Software Requirements Specification for the application : WhenBus

Rahul Ramesh, Girish Raguvir, R Gowrisankar, Seshagiri Rao, K Manoj Kumar
Computer Science and Engineering
Indian Institute of Techonology Madras

***Abstract*—WhenBus is a tool that facilitates Chennai MTC Bus users conveniently navigate bus stops, bus routes and find bus timings. The tool relies on the concept of Crowdsourcing in order to obtain accurate bus timings and positional tracking. Users, via this application, can find the nearest bus stop and search for any bus to track. The application on consulting its database, finds its estimated arrival time and position of the specified bus. Once the user is on the bus, the application will detect the movement of the user and consequently update the database based on a couple of queries to the user.**

I. Introduction

The Chennai MTC Buses are very affordable modes of transportation for many in the city. Our tool WhenBus aims make this process more convenient to bus users, by providing accurate bus timings and locations and nearby available bus stops based on real time data. The motivation behind building this tool is to ensure that the existing bus timing schedules are static in nature. Our application aims address this problem in a user friendly manner. The central theme of this application is the use of Crowdsourcing. The application will be built to be used on a mobile device and can possibly be extended to a variety of transportation networks since they all share multiple component features.

The basic features of the application and its usage is as follows:

- The user of app need not have any account to use the app.
- The user should provide access to his location and he is consequently mapped to the nearest bus stop.
- The user can enquire about the next bus at a given bus stop regarding the timing and position

The Software engineering approach that we will follow will be along the lines of incremental development. This software development philosophy is an agile model that borrows components from the rigid waterfall model. It is allows revisiting phases in the project and offers flexibility.

The first version is going to be a desktop version with the user input simulated and the backend API on the server with only few bus stops in the database. In next version, the app can possibly be migrated to mobile device running android/hybrid and the user inputs be more realistic data in order to test robustness and answers from users and API and database being the same. While in the final, the app can be made to work on larger group of bus stops by updating the database and by increasing user base.

The rest of this document deals with the description, requirement specification and control flow of the application.

II. General Description

The central motivation of WhenBus is crowdsourcing, basically, a set of users provide the required inputs which shall be processed and the results shall be used to

answer the queries of other users. So WhenBus have to provide two interfaces. One, to input the details of various MTC bus routes position and time. Second, for users to query the expected arrival of a bus at the nearest bus stop.

WhenBus shall primarily rely on the MTC databases of bus routes and tentative time tables. The data collected from crowdsourcing shall be stored in a separate database which shall be later used to run heuristics to modify the tentative schedule. We may also opt to retain some optimal number of histories of the timetable to determine the similarities(most probable time table) /anomalies (like the marina protest week when the MTC bus services were affected very much).

The first interface shall be simple and quick enough for the users to fill in the data on the go. Data, like location and the time at that location of the user shall be extracted by default in the background which shall undergo some sanity check(like the coordinates should be within 500m of the actual route and continue to do so) to prevent erroneous data from creeping into the database. The second interface shall be designed in a way that it is easier to interpret the response for the query of a particular route - time and its position.

III. Requirements specification

A. Functional Requirements

The main functional requirements of WhenBus are :

1) WhenBus shall provide a simple and quick interface (typically a single scroll page with minimal typing and buttons for selection) for the user end to extract the following:
   - Bus route (in which the user is currently)
   - Direction of the bus
   - Current location and time (extracted from the users mobile/system)
2) WhenBus shall maintain a database of all MTC bus routes and tentative schedule in the server (original and/or the updated one) which can be used to provide response for the queries of any bus route.
3) WhenBus shall check the validity of the data and run heuristics periodically to update the tentative schedule.
4) In response to a query about a bus route by a user, WhenBus shall provide the user with the expected arrival time of the bus to the bus stop nearest to the user and also maybe a map interface to track the bus in real time.
5) WhenBus shall provide means to download the tentative time table of the bus routes by the user.

B. Non-Functional Requirements

This section deals with requirements that generally deal with evaluation and performance of the applications on various aspects. Some reasonable expectations are :

1) The ideal response time for queries made by the user should be less than 2s.
2) WhenBus should refrain from extracting or asking for any other data from the user which will reveal the identity of the user and shall provide means to track them, thus respecting privacy. All users are anonymous with respect to the server.
3) The application must refrain from providing wildly incorrect responses and the errors must be within a reasonable range.

IV. Tentative Flow Diagram

The flow diagram describes a visual representation of how the entire applications works. The main entities present in the model are :

- Client Side program
- Application User
- Server Side program
- Database
- MTC Bus

The application can be understood as multiple client and server transactions and is composed primarily of two operations.

1) The User makes a request for a certain Bus. The Server side program consults the database and suggests nearest Bus stops and also furnishes details regarding the bus position and timing.
2) The Client side program asks the user if he has boarded the bus. The Server side program processes the response and the databases are updated accordingly.

