WhenBus

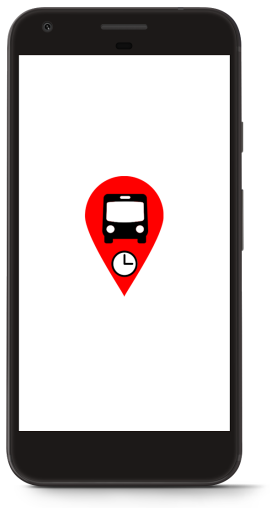
whenbus.software@gmail.com
IIT Madras, Chennai

Product Brochure

April 06, 2017

Introduction

WhenBus is a crowd-sourced application that provides accurate bus timings. The key motivation for this application is that, even though there are a lot of applications that provide the standard bus times available on government sites, it is highly unlikely that the buses do follow the same due to various factors like traffic, weather etc., . As the number of people who are taking the public transport is increasing and with mobile technology available, it is possible via crowd-sourcing to get accurate bus schedules based on user feedback.



Crowd-sourcing

Crowdsourcing allows businesses to use the input of multiple sources, both within the corporation and externally, to develop solutions for strategic issues or to find better ways to complete tasks. This new culture of innovation, supported by crowdfunding for worthwhile projects, allows for idea collaboration and technological innovation for the greater good. Further, our increasingly mobile world population allows for people from anywhere, and with any background, to give their input on a project.

App Overview

Bus location display

WhenBus has the feature of allowing the user to give the bus number as the input and get the live tracking information about the bus and the time expected for the bus to reach the nearest bus stop to the user.

Bus tracker

WhenBus has the feature of updating location of bus through gps location of users and some feedback taken from them. This allows other users to get real time information.

Best bus suggestion

WhenBus has the feature of allowing the user to give the source and destination bus stops as the input and get the best buses with their corresponding expected timings.

Software details

WhenBus application is built on Android, Nodejs backend and Mongo database. Express server is kept running at the backend and the android makes http requests for specific info to corresponding routes and information is processed from database and requested information is sent to the android phone as response.

The various API’s used for routes are shown below.

POST /suggest/nearestStop

description : *The API is used to suggest the geographically nearest bus stop to the user.*

Request :

```
{
  "coord" : {            [*] # User Location
    "lat" : float,
    "lng" : float
  }
}
```

Response :

```
{
  "distance": float,    # Distance to stop[m]
  "stopName": string,   # Stop Name
  "coord" : {           # Stop Location
    "lat" : float,
    "lng" : float
  }
}
```

POST	/suggest/bus

description : *The API suggests possible bus numbers for given destination, also suggesting the nearest starting bus stop too.*

```
Request :
{
  "src" : stop_name,
[-] # User needed source  (optional)
  "dest" : stop_name,
[*] # User destination
  "coord" : {
[*] # User Location
    "lat" : float,
    "lng" : float
  }
}
```

```
Response :
[
  {
    "bus_no": string,
# Bus   Number suggested
    "src"   : string,
# boarding stop
    "time"  : float,
# Expected Time of arrival at stop [min]
# (measured from 00:00 )
    "distance" : float
[-]  # (optional -- iff src is not specified
  }
]
```

POST /info/bus

description : *This API, returns the best possible bus [expected to arrive the earliest] given the bus number and the direction of the bus*

```
Request :
{
  "bus_no" : string,                     [*]
# Bus Number
  "start_point" : stop_name,             [*]
# Start Point of Bus
  "end_point" : stop_name,               [*]
# End Point of Bus
  "coord" : {                            [*]
# User Location
    "lat" : float,
    "lng" : float
  },
  "src" : stop_name,                     [-]
# User specified source
  "dest" : stop_name                     [-]
# User specified destination
}
```

```
Response :
{
  "id" : Number,
# bus id ( use for feedback )
  "stop" : stop_name,
# Nearest Stop Name
  "busLoc" : {
# Estimated Location of Bus
    "lat" : float,
    "lng" : float
  },
  "time" : float
# Expected time of arrival of bus
# at the stop[in minutes from 00:00]
  "stopList" : [stop_name]
# List of stops to destination
}
```

