On-the-go Flight cum Journey Tracker - Fleeto

Rupesh Joshi
SBU ID# 109486820
Department of Electrical & Computer Engineering
Stony Brook University

Vivek Ramji
SBU ID# 110084822
Department of Electrical & Computer Engineering
Stony Brook University

***Abstract*-This report provides a detailed explanation about the problem at hand, the motive behind the application and the various technicalities that went into the successful running of the application.**

I. INTRODUCTION

This Android application was developed in Java using Android Studio and serves to provide a real-time delay and flight tracking functionality to the user, especially on-the-go.

A. Motivation

Today's world is on the move, with a number of motive like business meetings, marriage, relocation, permanent immigrant migration, education, etc. Any form of journey demands punctuality, with respect to the mode of transportation, be it by train or flight.

Between 2012 and 2013, 35.9 million people 1 year and over living in the United States moved to a different residence. The mover rate for this period was 11.7 percent. According to the Bureau of Transportation Statistics, a total of 631,939,829 passengers boarded domestic flights in the United States in the year 2010. This averages to 1.73 million passengers flying per day.

We often encounter instances where we are not sure about the right time to leave home, in order to board a flight or we have no idea how the traffic or the weather would change and thereby, alter our time to reach the airport. Sometimes, change in timings by the airport authority also creates a lot of chaos and confusion.

All that is needed, in an on-the-go technology that keeps the user updated at any point, as to whether he is in a position to make it to the flight, and how much time he has, in hand so that he can plan for the journey well in advance. The technology should also incorporate unpredictable parameters like harsh weather, road constructions and other potential delays so that the user knows that the path he is taking or the weather after a few minutes is going to cause delay in the journey.

B. Project Idea

As a solution to the problem above, we have designed an application that will dynamically extract flight departure timings, any changes in status and suggestions, with a dynamic traffic update feature and alternate path suggestion feature in case of heavy traffic, taking into account your current position. It will also come with weather updates to

keep you notified about any unexpected climatic scenarios. On forecasting a rough weather along the path, due care is taken to add an appropriate amount of buffer time to the actual time for the journey. Additionally, the application will consider an average speed of journey and tell you if you are on time, ahead of time or before time, to give you a sense of urgency in your quest to reach the airport on time.

The most unique feature about our application is the fact that, it provides a user with a personal registered account. The user can log into his account at any point and make a quick search. The account also stores the history of all the user's past searches so that the user can repeatedly check the status of a particular search without having to type in all the input details again and again.

II. DESIGN

The Android application, in our project, spans over a number of activities or pages. In this section, we'll be describing the functional aspects of every activity in detail.

C. Module Design

To begin with, the application provides a user with his own account. The **Login Page** helps in establishing authentication of the identity by verifying the username and password. Both of these are **EditText** entities. In addition to this, the page also has a **Submit** button, which, when clicked, performs the authentication of the identity of the user. In case the user happens to be a first time user who does not have his own logic credentials, there exists a **Register** button, which when clicked, allows the user to register himself.

Submit Button – As mentioned earlier, this button, when clicked, helps perform verification and authentication of the user's credentials. When the user enters his/her username and password, the algorithm extracts these details and constructs an SQL query and runs it over the AMAZON Cloud EC2 Database, which holds the credentials of all the registered users. The query matches the username and password with the database entries and tries to establish a authentic connection with the details associated with the user. There are two possible scenarios that could arise at this point. Either the username and password do not match, which means that the user is concluded to be unregistered and is, therefore, redirected to the **Login Page**, where in, he is prompted to enter his username and password again; or there is a match found between the username and password entered, and his identity is authentic. The user, is then, concluded to be registered and genuine. The application, immediately takes the user to the History Page, the details of which will be described in the next section.

