8]

- gives information about ongoing flights
- shows departure and arrival points
- lists the duration of flights

Instead of switching between lots of websites and applications to gather simple but distinct information, LodeStar will offer all of them under one roof. Having to keep track of many different things across many different applications can easily become tiresome. Users might also forget the information they receive more quickly. By providing all the information in one common, united application, LodeStar will ensure the user can get what he wants immediately, and easily.

To build upon these listed features of already established applications, LodeStar adds a few more. For instance, guiding a user to the a SIM card store or rent-a-car store is inexistent in any preceding application. Moreover, there aren't any applications which explains the user how to get a ticket for bus or subway upon arrival, and which method to choose according to his/her needs and constraints. With a unified source for vital information, LodeStar will be the application for all who travel.

3. Subsystem Decomposition

LodeStar's main system is formed of many subsystems. These subsystems are vital to the main system's success and must work in harmony among each other. In this section, LodeStar's subsystems will be displayed. The interactions between the subsystems, the classes they consist of and their functions will be discussed.

3.1. Overview

LodeStar's system decomposition aims to present the structures of its systems and subsystems in great detail. Before getting into the diagrams, it is important to realize the importance of a well defined subsystem structure for a project. The subsystem decomposition showcases how each of our chosen subsystems (explained in greater detail in the following sections) interacts with each other. From the diagrams, it is possible to see the responsibilities of each subsystem, which information it receives, sends or calculates, and how the transition between different subsystems takes place. To get these traits of the subsystems correct before starting the implementation translates to fewer errors during coding, and less revision required.

For these reasons, we spent a lot of time and effort identifying the different subsystems that make up LodeStar when combined. We focused our resources into this aspect for we know this is the one with greatest return in the long run.

After realizing the importance of a proper subsystem decomposition, we are ready to dive deeper into the subject.

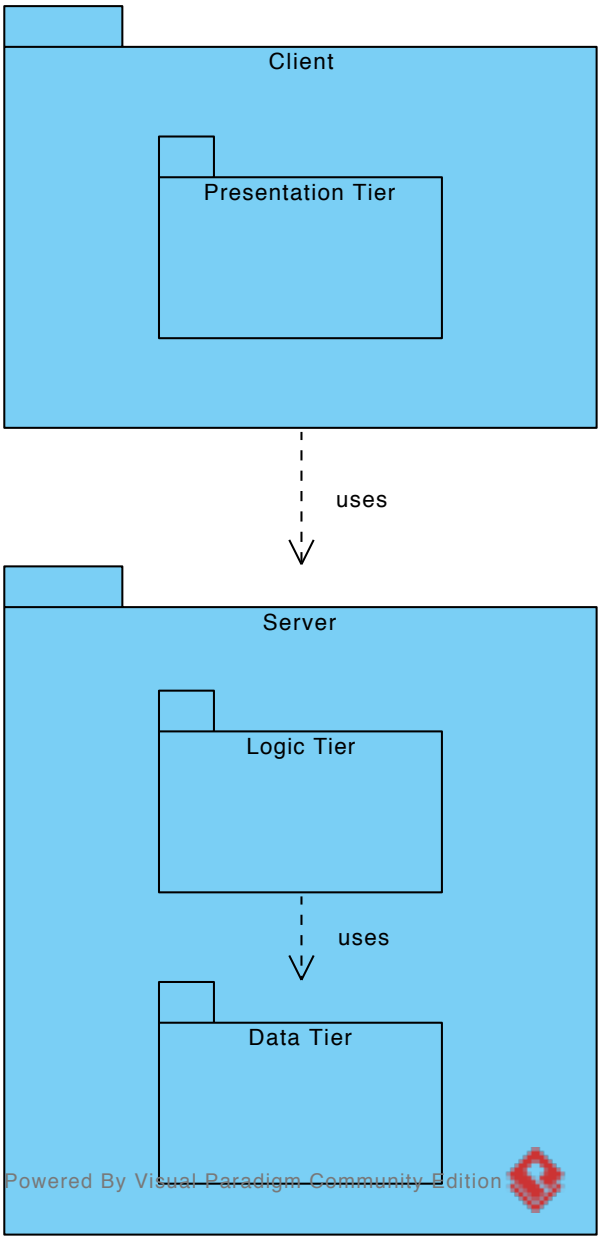

Figure 1 - Subsystem Decomposition

3.2.Subsystem Decomposition

LodeStar's system mimics a Client / Server architecture. With the image we have in mind for LodeStar, we aim to serve many different users simultaneously. When the expected audience gets as wide as it has here, it is important not to sacrifice performance. The Client / Server architecture fits our needs perfectly because it allows us to serve many (hopefully thousands or even millions!) of clients at a fast pace while still offering plenty of useful features.

