Fall 2012

MapMyVTA

Gaurav Sharma
San Jose State University

MapMyVTA

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

by

Gaurav Sharma

December 2012

SAN JOSE STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Title

MapMyVTA:

By

Gaurav Sharma

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Soon Tee Teoh, Department of Computer Science	Date
Dr. Mark Stamp, Department of Computer Science	Date
Bharat Agarwal, Senior Software Engineer, Cisco Systems	Date

ABSTRACT

Transportation is a very important part of our day-to-day life. Generally, it includes use of public transportation services like those provided by Valley Transportation Authority (VTA) to Santa Clara County.   VTA has reported total combined boarding of light rails and buses as more than a million on yearly basis. This fact clearly indicates the importance of public transportation in a society. Obviously trip planning and schedule matching are two very decisive factors to improve transit experiences. Information related to services makes it easy for users to plan their journey ahead. Still manual planning and information discovery is time consuming, tedious, and prone to human errors. Therefore need of a better, user-friendly transit information system has been long felt. MapMyVTA is a web application that provides detailed information about VTA services to its users. MapMyVTA keeps the users updated about the timings of the buses, positions of the buses at a given time, and expected time of arrival of a bus at a given stop in a route. These features help users to match their timings with expected timings of the buses at the stop, to see their options about the number of buses en-route, to look up their connecting lines by a simple click at the connecting stops, and to plan their journey quickly with all system supported routes. Additional features, such as stop locator is useful to find more information about a particular stop with a near around attractions list with addresses, and view lines information feature make it easy to view a very detailed information about the bus lines.

ACKNOWLEDGEMENTS

I am thankful to my project advisor Dr. Soon Tee Teoh, for his invaluable suggestions, insights and support throughout my master's project and I would like to thank my committee members Dr. Mark Stamp, and Mr. Bharat Agarwal for their time and suggestions.

I would like to thank my parents and my dear brother & sister in law for their continuous love, motivation, and support.

TABLE OF CONTENTS

LIST OF FIGURES

Introduction

1.1 Project Overview

Santa Clara Valley Transportation Authority (SCVTA) manages public transportation for Santa Clara County. VTA (in short) serves almost 326 square miles of area with 75 different routes with its fleet including light rails and express buses [8]. The Availability of connecting lines on transit centers for inter agency transfers, multiple connecting lines for intra agency transfers and frequent services to most of the stops, are some of the points which makes VTA a preferred choice to commute. VTA has reported a combined boarding of almost 3,660,722 annually in the month of May 2009 as a reference point on their website [8]. The reasons are obvious. Public transportation with frequent and mostly regular services makes them a promising choice to commute.   Hence public transportation with a better information representation infrastructure will not only fill the information gap between the services to its users but also will help to serve its users in a time savvy manner.

The basic idea behind this project is to make people more aware of their options of public transport, VTA in this case and make VTA journey more convenient for commuters by providing some very specific visual features, which can deliver the VTA transit information in a more detailed and user-friendly manner. Easy to use and detailed information about the transit is a necessity for better transit experience. It is quite interesting to discover that how a simple time-table can be made more user-friendly which can help users to make the most of their time by saving them those minutes which otherwise would have been wasted, just waiting at a stop for a bus. This transformation of information is obviously a very refined version of the textual form of information. In this project, we tried to provide the same. This project is a web application, which can be accessed over the internet from anywhere, any devices (PCs/Mobiles/Tablets) which support any web browser. Hence the project for its services has almost no specific requirements on the external environment on the user's part; therefore the project has very high accessibility. MapMyVTA allows users to view their buses visually, provides a time estimation of the buses en route for all stops in a dynamic display table and a trip planner to help users plan their trip. In addition to these features, this project also has features such as stop locator and line information dashboard. The goal is to help commuters to have a leading edge of information to save their time by reducing wait times at stops as much as possible, to reduce the necessity of tedious timetable lookups, calculations, manual planning, and to eliminate human errors, by providing coherent set of services to help the commuters to plan their trip more intelligently.

1.2 Report Structure

The project report is structured into the following sections:

Section 1 is an overview of this project and the report. Section 2 includes an analysis of existing systems and comparison. Section 3 discusses the details of software, tools, and technologies used to develop this application. Then Section 4 talks about the software architecture used in this project. Then Section 5 has details of the internal directory structure of the application. Section 6 provides the internal details of design and implementation of the system. Then section 7 has screen shots of the system. Finally, Section 8 concludes the project. References are included in the last part of this report.

2 Analysis of Existing Systems & Comparison

This section includes a summary of current systems, their advantages, and their possible inadequacies. By comparing these similar systems, it will be easier to understand the relevance of this project. As the web is increasingly getting popular and accessible almost from everywhere via wireless / 3G networks, using smart phones / tablets therefore any service available over the web will have high accessibility to its users. So now it is a user’s constraint that a system should be user-friendly and consistent enough to display the same information in mobile platforms without reducing its completeness.

All available applications have very limited set of features. Some of them which support VTA are the ones which do have a same time-table (stop & time combination) like those provided by VTA but without any additional visual support. Few of them navigate users back to the VTA site. However, in my research I have found no application which is supporting visual mapping of buses on a map for the VTA now. Hence I have analyzed systems being used in other parts of the USA by different transit agencies.

I have analyzed these systems:

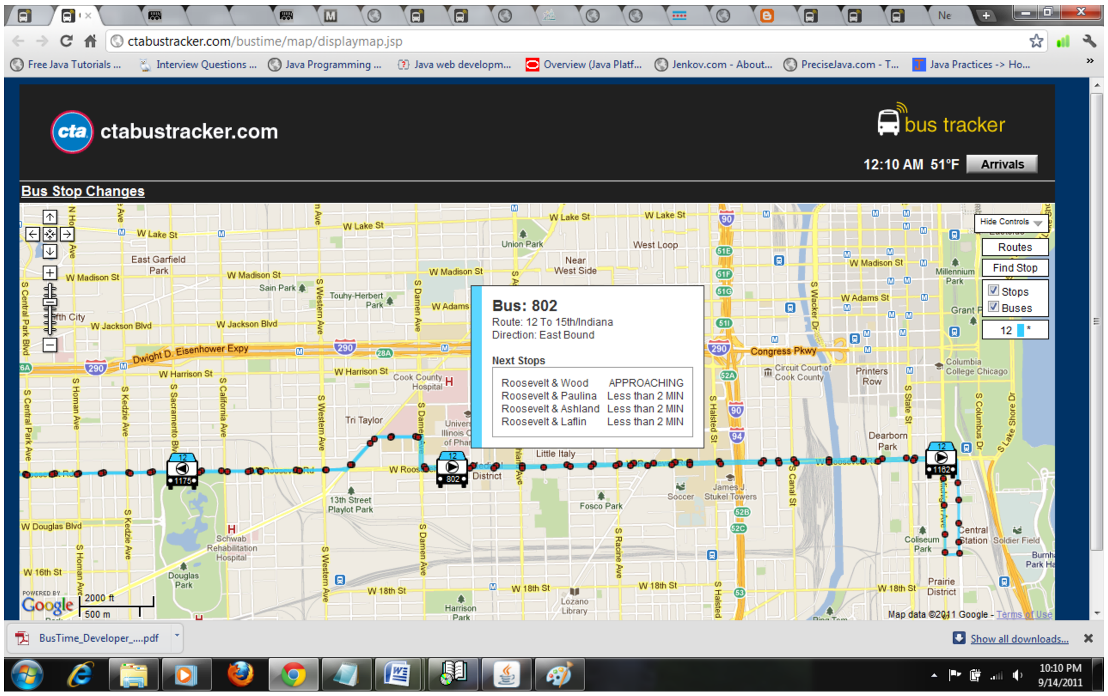
1.1 Chicago Transit Authority (CTA) Bus Tracker System

1.2 Next Bus

1.3 Washington Metro transit Authority (WMTA)

2.1 Chicago Transportation Authority (CTA) – Bus Tracker Application

According to the description of the application provided on the website of the CTA BusTracker, it says, "CTA Bus Tracker uses GPS devices to report bus location data (and more) back to our servers. We can then, in real time, show you where buses are on a map and estimate when they will arrive at your stop."



Bus: 802
Route: 12 To 15th/Indiana
Direction: East Bound

Next Stops

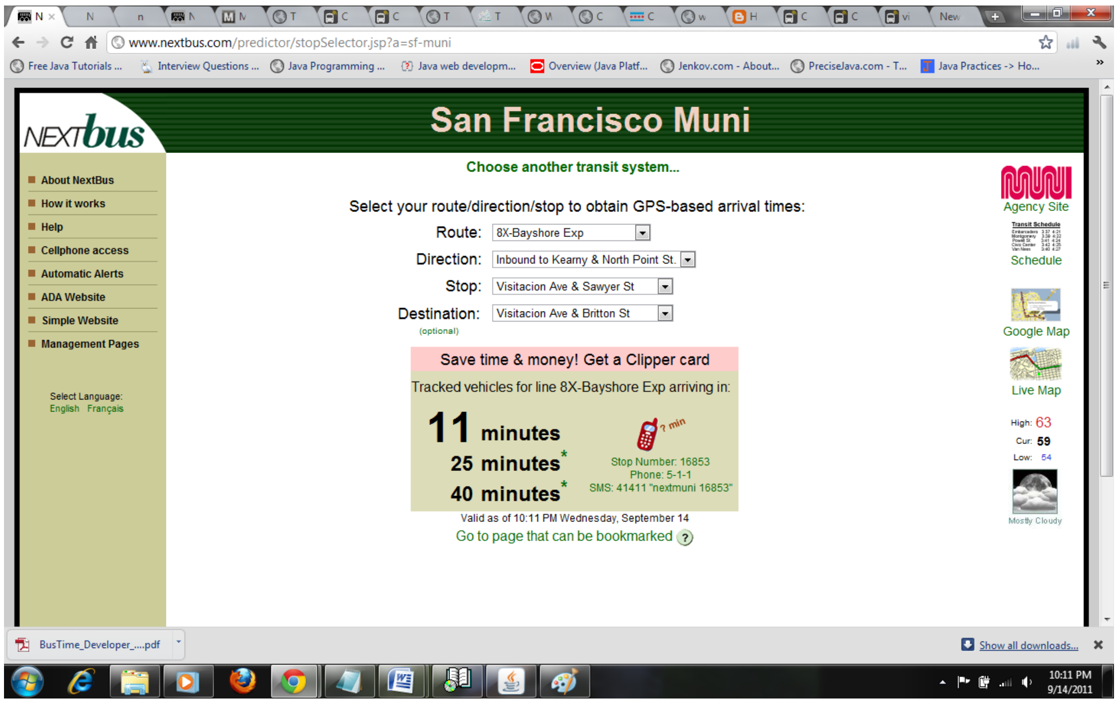
Roosevelt & Wood	APPROACHING
Roosevelt & Paulina	Less than 2 MIN
Roosevelt & Ashland	Less than 2 MIN
Roosevelt & Laflin	Less than 2 MIN

Figure. 2.1.1 CTA- visuals for Bus 802

CTA BusTracker [Image]. (2011). **Retrieved** September 14, 2011, **from** Ctabustracker.com/bustime/map/displaymap.jsp

2.2 NextBus Services

Next Bus services cover a variety of transportation authorities but in a selective manner. So here in California NextBus covers MUNI and SFBay Ferry only.



www.nextbus.com/predictor/stopSelector.jsp?a=sf-muni

San Francisco Muni

- About NextBus
- How it works
- Help
- Cellphone access
- Automatic Alerts
- ADA Website
- Simple Website
- Management Pages

Select Language:
English Français

Choose another transit system...

Select your route/direction/stop to obtain GPS-based arrival times:

Route: 8X-Bayshore Exp
Direction: Inbound to Kearny & North Point St.
Stop: Visitacion Ave & Sawyer St
Destination: Visitacion Ave & Britton St
(optional)

Save time & money! Get a Clipper card

Tracked vehicles for line 8X-Bayshore Exp arriving in:

11 minutes
25 minutes*
40 minutes*

Stop Number: 16853
Phone: 5-1-1
SMS: 41411 "nextmuni 16853"

Valid as of 10:11 PM Wednesday, September 14

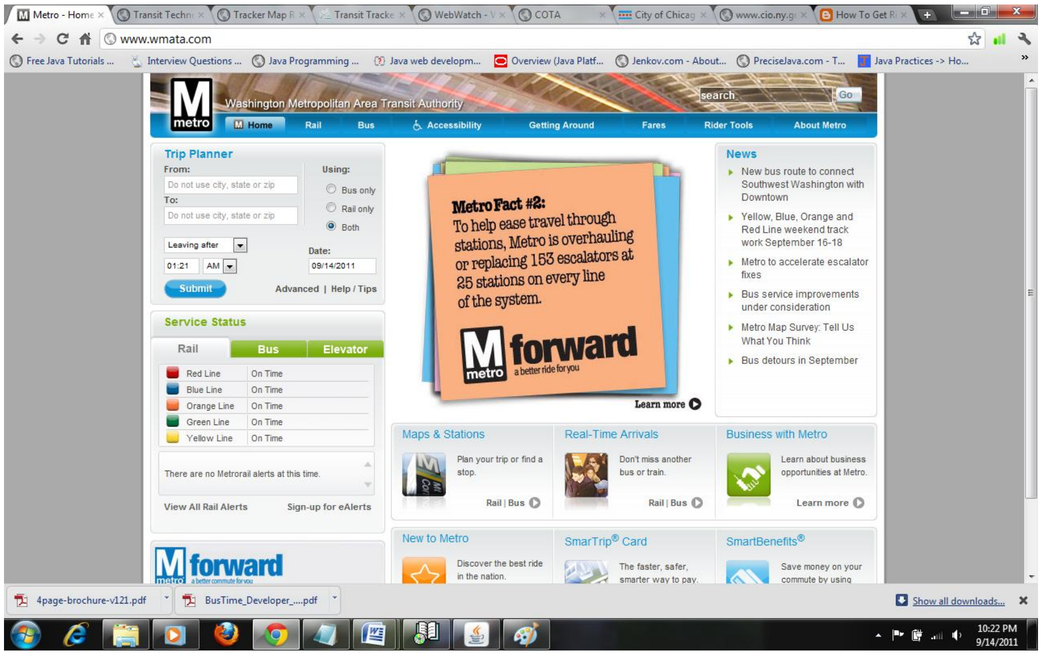
Go to page that can be bookmarked

Figure 2.2.1 NextBus Front page

NextBus [Image]. (2011). **Retrieved** September 14, 2011, **from** www.nextbus.com/predictor/stopSelector.jsp?a=sf-muni

2.3 Washington Metropolitan Area Transit Authority (WMTA)

WMTA provides metro rail & metro bus transit services in Washington, DC, Maryland and Virginia areas. For tracking buses WMTA uses next bus technology and services.



Using: Bus only ( ) Rail only ( ) [✓ Both]

Red Line	On Time
Blue Line	On Time
Orange Line	On Time
Green Line	On Time
Yellow Line	On Time

There are no Metrorail alerts at this time.

Metro Fact #2: To help ease travel through stations, Metro is overhauling or replacing 153 escalators at 25 stations on every line of the system.

M forward a better ride for you

News
- New bus route to connect Southwest Washington with Downtown
- Yellow, Blue, Orange and Red Line weekend track work September 16-18
- Metro to accelerate escalator fixes
- Bus service improvements under consideration
- Metro Map Survey: Tell Us What You Think
- Bus detours in September

Maps & Stations: Plan your trip or find a stop.

Real-Time Arrivals: Don't miss another bus or train.

Business with Metro: Learn about business opportunities at Metro.

New to Metro: Discover the best ride in the nation.

SmarTrip® Card: The faster, safer, smarter way to pay.

SmartBenefits®: Save money on your commute by using

M forward a better commute for you

Figure. 2.3.1 WMTA Front page

WMTA [Image]. (2011). **Retrieved** September 14, 2011, **from** www.wmta.com

In my detailed analysis, there were different areas of improvements I came across with all 3 different systems. Complete details of analysis are beyond the scope of this report. However, all 3 systems as official applications were able to display real time information and to plot the buses on the maps, to locate a particular stop, to show bus movements, and to list all schedules. MapMyVTA supports all these basic features except real time tracking as it does not have access to GPS data currently. But all 3 other systems lacked some vital features on common grounds such as, no user-friendly interface like icons selections, navigational issues, too much unnecessary data, similar icons for both directions, therefore it is difficult to sense the direction, no information of connecting lines at a particular stop, no complete snapshot of expected timings of the next bus arrival on a stop at one place for a particular route, no search feature for nearby attractions around a stop. Above all, clean and user-friendly interfaces are the top-most requirements of any application to attract any user; as these are the first few features that get noticed by a user. MapMyVTA supports all these features, in addition to the basic features.

3 Software, Tools & Technologies

3.1 J2EE

Java 2 Platform Enterprise Edition (J2EE) defines a standard for developing multi tier distributed enterprise applications. A platform-independent and component based J2EE framework makes it easy to develop web applications which are going to run on top of Java platform. The J2EE simplifies application development by providing reusable modules and very large libraries of APIs. The J2EE basically consists of several inter-connected yet independent set of services like ready to use APIs, which work together to ensure the success of the important aspects of a web application for example, transaction management, security, performance, scalability etc. As a framework all these services are provided and taken care of by the servers supporting the J2EE framework for developing enterprise level applications. The J2EE has two important constituents in it; they are JSPs & Servlets. Both of them are web components in the J2EE framework.

3.1.1 JSP

JSP technology is a dynamic web page generation technology. In an MVC architecture style, this technology is a part of a view layer. Any user specific customized page is a dynamic web page which is generated on the fly as a response. Page can also perform CRUD operations before it gets transmitted to a user for viewing and can interact with the user. The user gets the simple HTML/XHTML document and has no clue that the contents of the page have been changed selectively, very specific to him/her. JSP pages unlike simple HTML pages require some specific tools to translate them into an appropriate browser viewable document (into an HTML page) and this is why JSP technology requires servers with web containers. The motive behind the use of JSP pages is to save a developer's time from writing a tedious Servlets code for view component. This helps to reduce time, human errors, increase production time, and clarity. However, the server finally compiles all JSP pages into a Servlet. The view component of MapMyVTA is written using JSP technology.

3.1.2 Servlets

The Servlets are special java classes, which extend specific classes like HttpServlet, which make them able to talk to the server's web container for http protocol based request-response processing over the internet. In MVC technology, Servlets are generally part of "controller layer". Hence Servlets generally used as a controller for the application where they decide the flow of control, and do the decision making for the application like invoking service classes to process business logic, error handling, and selecting an appropriate response (view) as a result of the request. Whenever server (web container) receives a request for the application, it looks for a particular java Servlet whose binding with the request context is defined in an application's deployment descriptor files named as "web.xml". Deployment descriptor files are simple xml files consisting of relevant data for the server's to properly set the execution environment, initialize the application and context binding. Servlets, as they are java classes themselves make it easier to use communicate with other java classes and exchange information. Therefore, Most of the business logic is carried out in java classes also known as “model layer”, then the processed result is passed to a Servlet acting as a controller and then finally, the result is handed over to the view technology like JSP. The controller component of MapMyVTA is written using Servlets Technology.

3.2 JSON

There are two very popular data exchange formats available as XML and JSON. Out of these two, JSON is fastest growing widespread data exchange format. JSON stands for “JavaScript Object Notation”. Unlike XML, every well-formed JSON file is automatically a valid JavaScript object. That makes JSON data very easy to access, manipulate, and to retrieve them on a webpage. JSON is also known as lightweight, as JSON does not have a redundant information structure unlike </endtags> in XML and because of that reason resulting JSON file is very small compared to a similar XML file. Small size with the same amount of information reduces the number of HTTP response packets which results in faster and better efficient use of applications over the internet. The data representation and exchange format for MapMyVTA is JSON.

