Preface

This document contains all five reports for Bilkent Senior Design Project LodeStar. These reports are as follows:

- Project Specification Report
- Analysis Report
- High Level Design Report
- Low Level Design Report
- Final Report

During the development of this project, we have had the opportunity to learn about the newest tools and technologies. We. have been fortunate enough to work with Virtual Reality, which is one of today's hottest developments. Moreover, we have learnt how to work as a team (even under the most stressful deadline days) and built a friendship that we will never forget about. Thank you for giving us the chance to experience this project. We have definitely learnt a lot.

Special thanks to Bülent Özgüç, our project advisor for answering our burning questions even at 05.39AM in the morning. Thanks for keeping us motivated. With each one of or weekly meetings, you have shared many of your past experiences and pointed us towards the right direction. Without your help, we would not have been able to write these reports, draw these mockups and develop project LodeStar.

Also special thanks to our jury members, Uğur Güdükbay and Çiğdem Demir for providing helpful feedback during the development lifecycle.

Sincerely,
Team LodeStar

Scan a flight card and go. This is how LodeStar essentially works. Barış has managed to implement a fast barbode reader to decode Boarding Pass barcodes.

-m “Barcode reading works, weather page updated.” BP

Specifications Report

Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: LodeStar

Project Specifications 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

Oct 9, 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

1. Introduction 1
   1.1. Description 2
   1.2. Constraints 3
      1.2.1. Economic Constraints 3
      1.2.2. Environmental Constraints 3
      1.2.3. Social Constraints 4
      1.2.4. Political Constraints 4
      1.2.5. Ethical Constraints 4
      1.2.6. Health & Safety Constraints 4
      1.2.7. Manufacturing Constraints 5
      1.2.8. Sustainability Constraints 5
      1.2.9. Implementation Constraints 6
   1.3. Professional and Ethical Issues 6
2. Requirements 7
   2.1. Functional Requirements 7
      2.1.1. Data Resources 7
      2.1.2. User Specific Requirements 8
   2.2. Non-Functional Requirements 9
      2.2.1. Reliability 9
      2.2.2. Usability 9
      2.2.3. Accessibility 9
      2.2.4. Extensibility 9
      2.2.5. Portability 10
      2.2.6. Efficiency 10
3. References 11

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 the description of Project LodeStar. Then insight about the constraints surrounding economic, environmental, social, ethical, political, health, safety and sustainability issues regarding LodeStar will be given. Afterwards, the professional and ethical responsibilities LodeStar brings along will be listed. Finally, The functional and non-functional requirements of LodeStar will be provided.

1.1. Description

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.Constraints

1.2.1.Economic Constraints

- Virtual Reality Glasses will be needed for the virtual reality function of the application. The cheapest one (Google Cardboard™ VR platform[1]) costs around 1-2$ on various websites on the internet.
- For Android™ platform , a fee of 25$ needs to be paid so that our application can be published on Google Play™. This payment will be done only once.
- For iOS, a fee of 100$ will need to be paid. This payment has to be done every year so our application can remain publcarished on App Store®.
- All the APIs, frameworks and libraries used will be free. (For example, Google Street View™ mapping service API will be used for virtual reality functionality)
- There will be no in-app purchases. App will be free to download. Business model is described in the 1.2.8 Sustainability Constraints.
- Flight Tracker API[2] will be used. This API is free for use for 50 times a day. For the scope of this project, this number is more than enough.
- Domain of the website of LodeStar will be purchased. This costs 8.88$[3] per year.
- A server is required to contain necessary data. Examples of these data include: personal profile information and guides.

1.2.2.Environmental Constraints

- Our application may use virtual reality glasses (if the user wishes to use Virtual Reality functionality). Google Cardboard's environmental impact[4] is almost nonexistent.
- A smartphone will be required. Which contains a Li-ion battery which may need to be properly disposed[5] once the smartphone is out of order.

1.2.3.Social Constraints

- Our application will enable the users to interact with each other via Facebook integration. For example, while user A is virtually traveling the Atatürk Airport, he will be able to see the comments made by his friend user B about certain places in the virtual environment. This could be exploited where a Facebook user may follow advertisement company accounts.
- The application will not have any sort of mechanism that enables negativism while posting comments or likes.

1.2.4.Political Constraints

- Although the application will be able to show the surroundings of almost all airports around the world in virtual reality, there had been political conflicts with Google and some governments[6]. Thus, some parts inside the airport cannot be photographed due to security concerns.

1.2.5.Ethical Constraints

- We will abide by the Code of Ethics[8].
- The application will not distribute any personal user information to the third parties.
- The user will only be able to access their information after a successful authentication.
- User data on our servers will be encrypted.

1.2.6.Health & Safety Constraints

- Some people will feel a little bit sick after wearing VR glasses. Ten minutes of exposure can result in dizziness[7] for some people. On the flip side, many people don't have any issues.

1.2.7.Manufacturing Constraints

- One manufacturing constraint to fully utilize our application will be the Google Cardboard (if the user wishes to use the VR functionalities). Other than this material, the users will not need additional material and be able to download the application to their phones.
- There are several boarding pass scanning API's that free for use. These API's enable us to retrieve info about the user's flight number, the destination city, flight time and flight captain. Instead of manually typing some of the information to the application, the user will have the convenience of scanning the boarding pass. For this to happen, the user must posses a boarding pass.

1.2.8.Sustainability Constraints

- The application will receive user feedback from AppStore and PlayStore. The team will consider the comments from these source to be very valuable.
- We will also gather anonymous user statistics in order to understand user behavior. These statistics can be used to enhance the user experience.
- Our business model is to use indirect advertisements in our app. For example, when the user wants to learn how to get a SIM card form the airport, we can direct him to Vodafone instead of Turkcell.
- Agreements with airlines companies (i.e. Turkish Airlines[9]) can result in beneficial outcomes for both parties. For example, a tour company may utilize our application to show off the country of destination. A person wishing to travel in Europe can observe certain parts of the cities from our application's VR function and can plan his travel accordingly.

1.2.9.Implementation Constraints

- The application will be for both Android and iOS platforms. Java will be used for Android. Swift and Objective-C will be used for the iOS application.
- For git collaboration GitHub[10] will be used.
- Object Oriented Programming paradigm will be used while in development.
- The application will utilize third party data such as Google Maps and Facebook.
- The application will use third-party data obtained by the APIs provided by Google, Facebook and Linkedin.

1.3.Professional and Ethical Issues

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

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

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

Additionally, our application will be using login API's of Google and Facebook. In this way, we will be able to manage user information easier and safer, but if user does not want to login by their Google or Facebook accounts, our application will allow users to register

using an email and password. For those users, as a future work, a two way authentication system can be implemented.

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

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

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

2. Requirements

2.1. Functional Requirements

2.1.1. Data Resources

- The application will take image data from Google StreetView, flight information from FlightTracker™, maps and road information from Google Maps.
- The application should get travel and flight data by scanning the QR code on the boarding pass.
- The application will provide VR view of destination airport that shows the precise location of airport facilities and important stores for travelers, for example telecommunication shops SIM card purchase.
- Prominent touristic stops and most recommended 20 places from Foursquare at the destination city and, if available, street view of accommodation place will be provided in VR view.
- The application should provide information and prices about different transportation facilities and calculate the optimal routes for different choices.

- The application will gather data for city-wide events, special days and activities.
- The application will show FourSquare comments and recommendations for airport, stores and touristic city locations.
- The application should have up-to-date information about available hours of recommended airport shops and transportation facilities.
- The application should provide actual information about weather conditions of the destination point.
- Currency rates and currency exchange places near airport or staying place will be provided.
- Facebook integration should be able to show users information on their friends' activities on their trips using the application.
- In the case of contracted companies, the application should be able to guide user to those shops in the airport and see the available promotions.

2.1.2.User Specific Requirements

- The user should be able to choose different transportation facilities based on the preferred budget.
- The users will be able to enter their travel information either by scanning the QR code on the boarding pass, or manually entering the destination airport.
- According to their budget, the user will be recommended with different places to visit.
- The user will be seeing the comments made by other users for recommended places and leave comments themselves.
- The users will see the descriptions for reaching their target either on VR view, or on panoramic view of Google Street screen.
- Integrated small games for VR view, like Brick Breaker, will be provided for especially children who will use the application.
- The users will be provided with information on approximate living expenses, and may be able to calculate their own spending during their stay.
- In the future, depending on the contracts between airline companies, users may be able to unlock achievements as they travel more.

2.2.Non-Functional Requirements

2.2.1.Reliability

- The application should precisely display the targets and specify the routes to those targets both in regular view and in VR view.
- To keep the displayed travel information accurate, the application should update the data taken from other services constantly, and present real time information.
- The application should give up-to-date locations for recommended places, if a certain place is closed, recommendations list should be updated.
- The application should give transportation route advice, so application should find and evaluate different alternatives accurately according to the budget.

2.2.2.Usability

- The application should be easy to use for user all types of users and provide a user-friendly and a simple-enough interface.
- Application should provide necessary information on how to put on and use Google Cardboard, for VR view.
- Application should show as much as information it can by taking minimum information from the user.

2.2.3.Accessibility

- Google Cardboards that will be beneficial for the application use should be easy to obtain for airport passengers.

2.2.4.Extensibility

- The application should be able to include new features with ease, so it should be developed in a way that makes it easy to update
- The application should be allow feature additions, such as integration of bus terminals and train stations.

2.2.5.Portability

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

2.2.6.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.

3. References

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

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

[3] "Domain Name Registration Search Results • Namecheap.com," Namecheap. [Online]. Available: https://www.namecheap.com/domains/registration/results.aspx?domain=lodestarapp. [Accessed: 02-Oct-2017].

[4] "Manufacture Cardboard," Google. [Online]. Available: https://vr.google.com/cardboard/manufacturers/. [Accessed: 05-Oct-2017].

[5] "Lithium Ion Batteries," Lithium Ion Batteries - American Disposal. [Online]. Available: https://www.americandisposal.com/blog/lithium-ion-batteries. [Accessed: 04-Oct-2017].

[6] Claire Cain Miller and Kevin J. O'Brien, "Germany's Complicated Relationship With Google Street View," The New York Times, 23-Apr-2013. [Online]. Available: https://bits.blogs.nytimes.com/2013/04/23/germanys-complicated-relationship-with-google-street-view/. [Accessed: 08-Oct-2017].

[7] D. Magyari, Ceo, I. M. M. Y. Inc., and a V. R. headset maker, "Virtual reality: Are health risks being ignored?," CNBC, 08-Jan-2016. [Online]. Available: https://www.cnbc.com/2016/01/08/virtual-reality-are-health-risks-being-ignored-commentary.html. [Accessed: 09-Oct-2017].

[8] "Code of Ethics," Code of Ethics | National Society of Professional Engineers. [Online]. Available: https://www.nspe.org/resources/ethics/code-ethics. [Accessed: 01-Oct-2017].

[9] "Turkish Airlines ® | Flights to 110 countries from İstanbul," Turkish Airlines ® | Flights to 110 countries from İstanbul. [Online]. Available: https://p.turkishairlines.com/. [Accessed: 09-Oct-2017].

[10] "Build software better, together," GitHub. [Online]. Available: https://github.com/. [Accessed: 09-Oct-2017].

It was at this time we were still trying to figure out how we were going to handle user registrations on the database. Then, Efe figured it out.