LodeStar will have a server in which it stores most of its data and perform most of its logic operations. For example, if the user wishes to see the weather in London or see the currency rates of Krakow, the application will request this information from the server. The application itself will do few, if any computations. It will be responsible for fetching the requested information from LodeStar's server, where it will be stored, and presenting it to the user. Since we take most of the load off application's shoulders and hand it to the much more capable server, we aim to offer LodeStar's users a satisfying experience. One that is both fast and reliable.

The Server / Client architecture implemented in LodeStar also inherits the 3-Tier architectural style. What is meant by the 3-Tier architecture is that there will be three tiers named Presentation Tier, Logic Tier and Data Tier.

The Presentation Tier is what the user interacts with in the application. This is the only layer which the user is aware of its existence. For the other two, only the developers will have knowledge. Surely, that does not make them any less important. The Logic Tier does the important calculations like deriving a route from the airport to the hotel or deciding whether a taxi or bus will be the better option for transport. Finally, the Data Tier's responsibility is to manage and maintain LodeStar's database. Any information permanently stored in the database will, in truth, be asked from the Data Tier when requested.

As can be seen from the figure, Client tier uses information gathered from the Logic Tier, which demands information from the Data Tier. Below, you can find all the related classes in greater detail for each of these tiers.

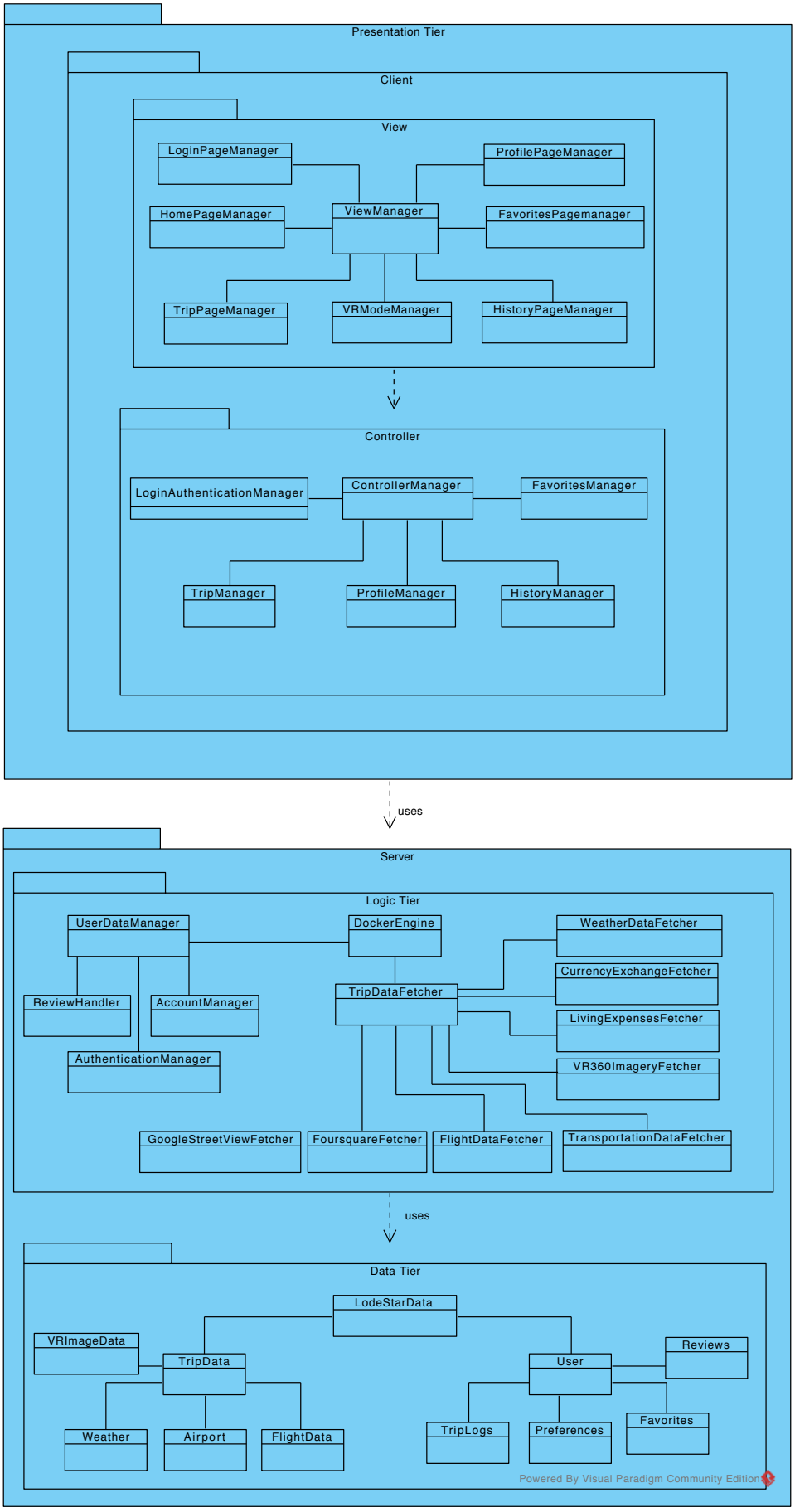

Figure 2 - Subsystem Decomposition Detailed View

As shown in Figure 2, The Presentation Tier is further divided into two packages as View and Controller. The View classes are LoginPageManager, HomePageManager, ProfilePageManager, FavoritesPageManager, HistoryPageManager, TripPageManager and VRModeManager. What combines all these different classes is the ViewManager. This ViewManager class will be one of the most crucial in our quest to present a functional and fast interface for the user to command.

What controls these View classes are shown in the Controller package. The ControllerManager class carries the most weight here, serving as the connection point for all the other classes in the package. These other classes include FavoritesManager, LoginAuthenticationManager, TripManager, ProfileManager and HistoryManager. These Manager classes will make sure everything on LodeStar's user interface (all the classes in the View package) run smoothly and error-free.