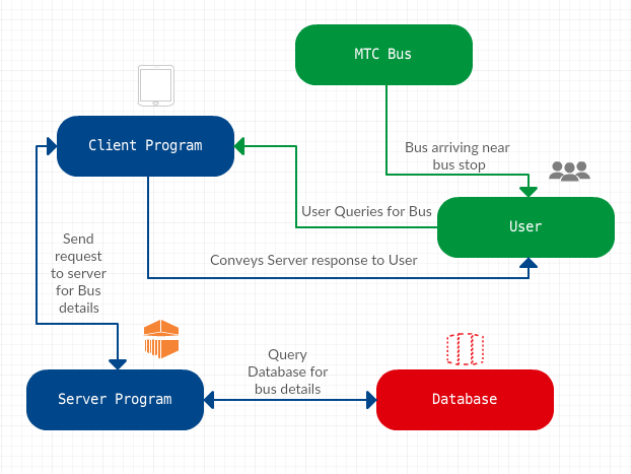

Fig. 1. User Query

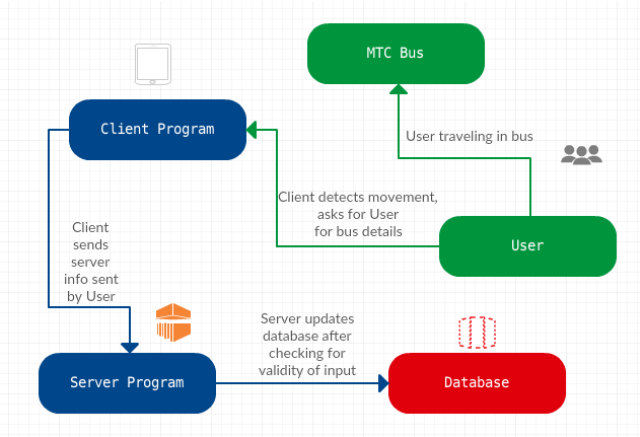

Fig. 2. Database Update

The Figure:1 describe a scenario where the user queries a certain bus number. The client sends the server a request. The server consults the database and after using multiple heuristics, gives a detailed reply to the user via the client.

Figure:2 describes the seconds scenario where a database update is required. This image, clearly highlights the crowd sourcing component of the application. The client sends the data collected from the user to the server. The server then updates the database using the client data after running a few validity checks.

V. Conclusion

The WhenBus application is currently under the planning phase. It aims to provide improved the usability of MTC buses in Chennai but the central idea of the application can be extended to any transportation network with dynamically changing timings. It is also possible to eliminate the crowdsourcing aspect of the application if we install physical sensors however this results in huge deployment and maintenance cost especially for large transport networks.

References

[1] Paper structure from : http://www1.cmc.edu/pages/faculty/DNeedell/papers/ucs_specs.pdf
[2] Software Engineering (9th Edition), Ian Sommerville, et al.
[3] Chennai MTC : http://www.mtcbus.org/

Z-Description : WhenBus

Rahul Ramesh, Girish Raguvir, R Gowrisankar, Seshagiri Rao, K Manoj Kumar

1 SCENARIO DESCRIPTION

The entire state of the WhenBus system can be characterized broadly via 7 states. We can consider WHENBUS to be a variable that captures the state of the application and it can be represented by :

$$WHENBUS\_STATE = STARTUP \land IDLE \land DETECT \land BUS\_TRAVEL \land STOP\_QUERY$$

$$\land BUS\_QUERY \land UPDATE\_TABLE$$

We will look at what each of these states comprise of and look at an in depth schema for the same. In this Z specification, we do not go into great details of the working of the algorithm but instead assume consider the heuristics and other approximation algorithms as black-boxes.

2 VARIABLE DESCRIPTION

This sections deals with the various variables that we will use in the Schema description and what they actually represent

- *status* : Indicates if the application is working correctly
- *stop_id* : A set of all bus stop ID's
- *stop_name* : A set of bus stop names(each for a particular ID)
- *stop_gps_coord* : A function mapping bus stop ID's to particular co-ordinates
- *gps_coord* : The GPS co-ordinate of the user
- *nearest_stop_loc* : The nearest bus stop location from current user
- *nearest_stop_name* : The name of the nearest bus stop from the current user
- *exists* : Variable indicating if a nearby bus stop exists
- *threshold* : Variable indicating the minimum distance threshold for the bus stop

- *bus_no* : The set of all possilble bus numbers
- *usr_id* : The set of all possible user ID's
- *time_stamp* : The timestamp of the transaction between user and client when user gives information regarding the bus when travelling in it
- *direc* : A function mapping *bus_no*, *usr_id* and *time_stamp* to the direction of the motion of a bus.
- *usr_gps_coord* : A function mapping *bus_no*, *usr_id* and *time_stamp* to the user's co-ordinate at the time of the transaction.
- *bus_stop* : A function mapping *bus_no*, *usr_id* and *time_stamp* to the bus stop that was last passed by the bus the user was on during the transaction.
- *on_bus* : Input variable indicating response of user. Variable is a true or false value.
- *velocity* : Variable indicating the velocity of the user at the time of the transaction.
- *usr_idnum* : Gives the user ID of the user making a transaction.
- *cur_gps_coord* : Gives the current co-ordinate of a user making a transaction.
- *dest_stop* : Response from user, which gives the destination of the bus.
- *cur_time* : The time stamp of at the time of transaction.
- *attempt_no* : This is the attempt to ask the user if he boarded the bus.
- *query_clock* : This is the timeout limit for querying the user boarding the bus.
- *dir_travel* : This finds the direction of the bus, with respect to the start and end locations stored for this bus.
- *trans_status* : Gives the status of the transaction with the user.
- *start_coord* : Co-ordinate of user who queries for bus.
- *end_coord* : Co-ordinate where user want to reach.
- *start_id* : The bus stop ID corresponding to start co-ordinate.
- *end_id* : The bus stop ID corresponding to end co-ordinate.
- *candidate_buses* : The buses that pass through the corresponding start and end point.
- *time_threshold* : The maximum estimated times of the buses to be displayed to the user.