History Page – The History Page displays all the past searches made by the user using the account. This is obtained from the **Amazon Cloud EC2 Database**, where, the details of his/her previous searches are populated to form a list associated with his user ID (UID). The retrieval of this list is done by constructing an SQL query as soon as the SUBMIT button is clicked and the identity of the user is authenticated. The list of past searches include details like Flight ID, the date for the search conducted and the Boarding Point. It should be noted here, that every history item obtained in the page could be susceptible to a normal-press and a long-press. The long-press on an item pops a choice for deletion of that entry, to the user. If the user selects YES, the entry is deleted from the database, and will not appear in the future, in the search history page. The normal-press takes the required credentials from that

particular search entry and make a fresh query to the API (FlightStats), and returns the necessary details on the Result Page, which is described after the Input Page below.

Input Page – Once the user is authenticated and is successfully logged into his account, he can make a fresh query to the API regarding details of a specific flight. The Input Page is basically a page form that enables him to enter certain required credentials required for querying the API, like the Flight Carrier, the Flight ID and the Date using the Date Spinner. The Date Spinner limits the user to request flight details over the next three days including the present day. This page contains two buttons – the **SUBMIT** button and the **CLEAR** button. The CLEAR button clear all the fields entered. The user can use this button if he feels that the credentials entered are incorrect. The SUBMIT button takes the user to the **RESULT PAGE**, where the necessary details about the flight and the journey is stated.

Fig 1. Login Page

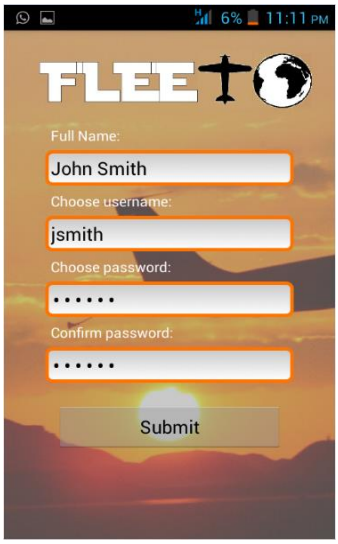


Fig 2. Registration Page

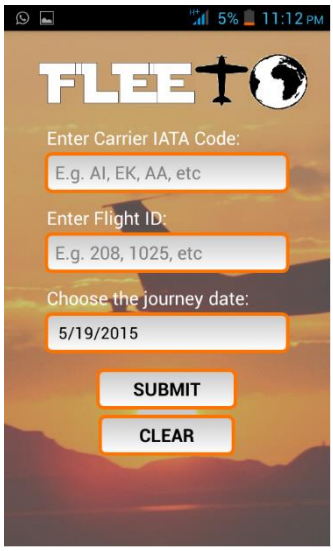


Fig 3. Input Page

Result Page – The Result Page displays necessary information pertaining to the flight and the journey. The API used to obtain the results is Flightstats, which is widely used service for tracking commercial aviation flights. The search credentials are extracted from the Input Page and then, used to build a URL which accesses the Flightstats service and database to return a XML or JSON response of the required information.

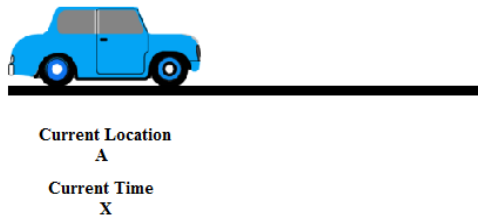
The Flightstats URL is built as follows:

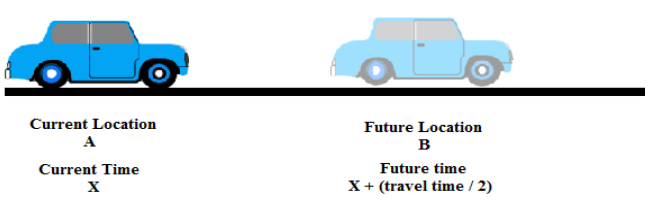
https://api.flightstats.com/flex/flightstatus/rest/v2/<response_format>/flight/status/<carrier>/<number>/dep/<year>/<month>/<day>?appId=<xxxxxxxxx>&appKey=<xxxxxxxxxxxxxxxxxxxx>&utc=false