POST	/feedback/access
description : *This API is called when the user is ready to provide valuable feedback, the server records the user and assigns an unique key, which has to be used for subsequent feedback transactions*	
Request (see below)	Response (see below)

```
Request :
{
  "busNo" : string,                 [*]
# The bus user is in
  "start_point" : String,           [*]
# Bus start point
  "end_point" : String,             [*]
# Bus Destination
  "id" : Number,                    [*]
# id returned from info module
  "src" : string,                   [*]
# user start point
  "dest" : string                   [*]
# user destination
}
```

```
Response :
{
  "key" : string
# Unique identifier for FPU
}
```

POST	/feedback/send
description : *This API will be pooled periodically by an Feedback Providing User (FPU), where he send the stop he is in and the timestamp. The server drops the FPU once he has reached his destination.*	

Request :

```
Request :
{
  "key" : string                          [*]
# FPU key
  "coord" : {                             [*]
# Location of the user
    "lat" : float,
    "lng" : float
  },
  "timestamp" : float,                    [*]
# Timestamp when sending feedback
  "stop" : string                         [*]
# Stop the user is in or the next stop user
is heading towards
}
```

Response :

```
Response :
{
  "status" : "OK"/"DROP"
# Accept/Drop Feedback
}
```

POST /feedback/end

description : *The user can voluntarily call this API, to stop sending feedback.*

```
Request :
{
  "key" : string                           [*]
# FPU key
}
```

```
Response :
# drop's the user if the record exists in DB
{
  "status" : "success"
}
```

GET /sync/stops

description : *This API is called to get all the bus stops information stored in the database*

```
Request : None
```

```
Response :
[
  {
    "stopName" : string,
# Bus Stop Name (unique)
    "coord" : {
# Stop Location
      "lat" : float,
      "lng" : float
    }
  }
]
```

GET /sync/bus

description : *This API is called to get the information of all the buses information stored in the database*

```
Request : None
```

```
Response :
[
  {
    "busNo" : string,
# Bus Number
    "source"   : string,
# Bus Starting Point
    "destination"  : string,
# Bus End Point
    "busStopList" : [string]
# List of Bus Stops the bus goes through
  }
]
# All three make a key
```

Data Extraction

MTC bus data is extracted by scraping the official mtc bus website “**mtcbus.org**” using tools beautifulsoup and selenium. The extracted data consists of list of bus numbers and their timings,list of bus stops,average travel time and service type. The coordinates of the bus stops are obtained from the google maps.

Data Cleanup

The extracted data from mtc is not in a proper format i.e there are some conflicts among the bus stop names, bus numbers and the bus stop list. So a lot of validation was needed to be done and now we have database consisting of bus numbers, which have timings and list of bus stops free from conflicts.

Database Setup

The mongo database has been setup and data scrapped is organised and stored in the database as documents in their respective collections.

BusStop Document Format:

```
{
 "busList" : [string]
 "coord" : {
   "lat" : float,
   "lng" : float
 },
 "stopName" : string
}
```

MasterBus Document Format:

```
{

     "busNo" : string,
     "source" : string,
     "destination" : string,
     "servicetype" : string,
     "averageTravelTime" :integer,
     "busStopList" : [string],
     "busList"   : [string]
    }
```

Bus Document Format:

```
      {
          "id" : integer,
          "busNo" : string,
          "source" : string,
          "destination" : string,
          "averageSpeed" : float,
          "Timings" : [{ "busStop" : string ,"time" : integer }]
         }
```

Project Members

Arjun Krishna, CS14B058 *BackEnd Developer*	● Developed the Application Program Interface (API) of the software, using Express, node JS and MongoDB. ● Documentation of API, Server code. ● Documentation of System Specifications, from the requirements phase to the development phase.

Vinay Reddy, CS14B006 *BackEnd Database Manager*	• Data is scraped from the MTC website using selenium and beautifulsoup. • Data is validated and is saved in MongoDB . • Documentation of System Specifications, from the requirements phase to the development phase.
Shashikanth Reddy, CS14B020 *QA engineer, UI/UX Designer*	• Testing android activities and API request responses. • Documentation of System Specifications, from the requirements phase to the development phase.
Sri Harsha, CS14B027 *Android Application Developer*	• Development of complete app interface and functioning. • Documentation of System Specifications, from the requirements phase to the development phase.
Hemanth Robbi, CS14B026 Android interface designer	• Design of app presentation • Documentation of System Specifications, from the requirements phase to the development phase.