3 SCHEMA DESCRIPTION

3.1 STARTUP

This state occurs immediately after the application is opened. It involves initializing the variables and also running a collection of sanity tests to ensure that WhenBus is functioning correctly. If any sanity test fails, an error report is sent to the server regarding the failed test.

STARTUP

$$status! : (error, success)$$

$$(status! = error \land \neg\, TEST) \lor (status! = success \land TEST)$$

3.2 STOP_INFO

This schema describes of the state space of the system with respect to the bus stops. This is a static database of information which contains various details regarding bus stops.

STOP_INFO

$$stop\_id : \mathbb{P}\,\mathbb{N}$$
$$stop\_name : \mathbb{N} \twoheadrightarrow string$$
$$stop\_gps\_coord : \mathbb{N} \rightarrowtail\!\!\!\!\!\rightarrow \mathbb{R}^2$$

$$stop\_id = \mathrm{dom}\, stop\_name$$
$$stop\_id = \mathrm{dom}\, stop\_gps\_coord$$

3.3 STOP_QUERY

This scenario describes the situation when the user makes a query to the bus stop. The GPS co-ordinates of the user are transmitted to the server. The server look for all bus stops within a 2.0 Km distance and returns it to the user. If a nearby stop, does not exist, we alert the user of the same.

STOP_QUERY

$$\Xi STOP\_INFO$$
$$gps\_coord? : \mathbb{R}^2$$
$$nearest\_stop\_loc! : \mathbb{P}\mathbb{R}^2$$
$$nearest\_stop\_name! : \mathbb{P}\, string$$
$$exists! : string$$
$$threshold : \mathbb{R}$$

$$threshold = 2.0Km$$
$$nearest\_stop\_loc! = \left\{ x : stop\_id \,\middle|\, (stop\_gps\_coord(x)_1 - gps\_coord_1?)^2 + (stop\_gps\_coord(x)_2 - gps\_coord_2?)^2 < threshold^2 \right\}$$
$$nearest\_stop\_name! = \left\{ y : nearest\_stop\_loc! \,\middle|\, stop\_name(y) \right\}$$
$$(exists! = true \land nearest\_stop\_name \neg\, \emptyset) \lor (exists! = false)$$

3.4 TIMETABLE_INFO

The schema describes the setup WhenBus will use to store the data that it collects from the users. This tables simply stores the raw data from the user and does where minor calculations on the data received. The data from multiple users are later collected and processed later to makes new time table estimates.

TIMETABLE_INFO

$$
\begin{aligned}
&bus\_no : \mathbb{P}\, string \\
&usr\_id : \mathbb{P}\, \mathbb{N} \\
&time\_stamp : \mathbb{P}\mathbb{R} \\
&direc : string \times \mathbb{N} \times \mathbb{R} \rightarrow\!\!\!\!\!\!\rightarrow (forward, backward) \\
&usr\_gps\_coord : string \times \mathbb{N} \times \mathbb{R} \rightarrow\!\!\!\!\!\!\rightarrow \mathbb{R}^2 \\
&bus\_stop : string \times \mathbb{N} \times \mathbb{R} \rightarrow\!\!\!\!\!\!\rightarrow \mathbb{R}^2
\end{aligned}
$$

$$
\begin{aligned}
&bus\_no \times usr\_id \times time\_stamp = \mathrm{dom}\, bus\_stop \\
&stop\_id \times usr\_id \times time\_stamp = \mathrm{dom}\, direc \\
&bus\_no \times usr\_id \times time\_stamp = \mathrm{dom}\, usr\_gps\_coord
\end{aligned}
$$

3.5 **BUS_TRAVEL**

This schema describes the situation when the user is on the bus and the app detects the same using the velocity of the user. The app the requests the user for the Bus number he is currently on which is then passed on to the server. The server then updates the database using this information which it can later use for some heuristics. This query will be made only if the user previously queried for a particular bus within the last 25 mins. The query will be attempted maximum of 2 times to any user and will not disturb the user further. Note we use an auxiliary function get_direction_travel in order to get the direction in which the bus is moving.