3.3 JavaScript

JavaScript is the scripting language of the front end. JavaScript makes it easier to manipulate the DOM structure of the HTML page thereby opens up a huge potential for dynamic content creation. Together with CSS, JavaScript is known as DHTML. JavaScript Makes it easy to interact with a user for trivial tasks which can be handled on the user’s side and hence saves a round trip network call to the server and time.

JavaScript is used as the client side scripting language for MapMyVTA.

3.4 Ajax

Ajax stands for “Asynchronous JavaScript & XML”. Apparently, Ajax is not a new technology; instead it is an intelligent approach to make the request to the server for fetching the new data without making a new request for the entire page to reload. Although XML is a preferred format to receive the data but using an XMLHttpRequest object Ajax’s abilities are unlimited. Ajax can send and receive the data in formats like JSON, HTML, TEXT files, and of-course XML. Hence it can also receive information from server-side scripts as well. Hence using Ajax, we can asynchronously make requests to the server while a client is still using the page and perform the necessary transformation and present the new data to the user in the same page. Hence it reduces the network calls; pay loads of HTTP response packets and keeps the client interaction while performing more action in the background. Ajax is used for the Quick Query Dashboard feature in this project.

3.5 Google Maps API

Google Maps APIs are a collection of maps APIs. These APIs are used to perform various operations over Google Maps. All Google Maps APIs are of two types: one is Google Maps API Web Services, which can be invoked over standard HTTP protocol to get geographic data in any map related application. So the use of web services makes it easy to build an application which can invoke some map related operation using the exposed set of web services and consume their responses. The format of the response is specified in its request URL as “output=json” or “output=xml”; as currently these are the only two formats which the web services support. Second one is the Google Maps JavaScript API. The current version of the JavaScript API is 3. The JavaScript API library is loaded just like any other library using a URL and invoked in your HTML page just like any other JavaScript method. Hence both forms of APIs are very easy to incorporate into any application. Google Maps APIs are further divided into different categories like Directions, Geocoding, Elevation, and Places etc. Each one of them includes interfaces to serve the particular need. For Geocoding API services, Google Maps APIs helps to project a point on our spherical earth onto a flat presentation of a map using one of the available projection techniques like Mercator projection. Some terminologies used in Google Maps are as follows: **1) GeoLocation:** Every point on the map is represented by a pair of latitude and longitude pair (Latitude, Longitude). The latitude and the longitude are together known as the Geolocation. **2) GeoCoding:** Geolocations are used to pinpoint any particular position on the map. This process of converting address to its Geolocation on the Map is known as Geocoding. **2) Reverse GeoCoding:** The process of converting a Geolocation back to its address form is known as reverse Geocoding. **3) Map Objects:** To incorporate a map in your web page we need to create an object of the type “google.maps.Map” – it’s a JavaScript class that represents a map. Every new instance of this class is a new map. Hence for multiple maps on the same page require an equal number of instances to be created using a new Operator. **4). Overlays (Decorator Pattern):** Overlays are different types of objects that we can create which are tied to a map. For a map, overlays objects include map points, map markers, map lines, map areas, or simply any MVC type collection of objects which have more detailed information about a Geolocation. Google maps JavaScript v3, makes it easy to integrate the Google maps APIs into custom JavaScript code and invokes them effortlessly like a local method.

3.6 Eclipse IDE

Eclipse IDE is an open source tool, which can be used to develop a variety of application for multiple platforms. Eclipse supports almost all major development languages with the help of plug-ins. Eclipse supports integration of third party software back-end like database server or the enterprise servers like tomcat apache. This integration gives complete control on your development environment from a single IDE. Using an appropriate IDE makes the development process very fast as syntax checks, missing library links and compile time errors are discovered and highlighted by the IDE instantly.

3.7 Tomcat Apache Server

Tomcat apache provides a pure java implementation of Servlet container. Servlet container is basically responsible for managing Servlets life cycle events, context mapping and provide a set of services like access right check on requester before handing over the request to a particular Servlet. Tomcat server is made of different components like Catalina, Coyote & Jasper.

A Catalina component is an actual Servlet container, which handles the Servlet life cycle events like create and destroy. To receive a request, the tomcat needs to listen to a specific port. To listen on a specific port tomcat needs a HTTP connector component. Tomcat’s HTTP connector component is Coyote. Finally, all JSP files are parsed into a compatible Java Servlets by a JSP-Engine. JASPER is the JSP-Engine for the tomcat. JASPER parses all JSP files in a web application into a compatible equivalent java Servlets file.

4 Software Architecture

This project is based on a very popular and successful software architecture style known as MVC2 architecture. In MVC2 architecture style the entire application is divided into layers. Each layer has separate responsibilities from another layer. Each layer cannot directly communicate with the other layer until they pass some criteria. This is known as "separation of concerns". Every layer addresses different concerns of the application. Here MVC2 architecture style has 3 layers a model, a view, and a controller.

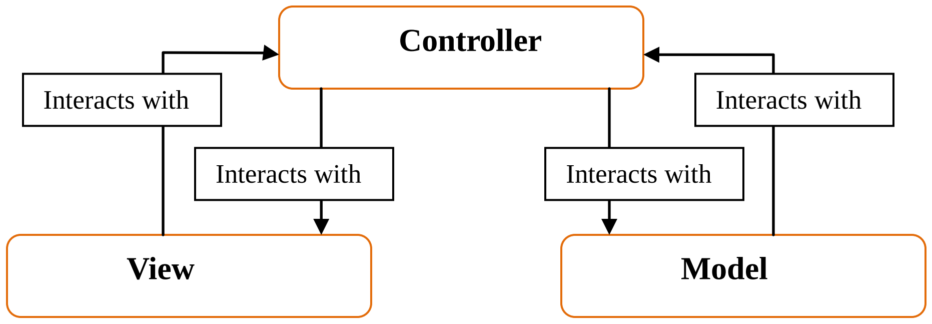


Fig 3.1 MVC2 Architecture

1) Model:

The model layer is comprised of model objects, which provide access to data, business logic to perform on data and methods to store the system state data. These data objects may represent the state of the system, the contents of a response, or the contents of a request depending upon the context. Once the request is given to the model object via a controller, the model applies appropriate business logic on the input based on the rules of the application to serve the request, generates results, and returns the response to the controller then the controller returns the response to the presentation layer i.e. to the client. In MVC2 architecture, models never talk directly to a view. Model objects are not concerned with the presentation of result data.

2) View:

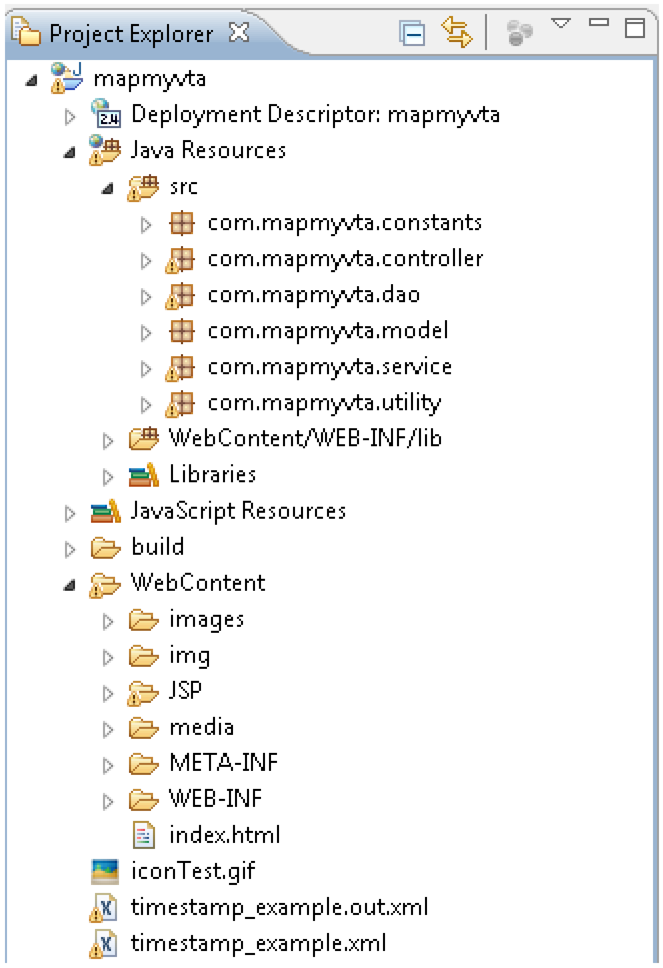
The view layer is the presentation layer of any application. The main purpose of the view layer is to provide a user-friendly interface of the system to the end user. View collects requests from the end user, hands it over to the controller and after getting a response from the controller, it prepares the data in a more presentable format for the user to view and to use. Views are not concerned with the execution of the business logic on the input data rather their work is just to render data, received from the controller in the correct format. The view layer in this project is written with JSP technology. Every JSP page in this project interacts only with the controller and renders its response on a JSP page. After this Apache server compiles this JSP page into a Servlet and then finally to a HTML static page which is delivered to the end user.

3) Controller:

Controller controls the behavior of the application. All requests for the application are directed to a single controller in the application. Controller then selects an appropriate model class with business logic to perform the requested operation, gets the response and hands it over to the appropriate view layer. Hence, the controller is an important link between the view and the model in the MVC2 architecture. In this project Servlet class (UserServices) is implemented as a controller. This controller class calls an appropriate model class for successful processing of request and then returns the response to a selected JSP page for presentation.

5 Web Application Directory Structure

This project is based on MVC2 architecture, therefore all inter connected modules are placed into the MVC layer like folder structure. All model classes are inside the model folder. Our controller is inside the controller folder. Constants for the entire application are inside the constant folder. Constants definition at one place improves code readability and increases code re-use. As constants can be made available to all classes using object composition therefore we don’t need to redefine them every time. Model classes use DAO (Data Access Objects) for data retrieval to generate response so they are placed inside the DAO folder. Our models have plenty of utility methods for various operations required for a proper response generation. Our views are placed inside the JSP folder of the directory structure. Hence, the view can now only be accessed via a Servlet as creating a directory structure prevents any invalid access to the view.



```
Project Explorer
◢ mapmyvta
    ▷ Deployment Descriptor: mapmyvta
    ◢ Java Resources
        ◢ src
            ▷ com.mapmyvta.constants
            ▷ com.mapmyvta.controller
            ▷ com.mapmyvta.dao
            ▷ com.mapmyvta.model
            ▷ com.mapmyvta.service
            ▷ com.mapmyvta.utility
        ▷ WebContent/WEB-INF/lib
        ▷ Libraries
    ▷ JavaScript Resources
    ▷ build
    ◢ WebContent
        ▷ images
        ▷ img
        ▷ JSP
        ▷ media
        ▷ META-INF
        ▷ WEB-INF
          index.html
      iconTest.gif
      timestamp_example.out.xml
      timestamp_example.xml
```

Fig 4.1 Web Application Directory Structure

Our icons for stops, buses, backgrounds, header, footers, are inside the img folder of the directory structure. The application's configuration file "web.xml" (DD) is inside the WEB-INF folder in which individual mappings of a context to a corresponding Servlet are defined.

6 Design & Implementation

6.1 Design

MapMyVTA is a 3-tier, MVC2 architecture based application. The 3 tiers are; the web browser – serves as the common standard universal interface for request and response to the application, a web server – receives the request and apply the application logic and returns the response to the user's browser, a data store- used as a repository of information in a customized version of GTFS information provided by VTA about the bus services, stops, and other line related information. In addition to it, the Google Maps JavaScript v3 gets loaded at the user browser via a CDN link. A CDN is known as Content Delivery Network, a connection of networked server for providing the content to the user's universally. So using a CDN link, a web browser can make a request for Google Maps API v3 on its own and load it from one of the CDN servers. Hence it removes the necessity of providing the Google Maps API file to the end users by the MapMyVTA application.

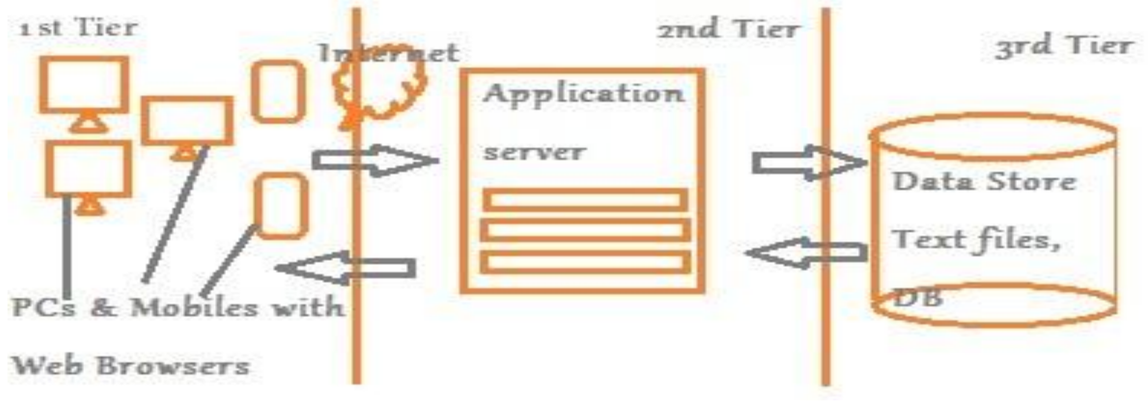


Figure 6.1.1: 3-Tier Architecture

The web browser and web server supports client-server architecture. The communication between them takes place over standard HTTP protocol. In this architecture client makes a request and thereby is an active component. Server on the other hand waits for a request and serves any request as they arrive so a server is a passive component. The application server and the data store communicate via model components (java classes which implements business logic for the request). Model components maintain the life cycle events of connections to the data stores. In case of MapMyVTA, the data stores are CSV text files (based on General Transit Feed Specification format) so the file handles are created and destroyed as and when needed.

For every software system, the design is the heart of the system. MapMyVTA is no exception to this rule. MapMyVTA is an attempt to provide every significant feature that a user may need while they wish to make a transit. MapMyVTA allows users to view VTA bus lines en route between a given pair of source and destination. It also allows users to view the timings of the buses in a dynamic display table. MapMyVTA is also equipped with a trip planner, which can plan a journey between a given pair of source and destination using VTA transit line services. MapMyVTA also has stop locator, attractions search features and a line services detailed dashboard for all the system supported line services.

It is evident for a map based trip planner, constraints are to make correct temporal, spatial, and system decisions while trip planning. According to Smith (2000)

[1] temporal and spatial decisions are two important decisions a map based trip planner has to make (p.47). Temporal Decisions include an ability of a trip planner to schedule the times to the source and to the destination, ability to determine the total trip time and to enforce a maximum trip time, constraints if any. Spatial decisions include the ability of a trip planner to identify all the transit stops surrounding the given source and destination stops. System constraints are the physical (memory/processing time) and virtual constraints (server uptime). According to Smith (2000) [1], goal of a good, map based trip planner is to fulfill all these requirements while keeping total trip time to the minimum (p.48). User constraints also include the user's freedom to choose from minimum transit time or minimum numbers of stops or minimum connections, or as minimum fare while requesting a trip planning service. Several authors have suggested similar opinions about this decision making process like Huang and Peng (2001) [1] and Donovan (1998) [1] to mention a few. To provide all these functionalities together in a single web application a simple design was needed which fulfill the constraints of the system and those of the user's.

MapMyVTA is designed using **"MVC2"** architecture style. The Entire web application is divided into 3 components namely Models, Views & Controller. The views contain the presentation logic which gives a user-friendly appearance to the result data. The Views in this application are JSP pages which takes the data from the service handler classes (model classes) applies the necessary presentation logic on the data and then these JSP pages are converted into a static HTML page by the server and are returned as a final response to the client(web browser). Hence the views are only concerned with the presentation of information. Then the Controllers are decision making Servlets classes which are invoked when a request is made at the view either by a click on a button or on a hyper link. As this project implements "Front Controller Design Pattern", hence only one controller Servlet receives every request. Therefore it is only a single controller which is responsible for making service decisions for every request received. The service decision to invoke an appropriate model method to serve a request is decided by the parameters that come along with the request. UserServices.java is the controller Servlet for MapMyVTA. This Controller Servlet makes appropriate calls to the ServiceHandler.java class which works as a class to locate services. ServiceHandler.java class has been designed to implement "Service Locator Design pattern" which helps Controller to access all the service of the system from a single instance of the ServiceHandler class.

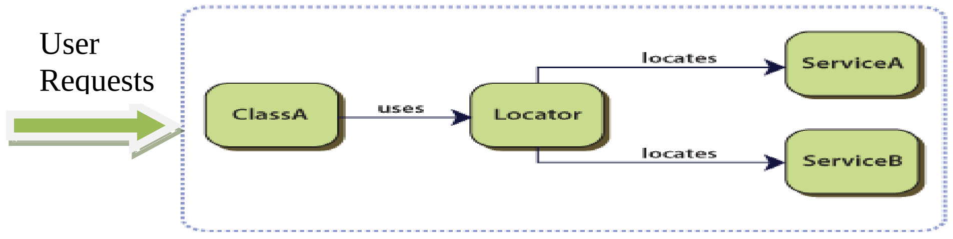
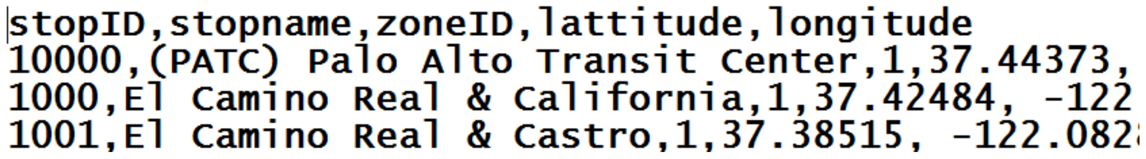


Figure 6.1.2:

Service locator design pattern [Image]. (2012). **Retrieved** April 14, 2012,
from: http://msdn.microsoft.com/en-us/library/ff648968.aspx

Third components as models are the classes which not only perform data store related activities but also perform the business logic associated with the request. So the actual data processing and application of business logic on the data for an appropriate response is performed by the model components. Models perform CRUD operations on the data. Models also perform the operation such as initialization of the system cache, creation of lines and stops objects from the data store files, then making uniform distribution of the time difference between any two stops based on the number of points between them, and preparing the output as in a JSON format for the response. The data store for this application is a collection of General Transit Feed Specification (GTFS) format files. The format of the data has been designed on top of the GTFS, which is a standard for transit authorities to provide transit information to any map related applications. These are simple csv text files which contain all information related to the bus lines, routes, stops, timing, hours of operations etc. Hence we grouped all related information under the same text file like a text file for stop, which will contain all information related to stops. Information stored in this manner also makes it very easy to represents them into appropriate objects in memory. Like a snapshot of the stopZone file is shown below.



```
stopID,stopname,zoneID,lattitude,longitude
10000,(PATC) Palo Alto Transit Center,1,37.44373,
1000,El Camino Real & California,1,37.42484, -122
1001,El Camino Real & Castro,1,37.38515, -122.082
```

Figure 6.1.3: CSV file example

There had been many challenges / design issues surfaced while balancing between the functionalities of the system and its simplicity. After all, the primary objective of this system is to present services in a very user friendly, simpler manner than to those currently available (MapMyVTA is the first application with respect to the set of services it offers now).