-m “user registration is DONE and improvements made on db” ES

Analysis Report

Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: LodeStar

Analysis 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

Nov 6, 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

1. Introduction 2
2. Current System 3
3. Proposed System 4
3.1.Overview 5
3.2.Functional Requirements 6
3.2.1.Data Resources 6
3.2.2.User-Specific Requirements 8
3.3.Non-Functional Requirements 9
3.3.1.Reliability 9
3.3.2.Usability 10
3.3.3.Accessibility 10
3.3.4.Extensibility 10
3.3.5.Portability 11
3.3.6.Efficiency 11
3.4.Pseudo Requirements 11
3.5.System Models 11
3.5.1.Scenarios 12
3.5.2.Use Case Model 26
3.5.2.1.Login Use Case 27
3.5.2.2.Create Account Use Case 28
3.5.2.3.Create Account Using Google Use Case 29
3.5.2.4.Create Account Using Facebook Use Case 29
3.5.2.5.Change Account Settings Use Case 30
3.5.2.6.Change Account Privacy Preferences Use Case 31
3.5.2.7.Select the City of Destination Use Case 32
3.5.2.8.Travel the Destination City Use Case 32
3.5.2.9.Select the Destination Airport Use Case 33
3.5.2.10.Travel the Destination Airport Use Case 33
3.5.2.11.Scan Flight Ticket Barcode Use Case 34
3.5.2.12.View Destination Weather Information Use Case 34
3.5.2.13.View Flight Information Use Case 35
3.5.2.14.View Flight Information Use Case 35
3.5.2.15.Manually Enter Flight Information Use Case 36
3.5.2.16.View Transportation Suggestions Use Case 36

3.5.2.17.Flight Time Notification Use Case 37
3.5.2.18.During Flight Virtual Exploration Use Case 37
3.5.2.19.During Flight Virtual Exploration Use Case 38
3.5.2.20.During Flight Reading Exploration Use Case 38
3.5.2.21.View Destination City Videos Use Case 39
3.5.2.22.View City Touristic Spots Use Case 39
3.5.2.23.View City Accommodation Use Case 40
3.5.2.24.View City Events Use Case 40
3.5.2.25.View Comments of Places Use Case 41
3.5.2.26.Enter Comment for Places Use Case 41
3.5.2.27.View Living Expenses Use Case 42
3.5.2.28.View Currency Rates Use Case 43
3.5.3.Object and Class Model 44
3.5.4.Dynamic Models 46
3.5.4.1.Sequence Diagrams 46
3.5.4.2.Activity Diagrams 50
3.5.5.User Interface 53
3.5.5.1.Login Page 53
3.5.5.2.Sign Up Page 55
3.5.5.3. Home Page 56
3.5.5.4.Trip Page 57
3.5.5.5.Weather Information Page 58
3.5.5.6.Flight Information Page 59
3.5.5.7.Currency Rates Page 60
3.5.5.8.Me Page 61
3.5.5.9.Preferences Page 62
4. References 63

List of Figures

I. Figure 1 - Use Case Diagram - Page 26
II. Figure 2 - Object and Class Diagram -Page 44
III. Figure 3 - Sign Up Sequence Diagram - Page 46
IV. Figure 4 - Log In Sequence Diagram - Page 47
V. Figure 5 - Provide SIM Card Information Sequence Diagram - Page 47
VI. Figure 6 - View Transportation Information Sequence Diagram - Page 48
VII. Figure 7- Travel City Sequence Diagram - Page 49
VIII. Figure 8- Scan Flight Ticket Sequence Diagram - Page 49
IX. Figure 9 - Application Flow Activity Diagram - Page 50
X. Figure 10 - Trip Creation Activity Diagram - Page 51
XI. Figure 11 - Login Page UI Mockup - Page 53
XII. Figure 12 - Sign In Page UI Mockup - Page 54
XIII. Figure 13 - Sign Up Page UI Mockup - Page 55
XIV. Figure 14 - Home Page UI Mockup - Page 56
XV. Figure 15- Trip Departure Page UI Mockup - Page 57
XVI. Figure 16 - Trip Arrival Page UI Mockup - Page 57
XVII. Figure 17 - Weather Information Page UI Mockup - Page 58
XVIII. Figure 18 - Flight Information Page UI Mockup - Page 59
XIX. Figure 19 - Currency Rates UI Mockup - Page 60
XX. Figure 20 - Me Page UI Mockup - Page 61
XXI. Figure 21 - Preferences Page UI Mockup - Page 62

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 development strategy of Project LodeStar. Information about the current technologies will be given as well as the difference LodeStar will make in contrast to these ones. Functional and non-functional requirements will be listed. The system models of LodeStar will be presented. Some use cases and possible scenarios will be explained. Following this, object models of the planned system and activity diagrams will be presented. At the end of the report, LodeStar's user interface mockups will be showcased.

2. Current System

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. Proposed System

In this section, the overview of the LodeStar application will be presented. On top of that, the necessary requirements (functional & non-functional) for the application will be specified.

3.1. Overview

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

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

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

In addition to airports, these features can also be used in favor of tourism companies. When customers who want to make an international trip and go to the offices of such companies, they can be shown, in virtual reality, the cities of their interest. The places to visit, 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.

3.2.Functional Requirements

In this section, functional requirements will be discussed. "Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform. " [9]. As part of functional requirements, we will discuss data resources requirements and user-specific requirements.

3.2.1.Data Resources

- The application should fetch data from many other sources using several APIs.
- The application should fetch:
    - normal or 360º images from Google Street View
    - maps from Google Maps
    - flight information from Flight Radar
    - tourist attraction information from FourSquare
    - weather from weather.com
    - currency exchange rates
- The application should show the opening and closing hours of the airports and tourist attractions.
- The application should provide real-time information about weather conditions of the destination point.
- application should show the available restaurants, cafes, and stores in the airport with their opening and closing hours. In the future, the system might favor some places over others and might list promoted items.
- The application should provide Facebook integration.
- The application will provide VR view of destination airport that shows the precise location of airport facilities and important stores for travelers, for example telecommunication shops SIM card purchase.
- The application should include taxi, subway and bus prices. Furthermore, the system should calculate the time and money needed to reach from the airport to target location using either of these methods.
- The application should be able to retrieve flight information from scanning the QR code on the boarding pass with a smart phone camera.

- The application should guide the user to target location in the airport, whether via VR images or else. Target locations can be for SIM card purchases, rent-a-car stores, baggage claim, or simply exit the airport.
- The application should guide the user to target location outside the airport with the help of Google Maps.
- The application should display necessary documents for Visa application if the user must get the Visa in the target country.
- The application should provide an estimation of the cost of general items in the target country. The countries can be ranked like expensive, moderately expensive, cheap etc.
- The application should display flight information like duration, set speed, plane attributes and the name of the captain.
- The application will gather data for city-wide events, special days and activities. The application might display additional graphics for these festivities in the VR images. This can include fireworks for New Year's Eve.
- The application should display helpful information for those with flight anxiety. Displayed information can include tips on how to avoid nausea or vomiting.
- The application should display images and information on any country during flight if the plane offers Wi-Fi service. This can be extended to show the galaxy via Google Earth.
- The application should show FourSquare comments and recommendations for airport, stores and touristic city locations. The application should allow a user to add his/her comment and rating as well.
- Facebook integration should be able to show the users information on their friends' activities on their trips using the application.
- The application can be enhanced to include 360º video clips of tourist attractions put together by staff. This can help tourism companies help their customers before choosing where to go.
- The application can show the food menus available during flight.
- The application can guide step by step on how to buy a bus or subway ticket.
- The application can have built-in games with VR support.
- In the case of contracted companies, the application should be able to guide user to those shops in the airport and see the available promotions.

3.2.2.User-Specific Requirements

- The user should be able to create accounts or use their preexisting Facebook information to log in.
- The user should be able to enter the name of the target country and city in order to receive information about both the airport and the tourist attractions. The user should be able to scan the QR code of his/her ticket with a smart phone camera in order to receive the same information.
- The user should be able to enter all the planned trip before going to the target country and receive guidance. That is to say, the user should be able to receive guidance on how to get from the airport to the hotel before departure. The system should save this and show it later in the target country, without requiring internet access.
- The user should be able to enter their budget. According to the entered budget, the system should recommend whether to take the taxi or the bus. Available time can be another variable here.
- According to their budget, the user will be recommended with different places to visit.
- The user should be able to see, step by step, how to buy a metro or bus ticket in the target city if such information is available.
- The user should be able to see where to eat before departure, or where the duty-free stores are.
- The user should be able to retrieve all the information for all the airports, not just the target one. This can help if they take more than one flight during the trip.
- The user should receive notifications or alarms as the flight time approaches. Notifications can be like: "30 minutes to boarding. Go to gate." To build upon this, the system must acquire real time flight information like delays or canceled flights.
- The users should be able to view 360º images of their target location if they have a Google Cardboard and the images are available from Google Street View.
- The user should be able to see comments or ratings about a given place. Should s/he want it, the user should be able to add comments or ratings as well.
- The application should notify the user if a Facebook friend of theirs had been to the target country and/or city as well.
- The application might recommend specific cities or countries to visit depending on the user's entered budget and time constraints.
- The user might be able to unlock achievements for doing various tasks while using the application. The achievements could be like, "Visit Turkey 5 times."

- In the case of contracted companies, the user can unlock achievements like "Fly with Turkish Airlines 10 times."
- Integrated small games for VR view, like Brick Breaker, will be provided for especially children who will use the application.

3.3.Non-Functional Requirements

In this section, non – functional requirements will be discussed. There are many definitions of non – functional requirements. "Colloquially speaking, NFRs have been referred to as "-ilities" (e.g., usability) or "-ities" (e.g., integrity), i.e., words ending with the string "-ility" or "-ity" [10] . A simpler definition is to think of them as quality attributes of the software. Nonfunctional requirements can be very challenging to handle but are crucial for successful implementation. [11] Because of this, we will spend a lot effort to make them as good as possible. Below you can find information about LodeStar's non-functional requirements such as reliability, usability, accessibility, extensibility, portability, and efficiency

3.3.1.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.

3.3.2.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.
- The application should present simply on how to get to common locations like SIM card stores or rent-a-cars.

3.3.3.Accessibility

- The application should provide a satisfying experience to users without using Google Cardboard as well.
- The application should remember the previously entered routes of the user (like airport to hotel map) after reaching the destination and show them without requiring further internet access.
- The application size should be sufficiently small to be downloaded while at the departure airport.

3.3.4.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.

3.3.5.Portability

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

3.3.6.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.

3.4.Pseudo Requirements

- The application will have a server and will be available on App Store & Google Play

3.5.System Models

To start off the system model, the report will be explaining the scenarios that a user in our application may go through. In this report twenty-six scenarios that a user may encounter are included. To explain each scenario in detail, they are divided into sub-sections as: Use Case Name, Actors, Entry Conditions, Exit Conditions and Main Flow of Events. Secondly, the system model will include a use-case diagram. The use-case diagram will be used to describe the events and actions between the user and the system. Furthermore, each use-case is explained in detail to for further explanation.

3.5.1.Scenarios

In scenarios section, we will discuss how our app will function during different userprogram interactions. Importance of this section is demonstrating the storyline of the application, as well as detecting possible alternative solutions to problems we are trying to solve.

Scenario 1

Use Case Name: LoginAsJack

Actors: Jack

Entry Conditions:

• User Jack is on Login screen

Exit Conditions:

• User Jack is on Home page

Main Flow of Events:

1. User Jack enters his email as "jack@mail.com."
2. User Jack enters his password.
3. User Jack taps the "Login" button.
4. LodeStar checks if the username and the password is correct
5. LodeStar confirms that the credentials are correct.
6. LodeStar navigates to the Home page of the user Jack.

Scenario 2

Use Case Name: CreateNewAccountForJack

Actors: Jack

Entry Conditions:

• User Jack is on Login screen

Exit Conditions:

• User Jack is on Home page

Main Flow of Events:

1. User Jack taps on the "Sign Up Using Email" button on the screen.
2. LodeStar navigates to Create New Account page.
3. User Jack enters his username as "jack."
4. User Jack enters his email address as "jack@mail.com."
5. User Jack enters his choice of password twice.

   a. User Jack taps the “Register” button.
6. LodeStar checks if the username “jack” exists in the database.
7. LodeStar checks if the mail address "jack@mail.com" exists in the database.
8. LodeStar checks if the rest of the information is accurate and complete.
9. LodeStar creates a new account.
10. LodeStar navigates to the Home page.