BUS_TRAVEL

$\Delta TIMETABLE\_INFO$
$on\_bus? : (yes, no)$
$time? : \mathbb{R}$
$velocity? : \mathbb{R}$
$user\_idnum? : string$
$cur\_gps\_coord? : \mathbb{R}^2$
$dest\_stop? : stringcur\_busno? : \mathbb{R}^2$
$cur\_time? : \mathbb{R}^2$
$attempt\_no : \mathbb{R}$
$query\_clock : \mathbb{R}$
$dir\_travel : (forward, backward)$
$trans\_status! : (failed, success)$

$(query\_clock < 90000s) \land (velocity? > 25Km/hr) \land (attempt\_no <= 2)$
$\Big((on\_bus? = false) \land (attempt\_no = attempt\_no + 1) \land trans\_status! = failed\Big)$
$\lor$
$\Big((attempt\_no = 3) \land$
$(dir\_travel = get\_direction\_travel(cur\_gps\_coord?, bus\_no, dest\_stop?, velocity?)) \land$
$(nearest\_stop = get\_nearest\_stop(cur\_gps\_coord?, bus\_no?, dest\_stop?, velocity?)) \land$

$(direc' = direc \cup \{(cur\_busno?, user\_idnum?, time?) \twoheadrightarrow dir\_travel\}) \land$
$(usr\_gps\_coord' = usr\_gps\_coord \cup \{(cur\_busno?, user\_idnum?, time?) \twoheadrightarrow cur\_gps\_coord?\}) \land$
$(bus\_stop' = bus\_stop \cup \{(cur\_busno?, user\_idnum?, time?) \twoheadrightarrow nearest\_stop\}) \land$
$trans\_status! = success\Big)$

3.6 BUS_QUERY

This scheme deals with a use Query for a bus. The user gives start and end points and asks for nearby buses. The client then finds all buses between stops near these start and end points. Then, using a heuristics, which is present in the auxiliary function HEURISTIC_TIME, we get all the buses, which are estimated to arrive within 5 minutes.

BUS_QUERY

$$
\begin{array}{l}
\Delta BUS\_TRAVEL \\
\Xi TIMETABLE\_INFO \\
\Xi STOP\_INFO \\
start\_coord? = \mathbb{R}^2 \\
end\_coord? = \mathbb{R}^2 \\
start\_stop\_dist = \mathbb{R} \\
end\_stop\_dist = \mathbb{R} \\
start\_stop\_id = \mathbb{N} \\
end\_stop\_id = \mathbb{N} \\
candidate\_buses = \mathbb{P}\, string \\
time\_threshold = \mathbb{R} bus\_estimate! = string \times \mathbb{R} query\_status! = (success, no\_bus, no\_route)
\end{array}
$$

$$
\begin{array}{l}
query\_clock = 0s \\
attempt\_no = 1 \\
\\
start\_stop\_dist = \min \left\{ (stop\_gps\_coord(x) - start\_coord?)^2 \middle| x \in stop\_id \right\} \\
end\_stop\_dist = \min \left\{ (stop\_gps\_coord(x) - end\_coord?)^2 \middle| x \in stop\_id \right\} \\
\\
start\_stop\_id = \left\{ x : stop\_id \middle| (stop\_gps\_coord(x) - start\_coord?)^2 = start\_stop\_dist \right\} \\
end\_stop\_id! = \left\{ x : stop\_id \middle| (stop\_gps\_coord(x) - end\_coord?)^2 = end\_stop\_dist \right\} \\
\\
candidate\_buses = \left\{ x : bus\_no \middle| goes\_via\_stop(x, start\_stop\_id) \land goes\_via\_stop(x, end\_stop\_id) \right\} \\
time\_threshold = 300s \\
\\
bus\_estimate! = \left\{ (x, y) \middle| (x \in candidate\_buses) \land heuristic\_time(start\_stop, x) < time\_threshold \right\} \\
\\
(candidate\_buses = \emptyset \Rightarrow query\_status! = no\_route) \\
(candidate\_buses \neq \emptyset \land bus\_estimate = \emptyset \Rightarrow query\_status! = no\_bus) \\
(bus\_estimate \neq \emptyset \Rightarrow query\_status! = success)
\end{array}
$$

Software Design Specification

1 Development Software

The software used in the project for the development are Nodejs framework for backend api, Mongodb for database, Android sdk for the UI app development.

- Backend was devoloped using the framework Nodejs and mongod for database.
- The Frontend is a simple mobile application developed for the Android platform using Android sdk.
- The Mocha framework was used for designing test cases and unit tests
- Code collaboration was facilitated via git and github.

2 Database ER Diagrams

The database is mainly for storing the following contents, namely :

- User login, registration tokens
- User sent, bus information
- Bus, Bus-stop details
- Real time predicted bus timings
- Bus route information