MapMyVTA implements an innovative approach for trip planning based on incremental-elimination filtering. This approach makes lesser use of space (memory) with respect to the other trip planning methods as this approach does not requires a road network and a transit network to be always available in memory. For example, there might be 50 additional points could have been between a pair of stops for displaying the motion of the bus on maps. As currently MapMyVTA serves two roles; one is of a service provider and other is of a service consumer, therefore to properly display the movement in a fraction of minute MapMyVTA requires more data where locations are mapped to timestamps. This data generated using the RouteQueryDashboard Servlet and later on the data is used as an input by MapMyVTA to display the route on the map. For one complete agency with all routes, the memory requirement could have been huge. But MapMyVTA implements incremental-elimination filtering to locate any stop presents in the system and to plan a trip. This technique does not require the road network or the transit network to be always available in memory and hence does not require much space. This approach also makes use of "divide n conquer" technique to plan a trip between two stops.

MapMyVTA utilizes a unique design which helps it to filter records incrementally until it reaches a situation of a match or no match. Incremental-elimination filtering is based on the facts that are true about all the elements in this project. Elements like transit lines, transit centers, stops etc. So the very facts, which are true about them help MapMyVTA to locate right information in a timely and efficient manner. Incremental-elimination filtering starts with the division of the service areas into a number of non-overlapping rectangular areas. Each rectangular area is tagged as a zone and assigned a zoneID. For example, in this case of a VTA service area, the area is divided into a total of 6 zones. Then each stop is assigned a unique systemID and a unique zoneID. Now the truth about each stopID is that the same stopID cannot belong to two different zones. If it would, that means two different zones have the same stop and it will refute the fact one that two zones cannot overlap. Now the truth about each transit line is that they will pass through one or more zones, therefore each zone will have one or more transit lines associated with it.

That means if stopID can be given a unique zone ID and then this zoneID can be utilized to fetch all the transit lines servicing in and/or around that particular zone. This will dramatically reduce the search space for all lines to a single zone out of “n” zones. Now the search space for a stop is limited only to lines, which serve in that particular zoneID. Now the search for a stop is limited to the lines servicing in that particular zone. This will make the search for a stop faster. Hence MapMyVTA also fulfills the spatial & temporal requirements as mentioned by Smith (1996) [5] (p.47).

Hence, MapMyVTA first filter the zone based on the zoneID and then it filters the transit lines by utilizing the zoneID. That is why it is called as incremental-elimination filtering. Now if source and destination stops are on the same line, the match will be found and a trip will be returned.

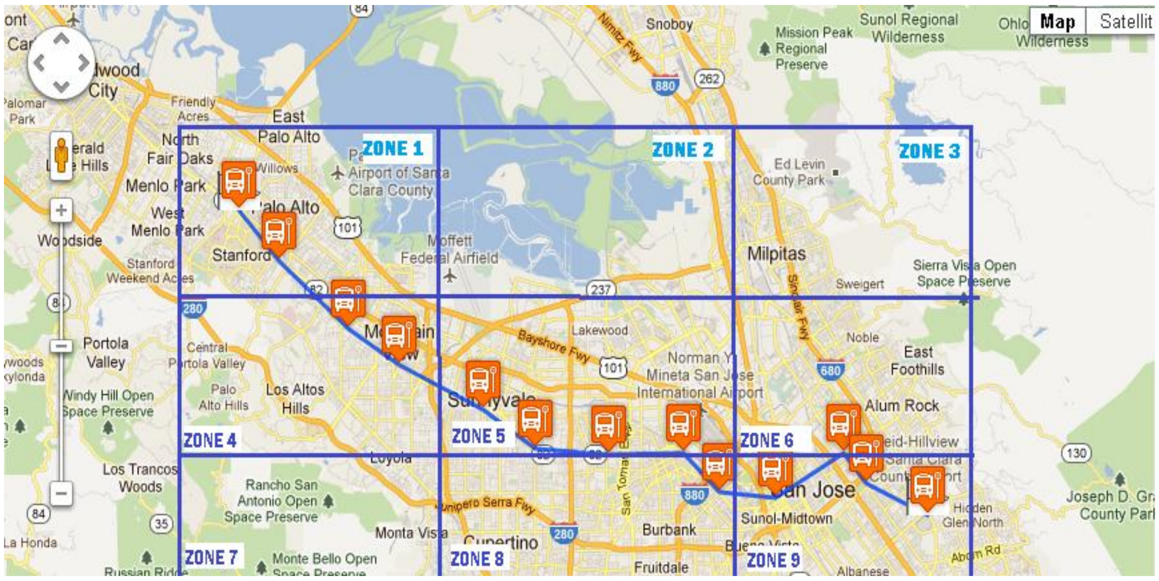


Figure 6.1.4 A sample non overlapping zones

(a division of a service area into unique zones)

Every stop and transit line model object also store the information regarding the connecting lines. Every time a stop is declared in the file as a transit center; it's connecting list also becomes the connecting list to that of line number, which serve the particular transit center. Whenever MapMyVTA search for a connection while planning a trip, it looks for this connection list of every transit line before looping through a stops list of any transit line. If any match with a transit line is found then that particular trip is added to the result object and search continues until the entire connection list of a source is compared against the list of lines that serve a destination. Now, 2 objects are created from the source to the connecting stop as trip 1 object and from connecting stop to destination stop as trip 2 object which then later on combined as one single object representing one complete planned trip.

Due to the nature of the application as to show and update the positions of the buses after every 15 seconds or less, the data collection rate for points between the stops is very high. Therefore a separate dashboard feature as a system service has been built which utilizes direction API of the Google maps and calls directions API for all stops pair within a route. The geocoding information for individual stop pair needs to be manually collected from a website known as itouchmap.com. Then dashboard gives a collection of points along the route between all stops pairs to show better movement of buses along the route. This process needs to be done exactly once for each route. Then these collections of points are stored in a csv file as route information. All stops objects for which VTA provides official timings are treated as major stops / transit centers and all other stops are treated as points along the route. So for all points between the two major stops the uniform distribution of time is performed so that every stop and every point are mapped to a single timestamp. uDistribute Method() of a model class **MappingServices.java** implements a bijective mapping function which maps every point from a route object to a unique timestamp. Then once the mapping is performed, the stop object and its corresponding timing are stored in a string which is in the JSON format. This JSON format information contains information about the route, stops, and timings. This JSON string is returned to the user as a part of the response on the appropriate page for bus mapping or planning a trip.

To accomplish all these functionalities successfully MapMyVTA utilizes a system cache, which helps to reduce response time and increase efficiency. A special Servlet is mapped to the context name of the application by setting a configuration file, web.xml of the application. Apache server uses this file to resolve the context mapping with the Servlets. Therefore, the first request to the system initializes the system cache.

6.2 Implementation

High level page flow diagram for the MapMyVTA Application:

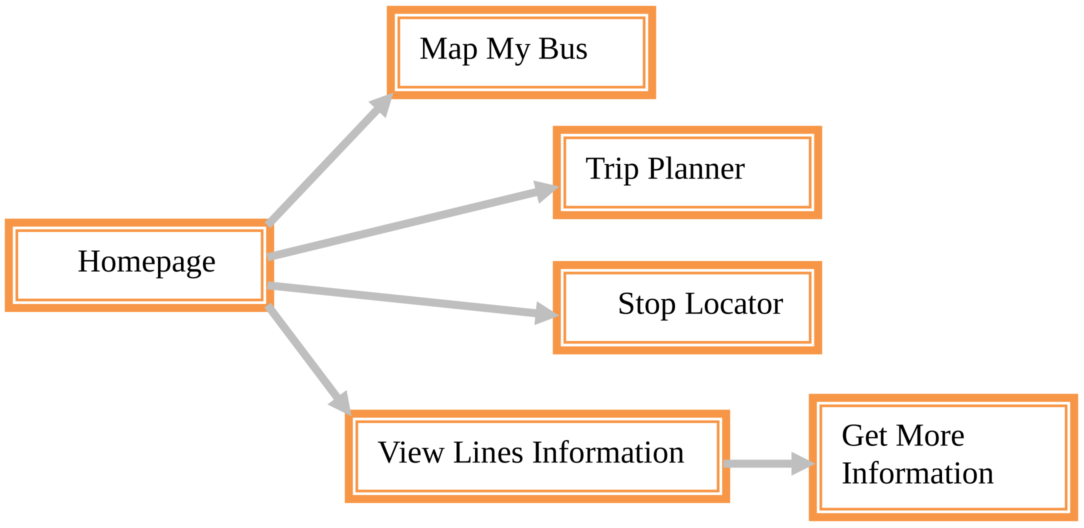
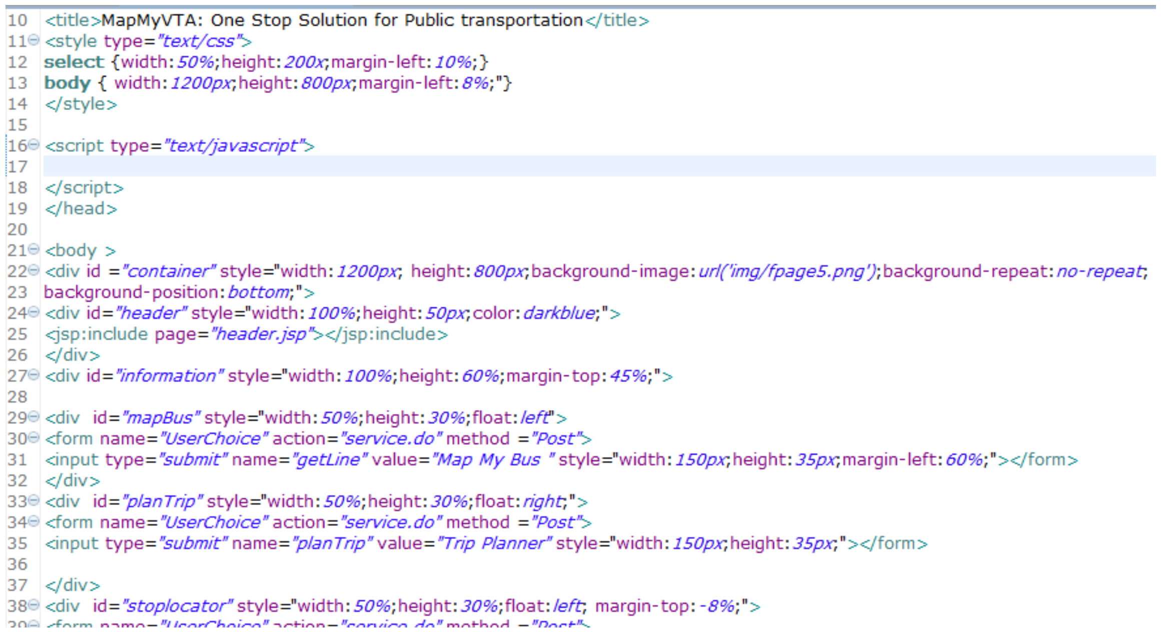


Figure 6.2.1: High level Page Flow Diagram

As the first feature user can select to view a particular service line on the map. For this user can select Map My Bus feature. The list of lines is shown as a drop down menu on the display page. Values of this drop down menu are set by the system cache. Once the request has been submitted to MapMyVTA, it will map all the buses, which are either at some stops or they are arriving at a stop soon.

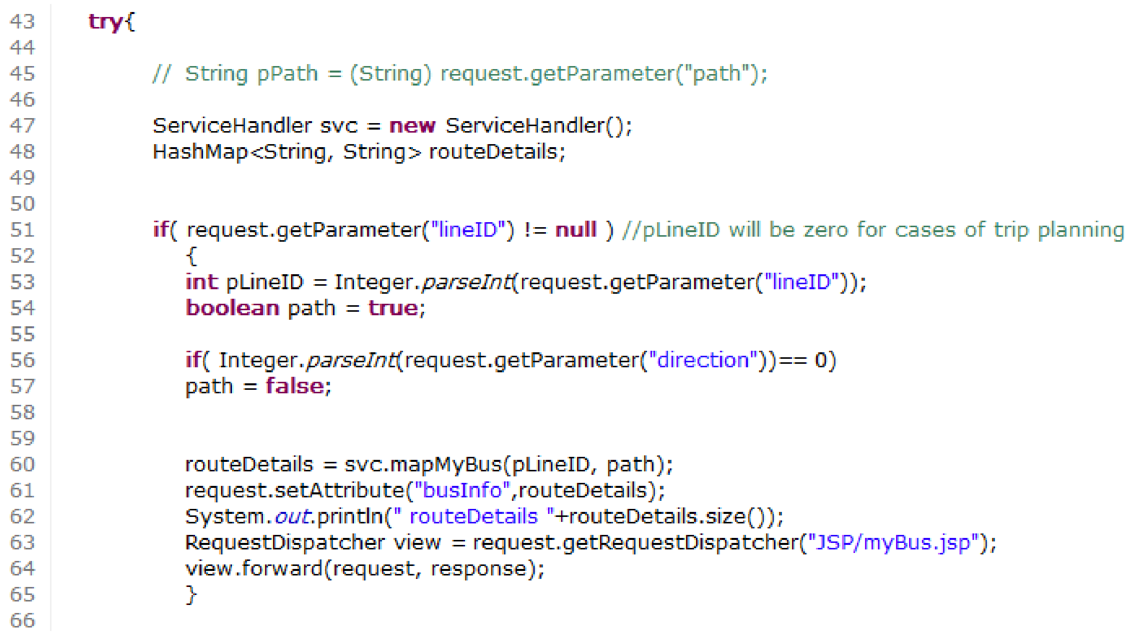


```
<title>MapMyVTA: One Stop Solution for Public transportation</title>
<style type="text/css">
select {width:50%;height:200x;margin-left:10%;}
body { width:1200px;height:800px;margin-left:8%;"}
</style>

<script type="text/javascript">

</script>
</head>

<body >
<div id ="container" style="width:1200px; height:800px;background-image:url('img/fpage5.png');background-repeat:no-repeat;
background-position:bottom;">
<div id="header" style="width:100%;height:50px;color:darkblue;">
<jsp:include page="header.jsp"></jsp:include>
</div>
<div id="information" style="width:100%;height:60%;margin-top:45%;">

<div  id="mapBus" style="width:50%;height:30%;float:left">
<form name="UserChoice" action="service.do" method ="Post">
<input type="submit" name="getLine" value="Map My Bus " style="width:150px;height:35px;margin-left:60%;"></form>
</div>
<div  id="planTrip" style="width:50%;height:30%;float:right;">
<form name="UserChoice" action="service.do" method ="Post">
<input type="submit" name="planTrip" value="Trip Planner" style="width:150px;height:35px;"></form>

</div>
<div  id="stoplocator" style="width:50%;height:30%;float:left; margin-top:-8%;">
<form name="UserChoice" action="service.do" method ="Post">
```

Figure 6.2.2 Homepage

From HomePage a user can select Map My bus feature which will direct him to the mapmybus page for making a selection about the bus line. Then mapmybus.jsp page utilizes a list, set by the controller in the request object under the attribute name “busList” for transit lines and “stopsList” for the list of stops.

Once a user has selected a transit line, a request is then sent to the single controller “UserServices.java”. There can be around 12 requests that can be made to the system and controller on the basis of the request parameters makes the decision about the action to take. For example, for the mapmybus feature the request will have a parameter lineID set as the number of bus service line.

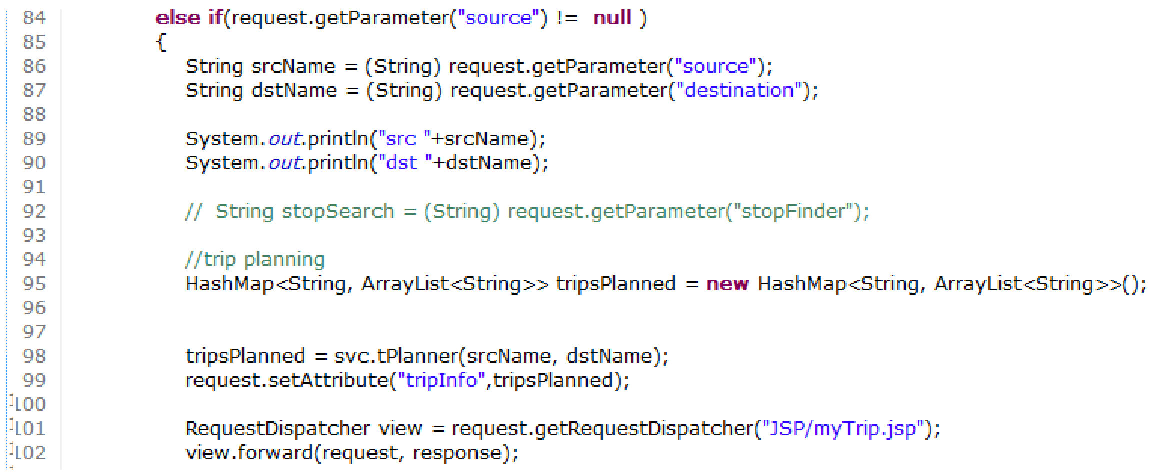


```
try{

        // String pPath = (String) request.getParameter("path");

        ServiceHandler svc = new ServiceHandler();
        HashMap<String, String> routeDetails;


        if( request.getParameter("lineID") != null ) //pLineID will be zero for cases of trip planning
            {
            int pLineID = Integer.parseInt(request.getParameter("lineID"));
            boolean path = true;

            if( Integer.parseInt(request.getParameter("direction"))== 0)
            path = false;


            routeDetails = svc.mapMyBus(pLineID, path);
            request.setAttribute("busInfo",routeDetails);
            System.out.println(" routeDetails "+routeDetails.size());
            RequestDispatcher view = request.getRequestDispatcher("JSP/myBus.jsp");
            view.forward(request, response);
            }
```

Figure 6.2.3 Input validation and direction checking

Then the controller also checks for the direction of the service requested if it is a one(1) means the direction is East/North bound zero(0) means the opposite direction.