Next, we move onto the Server package. Shown in more detail than the previous representation, the diagram displays the components of both Logic Tier and Data Tier.

Logic Tier, as discussed, is responsible from managing all the computational process and logic operations. LodeStar utilizes a DockerEngine class to shoulder much of the burden here. One of its connections is to the UserDataManager class, the class responsible from all things related to the users. These include reviews, user accounts and user authentication.

The other connection of DockerEngine is to the TripDataFetcher. This is a crucial class for LodeStar's Logic Tier. TripDataFetcher grabs the data from all the APIs LodeStar uses, makes the necessary calculations before presenting them to the user (via Presentation Tier).

Last comes the Data Tier. Here, LodeStar stores all its data for use at a later time. LodeStarData is the main class here, serving as a bridge between all user and trip data. User data includes the reviews, preferences, favorites and trip logs of a user. These will all be stored in a database which will make the information easily and quickly accessible. On the left side, it is possible to see TripData class. This class stores information about anything a user might require when going on a trip. Airport and weather information, flight data and images for VR display can all be found here.

The subsystems of LodeStar serve a great purpose. With the smart and clear identification of each tier and package with respect to our architecture, the high level design process approaches completion. The detailed subsystem decomposition is a true necessity of a successful implementation, and LodeStar's subsystem decomposition diagrams offers any aid and guidance required come development stage.

3.3.Hardware & Software Mapping

The clients' hardware will be their iOS or Android phones. Since the app will make use of location services and several API's for currency, flight status, foursquare integration, living expenses, transportation and weather, it requires a Web server. HTTP request/responses will be used to be able to communicate with the server. The server code and API integration will be written in NodeJS. The application itself will be coded in Java for Android & Objective-C for iOS.

There will also be a database to keep essential information (explained in 3.4 with more details). This database will have connections with the server and thus, provide and retrieve the information about the user and his/her trip. The following component diagram[9] shows this.