Scenario 3

Use Case Name: CreateNewAccountForJackUsingGoogleCredentials

Actors: Jack

Entry Conditions:

• User Jack is on Login screen

Exit Conditions:

• User Jack is on Home page

Main Flow of Events:

1. User Jack taps on the “Sign In With Google” button on the screen.
2. LodeStar navigates to Google Login website.
3. User Jack enters his email address as "jack@gmail.com."
4. User Jack enters his password.
5. User Jack taps the “Sign In Using Google” button.
6. LodeStar checks if the mail address "jack@gmail.com" exists in the database.
7. LodeStar creates a new account.
8. LodeStar navigates to the Home page.

Scenario 4

Use Case Name: CreateNewAccountForJackUsingFacebookCredentials

Actors: Jack

Entry Conditions:

• User Jack is on Login screen

Exit Conditions:

• User Jack is on Home page

Main Flow of Events:

1. User Jack taps on the “Sign In With Facebook” button on the screen.
2. LodeStar navigates to Facebook Login website.
3. User Jack enters his email address as "jack@facebookmail.com."

4. User Jack enters his password.
5. User Jack taps the "Sign In With Facebook" button.
6. LodeStar checks if the mail address "jack@facebookmail.com" exists in the database.
7. LodeStar creates a new account.
8. LodeStar navigates to the Home page.

Scenario 5

Use Case Name: ChangePreferencesOfJack

Actors: Jack

Entry Conditions:

• User Jack is on Me screen

Exit Conditions:

• User Jack is on Me screen

Main Flow of Events:

1. User Jack taps the "Gear" icon on the top right corner.
2. LodeStar navigates to the Preferences page.
3. User Jack views preferences for his LodeStar account.
4. User Jack navigates to the "Change Username" page of Preferences.
5. User Jack changes his username.
6. User Jack navigates to the "Privacy" page of Preferences.
    a. User Jack opts out of app analytics.
    b. User Jack selects "Do Not Share My LodeStar Activity on Facebook."
    c. User Jack selects "Do Not Notify My Facebook friends about my activity."
7. User Jack User Jack taps the "Clear History" button.
8. User Jack taps the "Save Preferences" button.
    a. LodeStar checks if all the given information are accurate and valid.
    b. LodeStar saves the changes of User Jack's account.
    c. LodeStar navigates to the Homepage of the User Jack.

Scenario 6

Use Case Name: ScanFlightTicketQR

Actors: Jack

Entry Conditions:

• User Jack is on Home page

Exit Conditions:

• User Jack is on Trip screen

Main Flow of Events:

1. User Jack taps on the "Scan Flight Ticket" button on the screen.
2. LodeStar navigates to the "Flight Ticket Scanner" screen.
3. User Jack allows LodeStar to use his device's camera.
4. User Jack points his device's camera to his flight ticket.
5. LodeStar reads the barcode on the flight ticket.
    a. LodeStar gets the flight number.
    b. LodeStar acquires the flight information using the flight number.
    c. LodeStar gets additional information about the destination city.
    d. LodeStar compiles the information into a readable format.
6. LodeStar navigates to the "Trip" page

Scenario 7

Use Case Name: ViewLodeStarDestinationWeatherInformation

Actors: Jack

Entry Conditions:

• User Jack is on Trip screen

Exit Conditions:

• User Jack is on Trip screen

Main Flow of Events:

1. User Jack taps on the "Weather" button on the screen.
2. LodeStar navigates to the "Destination Weather" screen.
3. LodeStar gathers information about the weather of the destination city.
4. LodeStar acquires information about temperatures.
5. LodeStar acquires information about weather conditions (rain, fog, etc.).
6. LodeStar acquires information about wind and humidity.
7. User Jack reads and digests this information.
8. User Jack taps the "Back" button.
9. LodeStar navigates back to the "Trip" page.

Scenario 8

Use Case Name: ViewLodeStarFlightInformation

Actors: Jack

Entry Conditions:

• User Jack is on Trip screen

Exit Conditions:

• User Jack is on Trip screen

Main Flow of Events:

1. User Jack taps the "Flight Information" button on the screen.
2. LodeStar navigates to the "Flight Information" page.
3. LodeStar gathers information about the flight using the flight number.
    a. LodeStar acquires departure and arrival time.
    b. LodeStar acquires aircraft brand and model.
    c. LodeStar acquires the pilot's name, if available.
    d. LodeStar acquires estimated flight speed, if available.
    e. LodeStar acquires flight distance.
4. User Jack reads and digests this information.
5. User Jack taps the "Back" button.
6. LodeStar navigates back to the "Trip" screen.

Scenario 9

Use Case Name: ViewLodeStarSIMCardInformation

Actors: Jack

Entry Conditions:

• User Jack is on Trip screen

Exit Conditions:

• User Jack is on Trip screen

Main Flow of Events:

1. LodeStar displays a menu option called "How to Buy a SIM Card" if the destination airport sells SIM Cards in the airport.
2. User Jack taps on the "How to Buy a SIM Card" button.
3. If the destination Airport has Google Street View 360 images available, LodeStar shows a "View In Virtual Reality" button .LodeStar shows the route from exit gates to SIM card shop in VR view.
4. User Jack travels in the route and digests the route information.

5. User Jack taps the "Back" button.
6. LodeStar navigates back to the "Trip" screen.

Scenario 10

Use Case Name: ManuallyEnterFlightInformation

Actors: Jack

Entry Conditions:

• User Jack is on Homepage

Exit Conditions:

• User Jack is on LodeStar Home screen

Main Flow of Events:

1. User Jack taps on the "Manually Search for Destination" button on the screen.
2. LodeStar navigates to the Destination Cities list.
3. User Jack selects his destination city in this list.
4. LodeStar lists available airports in that city (optional).
5. User Jack selects his destination airport in the list.
6. LodeStar navigates to the "LodeStar Flight" screen.

Scenario 11

Use Case Name: ViewLodeStarTransportationSuggestion

Actors: Jack

Entry Conditions:

• User Jack is on City screen

Exit Conditions:

• User Jack is on City screen

Main Flow of Events:

1. User Jack taps on the "Transportation Suggestions" button on the screen.
2. LodeStar navigates to Transportation Suggestions screen.
3. LodeStar asks User Jack the location he wishes to go.
4. LodeStar gathers information about transportation selections that are available to user at the time of the arrival and their approximate costs.
5. LodeStar displays the details of transportation choice.
    a. LodeStar acquires location of the transportation choice.
    b. LodeStar acquires the approximate price to use the transportation choice.
6. User Jack reads and digests this information.

7. User Jack presses the "Back" button.
8. User Jack navigates back to the Trip screen.

Scenario 12

Use Case Name: ViewLodeStarAirport

Actors: Jack

Entry Conditions:

• User Jack is on Destination screen

Exit Conditions:

• User Jack is on Destination screen

Main Flow of Events:

1. User Jack taps on the "Destination Airport" button on the screen.
2. LodeStar navigates to Destination Airport screen.
3. LodeStar asks User Jack about their destination airport he wishes to see.
4. LodeStar gathers information about the destination airport.
5. LodeStar displays the details of destination Airport.
    a. LodeStar displays the destination Airport virtually.
    b. User Jack can navigate in Virtual Reality.
    c. LodeStar displays the airport in Google Maps.
6. User Jack navigates back to the Destination screen.

Scenario 13

Use Case Name: TravelLodeStarAirport

Actors: Jack

Entry Conditions:

• User Jack is on Destination screen

Exit Conditions:

• User Jack is on Destination screen

Main Flow of Events:

1. User Jack taps on the "Destination Airport" button on the screen.
2. LodeStar navigates to Destination Airport screen.
3. LodeStar asks User Jack about the destination airport he wishes to see.
4. LodeStar gathers information about the destination airport.
5. LodeStar displays the details of destination city.
    a. LodeStar displays the destination city virtually.

   b. LodeStar displays the city in Google Maps.
6. User Jack navigates back to the Destination screen.

Scenario 14

Use Case Name: TravelLodeStarCity

Actors: Jack

Entry Conditions:

• User Jack is on Destination City screen

Exit Conditions:

• User Jack is on Destination City screen

Main Flow of Events:

1. User Jack taps on the "Destination City" button on the screen.
2. LodeStar navigates to Destination City screen.
3. LodeStar asks User Jack about the destination city he wishes to see.
4. User Jack enters the destination city.
5. LodeStar gathers information about the destination city.
6. LodeStar presents the information.
7. User Jack taps the "View In Virtual Reality" button.
8. LodeStar displays the destination city in Virtual Reality.
9. User Jack taps the arrows in the virtual environment to roam in the destination city.
10. User Jack navigates back to the Destination screen.

Scenario 15

Use Case Name: FlightTimeNotification

Actors: Jack

Entry Conditions:

• User Jack is on any page

Exit Conditions:

• User Jack taps the "Dismiss" button

Main Flow of Events:

1. User Jack is on any page.
2. 15 minutes remain until boarding time.
3. LodeStar gives a notification via an alarm to Jack.
4. User Jack taps the "Dismiss" button of the notification and goes to the plane.

Scenario 16

Use Case Name: DuringFlightVirtualExploration

Actors: Jack

Entry Conditions:

• User Jack is on board with Wi-Fi access

Exit Conditions:

• User Jack closes the "During Flight" screen

Main Flow of Events:

1. User Jack navigates to During Flight screen.
2. LodeStar gives information about the city the plane is flying over.
3. User Jack selects the Explore Virtually option.
4. User Jack gets a peek of the city's points of interest in a virtual reality environment.
5. User Jack closes the During Flight screen.

Scenario 17

Use Case Name: DuringFlightVideoExploration

Actors: Jack

Entry Conditions:

• User Jack is on board with Wi-Fi access

Exit Conditions:

• User Jack closes the "During Flight" screen

Main Flow of Events:

1. User Jack navigates to During Flight screen.
2. LodeStar gives information about the city the plane is flying over.
3. User Jack selects the Explore by Video option.
4. User Jack gets a peek of the city's points of interest in the form of a video.
5. User Jack closes the During Flight screen.

Scenario 18

Use Case Name: DuringFlightReadingExploration

Actors: Jack

Entry Conditions:

• User Jack is on board with Wi-Fi access

Exit Conditions:

• User Jack closes the "During Flight" screen

Main Flow of Events:

1. User Jack navigates to During Flight screen.
2. LodeStar gives information about the city the plane is flying over.
3. User Jack selects the Explore by Text option.
4. User Jack gets a peek of the city's points of interest in the form of textual description.
5. User Jack closes the During Flight screen.

Scenario 19

Use Case Name: ViewDestinationCityVideos

Actors: Jack

Entry Conditions:

• User Jack is on Destination City screen

Exit Conditions:

• User Jack is on the Destination City screen

Main Flow of Events:

1. User Jack is on Destination City screen.
2. User Jack presses the "Videos From The City" button.
3. LodeStar shows the most popular Youtube videos featuring the destination city.
4. User Jack selects one of the Youtube videos.
5. User Jack watches the video.
    a. If the video supports 360 view, Jack can watch it virtual reality.
    b. Jack can also watch the video through in app Youtube player.
6. User Jack closes the video.
7. User Jack returns to Destination City screen.

Scenario 20

Use Case Name: ViewCityTouristAttractions

Actors: Jack

Entry Conditions:

• User Jack is on Destination City page

Exit Conditions:

• User Jack is on Destination City page

Main Flow of Events:

1. User Jack is taps on the button "Tourist Attractions" on City page.
2. LodeStar lists the popular spot suggestions in a list.

3. Jack taps on a listed suggestion.
4. LodeStar shows the VR view of the city at the chosen location.
5. LodeStar displays some information about the location.
6. User Jack taps the "Back" button.
7. LodeStar navigates to Destination City screen.