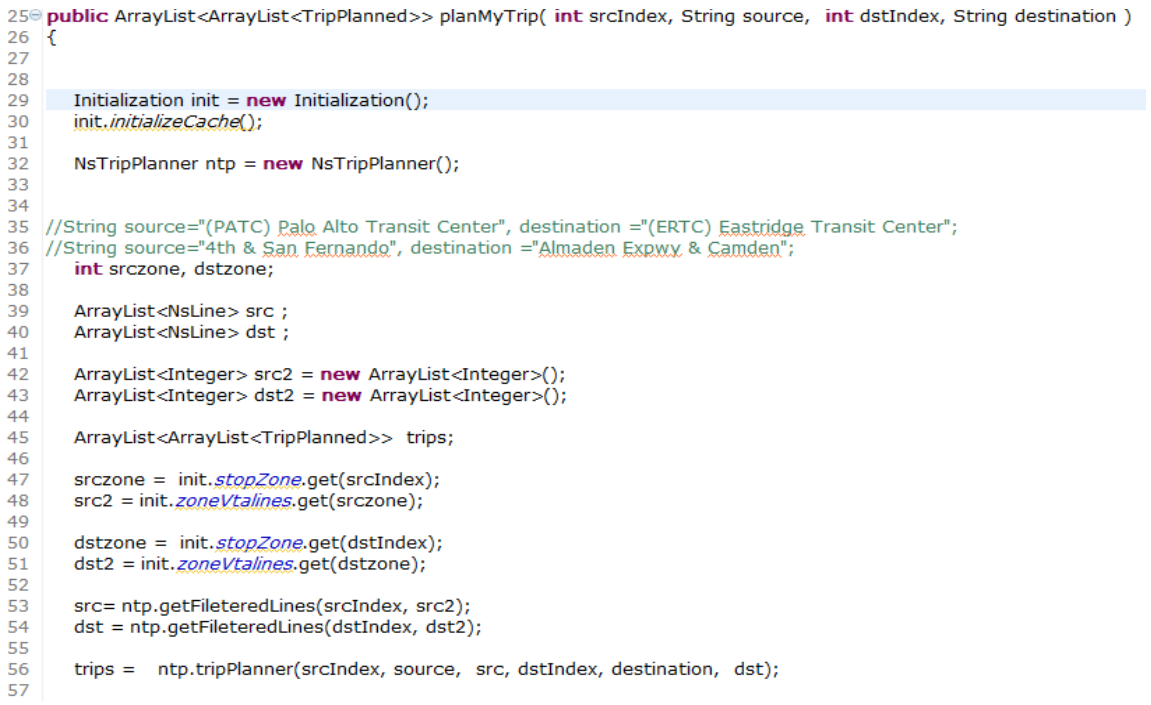
For trip planning, the parameters come as source and destination stop names. For every stop, the first task is to get the unique stopID which binds to this particular stop name hence the controller gets the appropriate stopID from the system cache. Then it passes the source ID and destination ID pair to the tPlanner method of the ServiceHandler class to get the result.



```
else if(request.getParameter("source") !=  null )
{
    String srcName = (String) request.getParameter("source");
    String dstName = (String) request.getParameter("destination");

    System.out.println("src "+srcName);
    System.out.println("dst "+dstName);

    //  String stopSearch = (String) request.getParameter("stopFinder");

    //trip planning
    HashMap<String, ArrayList<String>> tripsPlanned = new HashMap<String, ArrayList<String>>();


    tripsPlanned = svc.tPlanner(srcName, dstName);
    request.setAttribute("tripInfo",tripsPlanned);

    RequestDispatcher view = request.getRequestDispatcher("JSP/myTrip.jsp");
    view.forward(request, response);
```

Figure 6.2.4: Getting trips as result

tPlanner then filters the lines by zoneID to which the stops belong by calling a method getFileteredLines of NsTripPlanner class and then tPlanner calls the planMyTrip method of NsTripPlanner class to find out the no of trips that can be planned between source and destination stops.

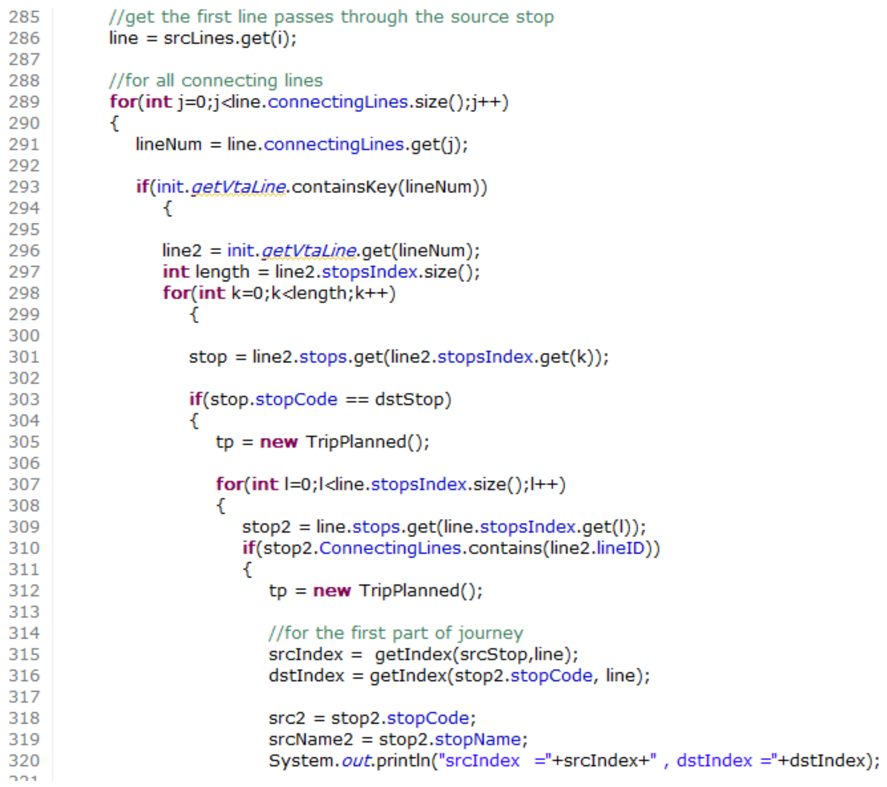


```
public ArrayList<ArrayList<TripPlanned>> planMyTrip( int srcIndex, String source,  int dstIndex, String destination )
{


    Initialization init = new Initialization();
    init.initializeCache();

    NsTripPlanner ntp = new NsTripPlanner();


//String source="(PATC) Palo Alto Transit Center", destination ="(ERTC) Eastridge Transit Center";
//String source="4th & San Fernando", destination ="Almaden Expwy & Camden";
    int srczone, dstzone;

    ArrayList<NsLine> src ;
    ArrayList<NsLine> dst ;

    ArrayList<Integer> src2 = new ArrayList<Integer>();
    ArrayList<Integer> dst2 = new ArrayList<Integer>();

    ArrayList<ArrayList<TripPlanned>>  trips;

    srczone =  init.stopZone.get(srcIndex);
    src2 = init.zoneVtalines.get(srczone);

    dstzone =  init.stopZone.get(dstIndex);
    dst2 = init.zoneVtalines.get(dstzone);

    src= ntp.getFileteredLines(srcIndex, src2);
    dst = ntp.getFileteredLines(dstIndex, dst2);

    trips =   ntp.tripPlanner(srcIndex, source,  src, dstIndex, destination,  dst);

```

Figure 6.2.5 Zone based filtering of lines & a call to the trip planning method

planMyTrip method then plans the trip for a source and a destination pair, if it can. For the direction of the service line, it calls another method getIndex, which helps the tripPlanner to ascertain the right direction and store the direction value in the object. Route objects also stores individual stop number in a route as well. Therefore direction calculation is based on the position of source and destination stops in a route. If a source stop's number is greater than a destination stop's number then the direction is opposite.

To plan a trip that requires one transfer planMyTrip method looks into the connecting list of individual lines originating from a source and matches it with the a destination line(s) if it matches, then planMyTrip adds those trips to the result.

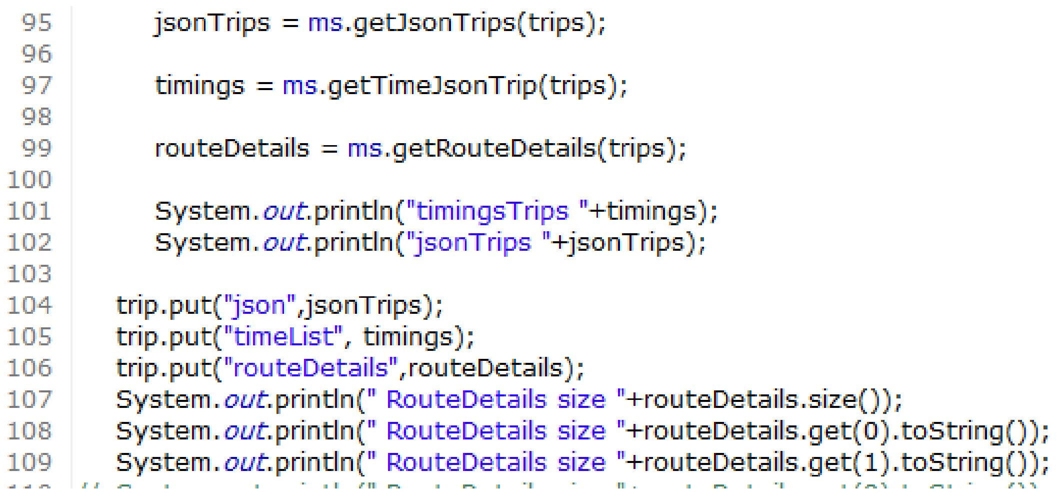


```
//get the first line passes through the source stop
line = srcLines.get(i);

//for all connecting lines
for(int j=0;j<line.connectingLines.size();j++)
{
    lineNum = line.connectingLines.get(j);

    if(init.getVtaLine.containsKey(lineNum))
        {

        line2 = init.getVtaLine.get(lineNum);
        int length = line2.stopsIndex.size();
        for(int k=0;k<length;k++)
            {

            stop = line2.stops.get(line2.stopsIndex.get(k));

            if(stop.stopCode == dstStop)
            {
                tp = new TripPlanned();

                for(int l=0;l<line.stopsIndex.size();l++)
                {
                    stop2 = line.stops.get(line.stopsIndex.get(l));
                    if(stop2.ConnectingLines.contains(line2.lineID))
                    {
                        tp = new TripPlanned();

                        //for the first part of journey
                        srcIndex =  getIndex(srcStop,line);
                        dstIndex = getIndex(stop2.stopCode, line);

                        src2 = stop2.stopCode;
                        srcName2 = stop2.stopName;
                        System.out.println("srcIndex  ="+srcIndex+" , dstIndex ="+dstIndex);
```

Figure 6.2.6 Connecting lines check for every stop

When planMyTrip returns the result to the tPlanner, the result is in a java object form, which needs to be converted into an appropriate JSON format so that the front end (view) can understand and use.

Hence tPlanner passes the results object to MappingServices class and asks an ArrayList of String data type which stores the result in JSON format



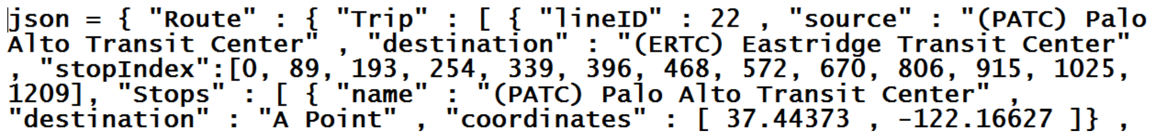
```
        jsonTrips = ms.getJsonTrips(trips);

        timings = ms.getTimeJsonTrip(trips);

        routeDetails = ms.getRouteDetails(trips);

        System.out.println("timingsTrips "+timings);
        System.out.println("jsonTrips "+jsonTrips);

    trip.put("json",jsonTrips);
    trip.put("timeList", timings);
    trip.put("routeDetails",routeDetails);
    System.out.println(" RouteDetails size "+routeDetails.size());
    System.out.println(" RouteDetails size "+routeDetails.get(0).toString());
    System.out.println(" RouteDetails size "+routeDetails.get(1).toString());
```

Figure 6.2.7 Getting results in JSON

Mapping services performs the following operations: 1. Data cleansing operation on the time-table, 2. Appropriate decisions making regarding the day of the service and selecting the corresponding published schedules, 3. Performing the Uniform distribution and bijective mapping of timestamps to points along a route, and 4. Populating the result in a JSON format which looks like similar to the one shown below. Mapping services utilizes different modules to accomplish all these functionalities and then put them together as a single result. It performs a pipeline operation where the output of one method becomes the input to another till the results finally become a JSON string.

This resulting JSON formatted String looks like this

For Planned Routes:

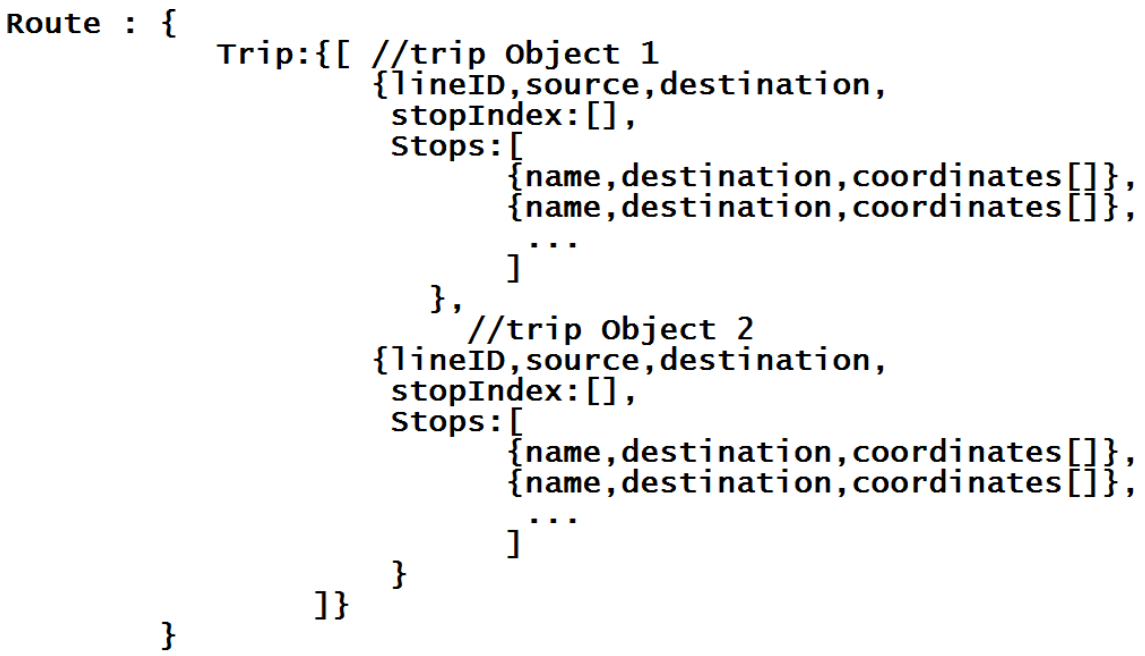


```
json = { "Route" : { "Trip" : [ { "lineID" : 22 , "source" : "(PATC) Palo
Alto Transit Center" , "destination" : "(ERTC) Eastridge Transit Center"
, "stopIndex":[0, 89, 193, 254, 339, 396, 468, 572, 670, 806, 915, 1025,
1209], "Stops" : [ { "name" : "(PATC) Palo Alto Transit Center" ,
"destination" : "A Point" , "coordinates" : [ 37.44373 , -122.16627 ]} ,
```

Figure 6.2.8 JSON Data – Routes (all planned trips as results)

Here Route is an object containing details of all routes for this particular trip; the trip is the type of arrays of objects. For every trip one trip type of object will be added to this JSON data. lineID is the number representing the transit line's number. A source and a destination pairs are source and destination stops for this particular trip. Then stop is another array of objects, which contains all information related to stops for this particular trip.

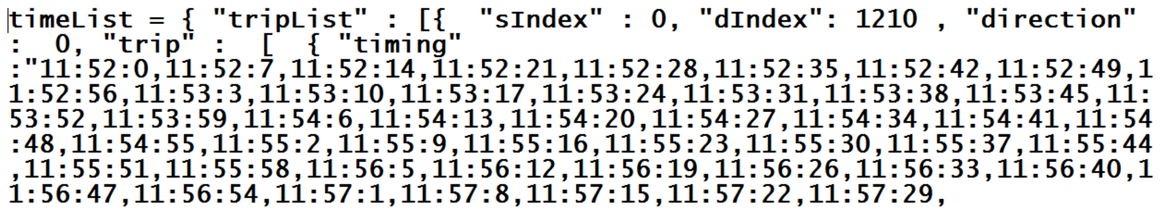
The general structure of json data for a planned route is as follows:



```
Route : {
           Trip:{[ //trip Object 1
                   {lineID,source,destination,
                    stopIndex:[],
                    Stops:[
                          {name,destination,coordinates[]},
                          {name,destination,coordinates[]},
                           ...
                          ]
                      },
                        //trip Object 2
                   {lineID,source,destination,
                    stopIndex:[],
                    Stops:[
                          {name,destination,coordinates[]},
                          {name,destination,coordinates[]},
                           ...
                          ]
                    }
               ]}
        }
```

Figure 6.2.9: General Structure – Routes JSON data

Similarly a JSON String for the timing also has arrays of timing information for a complete listing of timings for trips:

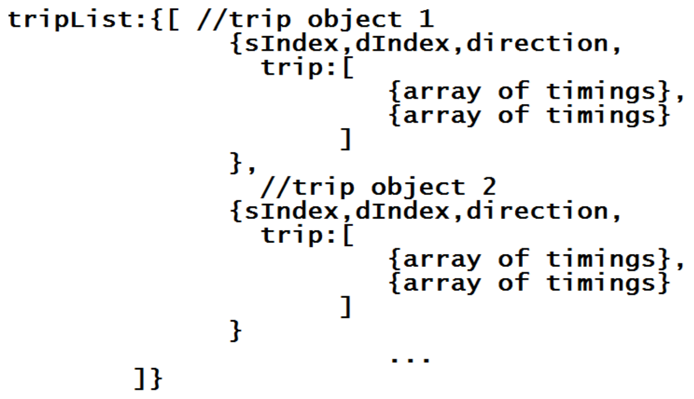


```
timeList = { "tripList" : [{  "sIndex" : 0, "dIndex": 1210 , "direction"
:  0, "trip" :  [  { "timing"
:"11:52:0,11:52:7,11:52:14,11:52:21,11:52:28,11:52:35,11:52:42,11:52:49,1
1:52:56,11:53:3,11:53:10,11:53:17,11:53:24,11:53:31,11:53:38,11:53:45,11:
53:52,11:53:59,11:54:6,11:54:13,11:54:20,11:54:27,11:54:34,11:54:41,11:54
:48,11:54:55,11:55:2,11:55:9,11:55:16,11:55:23,11:55:30,11:55:37,11:55:44
,11:55:51,11:55:58,11:56:5,11:56:12,11:56:19,11:56:26,11:56:33,11:56:40,1
1:56:47,11:56:54,11:57:1,11:57:8,11:57:15,11:57:22,11:57:29,
```

Figure 6.2.10 JSON Data – Timings (for each trip in a Route)

Here the triplist is an array of objects, which contains information for the entire timings schedule for a given source and destination pair.