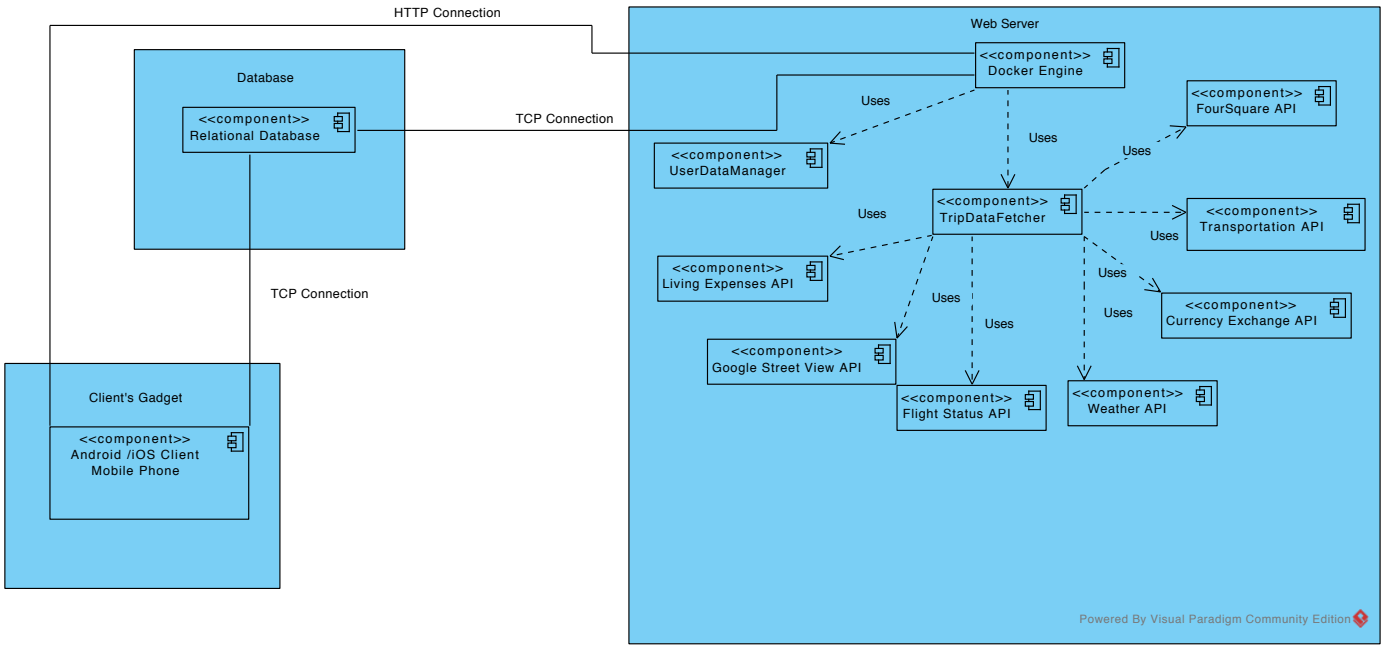

Figure 3 - Component Diagram

3.4.Persistent Data Management

Almost all unique features of our application are targeted to each individuals' needs. For example: proprietary trip planning, optimized route finder, airport navigation using virtual reality etc. Therefore for each user, our application will tailor a specific service. Therefore we will need to take this into consideration when designing our database.

All the essential information such as name, username, password (encrypted), email and phone number information will be kept in our database. Furthermore we also will keep information that is specific to an individual user such as: destination/source airport, destination/source city, accommodation locations. We will make sure that our data does not get lost. This will be done via a backup database.

3.5.Access Control and Security

Each user will be able to edit his/her credentials and will have access to their personal data. The password of users will be kept in the database with encryption. Users will also have access to the previous locations they have been in. However, the personal data of a user can only be accessed by himself/herself. Other users will not be allowed to access/see other users' credentials.

The application requires that a user signs up with a valid email address and a password. For security reasons, an email can only be used for a single user. This is intended to limit the number of accounts a person can create. In the case that the user forgets his/her password, an email will be sent to their respective emails.

User is also able to sign in to the system using either Google or Facebook account. Without signing in, users will not be allowed to the system. In order for our application to use the data of the user from Facebook and/or Google, their permission will be asked.

3.6.Global Software Control

The global software control of our application is mostly dependent on the server. It is based on events, which are mostly based on the user's requests.

The server of our application will work according to the requests of the user. When a user wishes to travel inside an airport or a tourist attraction, the server will use the respective APIs to set up the environment.

Furthermore when a user asks for directions, the server will provide the appropriate data the user requested. If the directions include budget constraints, the background algorithms combined with the server information will provide the user with an appropriate route.

3.7.Boundary Conditions

The application will have several boundary conditions. These include: Starting the application, terminating the application and failure in the application. These conditions are discussed in detail in the sections below.

3.7.1. Starting the Application

Both Android & iOS users need to download the app to their phones. The initial phase of the user should be to sign up. The user will be requested his/her credentials. The user will be given the opportunity to authenticate with Facebook. In the case that the user fails to sign up, the sign up page will be displayed again.

3.7.2.Terminating the Application

The user will be able to logout of the system via the logout button in the Preferences Page. In the case that the user does not logout, the system state will be saved and, if the user wishes to utilize the application again, respective data will be loaded.