Scenario 21

Use Case Name: ViewCityAccommodation

Actors: Jack

Entry Conditions:

• User Jack is on LodeStar Destination City page

Exit Conditions:

• User Jack is on the Destination City page

Main Flow of Events:

1. Use Jack taps on the "Accommodation View" button.
2. Jack types the words of the Hotel or address of the intended accommodation.
3. LodeStar shows the place on the map.
4. Tapping on the place on the map will open the VR view of the place(if available)or nearby street.
5. User Jack taps the "Back" button
6. LodeStar navigates to Destination City screen.

Scenario 22

Use Case Name: ViewCityEvents

Actors: Jack

Entry Conditions:

• User Jack is on Destination City screen

Exit Conditions:

• User Jack is on Destination City screen

Main Flow of Events:

1. User Jack taps "City Events" button on Destination City screen.
2. LodeStar shows the active and upcoming city events on the map.
3. User Jack selects one of the events by tapping on it.
4. LodeStar opens the VR view of the event location.
5. LodeStar gathers information about the events and shows it to Jack.

6. User Jack taps the "Back" button
7. LodeStar navigates to Destination City screen.

Scenario 23

Use Case Name: ViewCommentsOfPlaces

Actors: Jack

Entry Conditions:

• User Jack is on Destination City screen

Exit Conditions:

• User Jack is on Destination City screen

Main Flow of Events:

1. User Jack makes a selection of a place to visit on the Destination City screen
2. LodeStar displays the targeted place and shows a "Comments" button, if available comments exist.
3. User Jack taps on the "Comments" button.
4. LodeStar displays the comments.
5. User Jack taps the "Back" button.
6. LodeStar navigates to Destination City screen.

Scenario 24

Use Case Name: EnterCommentForPlaces

Actors: Jack

Entry Conditions:

• User Jack is on Destination City screen

Exit Conditions:

• User Jack is on Destination City screen

Main Flow of Events:

1. User Jack selects some places to visit on the Destination City screen.
2. LodeStar displays the targeted place and shows a "Comments" button, if available comments exist.
3. User Jack taps on the "Comments" button.
4. LodeStar displays the comments.
5. User Jack taps "Add A Comment" button.
6. LodeStar displays a title and text area.
7. User Jack enters the new comment for the place.

8. User Jack taps the "Submit" button.
9. LodeStar adds the new comment to its database and displays it.
10. User Jack taps "Done" button.
11. LodeStar navigates to Destination City screen.

Scenario 25

Use Case Name: ViewLivingExpenses

Actors: Jack

Entry Conditions:

• User Jack is on Destination City screen

Exit Conditions:

• User Jack is on Destination City screen

Main Flow of Events:

1. User Jack taps the "See Living Expenses" button on the Destination City screen.
2. LodeStar gathers data for the city.
3. LodeStar displays the living expenses on the city:
    a. LodeStar acquires daily restaurant costs,
    b. LodeStar acquires rent prices,
    c. LodeStar acquires daily approximate grocery costs,
    d. LodeStar acquires daily approximate transportation costs.
4. User Jack taps on the "Back" button.
5. LodeStar navigates to Destination City screen.

Scenario 26

Use Case Name: ViewCurrencyRates

Actors: Jack

Entry Conditions:

• User Jack is on Destination City screen

Exit Conditions:

• User Jack is on Destination City screen

Main Flow of Events:

1. User Jack taps the "See Currency Rates" button.
2. LodeStar gathers real time exchange data.
3. LodeStar displays the currency exchange rates between the visited country and User Jack's country.

4. User Jack taps on the "See Currency Exchange Shops" button on the screen.
5. LodeStar displays the currency exchange shops on a map.
6. User Jack taps on one of the displayed places.
7. LodeStar displays VR view of the shop if available.
8. User Jack taps on the "Back" button.
9. LodeStar navigates to Destination City screen.

3.5.2.Use Case Model

In the following section, we will introduce use-case models of our system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal [12]. This is important for showing our user-system interactions.

Figure 1 - Use Case Diagram

3.5.2.1.Login Use Case

Use Case Name: Login

Actors: User

Entry Conditions:

•User is on Login screen

Exit Conditions:

•User in on Home page

•Authentication failed and user is on Login screen

Main Flow of Events:

1. User enters email.
2. User enters password.
3. User taps the "Login" button.
4. LodeStar checks if the email and the password are correct.
5. LodeStar confirms that the credentials are correct.
6. LodeStar navigates to the Home page of the user.

Alternative Flow of Events:

A. User enters wrong email and/or wrong password.
    a. LodeStar detects that the credentials are incorrect.
    b. LodeStar navigates to Login screen.
    c. LodeStar informs user about the wrong entries.

B. User presses "Login with Google" button
    a. LodeStar looks for the user's Google credentials.
    b. LodeStar navigates to the Home page of the user.

C. User presses "Login with Facebook" button
    a. LodeStar looks for the user's Facebook credentials.
    b. LodeStar navigates to the Home page of the user.

3.5.2.2.Create Account Use Case

Use Case Name: Create New Account

Actors: User

Entry Conditions:

•User is on Login screen

Exit Conditions:

•User is on Home page

Main Flow of Events:

1. User taps the "Login Using Email" button.
2. LodeStar navigates to "Sign Up" page.
3. User enters username.
4. User enters email address.
5. User enters password twice.
6. User taps the "Sign up Using Email" button.
7. LodeStar checks if there is an already user registered with that username.
8. LodeStar checks if there is an already user registered with that mail address.
9. LodeStar checks if the information is accurate and complete.
10. LodeStar creates a new account for the user.
11. LodeStar navigates to the Home page of the user.

Alternative Flow of Events:

A. User entries are inaccurate or missing.
   a. LodeStar returns to Login screen.
   b. LodeStar informs the user about missing and/or inaccurate entries.
   c. User enter the missing and/or inaccurate informations again.
   d. LodeStar creates a new account for the user.
   e. LodeStar navigates to the Home page of the user.

3.5.2.3.Create Account Using Google Use Case

Use Case Name: Create New Account Using Google

Actors: User

Entry Conditions:

•User is on Login screen

Exit Conditions:

•User is on Home page

Main Flow of Events:

1. User presses on the "Login Using Google" button.
2. User enters his email address.3.User enters his password.
3. LodeStar checks if the mail address already exists in the database.
4. LodeStar creates a new account.
5. LodeStar navigates to the Home page of the user.

Alternative Flow of Events:

A. User has already logged into Google.
   a. User presses on "Login Using Google" button
   b. LodeStar checks the credentials.
   c. LodeStar creates a new account.
   d. LodeStar navigates to the Home page of the user.

B. User has already logged in using Google
   a. User presses on "Login Using Google" button.
   b. LodeStar checks the credentials.
   c. LodeStar navigates to the Home page of the user.

3.5.2.4.Create Account Using Facebook Use Case

Use Case Name: Create New Account Using Facebook

Actors: User

Entry Conditions:

•User is on Login screen

Exit Conditions:

•User is on Home page

Main Flow of Events:

1. User taps the "Sign in with Facebook" button.
2. User enters email address.

3. User enters password.
4. LodeStar checks if the mail address already exists in the database.
5. LodeStar creates a new account.
6. LodeStar navigates to the Home page of the user.

Alternative Flow of Events:

C. User has already logged into Facebook.
   a. User taps on "Sign in Using Facebook" button.
   b. LodeStar checks the credentials.
   c. LodeStar creates a new account.
   d. LodeStar navigates to the Homepage of the user.

D. User has already logged in using Facebook.
   a. User presses on the "Login Using Facebook" button.
   b. LodeStar checks the credentials.
   c. LodeStar navigates to Home page of the user.

3.5.2.5.Change Account Settings Use Case

Use Case Name: Change Account Settings

Actors: Jack

Entry Conditions:

• Change Account Setting includes Login use case

• User is on Me screen

Exit Conditions:

• User is on Me screen

Main Flow of Events:

1. User taps on the "Gear" icon.
2. LodeStar navigates to the Preferences screen.
3. User changes his username.
4. User changes his email address.
5. User changes his password.
6. User taps the "Save Preferences" button.
7. LodeStar checks if all the given information are accurate and valid.
8. LodeStar saves the changes of user's account.
9. LodeStar navigates to the Me screen of the user.

Alternative Flow of Events:

A. 1.User has signed in using Google.
    a. User cannot change his email address (Skip Step 5).
B. 2.User has signed in using Facebook.
    a. User cannot change his email address (Skip Step 5).
C. User does not change his username (Skip Step 4).
D. User does not change his email address (Skip Step 5).
E. User does not change his password (Skip Step 6).

3.5.2.6.Change Account Privacy Preferences Use Case

Use Case Name: Change Account Privacy Preferences

Actors: Jack

Entry Conditions:

• Change Account Privacy Preferences includes Login use case

• User Jack is on Me screen

Exit Conditions:

• User Jack is on Me screen

Main Flow of Events:

1. User taps on the "Gear" icon.
2. LodeStar navigates to the Preferences screen.
3. User views his preferences.
4. User navigates to the "Privacy" screen of Preferences.
    a. User opts out of app analytics.
    b. User selects "Do Not Share My LodeStar Activity on Facebook."
    c. User selects "Do Not Notify My Facebook Friends About My Activity."
5. User taps the "Clear History" button.
6. User taps the "Save Preferences" button.
7. LodeStar saves the new preferences of the user.
8. LodeStar navigates to the Me screen of the user.

Alternative Flow of Events:

A. 1.User does not opt out of app analytics (Skip Step 4.a).
B. 2.User does not select "Do Not Share My LodeStar Activity on Facebook" (Skip Step 4.b).
C. 3.User does not select "Do not Notify My Facebook Friends About My Activity" (Skip Step 4.c).

3.5.2.7.Select the City of Destination Use Case

Use Case Name: Select the City of Destination Use Case

Actors: Jack

Entry Conditions:

• Select the City of Destination includes Login use case

• User Jack is on Trip screen

Exit Conditions:

• User Jack leaves Trip screen

Main Flow of Events:

1. User taps the "Select Destination" button.
2. LodeStar navigates to Trip screen.
3. User selects the destination city.

Alternative Flow of Events:

A. User does not select the destination city. (In this case nothing is loaded).

3.5.2.8.Travel the Destination City Use Case

Use Case Name: Travel the Destination City Use Case

Actors: Jack

Entry Conditions:

• Travel the Destination City includes Login use case

• Travel the Destination City includes Select Destination use case

• User Jack is on Trip screen

Exit Conditions:

• User Jack leaves Trip screen

Main Flow of Events:

1. User taps the "Select Destination" button.
2. User selects the City of Destination.
3. User is redirected to the Trip screen.
4. User travels the tourist attractions of the Destination City with Google Street View.

Alternative Flow of Events:

A. User selects the virtual reality option in the 4th event to travel the tourist attractions

3.5.2.9.Select the Destination Airport Use Case

Use Case Name: Select the Destination Airport Use Case

Actors: Jack

Entry Conditions:

• Select the Destination Airport includes Login use case

• User Jack is on Trip screen

Exit Conditions:

• User Jack leaves the Trip screen

Main Flow of Events:

1. User taps the "Select Destination Airport" button.
2. User selects the Destination Airport.
3. User is redirected to the Trip screen.

3.5.2.10.Travel the Destination Airport Use Case

Use Case Name: Travel the Destination Airport Use Case

Actors: User

Entry Conditions:

• Travel the Destination City includes Login use case

• Travel the Destination City includes Select Destination Aİrport use case

• User Jack is on Trip screen

Exit Conditions:

• User Jack leaves the Trip screen

Main Flow of Events:

1. User taps the "Select Destination Airport" button.
2. User selects the Destination Airport
3. User is redirected to the Trip screen.
4. User travels the Destination Airport with Google Street View.