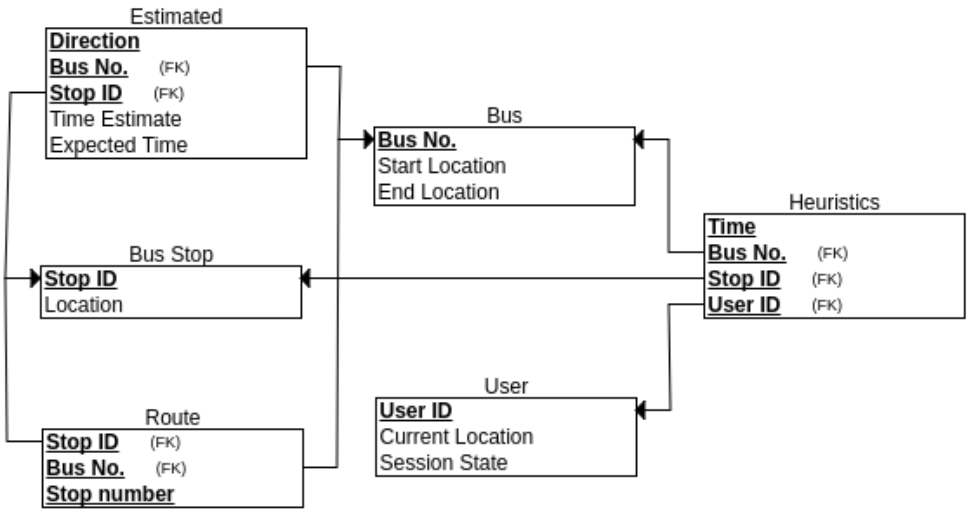

Figure 2.1: Schema-Diagram

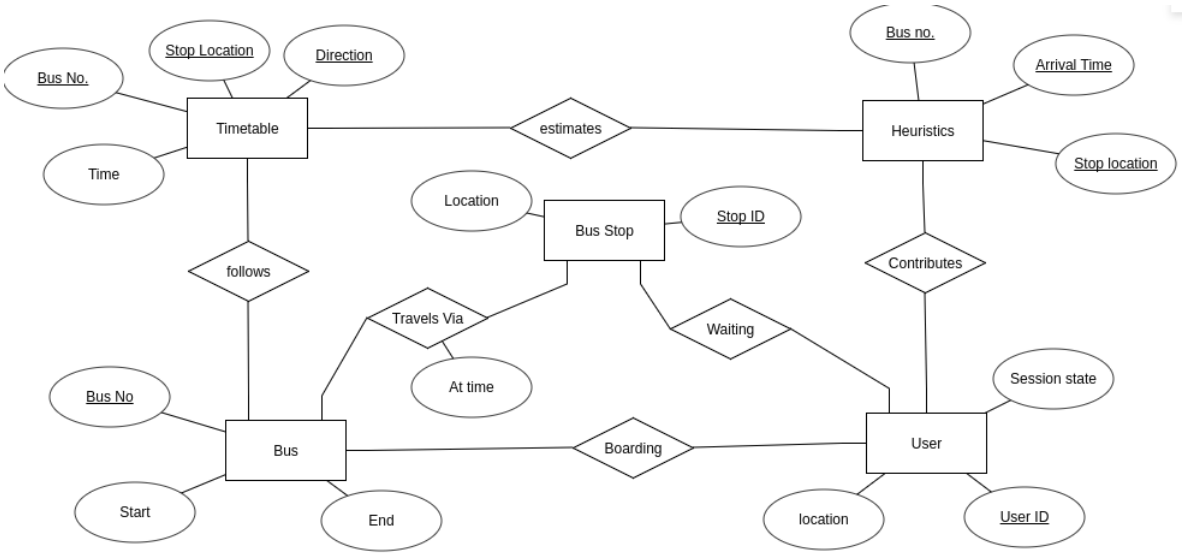

Figure 2.2: ER-Diagram

3 Software Models

3.1 Context Models

The context of the system for describes what is clearly defined as part of the system, and the environment. Our model has mainly three primary interacting external systems given by :

- Phone and its Physical sensors
- Google maps system API's
- Chennai MTC system

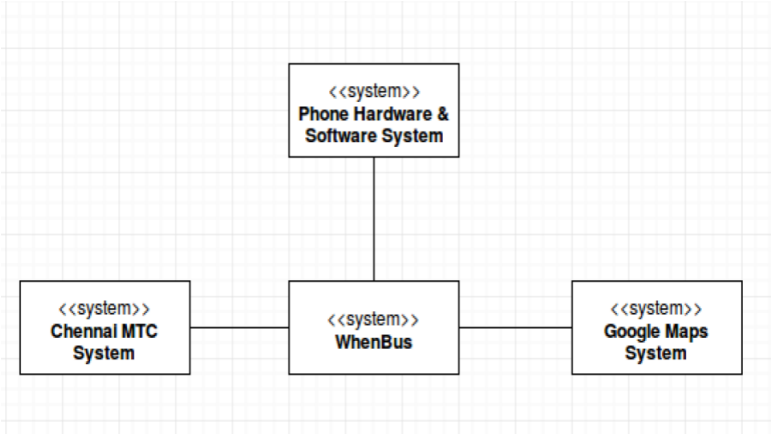

Figure 3.1: Context Model

3.2 Interaction Models

There are primarly, three types of interactions that one must consider, which are:

- User - System Interactions : Use Case Diagrams.
- System - System Interactions : Sequence Diagrams.
- Component Interactions : Components Model

Each of the models can be represented via appropriate representations We first look at use-case models :

User Registrations	
Actors	App Users
Description	The user is registered with the WhenBus application. The server and database are accordingly updated.
Data	User Name and password
Stimulus	The first time user opens the app after installation. Skip screen if already registered user
Responses	Acknowledge the user registration and store in the database

Figure 3.2: Use-Case Model-1

OnBoard Bus	
Actors	App Users
Description	This scenario describes the situation where the user shares data regarding his/her location when on board a bus. This is used for global heuristics calculations of ETA
Data	User response regarding whether user onboard bus, bus number and bus direction.
Stimulus	The velocity of the user is tracked and velocity goes above threshold, this interface is prompted
Responses	The user collects the User ID, time stamp, GPS location along with other details furnished by the user and updates the Database accordingly

Figure 3.3: Use-Case Model-2

Bus Stop Query	
Actors	App User
Description	User Query for nearby bus-stop. The application responds with a list of nearby bus stops after consulting with a server
Data	User GPS location or area
Stimulus	The user opens app and after the login, this screen occurs. Indicates the start of a transaction
Responses	The list of nearest bus stops(within certain threshold) are conveyed to the user. Atleast one bus stop is suggested to the user.