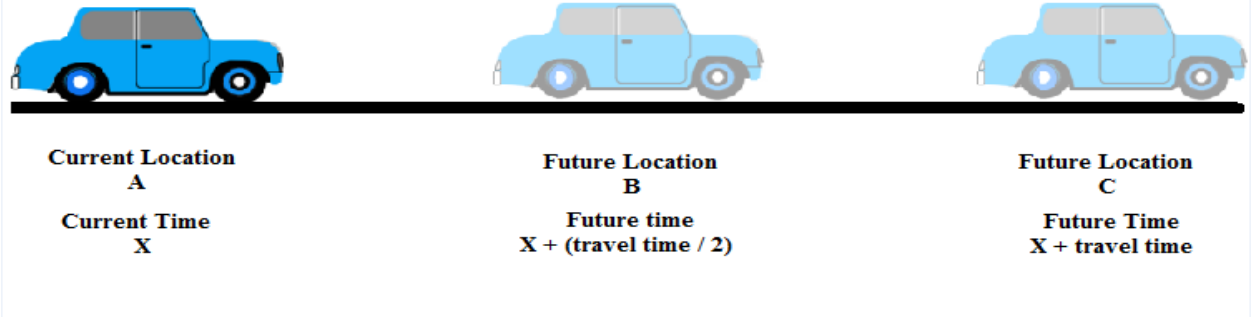
We, as the developers of this particular application, chose to work with the XML response of the API. The XML response needs to be parsed to obtain the essential details from the response. The response will contain details that might be irrelevant to the purpose of our application and therefore, due attention was paid in building the parser to fetch the desired results only. The SAX Parser was used for parsing the response and return the necessary details. The details include the flight carrier, the flight ID, the Airline service operating the flight, the boarding point, the

date of boarding, the scheduled gate departure time and the terminal number. The details presented in the Result Page and categorized into two groups. The first group deals with details related to the flight, and consist of the parameters mentioned above. The second group deals with delay statistics. This includes the number of minutes the flight is expected to be delayed, the time left for the user to board the flight from the current time, and the dalay that could be expected because of uncertain climatic conditions. The time left for the user to board the flight is represented in a “days:hour:minutes” format which is evaluated by subtracting the current time from the scheduled gate departure time.

The forecasted delay for the future is evaluated using an algorithm we have managed to devise for this particular application. Suppose that the user is at point A and the airport is located at point B, we’ve split this distance into four equal distances and determined a **QPF** at the three intermediate points and the QPF at the source and destination points, as well. However, this isn’t done simultaneously for all the points. The time required to reach B from A is first evaluated. The QPF at every point would be determined at the expected time of arrival at that point. For convenience of calculation, we’ve assumed that every point of QPF would add one minute of delay to the total travel time. Therefore, the total delay would be the sum of QPFs at all the points including the start and the destination points.







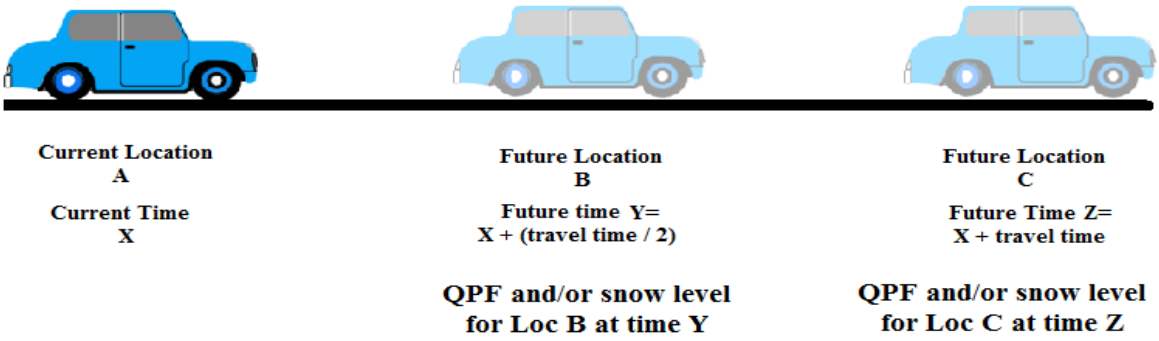


Fig 4. QPF to determine delay

This page also include three buttons – the Check Climate button, the Start Navigation button and the Get Real Time Data button.