Alternative Flow of Events:

A. User selects the virtual reality option in the 4th event to travel the destination airport

3.5.2.11.Scan Flight Ticket Barcode Use Case

Use Case Name: Scan Flight Ticket Barcode Use Case

Actors: User

Entry Conditions:

• Scan Flight Ticket Barcode includes Login use case

• User is on Home page

Exit Conditions:

• User is on LodeStar Flight screen

Main Flow of Events:

1. User taps the "Scan a Boarding Pass" button.
2. LodeStar navigates to the Flight Ticket Scanner screen.
3. User allows LodeStar to use his device's camera.
4. User points his device's camera to his flight ticket.
5. LodeStar reads the barcode on the flight ticket.
6. LodeStar gets the flight number and flight information using the flight number.
7. LodeStar gets additional information about the destination city and displays into a readable format.
8. LodeStar navigates to the Trip screen.

Alternative Flow of Events:

A. User doesn't give permission to use device camera (skip steps 4-6).

3.5.2.12.View Destination Weather Information Use Case

Use Case Name: View Destination Weather Information

Actors: User

Entry Conditions:

• View Destination Weather Information includes Login Use case

• User is on Trip screen

Exit Conditions:

• User is on LodeStar Trip screen

Main Flow of Events:

1. User taps the "Weather" button.
2. LodeStar navigates to the "Weather Information" screen.
3. LodeStar loads information about temperatures, weather conditions (rain, fog, etc.), and wind.

3.5.2.13.View Flight Information Use Case

Use Case Name: View Flight Information

Actors: User

Entry Conditions:

- View Flight Information includes Login Use case
- User is on Trip screen

Exit Conditions:

- User is on Trip screen

Main Flow of Events:

1. User taps the "Flight Information" button.
2. LodeStar opens "Flight Details" screen
3. LodeStar retrieves information about the flight using the flight number.
4. LodeStar loads departure and arrival time, aircraft brand, model and flight distance, if available.

3.5.2.14.View Flight Information Use Case

Use Case Name: View SIM Card Information

Actors: User

Entry Conditions:

- View SIM card Information includes Login Use Case
- User is on Trip screen

Exit Conditions:

- User is on Trip screen

Main Flow of Events:

1. LodeStar displays a menu option called "How to Buy a SIM Card" if the destination airport sells SIM Cards in the airport.
2. User taps the "How to Buy a SIM Card" button.
3. LodeStar lists the related shops.
4. If the destination Airport has Google Street View 360 images available, LodeStar shows a "View with VR" button next to the shops in the lists.
5. LodeStar shows the route from exit gates to SIM card shop in VR view.

6. User travels in the route in VR View.

Alternative Flow of Events:

A. Destination airport doesn't have SIM Card shops (skip 2-5).

B. Destination airport doesn't have Street View images available (skip 4-5).

3.5.2.15.Manually Enter Flight Information Use Case

Use Case Name: Manually Enter Flight Information

Actors: User

Entry Conditions:

•Manually Enter Flight Information includes Login use case

• User is on Home page

Exit Conditions:

• User is on Trip screen

Main Flow of Events:

1. User taps the "Manually Search for Destination" button.
   LodeStar navigates to the Destination cities list.
2. User selects his destination city in this list.
   LodeStar displays available airports in the city.
3. User selects his destination airport in this list.
4. LodeStar displays "Trip" screen.

Alternative Flow of Events:

A. Destination city aren't included in LodeStar database (skip 2-6).

B. Destination airport aren't included in LodeStar database (skip 4-6).

3.5.2.16.View Transportation Suggestions Use Case

Use Case Name: View Transportation Suggestion

Actors: User

Entry Conditions:

• View Transportation Suggestion includes Login use case

• User is on Trip screen

Exit Conditions:

• User is on Trip screen

Main Flow of Events:

1. User taps the "Transportation Options" button.
2. LodeStar opens to Transportation Options page.
3. LodeStar asks user about their location in the city that they want to go.
4. User types the name or address of the location.
5. LodeStar gathers and lists information about transportation selections that are available to user at the time of the arrival and their approximate costs.
6. User selects one of the options.
7. LodeStar displays the details of transportation choice.
    a. LodeStar loads location of the transportation choice.
    b. LodeStar loads the approximate price to use the transportation choice.

Alternative Flow of Events:

A. User typed location cannot be found in the city.
    a. LodeStar asks user again to type the information (return to step 4).

B. No available transportation suggestion found (skip 5-7).

3.5.2.17.Flight Time Notification Use Case

Use Case Name: Flight Time Notification

Actors: User

Entry Conditions:

• Flight Time Notification includes Login use case

• User is on any screen

Exit Conditions:

• User taps the Dismiss button

Main Flow of Events:

1. User is on any screen.
2. LodeStar detects only 15 minutes remain until boarding.
3. LodeStar gives a notification via an alarm to User.
4. User taps the "Dismiss" button.

3.5.2.18.During Flight Virtual Exploration Use Case

Use Case Name: During Flight Virtual Exploration

Actors: User

Entry Conditions:

• During Flight Virtual Exploration includes Login use case

• User is on board with Wi-Fi access

Exit Conditions:

• User closes the travel During Flight screen

Main Flow of Events:

1. User is on any page.
2. LodeStar gives information about the city the plane is flying over.
3. User selects the "Explore In Virtual Reality" option.
4. User gets a peek of the city's points of interest in virtual reality.

3.5.2.19.During Flight Virtual Exploration Use Case

Use Case Name: During Flight Video Exploration

Actors: User

Entry Conditions:

• During Flight Video Exploration includes Login use case

• User is on board with Wi-Fi access

Exit Conditions:

• User is on the travel during flight page

Main Flow of Events:

1. User is on any page.
2. LodeStar gives information about the city the plane is flying over
3. User selects the "Explore by Video" option
4. User gets a peek of the city's points of interest in video form.

3.5.2.20.During Flight Reading Exploration Use Case

Use Case Name: During Flight Reading Exploration

Actors: User

Entry Conditions:

•During Flight Reading Exploration includes Login use case

• User is on board with Wi-Fi access

Exit Conditions:

• User is on the travel during flight page

Main Flow of Events:

1. User is on any page.
2. LodeStar gives information about the city the plane is flying over.
3. User selects the "Explore by Text" option.
4. User Jack gets a peek of the city's points of interest in text form.

3.5.2.21.View Destination City Videos Use Case

Use Case Name: View Destination City Videos

Actors: User

Entry Conditions:

• User is on Trip screen

Exit Conditions:

• User Jack is on the Trip screen

Main Flow of Events:

1. User Jack is on Trip screen.
2. LodeStar shows most popular Youtube videos about that city.
3. User selects one of the Youtube videos.
4. User watches the video.
    a. If the video supports 360 view, user can watch it in virtual reality.
    b. User can watch the video through in app Youtube player.

3.5.2.22.View City Touristic Spots Use Case

Use Case Name: View City Tourist Attractions

Actors: User

Entry Conditions:

•View City Touristic Spots includes Login use case

• User is on Trip screen

Exit Conditions:

• User is on the Trip screen

Main Flow of Events:

1. User taps the "Tourist Attraction" button on City screen.
2. LodeStar lists the popular spot suggestions in a list.
3. User taps on a listed suggestion.

4. LodeStar shows the VR view of the city at the chosen location.
5. LodeStar displays some information about the location.

Alternative Flow of Events:

A. No suggested touristic spot is found (skip 2-5).

3.5.2.23. View City Accommodation Use Case

Use Case Name: View City Accommodation

Actors: User

Entry Conditions:

• View City Accommodation includes Login use case

• User is on Trip screen

Exit Conditions:

• User is on the Trip screen

Main Flow of Events:

1. Use taps the "Accommodation" button.
2. User types the words of the Hotel or address of the intended accommodation.
3. LodeStar shows the place on the map.
4. Tapping on the place on the map will open the VR view of the place (if available) or nearby street.

Alternative Flow of Events:

A. Typed words don't match with any city location, LodeStar asks the user to try again (return step 2).

B. No available panorama Google Street View is found near the area (skip 4).

3.5.2.24. View City Events Use Case

Use Case Name: View City Events

Actors: User

Entry Conditions:

• View City Events includes Login Use case

• User is on Trip page

Exit Conditions:

• User is on the Trip page

Main Flow of Events:

1. User taps the "City Events" button on Trip screen.
2. LodeStar shows the active and upcoming city events on the map.
3. User selects one of the events.
4. LodeStar will open the VR view of the event location.
5. LodeStar displays the information about the chosen event.

Alternative Flow of Events:

A. No Google Street View is available around the event location(skip 4-5).

B. No available information is found about the event (skip 5).

3.5.2.25.View Comments of Places Use Case

Use Case Name: View Comment Of Places

Actors: User

Entry Conditions:

•View Comment Of Places includes Login use case

• User is on Trip screen

Exit Conditions:

• User is on the Trip screen

Main Flow of Events:

1. User taps the "Places to See" button.
2. User makes a selection any place to visit on the "Places to See" screen.
3. When places are listed, "Comments" button are listed under the places.
4. User taps the "Comments" button of one of the places.
5. LodeStar gathers data of the place and displays the comments.

Alternative Flow of Events:

A. No available comment is found for the place (skip 3-4).

3.5.2.26.Enter Comment for Places Use Case

Use Case Name: Enter Comment For Places

Actors: User

Entry Conditions:

•Enter Comment For Places includes Login use case

• User is on Trip screen

Exit Conditions:

• User is on the Trip screen

Main Flow of Events:

1. User taps the "Places to See" button.
2. User makes a selection to visit on the "Places to See" screen.
3. When places are listed, "Comments" button are listed under the places, if available comments exists.
4. User taps on the Comments button of one of the places.
5. LodeStar gathers data of the place and displays the comments.
6. User enters a new comment for the place.
7. User taps the "Send" button.
8. LodeStar adds the new comment in the place data.
9. LodeStar navigates to Destination City screen.

Alternative Flow of Events:

A. User exceeds character limit when commenting (return step 4).

3.5.2.27.View Living Expenses Use Case

Use Case Name: View Living Expenses

Actors: User

Entry Conditions:

•View Living Expenses includes Login use case

• User is on Trip screen

Exit Conditions:

• User is on the Trip screen

Main Flow of Events:

1. User taps the "Living Expenses" button.
2. LodeStar gathers and displays the living expenses on the city:
    a. LodeStar shows daily restaurant costs.
    b. LodeStar shows average rent prices.
    c. LodeStar shows daily approximate grocery costs.
    d. LodeStar shows daily approximate transportation costs.

3.5.2.28.View Currency Rates Use Case

Use Case Name: View Currency Rates

Actors: User

Entry Conditions:

•View Currency Rates includes Login use case

• User is on Trip screen

Exit Conditions:

• User is on the Trip screen

Main Flow of Events:

1. User taps the "See Currency Rates" button.
2. LodeStar gathers and displays the currency exchange rates between the visited country and user's country.
3. User taps on the button "See Currency Exchange Shops" on the screen.
4. LodeStar displays the currency exchange shops on a map.
5. User taps on one of the displayed places.
6. If available, VR view of the nearby shops are displayed.

Alternative Flow of Events:

A. No shop for currency exchange is available (skip 5-7).

3.5.3.Object and Class Model

For demonstrating our initial design for program architecture, we will introduce object class diagram in this section. The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction[13].

Figure 2 - Object and Class Diagram

TripOrganizer: This class will control and maintain trip related information. It will communicate with DataManager to fetch information related to the specified airport.

DatabaseManager: This class will be responsible for all database communications.

DataFetcher: This class will fetch required information about the trips. Will contain all API communication code. As an example, the fetch function in this class can be called using a flight no as an argument to get specific information about a flight.

User: This class is a representation of a user in LodeStar. The user class will contain distinguishing information like name-surname, email, preferences, history and person

VirtualRealityManager: The task of this class is to manage the virtual reality aspects of our application. It will be responsible for initiating the Virtual Reality mode, fetching the VR image, displaying the image and adjust any preferences the user might have when the VR image is shown.

AuthenticationManager: The duty of this class is to enable the authentication via Google+ and Facebook. This class will give the user the option to sign in using one of these social media platforms.

Airport: This class will hold the critical information about an airport. It will hold the name of the airport, the city the airport is located, the country the airport is located in and the working hours of the airport.