Figure 3.4: Use-Case Model-3

Bus Query	
Actors	App User
Description	The user queries for a particular bus or particular start and end point. The app, with consultation with the server finds a list of buses and their ETA
Data	Start/End point or a particular Bus
Stimulus	The user selects a bus stop following which the Bus Query interface is prompted
Responses	List of Buses along with their estimated arrival time at the start location as per the heuristics used by the server backend

Figure 3.5: Use-Case Model-4

The we next have a look at the user-system interaction diagram

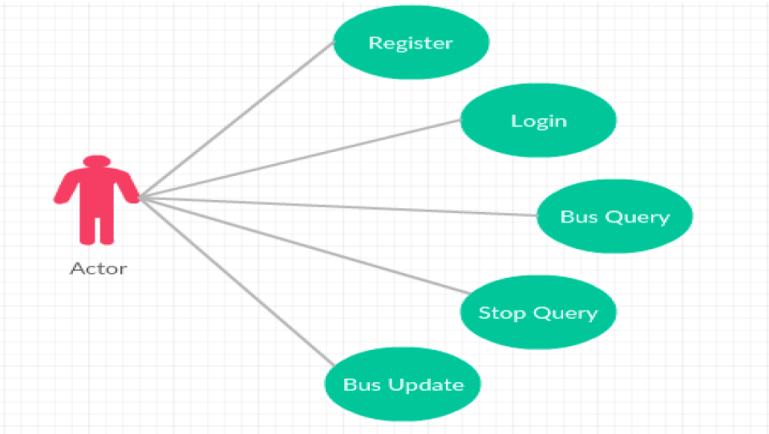

Figure 3.6: User Model

We next consider sequence diagram and a components model which describe system-system interaction, components interaction

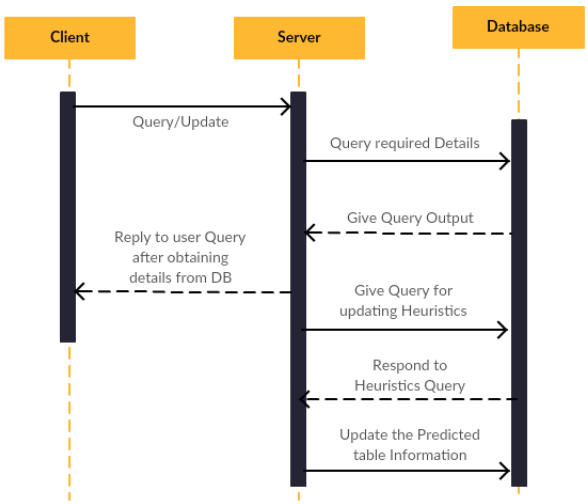

Figure 3.7: Sequence diagram

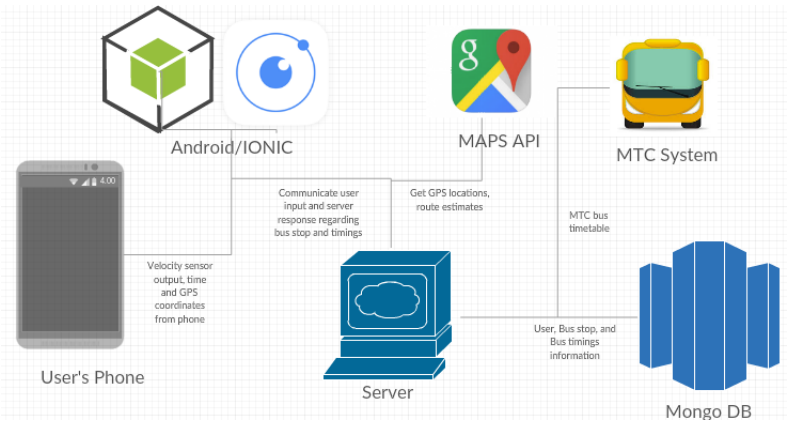

Figure 3.8: Components Model

3.3 Structural Models

Structural models describe the overall structure of the system and highlights the internal relation among various system modules. It is similar in flavour to a relational schema among various entities present in the overall system.