The **Check Climate** button takes the user to a page where hourly weather forecasts are given with a green tick or attention symbol, hinting to the user that he need not or has to worry about harsh weather, respectively.

The API used for this purpose is WUnderground whose URL is built as follows:

http://api.wunderground.com/api/50f4532de367c7e9/hourly/q/<latitude>,<longitude>.xml

The Start Navigation button takes the user to a Google Map interface, where he could visually observe or track details. More about this, is described later.

The **Get Real Time Data** button request a fresh query to the Flightstats API and returns an updates XML/JSON response, which is then parsed to display the updated results. It is to be noted that the request is made to the API again so that the delay, in specific, gets updated. The other details remain static.

The **Start Navigation** button enables the user to track his race against time, in real-time. This button takes the user to a Map fragment layout. We have used the Google Maps API for this purpose, as the Google Maps API comes with a number of useful functions that calculate the travel time and traffic density between locations, with a single line of code. The map layout enabled the user to choose the satellite view, a terrain view of a normal/default view. Three toggle buttons embedded on the map toggle navigation, icons and the traffic respectively. The navigation toggle button switched view from the default map view to a tracker, where the screen follows the location of the device. The icon toggle button hides or displays the construction/incidence icons, depending on the toggle status. The API used for this purpose is the Mapquest API whose URL is built as follows.

http://www.mapquestapi.com/traffic/v2/incidents?key=Jmjtd%7Clu6yn907nu%2C8w%3Do5-lw8x9&callback=handleIncidentsResponse&boundingBox=destlat,destlong,srclat,srclong&filters=construction,incidents&inFormat=kvp&outFormat=xml

These icons represent those areas in the route where construction work is in progress and there are good chances that this will add to the delay to the journey. The traffic toggle button sets the visibility of the traffic density lying along the route. The traffic lines give a general idea as to whether there exists heavy traffic along the route of the journey. This is indicated by red lines on the map, whereas, manageable traffic is indicated by yellow contour lines.

To provide a live tracking feel, the Start Navigation activity also has embedded on it, a textbox, which dynamically updates the distance and time remaining from the current location to the destination.

This update runs every minutes, so the distance and time is updated very frequently. If the user wants explores the map beyond the view visible surrounding his current location, he may find it difficult to navigate back to his current location on the map. A button embedded on the map does this for the user.

When the user clicks on it, the view automatically shifts to the frame containing the blue dot, which signifies the current location of the user. Another button named **Re-Calculate** re-plots everything including the shortest route and the markers to the destination from the user's current position. This replaces the previous route, thereby preventing the map from cluttering and making it difficult for the user to interpret and track.