3.7.3.Failure in the Application

The application will fail to work the user if there is no internet connection. When the application cannot connect to the internet it will go into offline state. In this mode, only cached data will be presented.

4. Subsystem Services

In this section, we will the describe the major components, i.e. subsystems, of our system. Our main two subsystems are client and server sides of the software which is a traditional method for implementing and governing mobile applications.

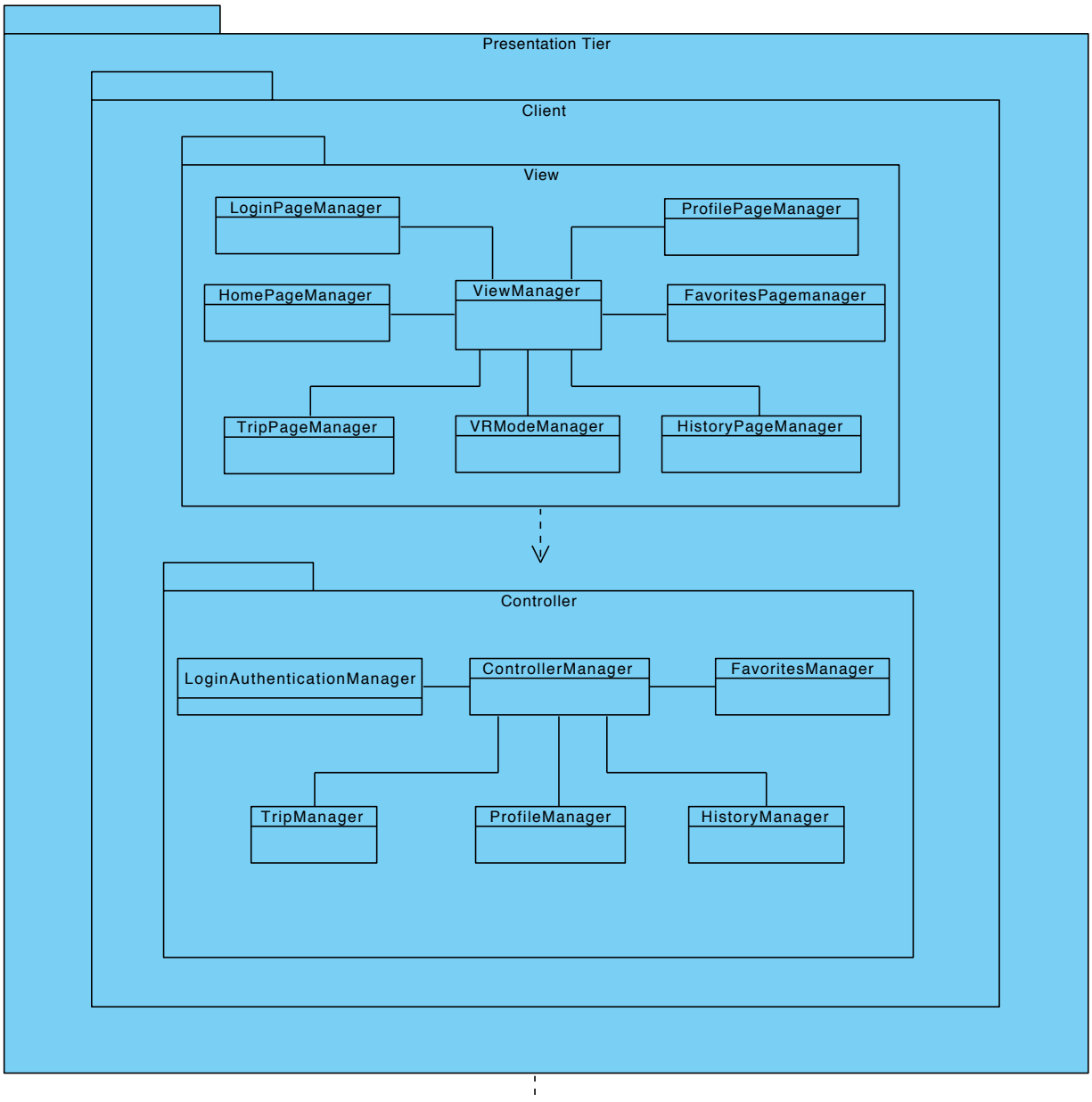

Figure 4 - Detailed View of Client Subsystem

4.1.Client

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.