TransportationManager: This class will contain information about transportation related activities such as time, location and budget constraints

3.5.4.Dynamic Models

Dynamic modeling of a system demonstrates the function calls and interaction between the system components, thus shows the inner processing of the system.

3.5.4.1.Sequence Diagrams

A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order with which function calls [14]. We introduced following six example sequences for our system.

3.5.4.1.1.Sign Up Sequence Diagram

Figure 3 - Sign Up Sequence Diagram

The figure shows information about how "Jack" signs up to use LodeStar. Initially Jack will need to provide his credentials. To do that, the SignUpScreen will be prompted to him. Having entered the credentials, the manager will contact the database to update it for the new user's credentials. After updating the database, a notification to the user will be given.

3.5.4.1.2.Log In Sequence Diagram

Figure 4 - Log In Sequence Diagram

The figure shows information about how user "Jack" logs in to use LodeStar. Initially Jack will need to provide his credentials. To do that, the LoginScreenView will be prompted to him. Having entered the credentials, the manager will contact the database to check the user's credentials. After receiving a verification from the database, LoginScreenView will direct Jack to Home Page.

3.5.4.1.3.Find SIM Card Information Sequence Diagram

Figure 5 - Provide SIM Card Information Sequence Diagram

The above sequence diagram displays the program call sequence when a user looks for a SIM card shop in during their travel. First, the LocationViewer will search the airport for desired criteria. To accomplish this, it will call LocationManager, which will return the locations of SIM card shops in the chosen airport. LocationViewer will list them to the user. Then the user will select a shop which will cause ViewManager to show a route to that shop in VR view. Following this, ViewManager will create a VRView, that will display the route to the shop from a chosen origin point in airport. It will also display information about the shop to the user.

3.5.4.1.4.View Transportation Suggestion Sequence Diagram

Figure 6 - View Transportation Information Sequence Diagram

This sequence diagram depicts the program functions when a user requires a transportation suggestion during their travels. First, TransportationManager will be called for possible means of transportation in a certain city for the user's airport. LocationManager will search the stops and terminals that user can choose and returns a list of suggestions. TransportationManager will show them to the user along with their details, like costs. The user will select one of these options. LocationManager will locate the chosen option, display related information and call ViewManager to show the route in VR view. A new VRView will be created and it will display the route to the user.

3.5.4.1.5.Travel City Sequence Diagram

Figure 7- Travel City Sequence Diagram

The above sequence diagram is intended for the user to travel the tourist attractions at the city of destination. The user initially searches the city using the LocationViewer. Then the manager finds the chosen city and returns the corresponding info (it's Google Street View). LocationViewer will also give the user the option to travel the city virtually.

3.5.4.1.6.Scan Flight Ticket Sequence Diagram

Figure 8- Scan Flight Ticket Sequence Diagram

This sequence diagram demonstrates the calls when users scan QR codes on their tickets at the beginning of their trip. When user wants to read their ticket, I/OManager call reads the QR code. Scanned information is sent to the TripManager to find the flight number and other details using the code. TripManager gets the flight details and creates a new Trip. Newly created Trip will be used in the application to update the flight information, so that the user will be kept up to date on delays and such. Also, it will display the flight info for other application functions to use, such as flight notifications.

3.5.4.2.Activity Diagrams

Activity diagram is another important diagram in unified modeling language to describe the dynamic aspects of the system that represents the flow from one activity to another activity[15]. We included following two examples of activity sequences for our system.

Application Flow Activity Diagram

Figure 9 - Application Flow Activity Diagram

LodeStar greets the user in its home page upon starting the application. From there, the user can either login with his/her credentials or set up a new account. Following successful login, LodeStar guides the user to the main page of the application. The user here is granted to choose between managing user profile, visiting history or favorites pages, or viewing information about an upcoming trip. To get to this trip screen, the user needs to either enter the destination city's name or scan the QR Code of a valid ticket. From any screen, it is possible to logout.

Trip Creating Activity Diagram

Figure 10 - Trip Creation Activity Diagram

This activity flow commences with a successful login into the system by the user. There, the user scans the QR Code of their plane ticket by using his/her smart phone's camera. Then, they are taken to the trip page. Here, the possibilities are endless. The user can check the weather of the destination city. S/he can check the living expenses, look for accommodation, get recommendations about transport or review the places to see. About the airport of interest, the user can see the restaurants or shopping stores available as well as the lounges present. For the flight, they can view all information like duration and expected landing time. From any of the screens, the user can logout.

3.5.5.User Interface

In this part of the report, the user-interface of the application will be presented. Each sketch is provided with an explanation to better specify how it address the requirements mentioned above.

3.5.5.1.Login Page

The user will log in to LodeStar from the Login page as showcased below. The user will have three different options. The user will also have an option to sign in using their social media accounts.

Figure 11 - Login Page UI Mockup

If the user already has an account, he will be able to sign into his account after tapping "Already have an account? Sign In" label from the previous page. After doing that, he will be navigated to the following page.

Figure 12 - Sign In Page UI Mockup

3.5.5.2.Sign Up Page

If the user does not wish to sign up using their social media accounts and wants to sign up using their personal email address, then the he will be navigated to the following page.

Figure 13 - Sign Up Page UI Mockup

3.5.5.3. Home Page

The user will be navigated to this page once they pass the log-in page. The page will have a big button at the top instructing the user to scan a flight card using their phone camera. Or the user will be able to enter a flight number manually if he does not have a flight card yet. (If the user did online check-in or the user basically wishes to view another flight). The user will also be able to manually search for cities just to learn more about them. We do not want to alienate users without flight cards. There might be users who want to learn about different cities without traveling there.

Towards the bottom of the page, a simple card will be shown to inform the user about the face that LodeStar experience is enhanced with VR glasses. LodeStar will also educate the user about VR glasses and tell them where to get one.

Figure 14 - Home Page UI Mockup

3.5.5.4.Trip Page

This page will be navigated upon a successful flight card scan. Using the QR code located on the flight card, the departure and arrival cities of the corresponding flight can be learnt. This way, LodeStar will fetch details about these cities and display them to the user in a user-friendly way.

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

Figure 15- Trip Departure Page UI Mockup

Figure 16 - Trip Arrival Page UI Mockup

3.5.5.5.Weather Information Page

If the user taps on the "Weather" card from the Trip page (see previous page), the user will see the weather information about the corresponding city. Weather information will be shown in a minimalistic and simple way, without any clutter. The user only needs to know if they need a coat or an umbrella in the city they are traveling. They do not need to be informed about the speed of wind or how much rain falls per square meter per year.

Figure 17 - Weather Information Page UI Mockup

3.5.5.6.Flight Information Page

This page will be navigated to when the user taps on the "Flight Details" card from the "Trip" page. This screen will gather all available data that can be gathered by scanning a flight card. Also, it will display a simple weather summary. Tech savvy users which are interested in the technical details of a flight will be able to learn the model of the plane and view the pictures of the aircraft.

Figure 18 - Flight Information Page UI Mockup

3.5.5.7.Currency Rates Page

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

Figure 19 - Currency Rates UI Mockup

3.5.5.8.Me Page

This page will contain a basic account summary at the top. Google and Facebook icons will be shown if the user has linked their corresponding accounts with LodeStar. Upon tapping the gear icon on the top right, the user will be taken to the preferences page.(see next page)

Figure 20 - Me Page UI Mockup

Trips: how many times jack scanned flight cards with LodeStar

Trip Logs: these will be comments that the user made during their trips. Just like tweeting, the user will write comments about the things they've seen, done, etc.

Last Trip: Destination city for the last flight card Jack has scanned.

3.5.5.9.Preferences Page

This page will be navigated to once the user taps the gear icon on the "Me" page. This page will allow user to change basic preferences.

Figure 21 - Preferences Page UI Mockup

4. 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] Malan R., Bredemeyer D. Functional Requirements and Use Cases. Architecture Resources For Enterprise Advantage. 2001. [Online]. Available: http://www.bredemeyer.com/pdf_files/functreq.pdf. [Accessed: 06-Nov-2017].

[10] Chung L., Leite J.C.S.D.P. On Non-Functional Requirements in Software Engineering. Conceptual Modeling: Foundations and Applications. Lecture Notes in Computer Science. 2009. [Online]. Available: https://doi.org/10.1007/978-3-642-02463-4_19. [Accessed: 06-Nov-2017].

[11] Chung L., et al. Non-Functional Requirements in Software Engineering. Springer Science+Business Media New York. 2000.

[12] SearchSoftwareQuality. (2017). What is use case? - Definition from WhatIs.com. [online] Available at: http://searchsoftwarequality.techtarget.com/definition/use-case [Accessed 4 Nov. 2017].

[13] www.tutorialspoint.com. (2017). UML - Class Diagram. [online] Available at: https://www.tutorialspoint.com/uml/uml_class_diagram.htm [Accessed 4 Nov. 2017].

[14] En.wikipedia.org. (2017). Sequence diagram. [online] Available at: https://en.wikipedia.org/wiki/Sequence_diagram [Accessed 2 Nov. 2017].

[15] www.tutorialspoint.com. (2017). UML - Activity Diagrams. [online] Available at: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm [Accessed 6 Nov. 2017].

Creating the exact same user interface as the mockup is quite a challenge. However, Berk has managed to create a solid codebase for the iOS backend.

–m “iOS app looks neat. And it just works” BA

High Level Design Report

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

1. Introduction 2
1.1.Purpose of the System 3
1.2.Design Goals 4
1.2.1.Reliability 4
1.2.2.Usability 4
1.2.3.Efficiency 5
1.2.4.Security 5
1.2.5.Maintainability 5
1.2.6.Extensibility 5
1.2.7.Portability 6
1.2.8.Supportability 6
1.3.Definitions, Acronyms and Abbreviations 6
1.4.Overview 6
2. Current Software Architecture 7
3. Subsystem Decomposition 9
3.1.Overview 9
3.2.Subsystem Decomposition 11
3.3.Hardware & Software Mapping 14
3.4.Persistent Data Management 15
3.5.Access Control and Security 15
3.6.Global Software Control 16
3.7.Boundary Conditions 16
3.7.1. Starting the Application 16
3.7.2.Terminating the Application 16
3.7.3.Failure in the Application 16
4. Subsystem Services 17
4.1.Client 18
4.1.1.Controller Subsystem 18
4.1.2.View Subsystem 19
4.2.Server 21
4.2.1.Logic Tier 22
4.2.2.Data Tier 24
5. Glossary 25
6. References 26

List of Figures

I. Figure 1 - Subsystem Decomposition - Page 10
II. Figure 2 - Subsystem Decomposition Detailed View -Page 12
III. Figure 3 - Component Diagram - Page 14
IV. Figure 4 - Detailed View of Client Subsystem - Page 17
V. Figure 5 - Controller Subsystem in Client - Page 18
VI. Figure 6 - View Subsystem in Client - Page 19
VII. Figure 7 - Detailed View of Server Subsystem - Page 21
VIII. Figure 8 - Logic Tier Subsystem in Server - Page 22
IX. Figure 9 - Data Tier Subsystem in Server - Page 24

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].

After 3 weeks of absolutely no progress on iOS VR
Çelik finally figured out what was missing.

-m “iOS GVR is now a reality.” ÇK

Low Level Design Report

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

1. Introduction 1
1.1. Design Trade-Offs 2
1.1.1. Functionality vs Usability 2
1.1.2. Security vs. Cost 2
1.1.3. Space vs. Time 3
1.1.4. Compatibility vs Extensibility 3
1.2. Engineering Standards 3
1.3. Use of New Tools and Technologies 3
1.4. Life Long Learning 4
1.5. Interface Documentation Guidelines 4
2. Packages 5
2.1. Client 5
2.1.1. View 6
2.1.2. Controller 8
2.2. Server 10
2.2.1. Logic Tier 11
2.2.2. Data Tier 13
3. Class Interfaces 15
3.1. Client 15
3.1.1. View 15
3.1.2. Controller 24
3.2. Server 27
3.2.1. Logic Tier 27
3.2.2. Data Tier 33
4. Glossary 37
5. References 38

List of Figures

I. Figure 1 - View Subsystem - Page 6
II. Figure 2 - Controller Subsystem - Page 8
III. Figure 3 - Logic Tier Subsystem - Page 11
IV. Figure 4 - Data Tier Subsystem - Page 13

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]

The most difficult task. Navgiating in a VR environment.
This one couldn’t be done without the expertise of Hüseyin.