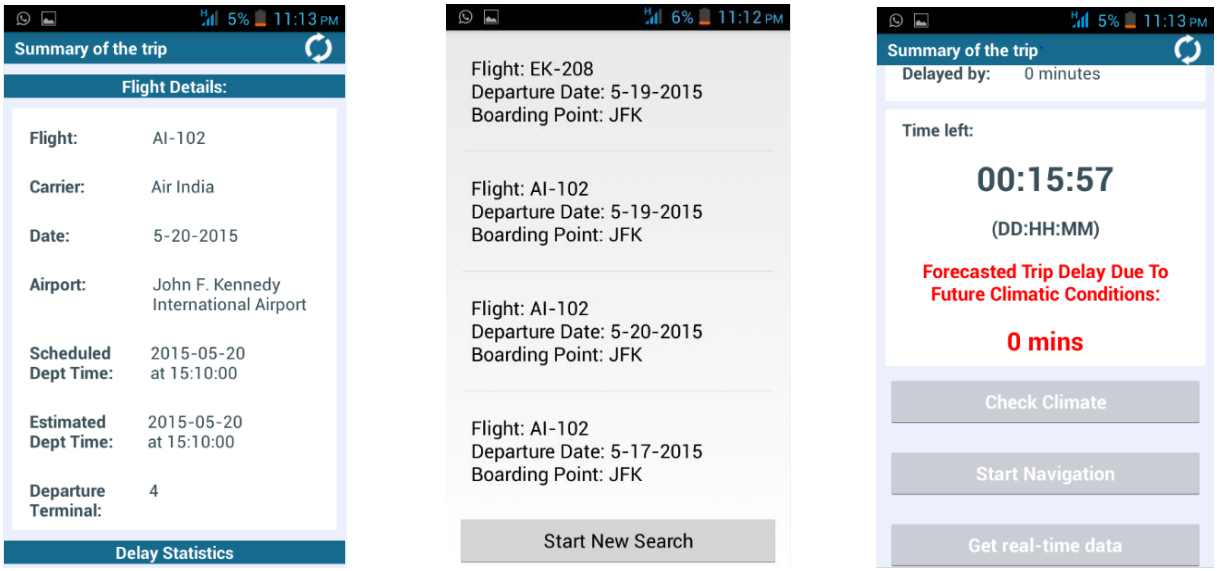


Fig 5. Result Page

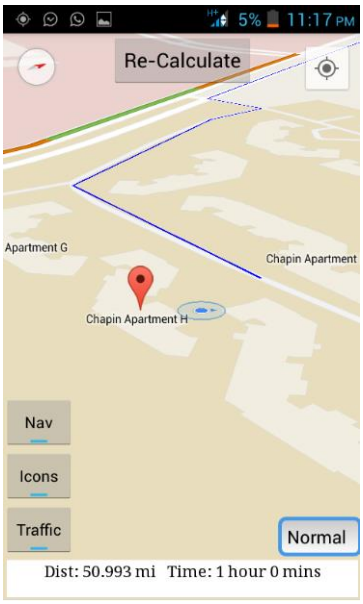


Fig 6. Start Navigation Activity

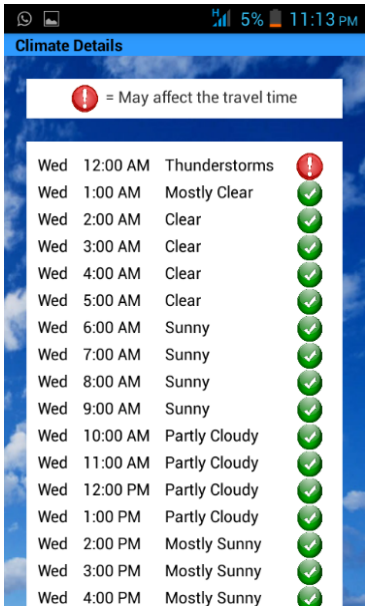


Fig 7. Check Climate Acitvity

D. Block Diagram

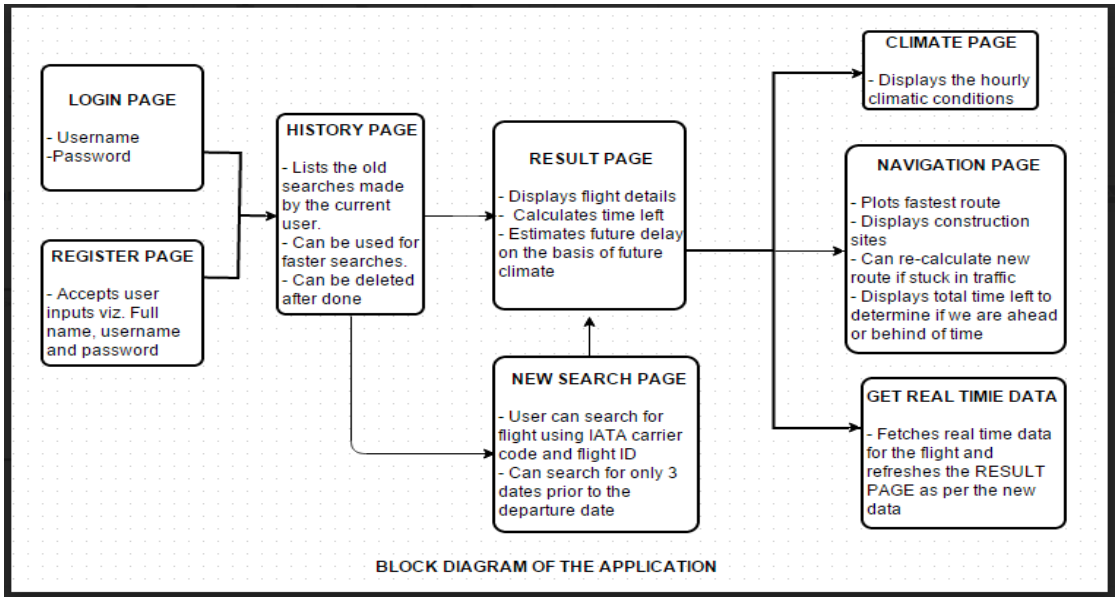


Fig 8. Block Diagram of the Application

E. Functions

During the running of the entire application, there are a number of functions that run to make it run successfully. The following segment describes every function that renders the application useful.

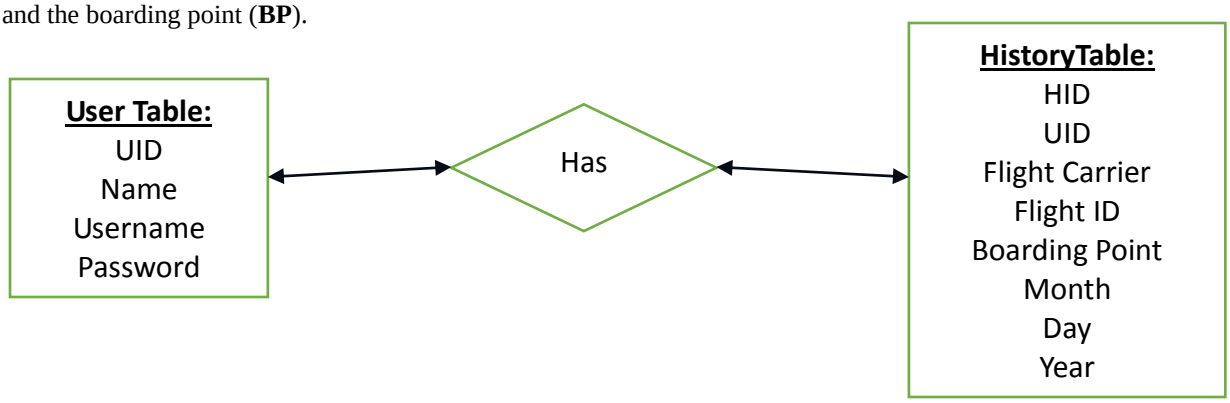
Popup – This is a small dialogue box that pops up as soon as Start Navigation is invoked. This popup dialogue box displays the distance to be covered as per the shortest distance, from the user's current location to the destination airport, the time left in hand before which, the user can successfully board the flight and the amount of fuel consumption in gallons, that is expected to take place in order to cover the said distance. In the background, as soon as the Map activity is started, the application does not wait for the device to connect to the GPS to obtain the co-ordinates and the location of the user. Instead, the application user the network tower to obtain an approximate location of the user and plots the point the on the map. But during this time, the GPS will be activated to connect to the satellite. As soon as the GPS connects to the satellite, the location service is handed off from over the network provider's service to GPS, which gives the precise location of the user.

While the GPS tries to connect to the satellite, the application plots the shortest path to the destination using the approximate location of the network service. As soon as the GPS connects to the satellite and the accurate location is obtained, the application recaliberates the shortest path to the destination and the previously plotted path will be removed and replaced by the new shortest path. Also, construction and incident locations will be obtained from Google Maps API and plotted on the map, as this helps the user determine the possible amount of delay that could be expected due to the constructions that lie on his path to the destination. If the user feels stranded on the way and wishes to calculate a new shortest path to the destination, he can simply click the **Re-Calc** button and a new route will be calculated, which will replace the previously calculated route as mentioned previously.