The general structure of json data for timing data is as follows:

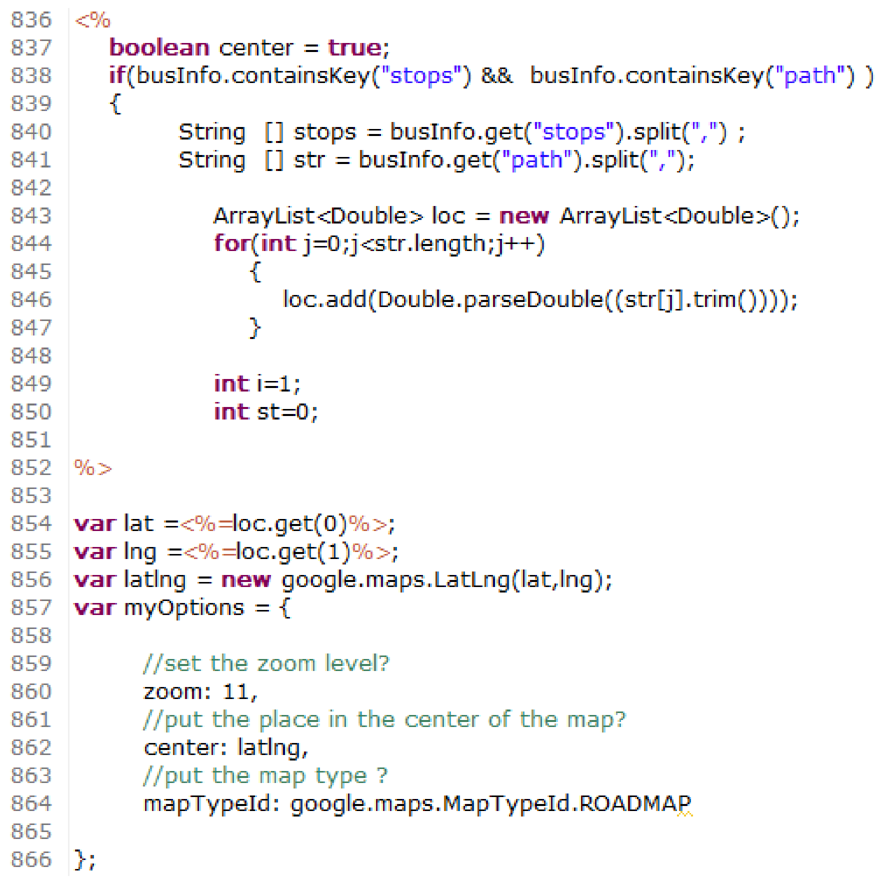


```
tripList:{[ //trip object 1
              {sIndex,dIndex,direction,
                trip:[
                        {array of timings},
                        {array of timings}
                     ]
              },
                //trip object 2
              {sIndex,dIndex,direction,
                trip:[
                        {array of timings},
                        {array of timings}
                     ]
              }
                        ...
        ]}
```

Figure 6.2.11: General Structure – timings JSON Data

Now we will see the details of implementation on the client side and how to interpret and use these details:

- Creating a map



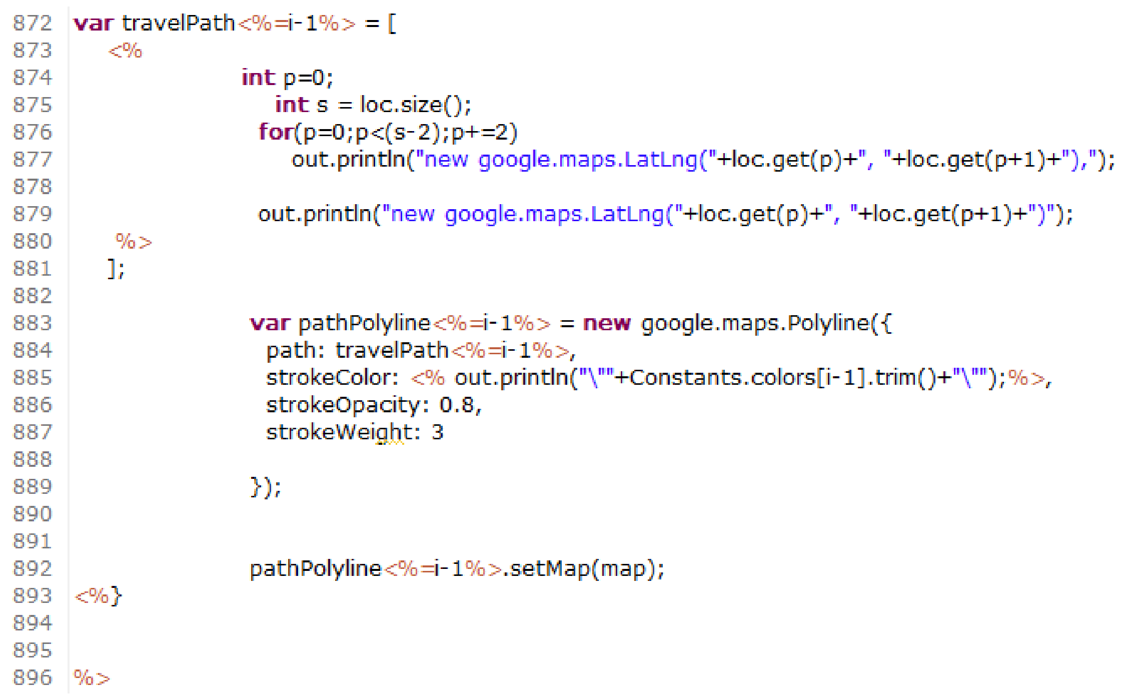
```
<%
   boolean center = true;
   if(busInfo.containsKey("stops") &&  busInfo.containsKey("path") )
   {
         String  [] stops = busInfo.get("stops").split(",") ;
         String  [] str = busInfo.get("path").split(",");

            ArrayList<Double> loc = new ArrayList<Double>();
            for(int j=0;j<str.length;j++)
               {
                  loc.add(Double.parseDouble((str[j].trim())));
               }

            int i=1;
            int st=0;

%>

var lat =<%=loc.get(0)%>;
var lng =<%=loc.get(1)%>;
var latlng = new google.maps.LatLng(lat,lng);
var myOptions = {

      //set the zoom level?
      zoom: 11,
      //put the place in the center of the map?
      center: latlng,
      //put the map type ?
      mapTypeId: google.maps.MapTypeId.ROADMAP

};
```

Figure 6.2.12 Creating a map

This function “initialize()” called only once when MapMyBus page loads for the first time.

This function is called by the body =onload() function to initialize the map, to color the routes, and to geocode all the stops for this trip as these settings are going to remain same for this page. This function uses Google Maps API service for drawing a map and then setting the necessary options for the map and then uses the div section of the HTML page named as “animap” to draw the map.

- Color coded Routes

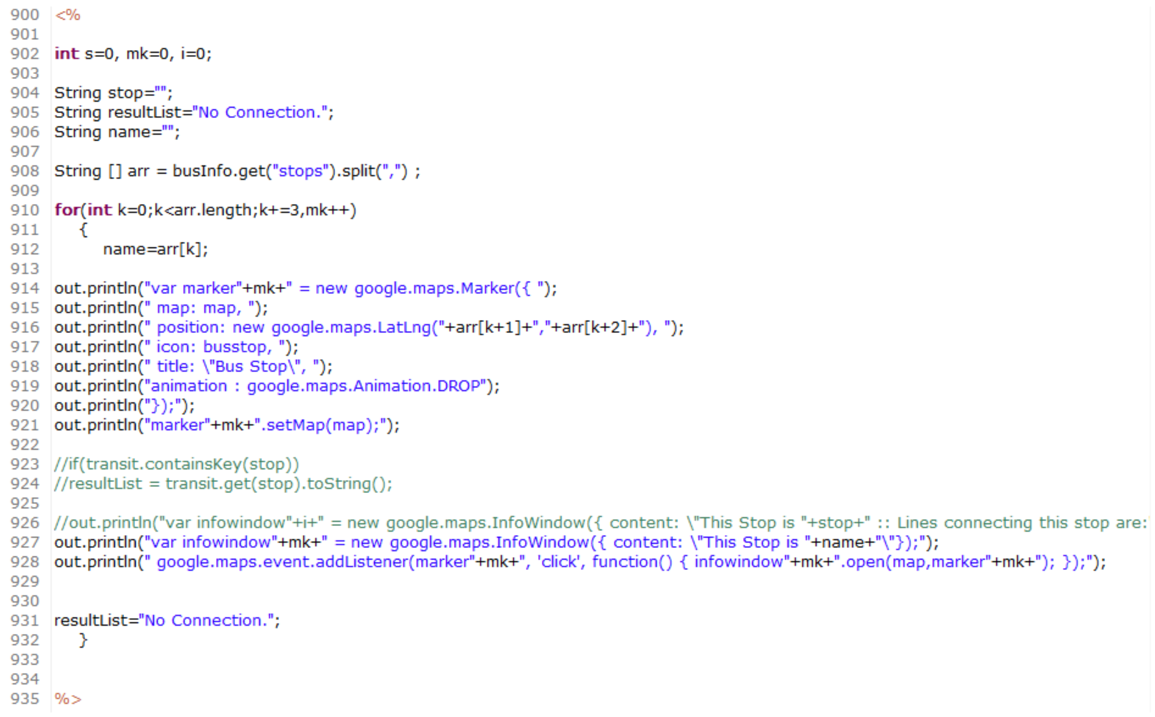


```
var travelPath<%=i-1%> = [
    <%
                   int p=0;
                       int s = loc.size();
                     for(p=0;p<(s-2);p+=2)
                         out.println("new google.maps.LatLng("+loc.get(p)+", "+loc.get(p+1)+"),");

                     out.println("new google.maps.LatLng("+loc.get(p)+", "+loc.get(p+1)+")");
     %>
    ];

                    var pathPolyline<%=i-1%> = new google.maps.Polyline({
                      path: travelPath<%=i-1%>,
                      strokeColor: <% out.println("\""+Constants.colors[i-1].trim()+"\"");%>,
                      strokeOpacity: 0.8,
                      strokeWeight: 3

                    });


                    pathPolyline<%=i-1%>.setMap(map);
<%}


%>
```

Figure 6.2.13 Routes coloring

This part comes after a map is initialized. In this part we retrieve a list of geocodes for all the stops from the request object as an array of stops and their geocodes, and we set these arrays of geocodes as polyline objects. A polyline is shown as a colored route, which passes through all points of a given array of stops on the map.

- Stop Positions

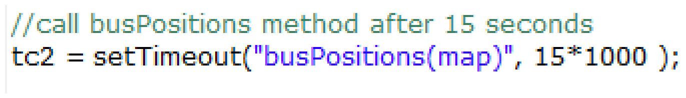


```
<%

int s=0, mk=0, i=0;

String stop="";
String resultList="No Connection.";
String name="";

String [] arr = busInfo.get("stops").split(",") ;

for(int k=0;k<arr.length;k+=3,mk++)
   {
      name=arr[k];

out.println("var marker"+mk+" = new google.maps.Marker({ ");
out.println(" map: map, ");
out.println(" position: new google.maps.LatLng("+arr[k+1]+","+arr[k+2]+"), ");
out.println(" icon: busstop, ");
out.println(" title: \"Bus Stop\", ");
out.println("animation : google.maps.Animation.DROP");
out.println("});");
out.println("marker"+mk+".setMap(map);");

//if(transit.containsKey(stop))
//resultList = transit.get(stop).toString();

//out.println("var infowindow"+i+" = new google.maps.InfoWindow({ content: \"This Stop is "+stop+" :: Lines connecting this stop are:
out.println("var infowindow"+mk+" = new google.maps.InfoWindow({ content: \"This Stop is "+name+"\"});");
out.println(" google.maps.event.addListener(marker"+mk+", 'click', function() { infowindow"+mk+".open(map,marker"+mk+"); });");


resultList="No Connection.";
   }


%>
```

Figure 6.2.14 Mapping Stops (Plotting Stops on the map for a route)

Then for every stop name we recursively call Google Maps API with the geocodes of the stop to plot them on the map. While doing so it is important to give a unique name to each marker else without a unique name when we attach information window for displaying information it would not store a proper message.

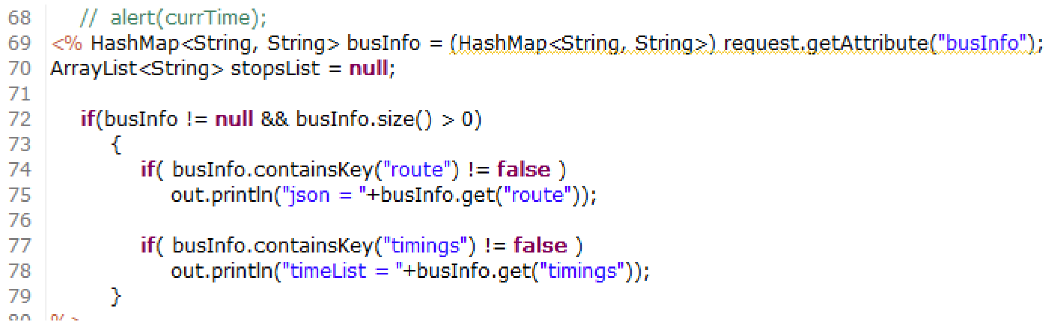
Finally, we have set our map then we call another JavaScript method “busPositions()” to start interpreting the result data and start showing the buses and timing information. SetTimeout function calls the method after the delay specified in microseconds. Here in this case, busPositions is called after every 15 seconds once control reaches at this position.



```
//call busPositions method after 15 seconds
tc2 = setTimeout("busPositions(map)", 15*1000 );
```

Figure 6.2.15 Caculating bus positions (recursive call to calculate bus positions)

- Calculating bus positions

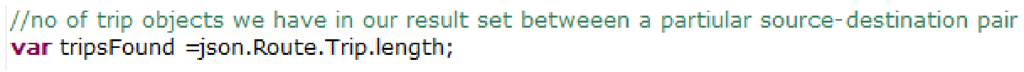


```
   // alert(currTime);
<% HashMap<String, String> busInfo = (HashMap<String, String>) request.getAttribute("busInfo");
ArrayList<String> stopsList = null;

   if(busInfo != null && busInfo.size() > 0)
      {
         if( busInfo.containsKey("route") != false )
            out.println("json = "+busInfo.get("route"));

         if( busInfo.containsKey("timings") != false )
            out.println("timeList = "+busInfo.get("timings"));
      }
```

Figure 6.2.16 Calculating bus positions

This JavaScript function first gets the data from the request object. It stores the route information in a JavaScript variable “json” and it stores the timing information in another JavaScript variable called as “timeList”. As this information is in the JSON format, therefore these variables are inherently proper JavaScript objects.

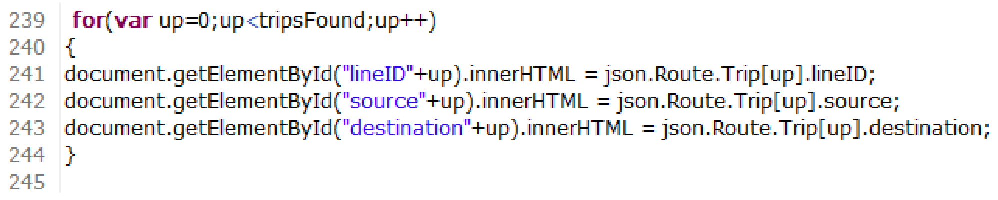
This function then initializes the number of routes returned by the server for the requested source and destination pair, By



```
//no of trip objects we have in our result set betweeen a partiular source-destination pair
var tripsFound =json.Route.Trip.length;
```

Figure 6.2.17 Getting total trips (No of planned trips for the route)

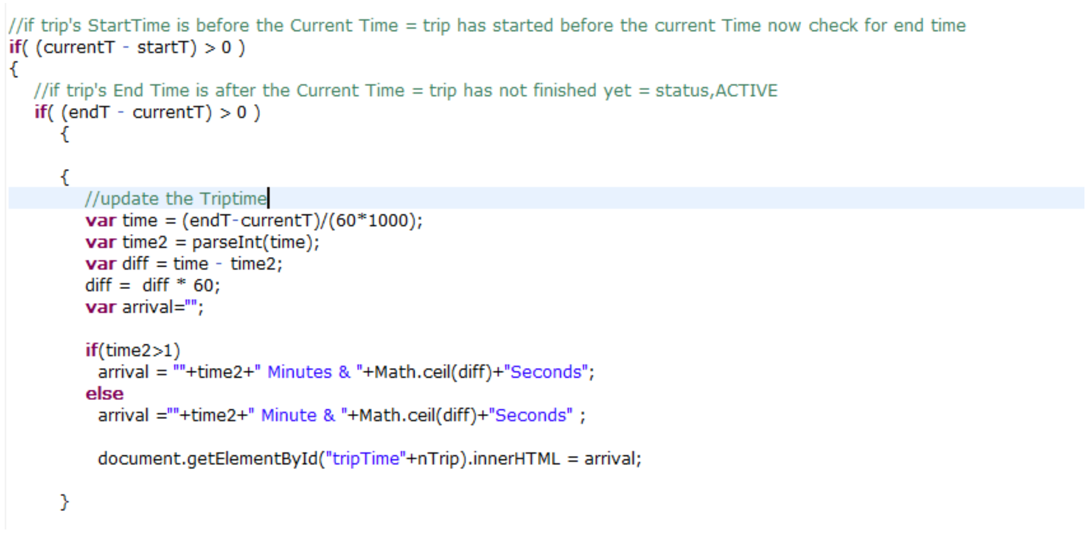
Then it puts a for loop to estimate the positions of the buses for all available timings list in the variable timeList & initializes the MapMyVTA information Panel once for one iteration as



```
for(var up=0;up<tripsFound;up++)
{
document.getElementById("lineID"+up).innerHTML = json.Route.Trip[up].lineID;
document.getElementById("source"+up).innerHTML = json.Route.Trip[up].source;
document.getElementById("destination"+up).innerHTML = json.Route.Trip[up].destination;
}
```

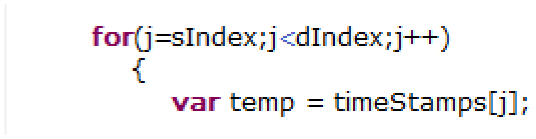
Figure 6.2.18 Iterating for all routes

If the trip’s timing’s start time is less than the current time and journey’s end time is greater than the current time that means the bus is live on the route somewhere so to locate its position between the given pair of,



```
//if trip's StartTime is before the Current Time = trip has started before the current Time now check for end time
if( (currentT - startT) > 0 )
{
   //if trip's End Time is after the Current Time = trip has not finished yet = status,ACTIVE
   if( (endT - currentT) > 0 )
      {

      {
         //update the Triptime
         var time = (endT-currentT)/(60*1000);
         var time2 = parseInt(time);
         var diff = time - time2;
         diff =  diff * 60;
         var arrival="";

         if(time2>1)
           arrival = ""+time2+" Minutes & "+Math.ceil(diff)+"Seconds";
         else
           arrival =""+time2+" Minute & "+Math.ceil(diff)+"Seconds" ;

           document.getElementById("tripTime"+nTrip).innerHTML = arrival;

      }
```

Figure 6.2.19: Trips Filtering

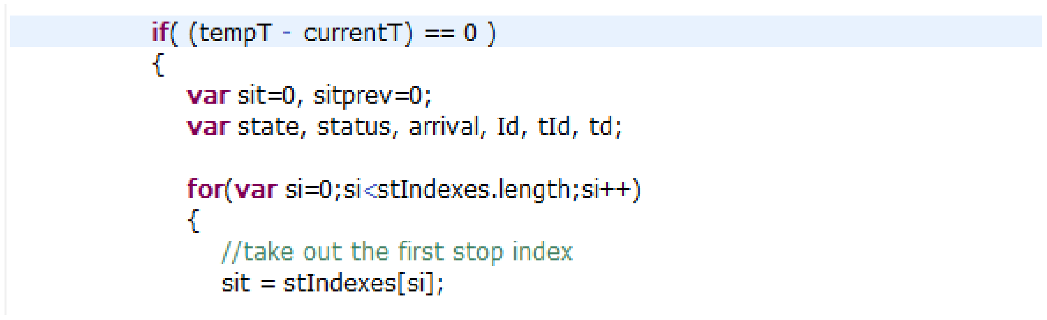


```
for(j=sIndex;j<dIndex;j++)
   {
      var temp = timeStamps[j];
```

Figure 6.2.20: iterating for timings between source and destination stops

sIndex= source stop position in the current list,

dIndex = destination stop position in the current list

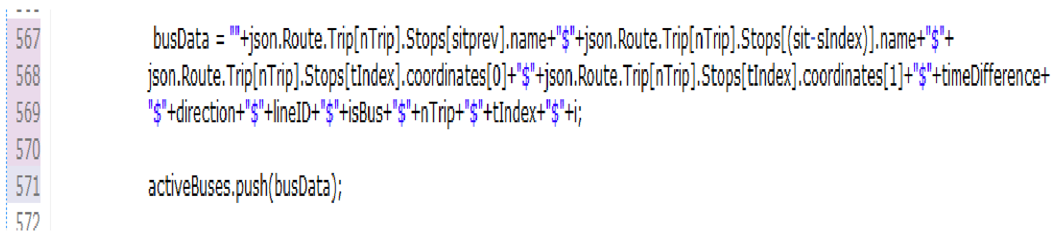


```
if( (tempT - currentT) == 0 )
{
   var sit=0, sitprev=0;
   var state, status, arrival, Id, tId, td;

   for(var si=0;si<stIndexes.length;si++)
   {
      //take out the first stop index
      sit = stIndexes[si];
```

Figure 6.2.21 Time comparison CASE 1(current time == bus time)

If a match is found for timings between source and destination stops then add it to an array called as activeBuses holding information about identified active buses, the String busData will contain all information for the bus including the time to next stop, direction, and type of vehicle (bus/rail).