–m “Android VR Navigation is a success.” HB

Final Report

Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: LodeStar

Final Report

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

Supervisor: Halil Bülent Özgüç

Jury Members: Çiğdem Gündüz Demir and Uğur Güdükbay

Progress Report

May 3, 2018

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

Contents

1. Introduction 1
1. Engineering and Report Writing Standards 2
1.2. Use of New Tools and Technologies 2
1.3. Life Long Learning 3
1.4. Implementing Creative and Impactful Solutions 3
2. System Overview 4
3. LodeStar's Algorithmic and Application Design 5
3.1. Virtual Reality Algorithm and Implementation 5
3.2. User Experience and Interface Design 9
4. Final System Architecture Design 10
4.1. Subsystem Overview 10
4.2. Client 10
4.2.1. View 11
4.2.2. Controller 13
4.3. Server 15
4.3.1. Logic Tier 15
4.3.2. Data Tier 18
5. Impact of Engineering Solutions 20
5.1.1. Economical Impacts 20
5.1.2. Environmental Impacts 20
5.1.3. Social Impacts 21
5.1.4. Political Impacts 21
5.1.5. Impacts on Security 21
5.1.6. Impacts on Health & Safety 21
6. Tools and Technologies Used 22
6.1. Development Tools and Frameworks 22
6.1.1. Android Studio 22
6.1.2. Xcode 23
6.1.3. CocoaPods 23
6.1.4. Github 24
6.1.5. NodeJS 24
6.1.6. Firebase 25
6.1.7. Docker 25
6.1.8. ImageMagick 26

6.1.9. Paw and Postman 26
6.1.10.Sketch 27
6.1.11.Adobe Indesign 27
6.1.12. Adobe Illustrator 28
6.1.13. Google Cardboard 28
6.2. Library Resources 29
6.2.1. Google Play Services 29
6.2.2. Android Libraries 29
6.2.3. Google VR Toolkit 30
6.3. APIs Used 30
6.3.1. OpenGL 30
6.3.2. OpenWeatherMap 30
6.3.3. Google Maps 31
6.3.4. Google StreetView 31
6.3.5. Numbeo 31
6.3.6. Foursquare 32
6.3.7. FlightAware 32
6.3.8. OpenExchangeRates 32
7. Engineering Solutions and Contemporary Issues 33
7.1.1. Language and Communication 33
7.1.2. Internet Access and Speed Issues 33
7.1.3. Security Issues 33
7.1.4. Platform Development Issues 34
7.1.5. Data Privacy Issues 34
7.1.6. Gender Issues 35
7.1.7. Future Work 35
8. Appendix A - User Manual 36
8.1.1. Splash Screen 36
8.1.2. Log In / Sign Up Page 37
8.1.3. Home Page 38
8.1.4. Manually Enter Flight Number Page 39
8.1.5. Trip Overview Page 40
8.1.6. Flight Information Page 41
8.1.7. Landmarks Page 42
8.1.8. Near Me Page 43
8.1.9. Venue Details Page 44

8.1.10.Currency Rates Page 45
8.1.11.Currency Rates Page 46
9. Appendix B - Class Diagrams 47
9.1.1. Controllers 51
9.1.2. Models 52
9.1.3. Virtual Reality Related Classes 56
9.1.4. Activities 59
10. Glossary 69
10.1.1.Development Strategy and Methodology 69
10.1.2. Server and Deployment Technology 69
11. References 70

List of Figures

I. Figure 1 - Virtual Reality Panorama Mode Showcase - Page 5

II. Figure 2 - Virtual Reality Stereoscopic Mode Showcase - Page 6

III. Figure 3 - Natural vs Stereoscopic Vision - Page 7

IV. Figure 4 - View Subsystem - Page 11

V. Figure 5 - Controller Subsystem - Page 13

VI. Figure 6 - Logic Tier Subsystem - Page 16

VII. Figure 7 - Data Tier Subsystem - Page 18

VIII. Figure 8 - Figure 8 - Android Logo - Page 22

IX. Figure 9 - Xcode Logo - Page 23

X. Figure 10 - CocoaPods Logo - Page 23

XI. Figure 11 - Github Logo - Page 24

XII. Figure 12 - NodeJS Logo - Page 24

XIII. Figure 13 - Firebase Logo - Page 25

XIV. Figure 14 - Docker Logo - Page 25

XV. Figure 15 - ImageMagick Logo - Page 26

XVI. Figure 16 - Paw Logo - Page 26

XVII. Figure 17 - Postman Logo - Page 26

XVIII. Figure 18 - Sketch Logo - Page 27

XIX. Figure 19 - Adobe Indesign Logo - Page 27

XX. Figure 20 - Adobe Illustrator Logo - Page 28

XXI. Figure 21 - Google Cardboard Illustration - Page 28

XXII. Figure 22 - Google Play Services Logo - Page 29

XXIII. Figure 23 - Android Libraries Logo - Page 29

XXIV. Figure 24 - Google VR Logo - Page 30

XXV. Figure 25 - OpenGL Logo - Page 30

XXVI. Figure 26 - OpenWeatherMap Logo - Page 30

XXVII. Figure 27 - GoogleMaps Logo - Page 31

XXVIII. Figure 28 - Google StreetView Logo - Page 31

XXIX. Figure 29 - Numbeo Logo - Page 31

XXX. Figure 30 - Foursquare Logo - 32

XXXI. Figure 31 - FlightAwareLogo Logo - Page 32

XXXII. Figure 32 - OpenExchangeRates Logo - Page 32

XXXIII.Figure 33 - Splash Screen Page - Page 36
XXXIV.Figure 34 - Welcome Page - Page 37
XXXV.Figure 35 - Sign Up Page - Page 37
XXXVI.Figure 36 - Log In Page - Page 37
XXXVII.Figure 37 - Home Page - Page 38
XXXVIII.Figure 38 - Manually Enter Flight Number - Page 39
XXXIX.Figure 39 - Confirm Flight Number Page - Page 39
XL. Figure 40 - Trip Departure Page - Page 40
XLI.Figure 41 - Trip Arrival Page - Page 40
XLII.Figure 42 - Flight Information Page - Page 41
XLIII.Figure 43 - Landmarks Page - Page 42
XLIV.Figure 44 - Near Me Page - Page 43
XLV.Figure 45 - Venue Details Page - Page 44
XLVI.Figure 46 - Venue Details Page (Scrolled Down) - Page 44
XLVII.Figure 47 - Weather Information Page - Page 45
XLVIII.Figure 48 - Currency Rates Page - Page 46

Definitions, Acronyms and Abbreviations

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

VR: Virtual Reality

API: Application Programming Interface

HTTP: Hypertext Transfer Protocol

SSL: Secure Sockets Layer

UI: User Interface

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

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

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

3D: Three-Dimensional

1. Introduction

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

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

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

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

1. Engineering and Report Writing Standards

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

1.2. Use of New Tools and Technologies

Firebase: Firebase is a mobile and web application development platform. This platform offers users to rapidly develop applications that require user logins, basic databases, encryption facilities and media sharing services[3].

NodeJS: Allows rapid development and deployment for web micro-services. Allows the developer to save precious development time by providing many simple libraries for building applications[4].

Docker: Allows deployment of NodeJS applications with a simple click. Stores all program code in containers and automatically manages them across a wide variety of platforms[5].

Google VR: Provides virtual reality rendering facilities for VR glasses[6].

Swift Lang: Swift is the programming language use to code applications for iOS. Swift is fast, safe and interactive and gives the ability to program for phones, desktops, servers and anything else that runs code[7]. Because LodeStar will have an iPhone application, it was necessary to use Swift for implementation.

Unity: Game Engine. Allows game developers to build sophisticated games without getting involved into shader programming[8].

1.3. Life Long Learning

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

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

1.4. Implementing Creative and Impactful Solutions

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

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

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

2. System Overview

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

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

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

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

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

3. LodeStar's Algorithmic and Application Design

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

3.1. Virtual Reality Algorithm and Implementation

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

Figure 1 - Virtual Reality Panorama Mode Showcase

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

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

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

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

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

Figure 2 - Virtual Reality Stereoscopic Mode Showcase

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

Figure 3 - Natural vs Stereoscopic Vision[53]

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

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

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

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

3.2. User Experience and Interface Design

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

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

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

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

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

4. Final System Architecture Design

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

4.1. Subsystem Overview

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

4.2. Client

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

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

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

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

4.2.1. View

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

Figure 4 - View Subsystem

LoginActivity: This class accomplishes the Login related duties of the application. If the user has an account he/she will be able to use the methods of this class for authentication.
SignUpActivity: This class will accomplish the Signing-up related activities of the user. This will be the first authentication step the user will need to pass in order to be able able to sign-up for the account.
HomeActivity: This class is the view for main page which will open after user authenticates to the application. It will display interfaces for entering flight number so that user may initialize their trip page.
QRCodeActivity: This class is responsible for detecting the Barcode objects, specifically for the type in boarding passes, then parses the information and sends to the TripActivity.
TripActivity: This class is responsible for displaying the Trip page that will show cities in the travel and interfaces to related pages.
TransportActivity: This class is responsible for showing the transportation options with an estimated cost with respect to the transportation costs, such as taxi fare rates, for that country
WeatherInformationActivity: This class is responsible for creating WeatherInformation objects for a given city by fetching 5-day forecast of that city from the server. This class is also responsible for notifying the adapters so that view can be changed.
FlightInfoActivity: This class is responsible for displaying flight details taken from server side of the system according to flight number.
ShoppingActivity: This class displays the shopping areas in that location
CurrencyActivity: This class displays the currency rates between travelled countries.
RestaurantsActivity: This class displays the top quality restaurants in that location
LivingExpensesActivity: This class is responsible for displaying living expenses by giving most common used things as examples.
PlacesToSeeActivity: This class is responsible for fetching the top places to visit in the specified location
AccommodationActivity: This class is for displaying accommodation information taken from the server.
HistoryFragment: This class is responsible for displaying user history of user's trips.
FavoritesFragment: This class is responsible for displaying the user favorites.
VRGamesActivity: This class displays the VR Game/s and allow users to choose a game
GetSIMCardActivity: This class displays instructions on how to get a SIM card on the arrival airport.