F. The Database

The database used to store credentials of registered users and their search history is the **Amazon EC2 Web Service (AWS)**, an instance of which runs on a Linux Server. The two tables in the database has a many-to-many relationship between themselves. Table 1 is named **Users** as it stores the users' account credentials which are used at the time of authentication and log in. This table consists for four columns – the user ID (**UID**) which acts as the primary key for this table, the full name of the user (**Fullname**), the username (**Username**) and the password (**Password**). The password is encrypted to the MD5 format.

Table 2 is named **History_table** as it stores the Users' search history. Clearly, this should be related to Table 1 in some manner, as it tries to establish a relation between each user and his search history. Therefore, the user ID (**UID**) from Table 1 acts as the foreign key to this table. The primary key to this table is the History ID (**HID**). The other tables include details a particular search like the Flight Carrier, Flight ID (**FID**), the month, the day, the year and the boarding point (**BP**).



Register Page – The Login Page redirects new users to the Register Page. This helps a new user create an account for himself. This page asks the user to enter his full name, a username of his choice, a password of his choice and a re-entry of the same password to check for mismatch. The validation of the username is done, in order to avoid duplicates and prevent two users from having the same username. If all of the above are successful, the registration is successful and an SQL query pushes all the details to the database and creates a new User ID (UID) for the user. Henceforth, if the user wants to log in, he can simply enter the username and password in the Login Page and access his account.

II. TECHNICAL CHALLENGES

G. Configuring Database to AWS

Configuring Database to AWS – Once we managed to create an instance of a Linux virtual machine, we tried to run a database on the virtual machine. After initial hiccups, we managed to get the database running on the machine. However, we still couldn't connect to the database from our remote machine. This problem was resolved by adding the Group Admin to the Security Group and then making due changes to the Config file on the virtual machine.

H. Route Decoding Algorithm

This algorithm helps in plotting the path or the 'polyline' on the Google Map based on a set of coordinates. This is fondly referred to as Encoded Polyline Algorithm Format by Google. A brief description of the algorithm is as follows. A decimal value of a coordinate is first multiplied by 10^5 and then the result is rounded off. This value is then converted to binary. Note that a negative value must be calculated using it's two's complement by inverting the binary value and adding one to the result. This binary value is if left-shifted by one bit. If the original decimal value is negative, this encoded value is inverted. This string now is broken into 5-bit chunks starting from the right hand side. These 5-bit chunks are placed in reverse order. Every chunk is then OR'd with 0x20 if there's another chunk follows the current one. Every resulting chunk is then converted to decimal and then summed with 63. The resulting chunks are then converted to its ASCIII equivalent.

The following is an example using this algorithm.

Let the coordinate be -179.9832104

The Decimal value is multiplied by 105 and then rounded off.

-17998321

Negative values are two's complemented and 1 is added.

00000001 00010010 10100001 11110001

11111110 11101101 01011110 00001110

11111110 11101101 01011110 00001111

Left-shift by one bit

11111101 11011010 10111100 00011110

Invert this is the original value is negative

00000010 00100101 01000011 11100001

Break the binary value into 5-bit chunks from the right.

00001 00010 01010 10000 11111 00001

Reverse the order

00001 11111 10000 01010 00010 00001

OR each chunk with 0x20 if there's a chunk following the current one.

100001 111111 110000 101010 100010 000001

Convert each value to decimal

33 63 48 42 34 1

Add 63 to each value

96 126 111 105 97 64

Convert each value to its ASCII equivalent

`~oia@

I. Hand-off from network service to GPS service

This point was also discussed earlier. Initially, while developing the application, we did not expect that the GPS would take a few minutes before connecting to the satellite and extracting the exact location of the device. After a number of trials, we decided to proceed with extracting the location from the network provider initially, and then handing off the service to the GPS once the GPS successfully establishes connection with the satellite. We had to compromise on accuracy initially, as the location obtained from the network provider is not accurate as that of the GPS.

J. Future Delay Forecast

The QPF is used to determine the delay, as described earlier.