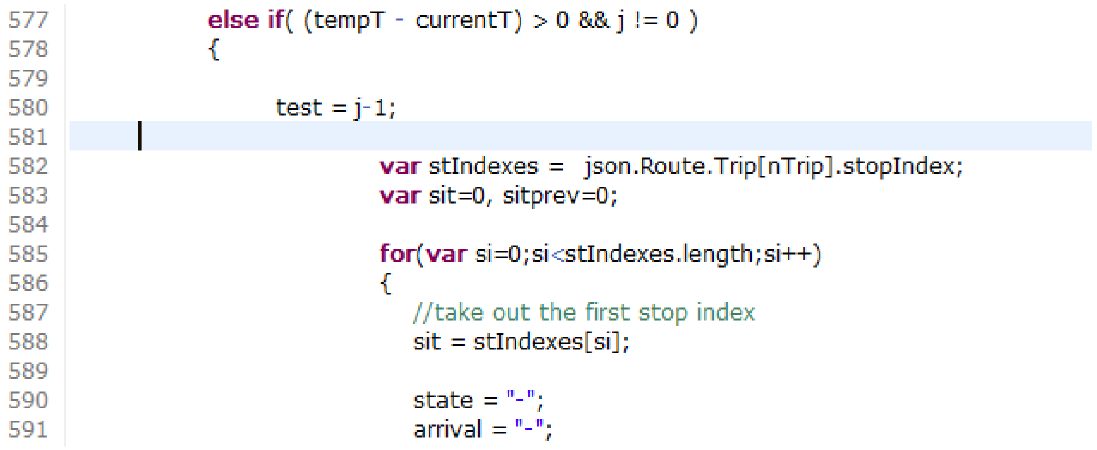
The data inside this string are added with a token "$" which will be used to split this string into a number of individual string of individual data items. This will help to get all the individual data back from this String.



```
 busData = ""+json.Route.Trip[nTrip].Stops[sitprev].name+"$"+json.Route.Trip[nTrip].Stops[(sit-sIndex)].name+"$"+
json.Route.Trip[nTrip].Stops[tIndex].coordinates[0]+"$"+json.Route.Trip[nTrip].Stops[tIndex].coordinates[1]+"$"+timeDifference+
"$"+direction+"$"+lineID+"$"+isBus+"$"+nTrip+"$"+tIndex+"$"+i;

activeBuses.push(busData);
```

Figure 6.2.22 Populating result set (Adding result data to an array of results)

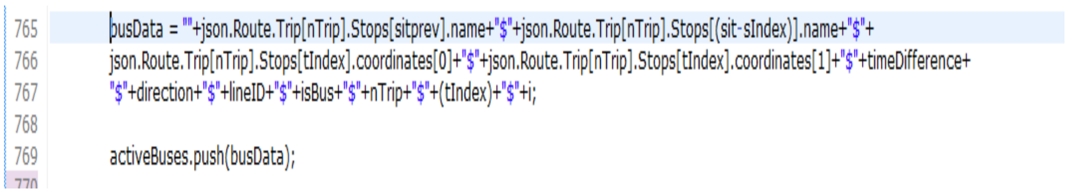
At the same time set the dynamic display table for this particular bus as "Arrived". Now,



```
else if( (tempT - currentT) > 0 && j != 0 )
{

    test = j-1;

                var stIndexes =  json.Route.Trip[nTrip].stopIndex;
                var sit=0, sitprev=0;

                for(var si=0;si<stIndexes.length;si++)
                {
                   //take out the first stop index
                   sit = stIndexes[si];

                   state = "-";
                   arrival = "-";
```

Figure 6.2.23 Time comparison case 2 (current time > bus time)

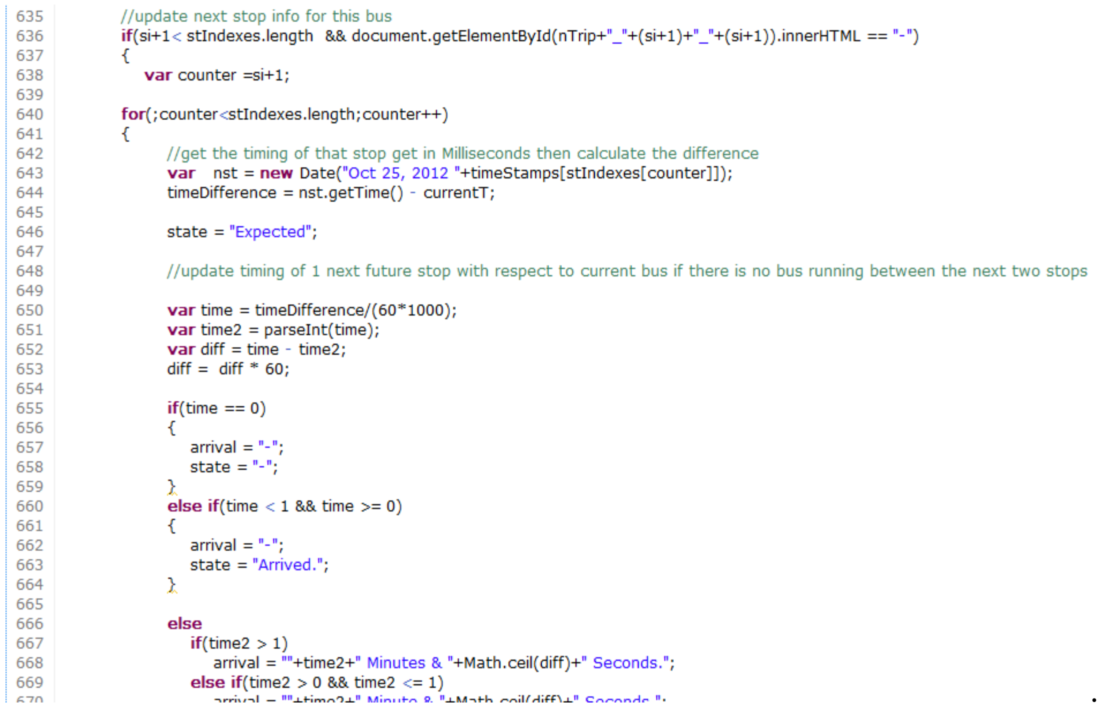
if time of the bus is greater than current time but not equal to timings of any of the stops timing then the bus must be somewhere between the current stop and one stop before it so add the bus info with all other details consider the current stop as destination and one stop before it as a source stop.



```
busData = ""+json.Route.Trip[nTrip].Stops[sitprev].name+"$"+json.Route.Trip[nTrip].Stops[(sit-sIndex)].name+"$"+
json.Route.Trip[nTrip].Stops[tIndex].coordinates[0]+"$"+json.Route.Trip[nTrip].Stops[tIndex].coordinates[1]+"$"+timeDifference+
"$"+direction+"$"+lineID+"$"+isBus+"$"+nTrip+"$"+(tIndex)+"$"+i;

activeBuses.push(busData);
```

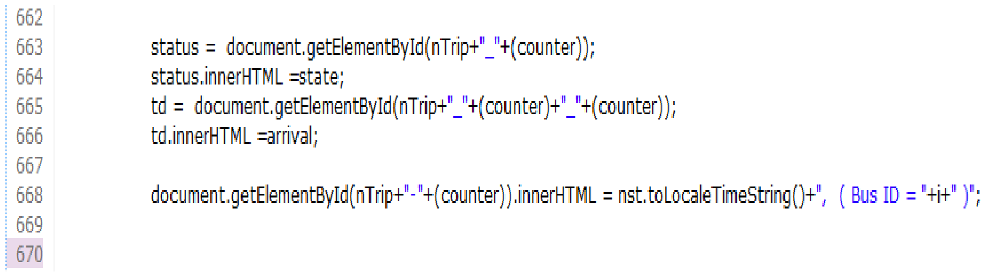
Figure 6.2.24 Populating result set

At the same time, to set the dynamic display table value of this particular bus as “Arriving” and set the time to the next stop. Every major stop in a route has a corresponding element id in the HTML table. This element id is used to access a specific id, to store the information about the particular stop and to update the table for every run of bus positions calculation method. The id’s of the elements and the corresponding trip indexes have been matched to avoid ambiguity and the above two cases are handled something like this for the table,



```
//update next stop info for this bus
if(si+1< stIndexes.length  && document.getElementById(nTrip+"_"+(si+1)+"_"+(si+1)).innerHTML == "-")
{
    var counter =si+1;

for(;counter<stIndexes.length;counter++)
{
        //get the timing of that stop get in Milliseconds then calculate the difference
        var   nst = new Date("Oct 25, 2012 "+timeStamps[stIndexes[counter]]);
        timeDifference = nst.getTime() - currentT;

        state = "Expected";

        //update timing of 1 next future stop with respect to current bus if there is no bus running between the next two stops

        var time = timeDifference/(60*1000);
        var time2 = parseInt(time);
        var diff = time - time2;
        diff =  diff * 60;

        if(time == 0)
        {
            arrival = "-";
            state = "-";
        }
        else if(time < 1 && time >= 0)
        {
            arrival = "-";
            state = "Arrived.";
        }

        else
            if(time2 > 1)
                arrival = ""+time2+" Minutes & "+Math.ceil(diff)+" Seconds.";
            else if(time2 > 0 && time2 <= 1)
                arrival = ""+time2+" Minute & "+Math.ceil(diff)+" Seconds.";
```

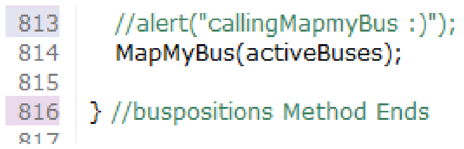
Figure 6.2.25 Calculating time & status for the dynamic display table



```
status =  document.getElementById(nTrip+"_"+(counter));
status.innerHTML =state;
td =  document.getElementById(nTrip+"_"+(counter)+"_"+(counter));
td.innerHTML =arrival;

document.getElementById(nTrip+"-"+(counter)).innerHTML = nst.toLocaleTimeString()+",  ( Bus ID = "+i+" )";
```

Figure 6.2.26 Updating time & status in the dynamic display table

Once details for all the buses are added, now call mapping method to draw them on the map by calling MapMyBus method



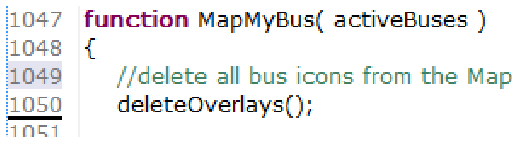
```
    //alert("callingMapmyBus :)");
    MapMyBus(activeBuses);

} //buspositions Method Ends
```

Figure 6.2.27 A call to map my bus method

MapMyBus takes an array of strings, which has all the details of the buses active at a particular point of time.

- Bus Mapping

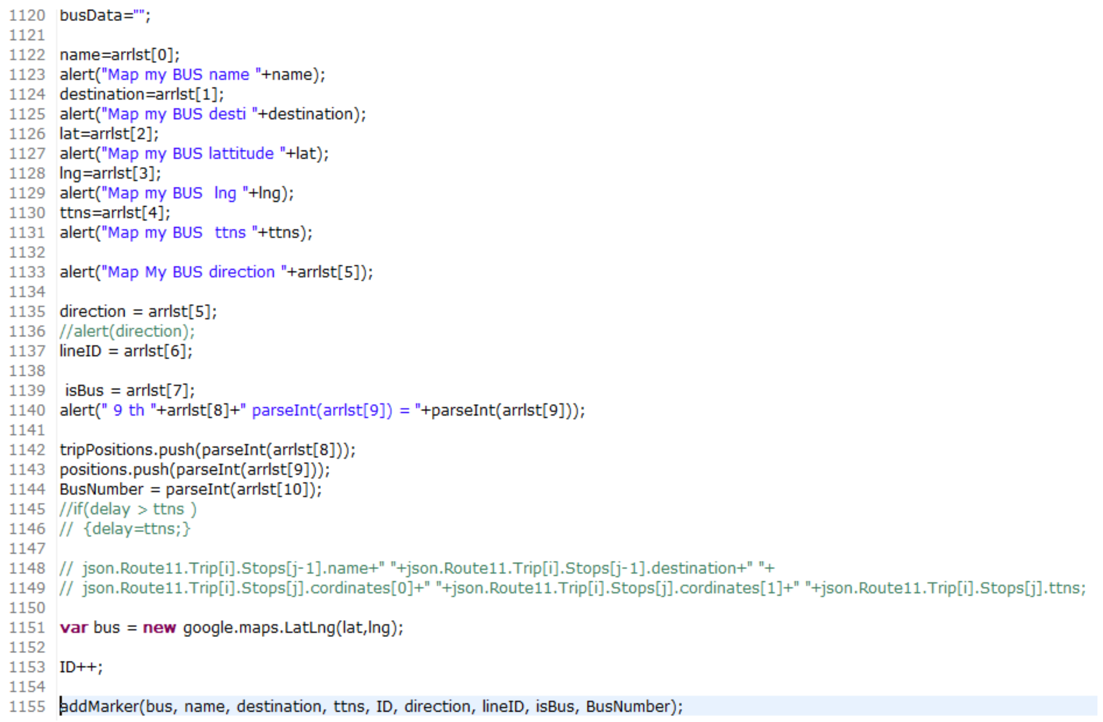
MapMyBus method() takes the array of information about active buses in the form of string array. But before mapping new buses we need to remove the buses mapped before. So it calls a mehod deleteOverlays() which deletes all previous bus icons from the map. Complete descriptions of the Google Maps APIs are beyond the scope of this report. However, the official documentation available online can be consulted for further information.



```
function MapMyBus( activeBuses )
{
   //delete all bus icons from the Map
   deleteOverlays();
```

Figure 6.2.28 Delete Overlays (on the Google map)

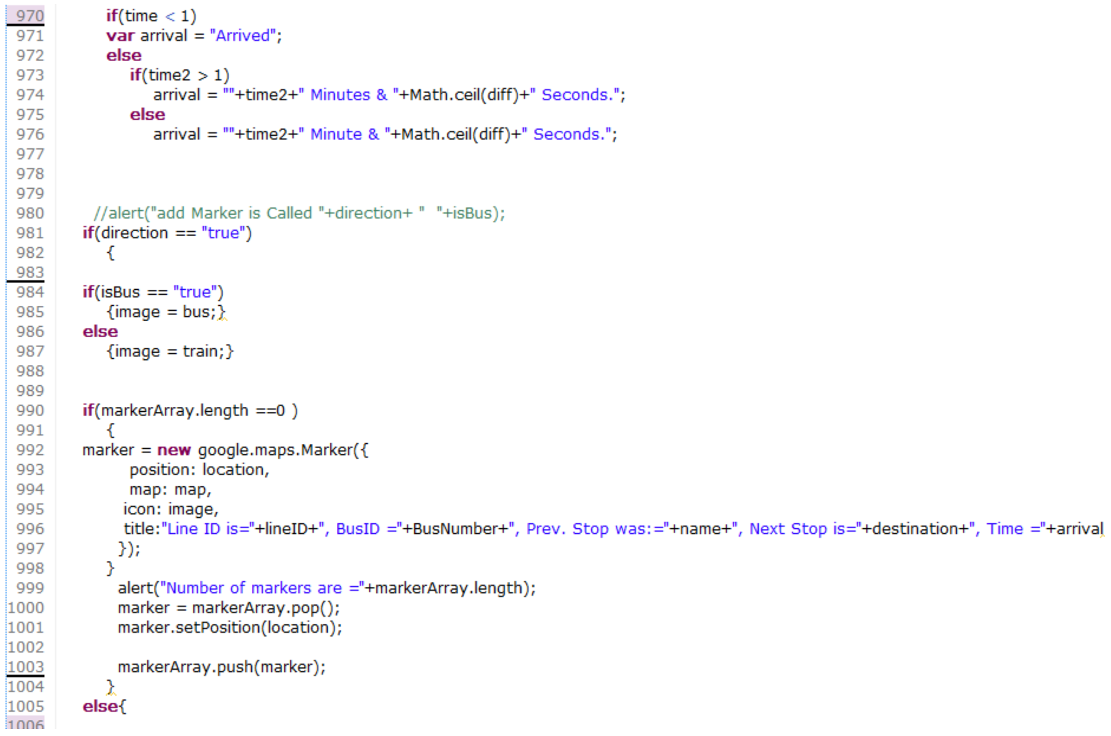
Then for the entire array of active buses, we retrieve the individual data items using the string split by the same token ($) we used and call an addMarker() method which is another method to add one marker at a time on the map.



```
busData="";

name=arrlst[0];
alert("Map my BUS name "+name);
destination=arrlst[1];
alert("Map my BUS desti "+destination);
lat=arrlst[2];
alert("Map my BUS lattitude "+lat);
lng=arrlst[3];
alert("Map my BUS  lng "+lng);
ttns=arrlst[4];
alert("Map my BUS  ttns "+ttns);

alert("Map My BUS direction "+arrlst[5]);

direction = arrlst[5];
//alert(direction);
lineID = arrlst[6];

 isBus = arrlst[7];
alert(" 9 th "+arrlst[8]+" parseInt(arrlst[9]) = "+parseInt(arrlst[9]));

tripPositions.push(parseInt(arrlst[8]));
positions.push(parseInt(arrlst[9]));
BusNumber = parseInt(arrlst[10]);
//if(delay > ttns )
//  {delay=ttns;}

//  json.Route11.Trip[i].Stops[j-1].name+" "+json.Route11.Trip[i].Stops[j-1].destination+" "+
//  json.Route11.Trip[i].Stops[j].cordinates[0]+" "+json.Route11.Trip[i].Stops[j].cordinates[1]+" "+json.Route11.Trip[i].Stops[j].ttns;

var bus = new google.maps.LatLng(lat,lng);

ID++;

addMarker(bus, name, destination, ttns, ID, direction, lineID, isBus, BusNumber);
```

Figure 6.2.29 Mapping all the Buses

Add marker method() then checks for a vehicle type, its direction and based on those conditions it makes a decision about icons and maps the particular vehicle on the map



```
        if(time < 1)
        var arrival = "Arrived";
        else
            if(time2 > 1)
                arrival = ""+time2+" Minutes & "+Math.ceil(diff)+" Seconds.";
            else
                arrival = ""+time2+" Minute & "+Math.ceil(diff)+" Seconds.";



      //alert("add Marker is Called "+direction+ "  "+isBus);
    if(direction == "true")
        {

    if(isBus == "true")
        {image = bus;}
    else
        {image = train;}


    if(markerArray.length ==0 )
        {
    marker = new google.maps.Marker({
            position: location,
            map: map,
           icon: image,
           title:"Line ID is="+lineID+", BusID ="+BusNumber+", Prev. Stop was:="+name+", Next Stop is="+destination+", Time ="+arrival
          });
        }
          alert("Number of markers are ="+markerArray.length);
          marker = markerArray.pop();
          marker.setPosition(location);

          markerArray.push(marker);
        }
    else{
```

Figure 6.2.30 Adding bus icons on the map

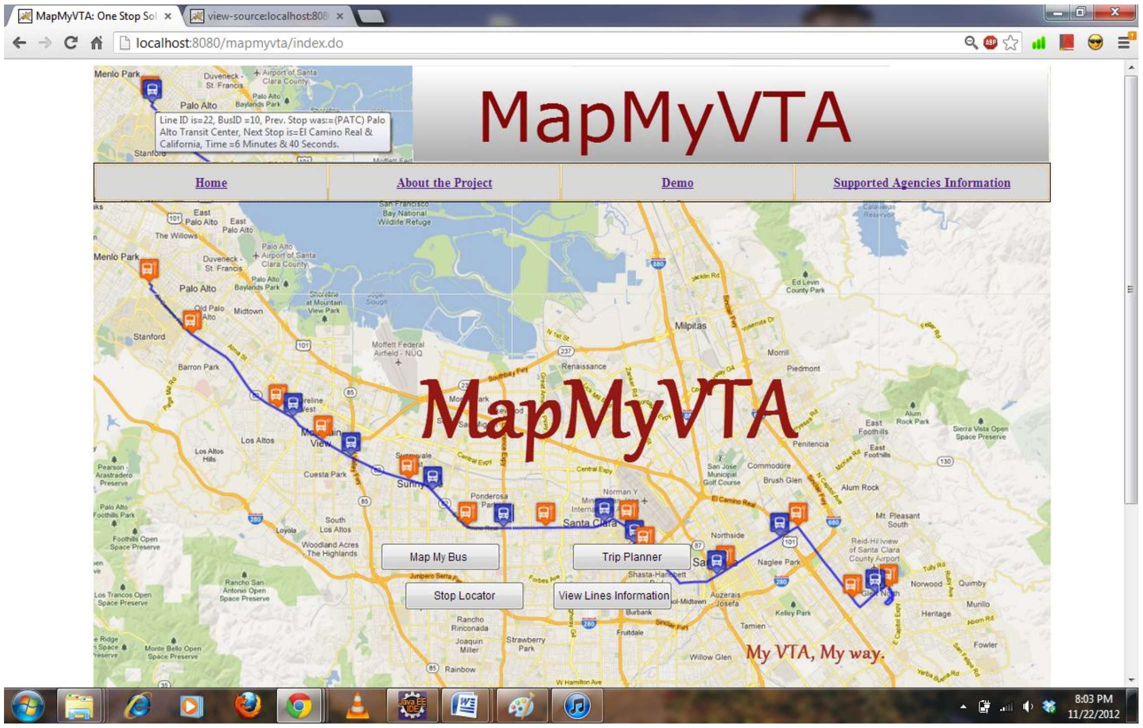
Finally, once after all markers are added to the map, MapMyBus calls showOverlay() method, which shows all the buses on the map by setting visible property of each marker to true. Then as MapMyBus() method has processed all active buses so it sets activeBuses array to empty and calls again the busPositions() method with a delay of some seconds.