4.1.1.Controller Subsystem

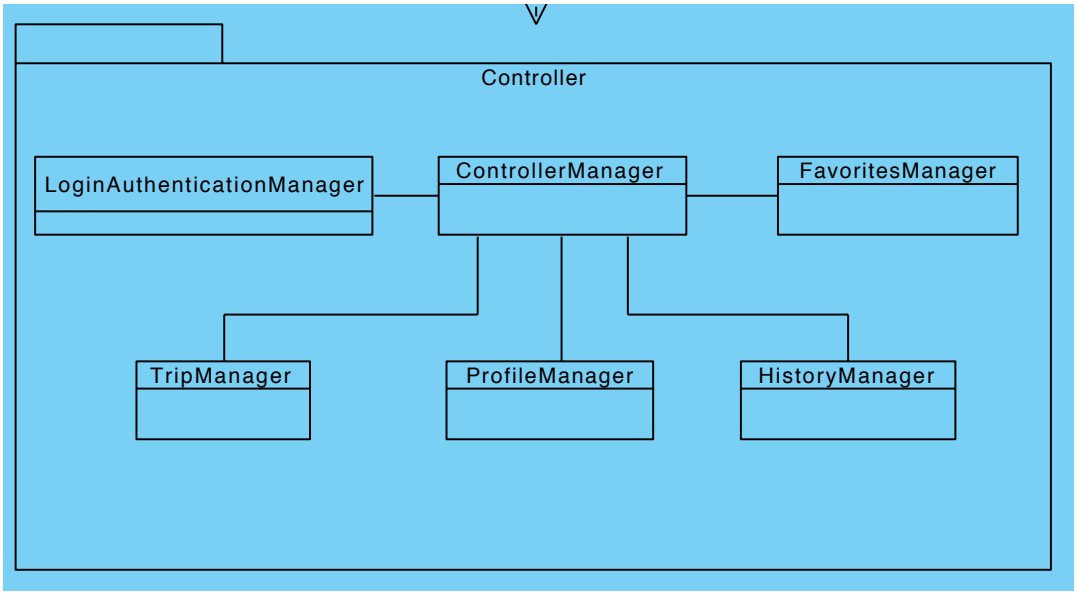

Figure 5 - Controller Subsystem in Client

The controller subsystem will direct the data transfer between client and server side of the system. It will consist of following parts:

ControllerManager: This class will administer other classes to maintain connection to server and keep the information updated.

ConnectionManager: Main information exchange between client and server side will be done by this class.

AuthenticationManager: Receiving user login information and updating the application user information will be done by this class.

TripManager: Receiving and updating the information displayed on trip page, such as "Places to See" or "Transportation Options" will be done by this class.

ProfileManager: User profile information such as name, trips or comments will be done using this class.

HistoryManager: User history will be received and updated using this class.

FavoritesManager: The information exchange for user favorites will be managed by this class.

4.1.2.View Subsystem

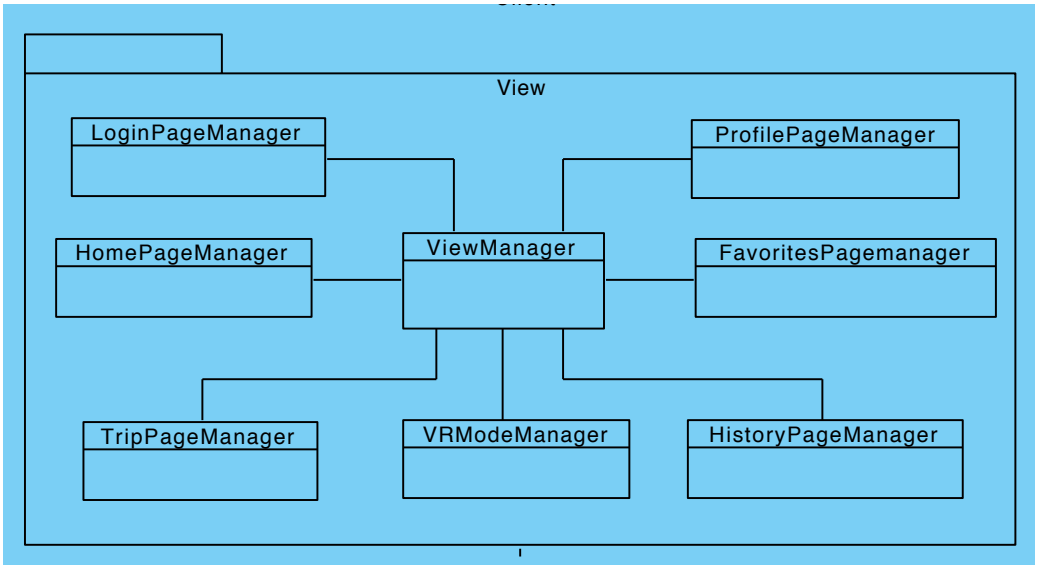

Figure 6 - View Subsystem in Client

The view subsystem will be responsible for managing the user interface operations.