K. API Interaction

As mentioned earlier in the report, the application forms a URL based on the input details given by the user and fetches data from the API. The API returns a response in either JSON or XML format. This needs to be parsed and the required data needs to be carefully extracted to utilize. We chose to work with the XML response and we used the SAX Parser to parse the response. The Simple API for XML parser (commonly called the SAX parser) is an event-driven online algorithm for parsing XML documents, with an API interface developed by the XML-DEV mailing list. SAX provides a mechanism for reading data from an XML document that is an alternative to that

provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially.

A parser that implements SAX (i.e., a SAX Parser) functions as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include (among others):

XML Text nodes

XML Element Starts and Ends

XML Processing Instructions

We spent a good many days trying to get the logic of the parser right, since it was difficult to decipher the response easily. After a number of attempts, we started to realize that the format of the response is little different for a different combination of inputs. We then managed to collect all the possible formats of response and come up with a common, robust logic for the parser, after which, the application began to run.

III. Evaluation

LOGIN PAGE			
CASES	**EXPECTED OUTPUT**	**ACTUAL OUTPUT**	**RESULT**
Incorrect Username	*"Incorrect User credentials. Try Again"*	*"Incorrect User credentials. Try Again"*	Pass
Inocrrect Password	*"Incorrect User credentials. Try Again"*	*"Incorrect User credentials. Try Again"*	Pass
Incorrect Username & Password	*"Incorrect User credentials. Try Again"*	*"Incorrect User credentials. Try Again"*	Pass
Correct Username & Password	*"Successful Login"* History Page activity begins	*"Successful Login"* History Page activity begins	Pass

REGISTER PAGE			
CASES	**EXPECTED OUTPUT**	**ACTUAL OUTPUT**	**RESULT**
Non-unique Username	*"Username taken. Try a different name"*	*"Username taken. Try a different name"*	Pass
Password mismatch	*"Passwords do not match. Try again"*	*"Passwords do not match. Try again"*	Pass
Field(s) left blank	*"Fields cannot be left blank. Try again"*	*"Fields cannot be left blank. Try again"*	Pass
Accurate Details	*"Successful registration"*	*"Successful registration"*	Pass

INPUT PAGE			
CASES	**EXPECTED OUTPUT**	**ACTUAL OUTPUT**	**RESULT**
Invalid Carrier Name	*"Invalid Flight Details. Try again"* Destroy result activity and return	*"Invalid Flight Details. Try again"* Destroy result activity and return	Pass
Invalid Flight ID	*"Invalid Flight Details. Try again"* Destroy result activity and return	*"Invalid Flight Details. Try again"* Destroy result activity and return	Pass
Accurate Details	Result Page activity begins	Result Page activity begins	Pass

IV. Future Scope

In the future, we plan on implementing a few features that would further enhance user experience and provide additional information for navigation. The following new features are what's on our mind –

Modes of transport – Along with the delay prediction, we would ideally like to implement a feature that could suggest the best mode of transport to reach the destination, including public transport systems like subways and buses.

Void guided navigation – this is essential for car owners as it helps the users to utilize the application hands-free and get turn-to-turn voice assistance.

Accurate Future Delay Predictor – The future delay prediction algorithm can be further enhanced and calibrated to be more accurate.

Push Notification – This is important as any changes in the flight status would affect the user. So, to keep him updated and not wait for him to manually check the status, we would like to incorporate a push notification system that would alert him of the changes irrespective of whether he checks manually or not.

IV. Conclusion

To conclude, we would like to reiterate the importance of this application in day-to-day lives of a number of travelers and businessmen who are constantly on the road for business trips, tours and vacations. This application is essential to the user to keep him updated with the status of the flight and assist him in reaching the airport at any condition.

V. References

1) FlightStats API - https://developer.flightstats.com/api-docs/flightstatus/v2/flight

2) WUndergound API for weather updates - http://www.wunderground.com/weather/api/

3) Google Maps API v2 for directions - https://developers.google.com/maps/documentation/directions/

4) Mapquest API for Traffic Updates - http://developer.mapquest.com/web/products/dev-services/traffic-ws

5) Google Maps API v2 - https://developers.google.com/maps/documentation/android/