As the 3rd feature, stop locator helps a VTA user to search for a particular stop and to view more information about the stop. User is presented with a list of drop down values to select a stop and to make a request. As a response MapMyVTA returns the lines information serving that particular stop. Front end scripting has been done using JavaScript and HTML tags along with a stop mapped on a map using Google Maps API v3 for better location identification. Also a drop down menu to search and locate the attractions around a particular bus stop using Google Maps location services is provided. Almost always VTA users want to reach a school, market, station, or a mall from a VTA stop hence this feature with attraction mapped on map and address visibility, makes it easy to search places around.

As the 4th feature view lines information presents user an option to view the complete information about all the lines supported by the system. On the first page it gives a snapshot of all the bus lines currently supported by the system with an additional navigational button to display more information about a particular bus line. Also at the bottom of more information page there is a **"Quick Query Dashboard"** which helps user to select bus lines, directions, and days of service and to see the line's information for the specified criteria instantly. The view is a JSP page with scripting using Ajax, JavaScript & HTML.

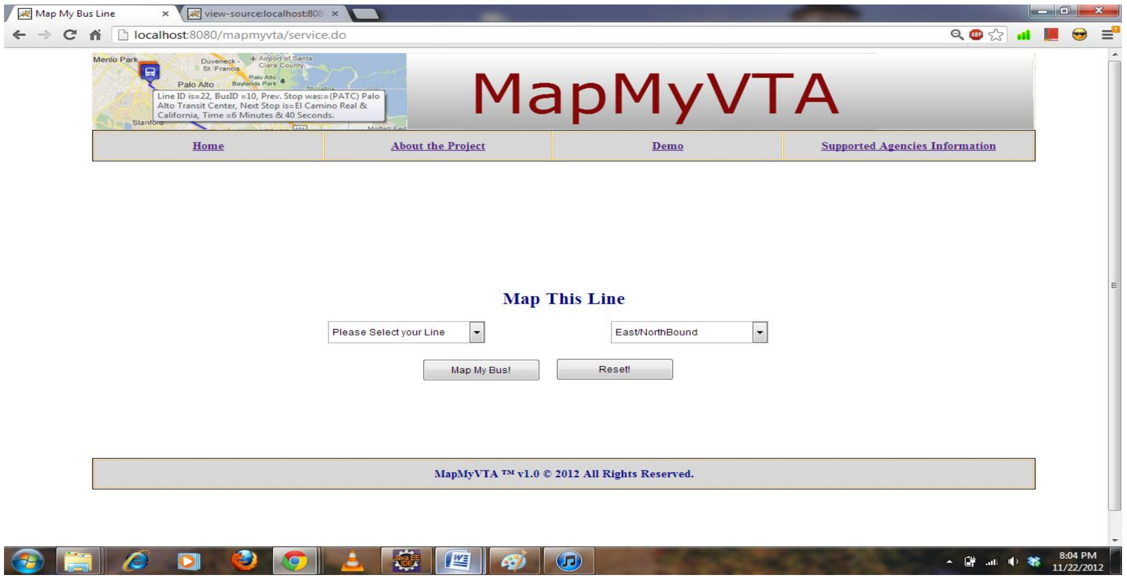
7 Web Application UI

7.1 Index Page



Line ID is=22, BusID =10, Prev. Stop was:=(PATC) Palo Alto Transit Center, Next Stop is=El Camino Real & California, Time =6 Minutes & 40 Seconds.

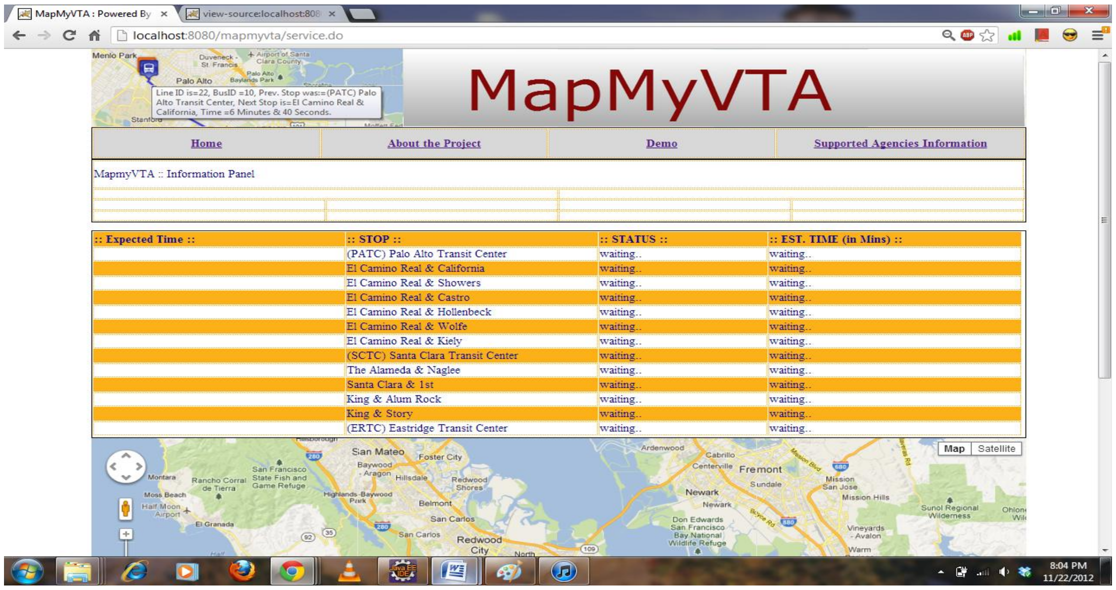
7.2 Map this Bus option



Line ID is=22, BusID =10, Prev. Stop was:=(PATC) Palo Alto Transit Center, Next Stop is=El Camino Real & California, Time =6 Minutes & 40 Seconds.

MapMyVTA ™ v1.0 © 2012 All Rights Reserved.

7.3 MapMyBus Initialized page



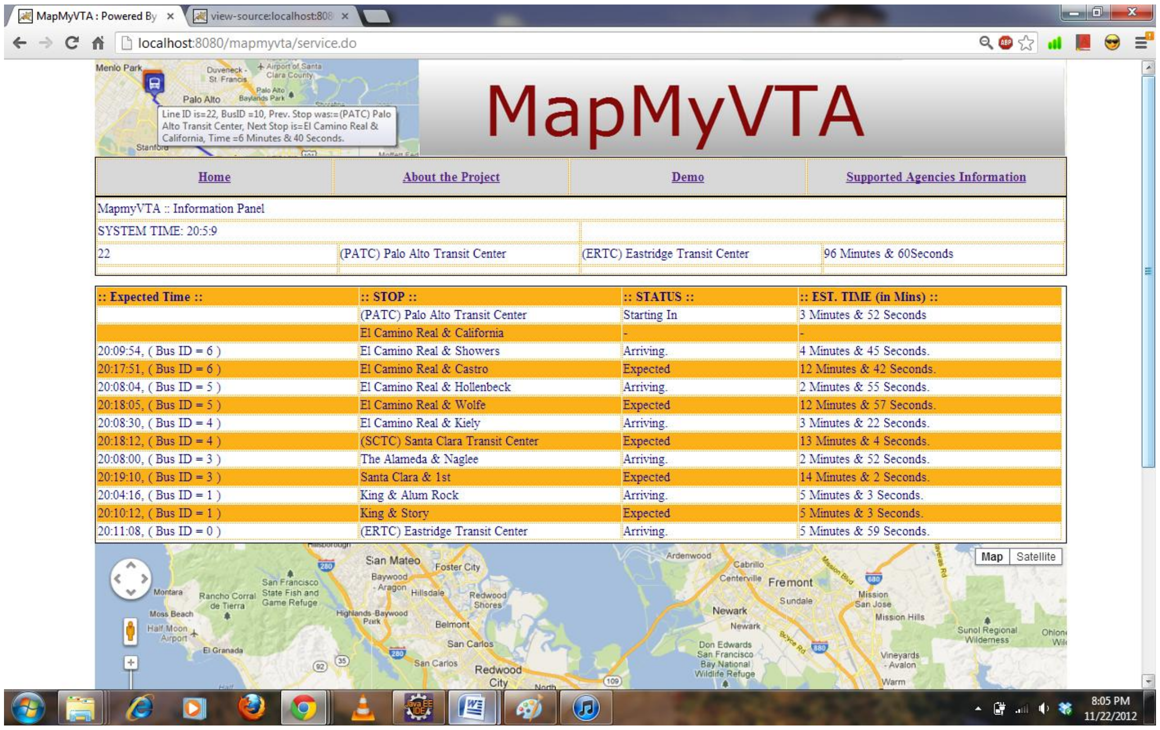
MapMyVTA : Powered By
view-source:localhost:808
localhost:8080/mapmyvta/service.do
Line ID is=22, BusID =10, Prev. Stop was:=(PATC) Palo Alto Transit Center, Next Stop is=El Camino Real & California, Time =6 Minutes & 40 Seconds.
MapMyVTA
Home
About the Project
Demo
Supported Agencies Information
Map
Satellite
8:04 PM
11/22/2012

MapmyVTA :: Information Panel

:: Expected Time ::	:: STOP ::	:: STATUS ::	:: EST. TIME (in Mins) ::
	(PATC) Palo Alto Transit Center	waiting..	waiting..
	El Camino Real & California	waiting..	waiting..
	El Camino Real & Showers	waiting..	waiting..
	El Camino Real & Castro	waiting..	waiting..
	El Camino Real & Hollenbeck	waiting..	waiting..
	El Camino Real & Wolfe	waiting..	waiting..
	El Camino Real & Kiely	waiting..	waiting..
	(SCTC) Santa Clara Transit Center	waiting..	waiting..
	The Alameda & Naglee	waiting..	waiting..
	Santa Clara & 1st	waiting..	waiting..
	King & Alum Rock	waiting..	waiting..
	King & Story	waiting..	waiting..
	(ERTC) Eastridge Transit Center	waiting..	waiting..

7.4 MapMyBus – after Initialization

-Dynamic Display Table



MapMyVTA : Powered By
view-source:localhost:808
localhost:8080/mapmyvta/service.do
Line ID is=22, BusID =10, Prev. Stop was:=(PATC) Palo Alto Transit Center, Next Stop is=El Camino Real & California, Time =6 Minutes & 40 Seconds.
MapMyVTA
Home
About the Project
Demo
Supported Agencies Information
Map
Satellite
8:05 PM
11/22/2012

MapmyVTA :: Information Panel

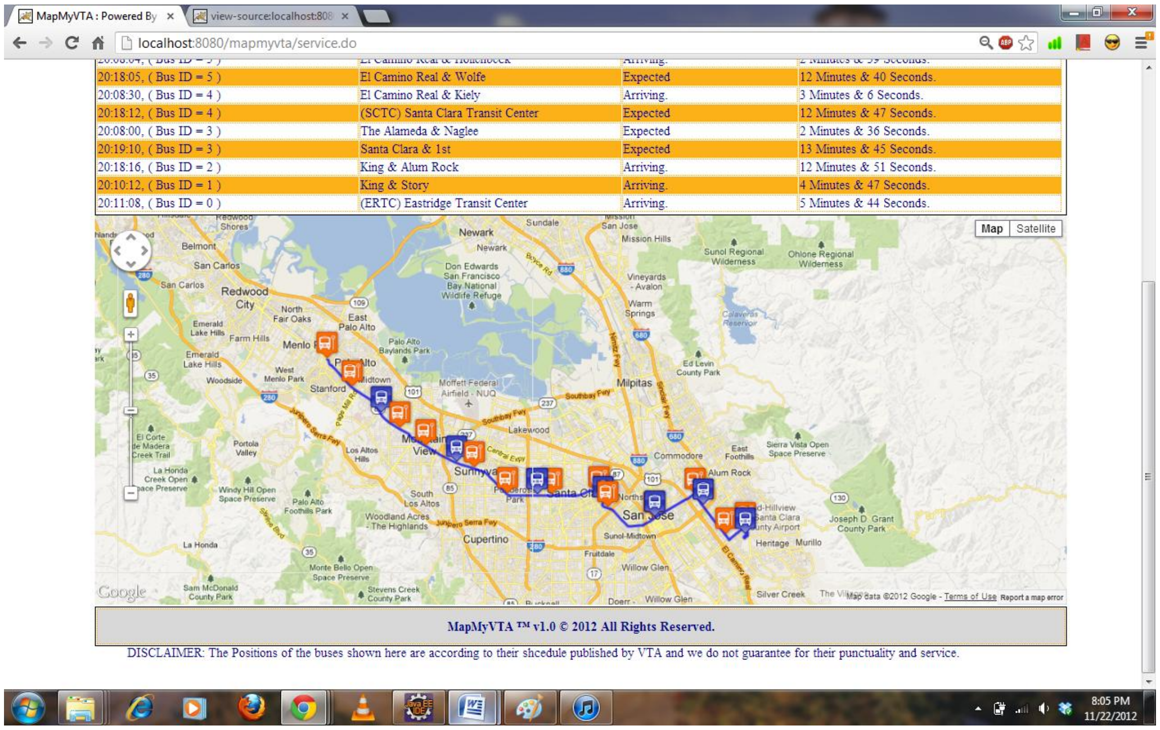
SYSTEM TIME: 20:5:9

22	(PATC) Palo Alto Transit Center	(ERTC) Eastridge Transit Center	96 Minutes & 60Seconds

:: Expected Time ::	:: STOP ::	:: STATUS ::	:: EST. TIME (in Mins) ::
	(PATC) Palo Alto Transit Center	Starting In	3 Minutes & 52 Seconds
	El Camino Real & California	-	-
20:09:54, (Bus ID = 6)	El Camino Real & Showers	Arriving.	4 Minutes & 45 Seconds.
20:17:51, (Bus ID = 6)	El Camino Real & Castro	Expected	12 Minutes & 42 Seconds.
20:08:04, (Bus ID = 5)	El Camino Real & Hollenbeck	Arriving.	2 Minutes & 55 Seconds.
20:18:05, (Bus ID = 5)	El Camino Real & Wolfe	Expected	12 Minutes & 57 Seconds.
20:08:30, (Bus ID = 4)	El Camino Real & Kiely	Arriving.	3 Minutes & 22 Seconds.
20:18:12, (Bus ID = 4)	(SCTC) Santa Clara Transit Center	Expected	13 Minutes & 4 Seconds.
20:08:00, (Bus ID = 3)	The Alameda & Naglee	Arriving.	2 Minutes & 52 Seconds.
20:19:10, (Bus ID = 3)	Santa Clara & 1st	Expected	14 Minutes & 2 Seconds.
20:04:16, (Bus ID = 1)	King & Alum Rock	Arriving.	5 Minutes & 3 Seconds.
20:10:12, (Bus ID = 1)	King & Story	Expected	5 Minutes & 3 Seconds.
20:11:08, (Bus ID = 0)	(ERTC) Eastridge Transit Center	Arriving.	5 Minutes & 59 Seconds.

7.5 MapMyBus – – after Intialization

-Bus Icons



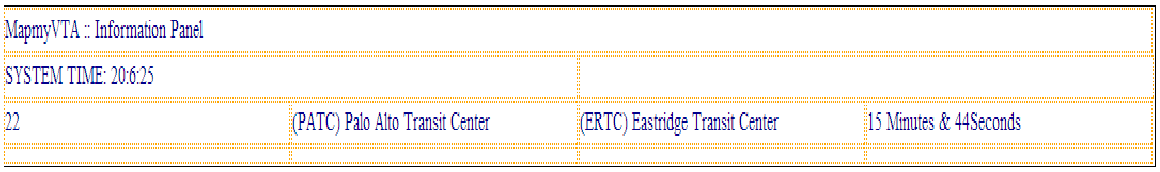
20:08:04, (Bus ID = 5)	El Camino Real & Hollenbeck	Arriving.	2 Minutes & 59 Seconds.
20:18:05, (Bus ID = 5)	El Camino Real & Wolfe	Expected	12 Minutes & 40 Seconds.
20:08:30, (Bus ID = 4)	El Camino Real & Kiely	Arriving.	3 Minutes & 6 Seconds.
20:18:12, (Bus ID = 4)	(SCTC) Santa Clara Transit Center	Expected	12 Minutes & 47 Seconds.
20:08:00, (Bus ID = 3)	The Alameda & Naglee	Expected	2 Minutes & 36 Seconds.
20:19:10, (Bus ID = 3)	Santa Clara & 1st	Expected	13 Minutes & 45 Seconds.
20:18:16, (Bus ID = 2)	King & Alum Rock	Arriving.	12 Minutes & 51 Seconds.
20:10:12, (Bus ID = 1)	King & Story	Arriving.	4 Minutes & 47 Seconds.
20:11:08, (Bus ID = 0)	(ERTC) Eastridge Transit Center	Arriving.	5 Minutes & 44 Seconds.

MapMyVTA ™ v1.0 © 2012 All Rights Reserved.

DISCLAIMER: The Positions of the buses shown here are according to their shcedule published by VTA and we do not guarantee for their punctuality and service.