ViewManager: Main class that will direct the operations of other interface classes.

LoginPageManager: Displaying Login/Sign Up Page will be done by this class.

HomePageManager: Displaying home page content, such as destination city selection or scanning QR code, will be done by this class.

TripPageManager: Trip page content and information displayed there, transport options, weather, flight information, shopping, lounge services, restaurants, places to see, living expenses and accommodation, will be displayed by this class.

HistoryPageManager: The page for displaying user history data will use this class.

FavoritesPageManager: User favorites will be displayed by using this class.

ProfilePageManager: Profile page of the content will be displayed by this class.

VRModeManager: When user selects a place to see in VR screen, this class will be used for displaying the VR view of the location.

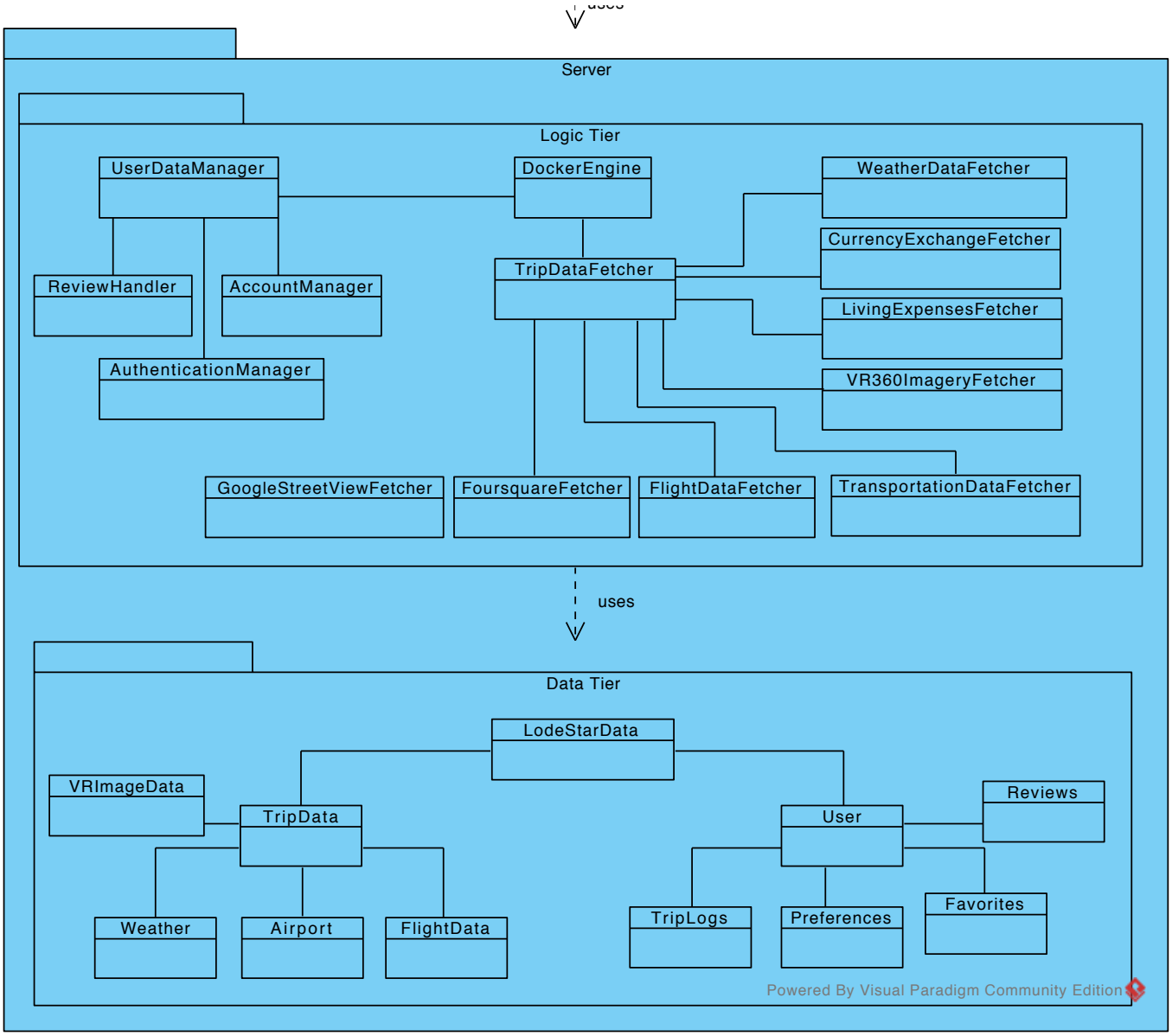

Figure 7 - Detailed View of Server Subsystem

4.2.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.