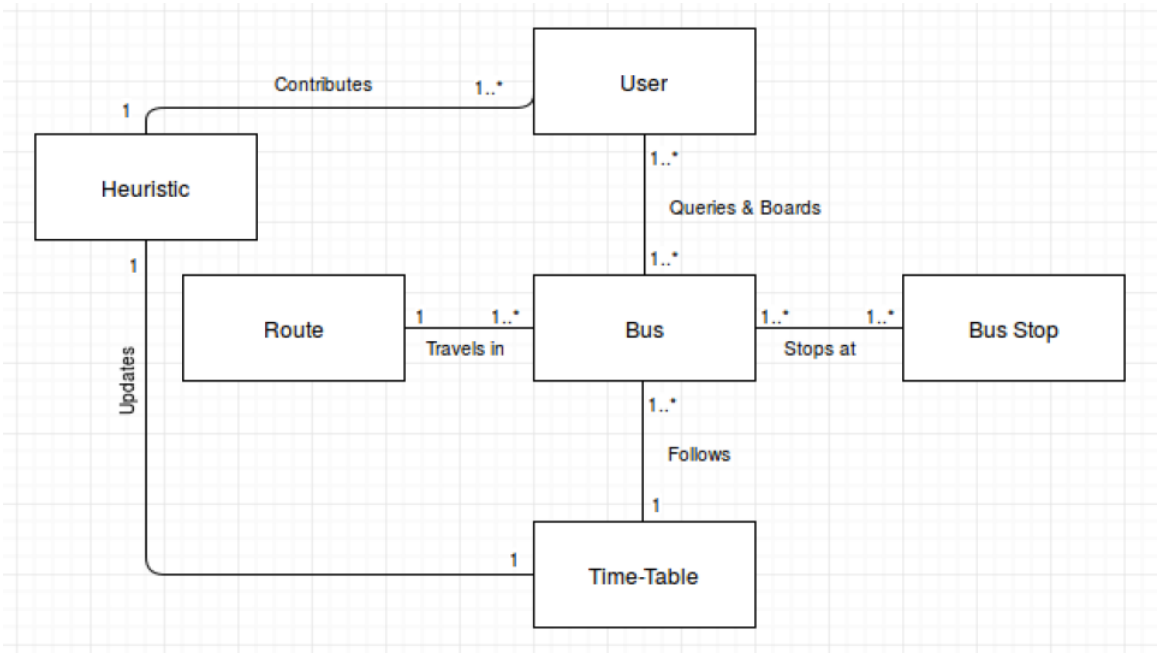

Figure 3.9: Structural Model

3.4 Behavioural Models

The behavioural model describes the behaviour of the system in response to various stimuli and is similar in flavour to a finite state automaton. The system is present in one of the describes states, and based on external or internal actions, the system transitions to a new state.