4.2.2. Controller

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

Figure 5 - Controller Subsystem

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

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

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

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

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

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

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

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

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

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

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

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

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

4.3. Server

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

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

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

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

4.3.1. Logic Tier

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

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

Figure 6 - Logic Tier Subsystem

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

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

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

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

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

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

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

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

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

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

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

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

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

4.3.2. Data Tier

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

Figure 7 - Data Tier Subsystem

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

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

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

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

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

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

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

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

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

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

5. Impact of Engineering Solutions

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

5.1.1. Economical Impacts

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

5.1.2. Environmental Impacts

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

5.1.3. Social Impacts

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

5.1.4. Political Impacts

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

5.1.5. Impacts on Security

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

5.1.6. Impacts on Health & Safety

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

6. Tools and Technologies Used

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

6.1. Development Tools and Frameworks

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

6.1.1. Android Studio

Figure 8 - Android Logo[26]

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

6.1.2. Xcode

Figure 9 - Xcode Logo[28]

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

6.1.3. CocoaPods

Figure 10 - CocoaPods Logo[29]

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

6.1.4. Github

Figure 11 - Github Logo[27]

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

6.1.5. NodeJS

Figure 12 - NodeJS Logo[4]

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

6.1.6. Firebase

Figure 13 - Firebase Logo[3]

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

6.1.7. Docker

Figure 14 - Docker Logo[30]

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

6.1.8. ImageMagick

Figure 15 - ImageMagick Logo[31]

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

6.1.9. Paw and Postman

Figure 16 - Paw Logo[32]

Figure 17 - Postman Logo[33]

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

6.1.10.Sketch

Figure 18 - Sketch Logo[13]

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

6.1.11.Adobe Indesign

Figure 19 - Adobe InDesign Logo[14]

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

6.1.12. Adobe Illustrator

Figure 20 - Adobe Illustrator Logo[35]

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

6.1.13. Google Cardboard

Figure 21 - Google Cardboard Illustration[12]

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

6.2. Library Resources

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

6.2.1. Google Play Services

Figure 22 - Google Play Services Logo[37]

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

6.2.2. Android Libraries

Figure 23 - Android Libraries Logo[26]

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

6.2.3. Google VR Toolkit

Figure 24 - Google VR Logo[6]

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

6.3. APIs Used

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

6.3.1. OpenGL

Figure 25 - OpenGL Logos[39]

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

6.3.2. OpenWeatherMap

Figure 26 - OpenWeatherMap Logo[20]

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

6.3.3. Google Maps

Figure 27 - GoogleMaps Logo[41]

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

6.3.4. Google StreetView

Figure 28 - Google StreetView Logo[42]

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

6.3.5. Numbeo

Figure 29 - Numbeo Logo[19]

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

6.3.6. Foursquare

Figure 30 - Foursquare Logo[43]

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

6.3.7. FlightAware

Figure 31 - FlightAwareLogo Logo[17]

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

6.3.8. OpenExchangeRates

Figure 32 - OpenExchangeRates Logo[21]

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

7. Engineering Solutions and Contemporary Issues

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

7.1.1. Language and Communication

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

7.1.2. Internet Access and Speed Issues

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

7.1.3. Security Issues

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

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

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

7.1.4. Platform Development Issues

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

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

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

7.1.5. Data Privacy Issues

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

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

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

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

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

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

7.1.6. Gender Issues

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

7.1.7. Future Work

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

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

8. Appendix A - User Manual

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

8.1.1. Splash Screen

Figure 33 - Splash Screen Page

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

8.1.2. Log In / Sign Up Page

Figure 34 - Welcome Page

Figure 35 - Sign Up Page

Figure 36 - Log In Page

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

8.1.3. Home Page

Figure 37 - Home Page

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

8.1.4. Manually Enter Flight Number Page

Figure 38 - Manually Enter Flight Number

Figure 39 - Confirm Flight Number Page

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

8.1.5. Trip Overview Page

Figure 40 - Trip Departure Page

Figure 41 - Trip Arrival Page

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

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

8.1.6. Flight Information Page

Figure 42 - Flight Information Page

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

8.1.7. Landmarks Page

Figure 43 - Landmarks Page

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

8.1.8. Near Me Page

Figure 44 - Near Me Page

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

8.1.9. Venue Details Page

Figure 45 - Venue Details Page

Figure 46 - Venue Details Page (Scrolled Down)

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

8.1.10.Currency Rates Page

Figure 47 - Weather Information Page

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

8.1.11.Currency Rates Page

Figure 48 - Currency Rates Page

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

9. Appendix B - Class Diagrams

NearMeAdapter
-placesList : List<Places>
-context : Context
+NearMeAdapter()
+onCreateViewHolder()
+onBindViewHolder()
+getItemCount()
NearMeHolder
~placePicture : ImageView
~placeName : TextView
~placeIcon : ImageView
~placePhotoAttribution : TextView
~placeStarImages : ImageView[]
~itemView : View
~NearMeHolder()

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

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

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

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

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

9.1.1. Controllers

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

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

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

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

9.1.2. Models

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

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

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

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

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

9.1.3. Virtual Reality Related Classes

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

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

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

9.1.4. Activities

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

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

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

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

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

UserPage

-mRecyclerView : RecyclerView
-mAdapter : Adapter
-mLayoutManager : LayoutManager
-mAuth : FirebaseAuth
~tw : TextView
~tripLogs : TextView
~lastTrip : TextView
-tripCount : TextView
-posts : EditText
~user : FirebaseUser
~userMe : FirebaseUser
+mUser : FirebaseUser
-mDatabase : DatabaseReference
~au : ADDITIONAL_USER
~au2 : ADDITIONAL_USER
~ref : DatabaseReference
-userInfoWithPosts : List<String>
-view : View

+onCreate()
+onCreateView()
+onDataChange()
+onCancelled()
+onStart()
+writePost()
+pickImageAndChangeTheProfilePicture()
+onActivityResult()

Created By: Visual Paradigm Community Edition

VenueActivity

-placeId : String
-photoReferences : ArrayList<String>
-venueReviews : ArrayList<String>
-reviewers : ArrayList<String>
-venueBitmaps : ArrayList<Bitmap>

#onCreate()
-sendPlaceIdToDatabase()
+nearStart()
+goSmall()
+goVr()
+goFullscreen()

Created By: Visual Paradigm Community Edition

LoginActivity

-googleSignInClient : GoogleSignInClient
-mGoogleApiClient : GoogleApiClient
-mLayoutManager : FirebaseAuth
-dbitemcouner : int
~signInButton : SignInButton
~callbackManager : CallbackManager
~lastTrip : DatabaseReference
-userList : ArrayList<String>

+onCreate()
+getArray()
+onDataChange()
+onClick()
+onStart()
+updateUI()
+pickImageAndChangeTheProfilePicture()
+onActivityResult()
+firebaseAuthWithGoogle()
+panoStart()
+VRstart()
+meStart()
+tabStart()
+placesToSeeStart()
+signUpStart()
+signUpPlease()
+userSearchStart()
+onConnectionFailed()
+signIn()

Created By: Visual Paradigm Community Edition

ManualDestinationActivity

-dropdown : Spinner
-countries : String[]

+onCreate()

NearMeActivity

-recyclerView1 : RecyclerView
-recyclerView2 : RecyclerView
-recyclerView3 : RecyclerView
-REQUEST_LOCATION : int
~fusedLocationProviderClient : FusedLocationProviderClient
~ll1 : LinearLayout
~ll2 : LinearLayout
~restList : List<Places>
~parkList : List<Places>
~museumList : List<Places>
~nmc : NearMeController
~venueImageURL : String[]
~iconURLs : ArrayList<String>
~ll3 : LinearLayout
~adapter : Adapter<ViewHolder>
~adapter2 : Adapter<ViewHolder>
-adapter3 : Adapter<ViewHolder>

+onCreate()
+tripStart()
+checkForLocationPermission()
+requestPermissionForLocation()
+getLastKnowLocation()
+onRequestPermissionsResult()
+parseTheJSONObject()
+getPlacesIcon()

PanaromaActivity

-pv : PanoramaView
-sManager : SensorManager
-rotationVectorSensor : Sensor
-gyroscopeSensor : Sensor
~gameRotatiton : Sensor
~accelSensor : Sensor
~magnSensor : Sensor
~angle : float

+onCreate()
+onResume()
+onStop()
+goFullscreen()
+goSmall()
+goVr()

PlacesToSee

-recyclerView : RecyclerView
-googleMap : GoogleMap
-placesToSeeController : PlacesToSeeController
-REQUEST_LOCATION : int
~fusedLocationProviderClient : FusedLocationProviderClient
~layoutManager : LayoutManager
~ll1 : LinearLayout
~placesList : List<Places>
~photoReferences : ArrayList<String>
~iconURL : ArrayList<String>
~TAG : String
~builder : Builder
~adapter : Adapter<ViewHolder>

+onCreate()
+onMapReady()
+checkForLocationPermission()
+requestPermissionForLocation()
+getLastKnowLocation()
+onRequestPermissionsResult()
+parseTheJSONObject()
+tripStart()
+getPlacesPhotos()

PlacesToSeeExpandedActivity

-recyclerView : String
-placeId : String
-placesToSeeController : ArrayList<String>
-landmarkReviews : ArrayList<String>
~reviewers : ArrayList<String>
~ratings : ArrayList<Double>
~landmarkBitmaps : ArrayList<Bitmap>
~au : ADDITIONAL_USER
~mDatabase : DatabaseReference
~pv : PanoramaView
~adapter : Adapter
~adapter3 : Adapter
~adapter2 : Adapter
~slider : Slider
~slider3 : Slider
~slider2 : Slider
~placeName : String
~placeStarImages : ImageView[]
~t11 : TextView
~gameRotatiton : Sensor
~sManager : SensorManager
~API : String
~cv1 : CardView
~cv2 : CardView
~cv3 : CardView

+onCreate()
+sendLandmarkPlaceIdsToDatabase()
+onResume()
+onCancelled()
+getLastKnowLocation()
+retrieveMoreInformationsAndPhotos()
+goVr()
+onPageScrollStateChanged()
+goSmall()
+goFullscreen()
+onSliderClick()
+onStop()
+onPageSelected()
+parseMoreInformationAndPhotos()
+setSlider()
+onPageScrolled()

PreferencesActivity

-changePassword : Button
-changePassword3 : Button
-logout : Button
-aboutLDSTR : Button
-changeUsername : Button
-db : FirebaseDatabase
~mAuth : FirebaseAuth

+onCreate()
+meStart()
+changeUNAME()
+changePasswordPref()

DirectedUserPage

-myEmail : String
-tw : TextView
-logout : Button
-tripLogs : TextView
-lastTrip : TextView
-tripCounr : TextView
-user : FirebaseUser
-au : ADDITIONAL_USER
-ref : DatabaseReference
-changeUsername : Button
-db : FirebaseDatabase
~mAuth : FirebaseAuth

+onCreate()
+setup()
+onCancelled()

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

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

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

SignInActivity
-emailF : EditText -passwordF : EditText -signButton : Button -txtViewtoSignUp : TextView -pDialog : ProgressDialog -managerDB : FirebaseAuth
+onCreate() +onClick() +onBackPressed() +tryToLogIn()

Powered By Visual Paradigm Community Edition

TripActivity
-view_flipper : ViewFlipper -firstView : View -secondView : View -flightInfo : FlightInfo -qrCodeInfo : QRCodeInfo -origCity : String -destCity : String -flc : FlightInfoController -trp : TripController -photoReferenceOrig : String -photoReferenceDest : String -backgroundImageWidth : int
+onCreate() +onCreateView() +onWindowFocusChanged() +updateSizeInfo() +flightInfoStart() +sendRequest() +sendImageRequestOrig() +sendImageRequestDest() +isStoragePermissionGranted()

Powered By Visual Paradigm Community Edition

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

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

FavoritesFragment

-favPlaces : ArrayList<String>
-ref : DatabaseReference
-mParam1 : String
-mParam2 : String

+onCreate()
+databaseToArrayList()
+onCreateView()

PlacesToSeeActivity

-googleMap : GoogleMap
-placesToSeeController : PlacesToSeeController
-fusedLocationProviderClient : FusedLocationProviderClient
-recyclerView : RecyclerView
-adapter : Adapter
-layoutManager : LayoutManager
-ll1 : LinearLayout
-placesList : List<Places>
-photoReferences : ArrayList<String>
-iconURL : ArrayList<String>
-builder : Builder

+onCreate()
+onMapReady()
-getLastKnowLocation()
+onSuccess()
+onPlaceImageSuccess()
-checkForLocationPermission()
-requestPermissionForLocation()
+onRequestPermissionsResult()
-parseTheJSONObject()
-getPlacesIcons()
-getPlacesPhotos()
+tripStart()

DirectedUserPage

-tripCountr : TextView
-tw : TextView
-logout : FirebaseUser
-tripLogs : ADDITIONAL_USER
-lastTrip : TextView
-tripCounr : TextView
-user : FirebaseUser
-au : ADDITIONAL_USER
-ref : DatabaseReference

+onCreate()
+setup()

10. Glossary

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

10.1.1.Development Strategy and Methodology

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

10.1.2. Server and Deployment Technology

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

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

11. References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[This page is intentionally left blank]

[end of report]