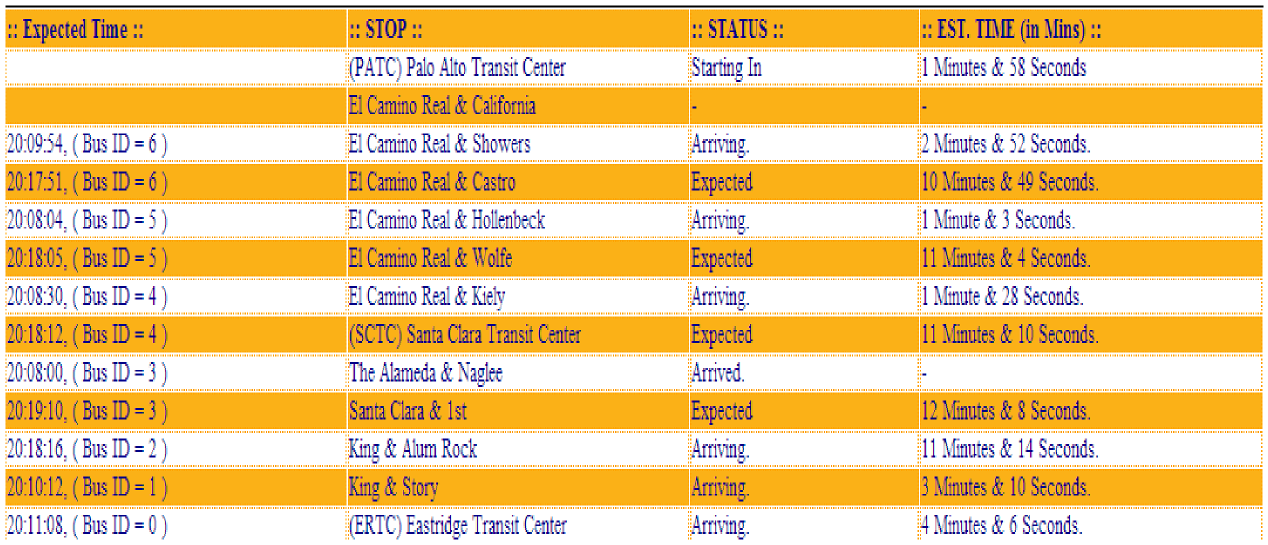
MapMyVTA : Powered By
view-source:localhost:808
localhost:8080/mapmyvta/service.do
Map
Satellite
Map data ©2012 Google - Terms of Use
Report a map error
8:05 PM
11/22/2012

7.6 MapMyVTA information panel(after system initialization)



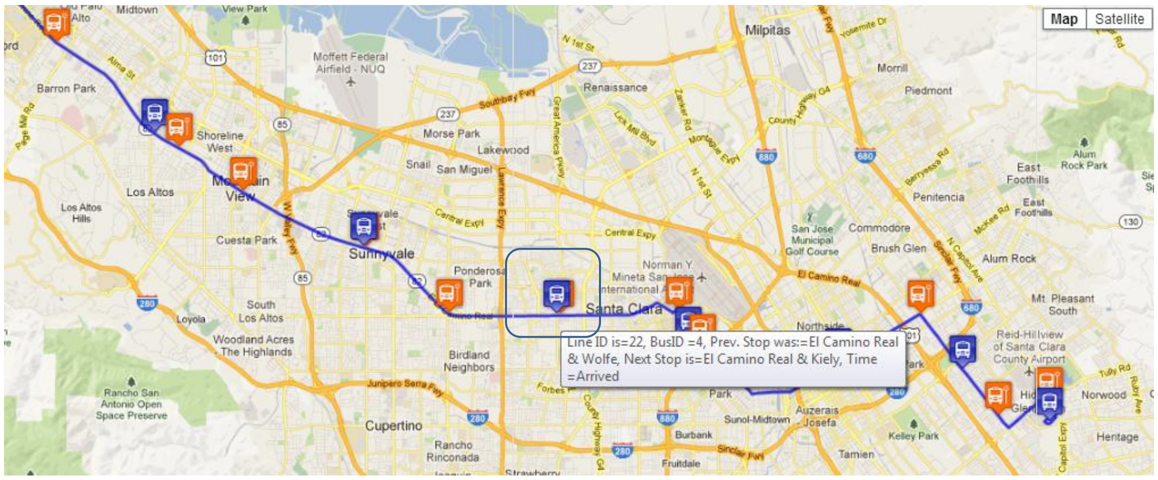
MapmyVTA :: Information Panel			
SYSTEM TIME: 20:6:25			
22	(PATC) Palo Alto Transit Center	(ERTC) Eastridge Transit Center	15 Minutes & 44Seconds

7.7 Dynamic Display timing display



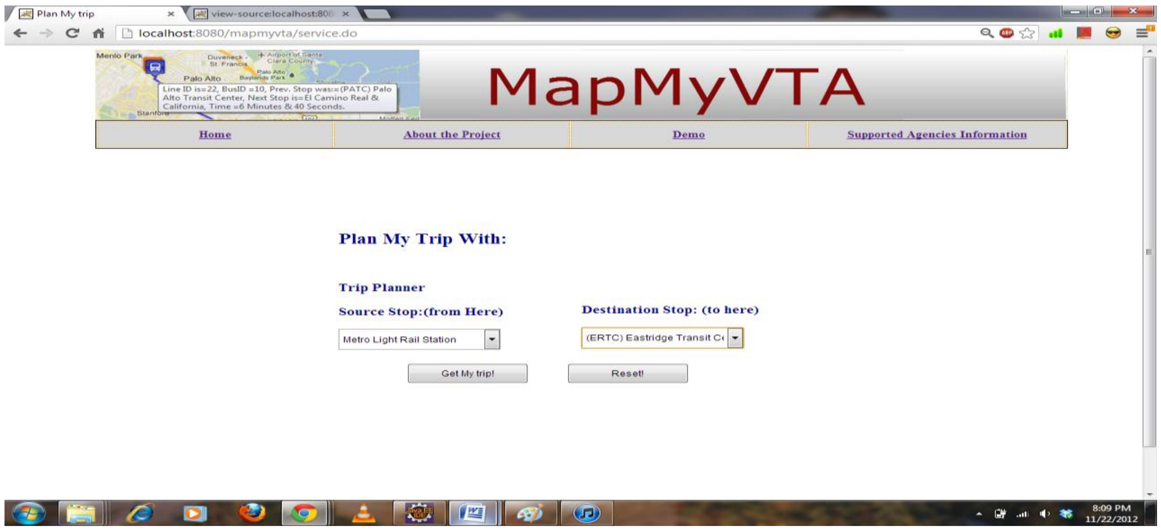
:: Expected Time ::	:: STOP ::	:: STATUS ::	:: EST. TIME (in Mins) ::
	(PATC) Palo Alto Transit Center	Starting In	1 Minutes & 58 Seconds
	El Camino Real & California	-	-
20:09:54, (Bus ID = 6)	El Camino Real & Showers	Arriving.	2 Minutes & 52 Seconds.
20:17:51, (Bus ID = 6)	El Camino Real & Castro	Expected	10 Minutes & 49 Seconds.
20:08:04, (Bus ID = 5)	El Camino Real & Hollenbeck	Arriving.	1 Minute & 3 Seconds.
20:18:05, (Bus ID = 5)	El Camino Real & Wolfe	Expected	11 Minutes & 4 Seconds.
20:08:30, (Bus ID = 4)	El Camino Real & Kiely	Arriving.	1 Minute & 28 Seconds.
20:18:12, (Bus ID = 4)	(SCTC) Santa Clara Transit Center	Expected	11 Minutes & 10 Seconds.
20:08:00, (Bus ID = 3)	The Alameda & Naglee	Arrived.	-
20:19:10, (Bus ID = 3)	Santa Clara & 1st	Expected	12 Minutes & 8 Seconds.
20:18:16, (Bus ID = 2)	King & Alum Rock	Arriving.	11 Minutes & 14 Seconds.
20:10:12, (Bus ID = 1)	King & Story	Arriving.	3 Minutes & 10 Seconds.
20:11:08, (Bus ID = 0)	(ERTC) Eastridge Transit Center	Arriving.	4 Minutes & 6 Seconds.

7.8 Bus position & time display on mouse hover



Line ID is=22, BusID =4, Prev. Stop was:=El Camino Real & Wolfe, Next Stop is=El Camino Real & Kiely, Time =Arrived

7.9 Trip planning from Metro Light Rail Station to East ridge transit Center


Plan My trip
view-source:localhost:808
localhost:8080/mapmyvta/service.do
MapMyVTA
Home
About the Project
Demo
Supported Agencies Information
Get My trip!
Reset!
8:09 PM
11/22/2012

Line ID is=22, BusID =10, Prev. Stop was:=(PATC) Palo Alto Transit Center, Next Stop is=El Camino Real & California, Time =6 Minutes & 40 Seconds.

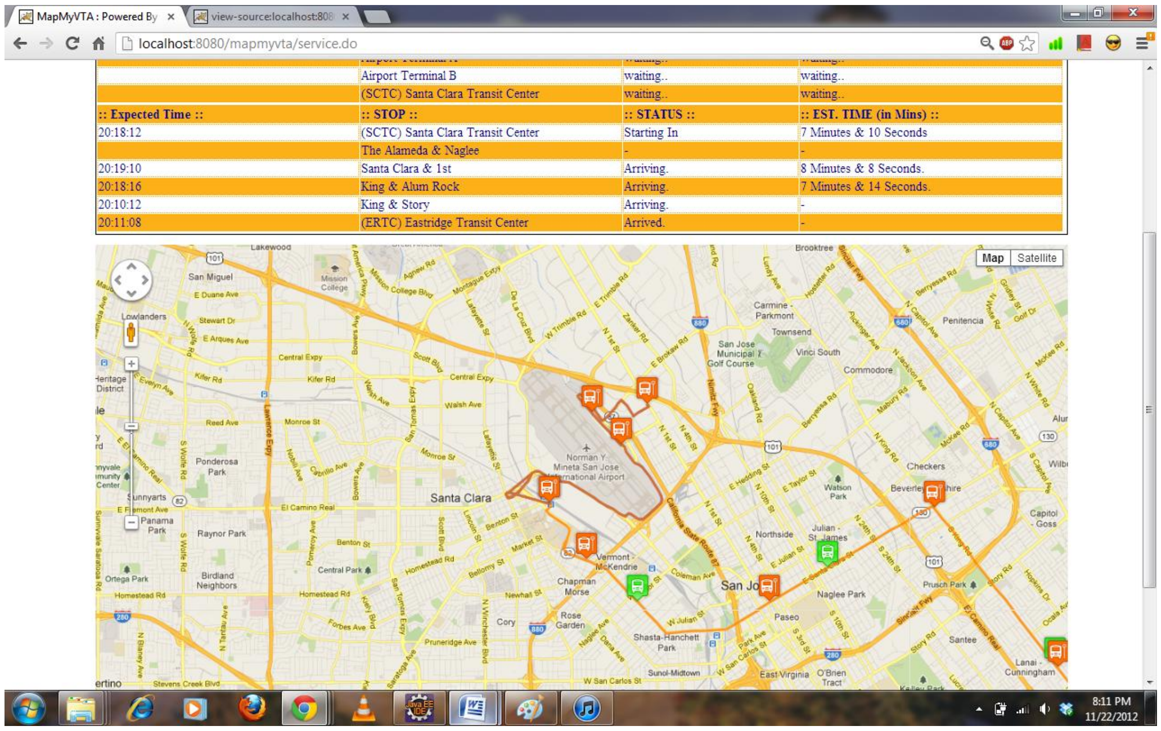
Plan My Trip With:

Trip Planner

Source Stop:(from Here) Metro Light Rail Station

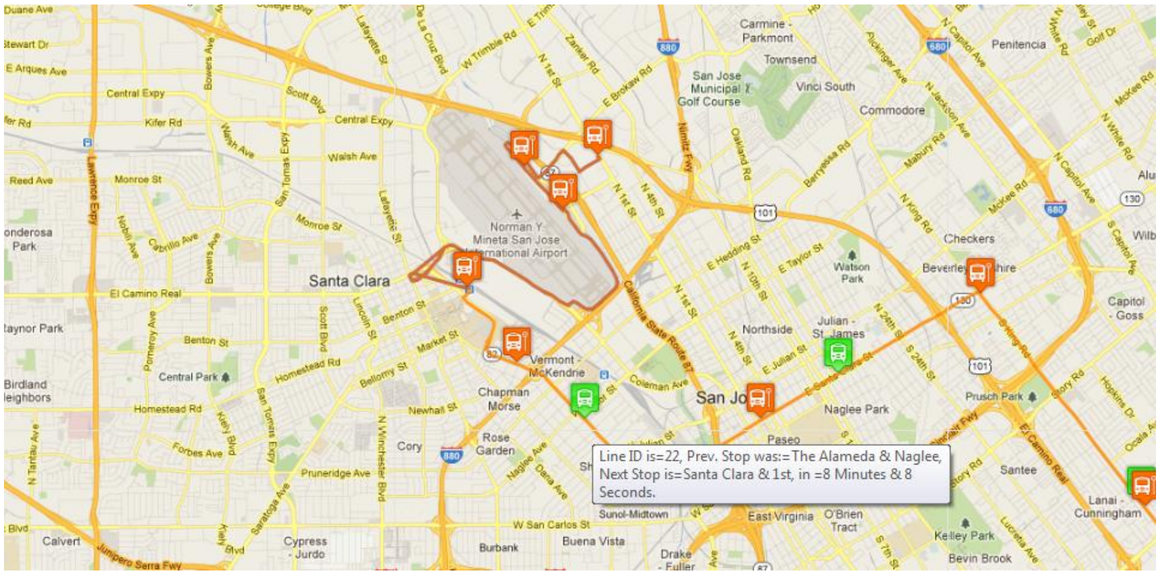
Destination Stop: (to here) (ERTC) Eastridge Transit Ce

7.10 Initial map



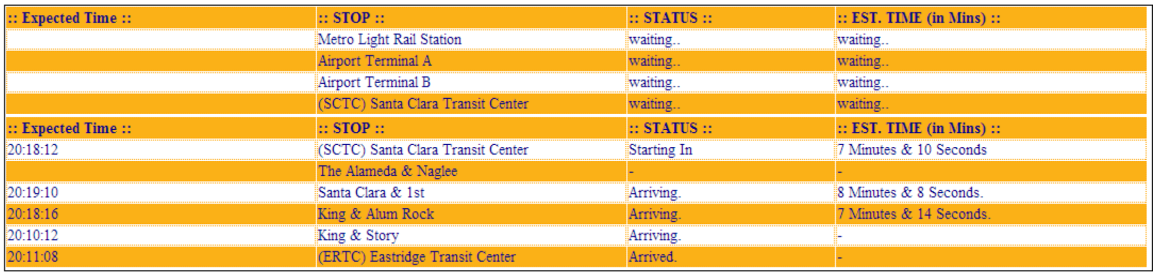
	Airport Terminal B	waiting..	waiting..
	(SCTC) Santa Clara Transit Center	waiting..	waiting..
:: Expected Time ::	**:: STOP ::**	**:: STATUS ::**	**:: EST. TIME (in Mins) ::**
20:18:12	(SCTC) Santa Clara Transit Center	Starting In	7 Minutes & 10 Seconds
	The Alameda & Naglee	-	-
20:19:10	Santa Clara & 1st	Arriving.	8 Minutes & 8 Seconds.
20:18:16	King & Alum Rock	Arriving.	7 Minutes & 14 Seconds.
20:10:12	King & Story	Arriving.	-
20:11:08	(ERTC) Eastridge Transit Center	Arrived.	-

7.11 Active map



Line ID is=22, Prev. Stop was:=The Alameda & Naglee,
Next Stop is=Santa Clara & 1st, in =8 Minutes & 8
Seconds.

7.12 interactive timing display of stops:



:: Expected Time ::	:: STOP ::	:: STATUS ::	:: EST. TIME (in Mins) ::
	Metro Light Rail Station	waiting..	waiting..
	Airport Terminal A	waiting..	waiting..
	Airport Terminal B	waiting..	waiting..
	(SCTC) Santa Clara Transit Center	waiting..	waiting..
:: Expected Time ::	:: STOP ::	:: STATUS ::	:: EST. TIME (in Mins) ::
20:18:12	(SCTC) Santa Clara Transit Center	Starting In	7 Minutes & 10 Seconds
	The Alameda & Naglee	-	-
20:19:10	Santa Clara & 1st	Arriving.	8 Minutes & 8 Seconds.
20:18:16	King & Alum Rock	Arriving.	7 Minutes & 14 Seconds.
20:10:12	King & Story	Arriving.	-
20:11:08	(ERTC) Eastridge Transit Center	Arrived.	-

8. Conclusion

VTA provides multiple options of transit lines with frequent services towards the important destinations. Even though VTA provides booklets of schedule and connections, it is still a tiresome work to search and / or plan a trip using this static information that needs to be repeated for every journey. The internet is available and free, at most of the places. VTA itself provides free internet wireless access to its users on its many of the transit lines. Almost all handheld devices now support internet browsing using these wireless connections. Therefore, the infrastructure for a better IT based option to provide service information for public transportation is already available. Report on a study done by Transportation Research Board, Washington DC [7], has several encouraging facts including this, about 38% of current non users will opt for public transportation if better information is presented about the transit services. Therefore, there is a huge scope of improvements for a better information representation. Even then, there are very few dedicated applications are available for public transport. In this case for VTA, there is none with such details that provides information about the services of the VTA in such a detailed and user-friendly manner like MapMyVTA.   All these features not only help the public transport users, VTA users in this case, to use and utilize the services of the VTA in an efficient manner but also help VTA to serve its users to its full potential. As the knowledge gap between the user and that of VTA services gets drastically reduced because of this project. MapMyVTA has the potential to be an asset to the VTA users and to the VTA, both.

REFERENCES

1. Cherry, C., Hickman, M., & Garg, A. (2006). Design of a Map-Based Transit Itinerary Planner. *Journal of Public Transportation, 9*(2). **Retrieved from** http://nctr.usf.edu/jpt/pdf/JPT%209-2%20Cherry.pdf

2. Google Maps Javascript API V3 Basics - Google Maps JavaScript API V3 - Google Code. (n.d.). *Google Code*. **Retrieved from** http://code.google.com/apis/maps/documentation/javascript/basics.html

3. Guo, Z. (2010) *Mind the Map! The Impact of Transit Maps on Travel Decisions in Public Transit*. Graduate. New York University. **Retrieved from** http://wagner.nyu.edu/faculty/publications/files/Mind_the_Map_Guo_Zhan_2010.pdf

4. Li, J., Zhou, K., & Zhang, W. (n.d.). *A Multimodal Trip Planning System Incorporating the Park-and-Ride Mode and Real-time Traffic/Transit Information*. **Retrieved from** http://www.networkedtraveler.org/tripplanner.pdf

5. Model view controller - Wikipedia, the free encyclopedia. (n.d.). *Wikipedia, the free encyclopedia*. **Retrieved from** http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

6. The Java Tutorials. (n.d.). *Oracle Documentation*. **Retrieved from** http://docs.oracle.com/javase/tutorial/

7. Transportation Research Board 500 Fifth Street, NW Washington, DC 20001 USA (1996). *INVESTIGATING EFFECT OF ADVANCED TRAVELER INFORMATION ON COMMUTER TENDENCY TO USE TRANSIT*. **Retrieved**

from http://trid.trb.org/view.aspx?id=471018

8. VTA Newsroom: VTA Media Relations Frequently Asked Questions. (n.d.). *Santa Clara Valley Transportation Authority*. **Retrieved from** http://www.vta.org/news/media_relations_faq.html