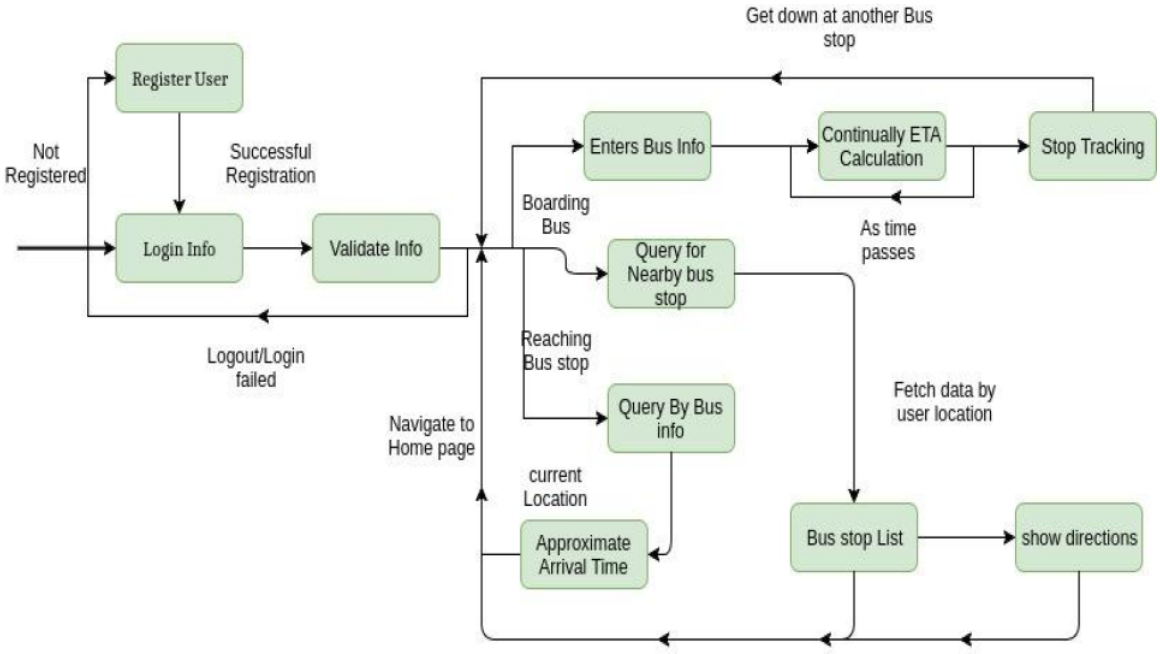

Figure 3.10: Behavioural Model

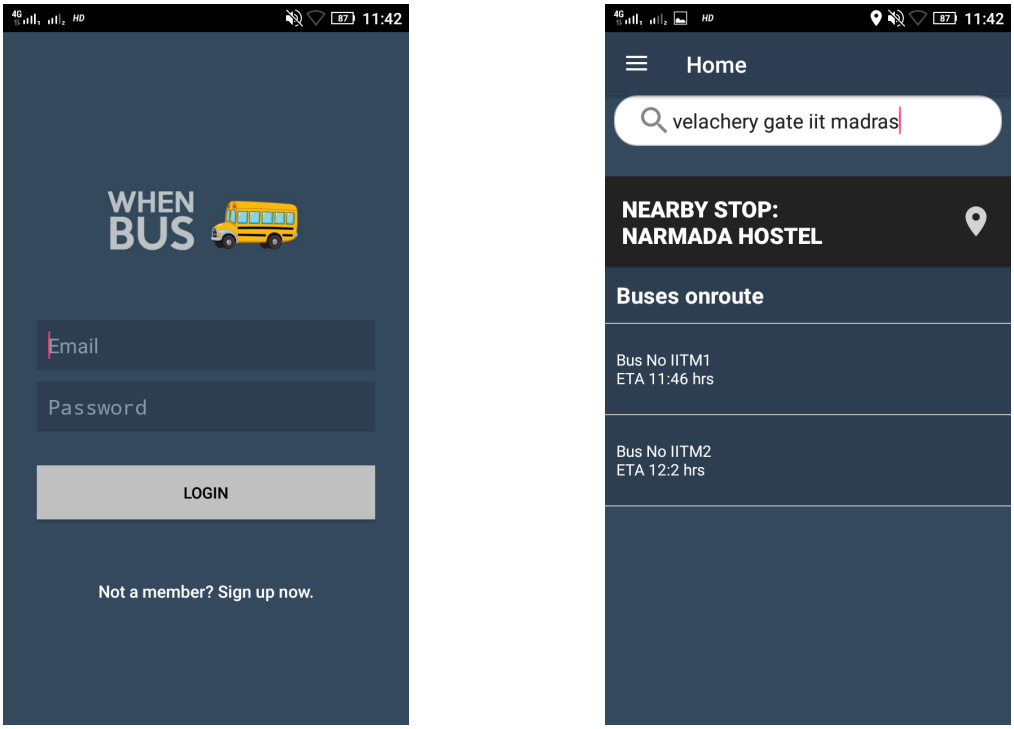

Figure 4.1: UI Design

4 UI Design

Presented below, is a template for the UI design proposed :

5 Software Architecture

WhenBus follows a client-server architecture whereby there are multiple clients that asynchronously communicate with one server. Our code base also is also closely structured around a MVC system and the code is segregated into different models and controllers.

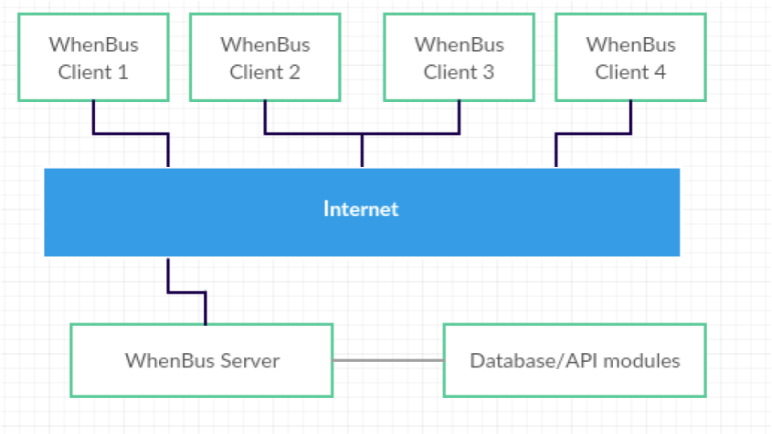

Figure 5.1: Architecture Type

The system comprises of three major components which are Frontend, Backend and the database management system. The external components are

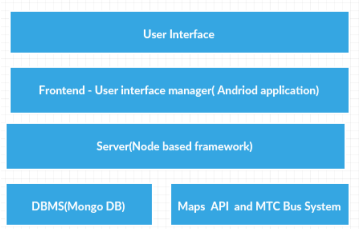

Figure 5.2: Architecture View

The development view describes the segregation of various tasks into corresponding software components. It clearly describes the role of all major modules

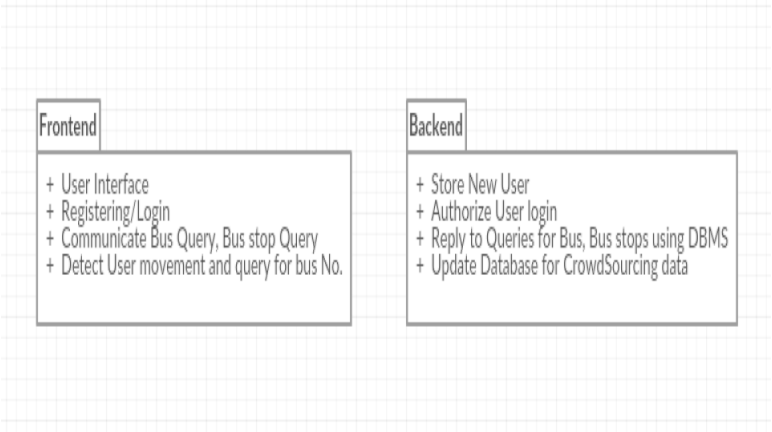

Figure 5.3: Development View