4.2.1.Logic Tier

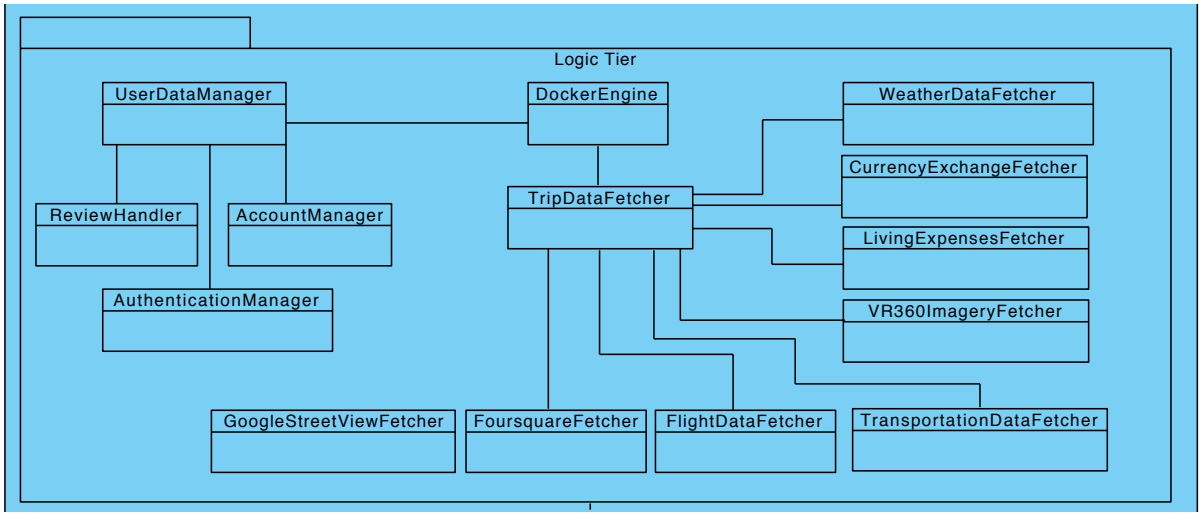

Figure 8 - Logic Tier Subsystem of Server

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]

4.2.2.Data Tier

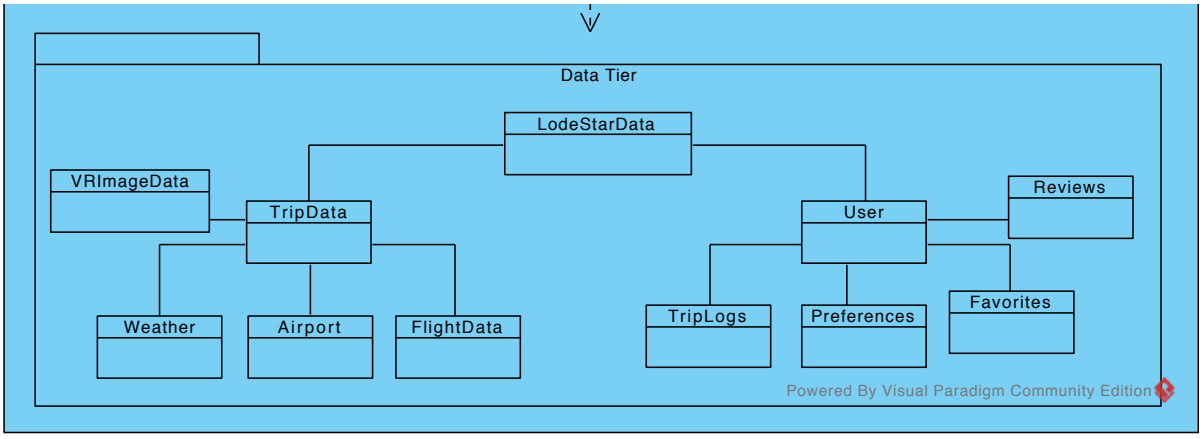

Figure 9 - Data Tier Subsystem of Server

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

5. 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.

6. 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].