San Luis Obispo Transit Tracker System Back-End Server

Daryl Alan A Dimalanta
Advisor: Christopher Lupo
Senior Project | Spring 2011
Computer Engineering Program, Department of Electrical Engineering
California Polytechnic State University
San Luis Obispo, CA 93407
ddimalan@calpoly.edu
clupo@calpoly.edu

***Abstract*-- This document gives the rationale, design process, technical implementation, testing procedures, and testing results of a back-end server used for the San Luis Obispo (SLO) Transit Tracker System. The SLO Transit Tracker System (SLOTTS) includes client software installed on mobile devices and a back-end server. The purpose of this system is to encourage SLO residents to utilize the transit system by having the client software present current bus location, navigation instructions, and bus schedule in a simple, fast, and easy to use mobile application.**

Due to the limited central processing unit (CPU) and limited power supplied by the battery, route navigation calculation and database queries done on the client device is slow and quickly drains the available power of the mobile device.

Alternatively, route navigation calculation and database query requests done through hypertext transfer protocol (HTTP) to a server is much quicker to do with minimal computational or power costs to the mobile device.

Additionally, a back-end server allows for a central point for database changes. Maintainers of SLOTTS can easily change schedule entries on the server database. With each client device periodically syncing its own database with the server database, manually providing client software updates with an updated database hard-programmed into the software is no longer needed.

The SLOTTS back-end server aims to solve these problems and increase the effectiveness of the overall system.

I. INTRODUCTION

This report, made in partial fulfillment of the requirements for the Degree in Bachelor of Science, outlines my project in which a server was designed and implemented for SLOTTS. The project began in winter of 2011 with research and planning, and ended in spring of 2011 with the server physically built and configured.

The report will cover the problem faced by the San Luis Obispo Transit System and the residents of San Luis Obispo which is the basis for the creation of SLOTTS.

The limitation of mobile devices will be discussed. This limitation is the rationale behind splitting SLOTTS into two parts: front-end client side and back-end server side.

The design process in planning the implementation of the back-end server and technologies researched will be explained.

The steps taken to create the physical server and to configure the server will be listed, along with any problems encountered. For explaining the steps conducted to configure the server, the software logic will be illustrated.

Lastly, testing methodology will be explained and followed with suggestions for improvements.

II. INDIVIDUALS AND PARTIES INVOLVED

This project involved the participation of several individuals and parties. Other than myself, participants include my senior project advisor, the City of San Luis Obispo, and Digital Recorders Incorporated (DRI), which is a third party company contracted by the City of San Luis Obispo to track each running bus by GPS and maintain a city server which contains this information.

A. Developers

- Daryl Dimalanta – Back-End Server
- Jeff Brown – Front-End Client Application
- Zach Negrey – Front-End Client Application

B. Senior Project Advisor

- Christopher Lupo, PhD

C. City of San Luis Obispo

- John Webster – San Luis Obispo Transportation Manager

D. Digital Recorders Incorporated

- Dan Trujillo – Sales Manager, Western Region

III. BACKGROUND

The City of San Luis Obispo Transit System is decreasing despite the city of San Luis Obispo's population increasing [1]. The decreased number of riders can be attributed to many factors, however, the lack of awareness and perceived inconvenience of the bus system are likely contributors to this decrease.

To increase ridership by decreasing the inconvenience of using the transit system, the California Department of Transportation along with faculty from California Polytechnic State University developed the Efficient Deployment of Advanced Public Transportation System (EDAPTS) Intelligent Transportation System (ITS). The EDAPTS ITS system uses global positioning system (GPS) devices on each individual bus to determine how long each bus will arrive at a stop fitted with real-time arrival signs. By informing the user when a bus is to arrive, this improved the rider's confidence in using the transit system. EDAPTS ITS was installed and deployed to the city of San Luis Obispo in early 2001. [2]

In June of 2009, the city of San Luis Obispo decommissioned EDAPTS ITS. To replace EDAPTS ITS, the city of San Luis Obispo uses a third party company, Digital Recorders Incorporated (DRI), to track GPS location of each bus and to display real-time location of each bus on the city website [3]. The DRI system which the city of San Luis Obispo uses is called On-Time Vehicle Information Access (OTvia2). This system is not incorporated with the EDAPTS arrival signs placed throughout the city and therefore the EDAPTS arrival signs rely on predicting bus arrival times strictly by bus schedule.

The wide spread use of smartphones allows riders to access the city website and view the OTvia2 web-application which displays bus information. However, the OTvia2 web-application is optimized to be used on personal computers with a Firefox browser [4]. As a result, usage on a smartphone web browser is not always ideal and may be difficult for certain users based on the capabilities and screen size of their internet capable smartphones.

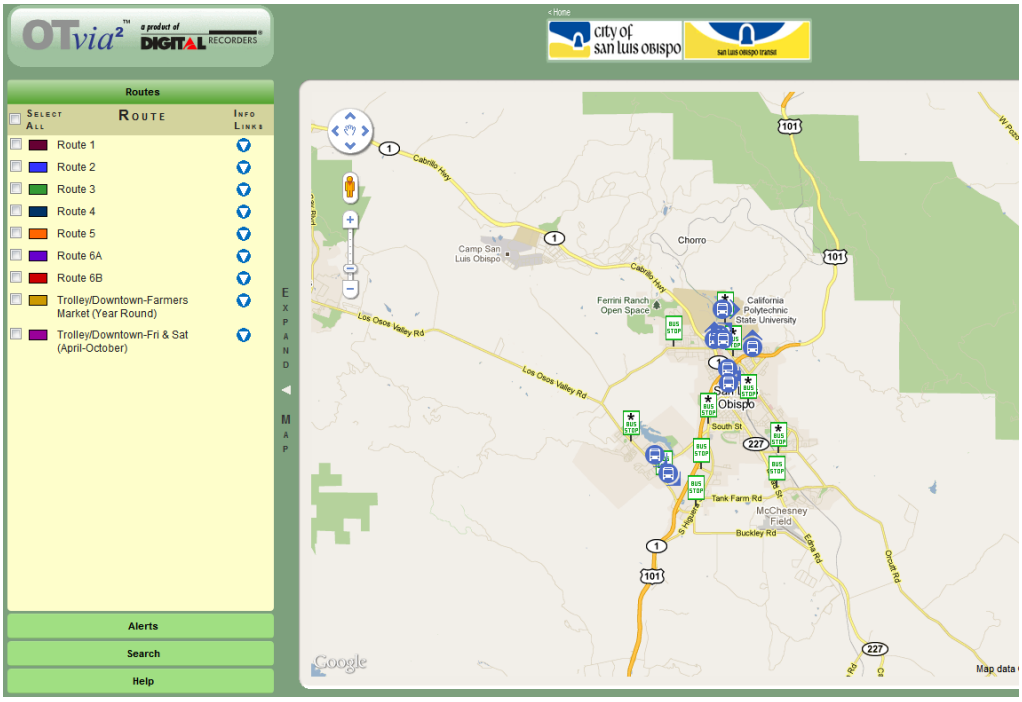


Fig. 1 The San Luis Obispo transit website implementation of the OTvia2 system by DRI

With as much as 50% of smartphone users preferring to use natively installed applications on their mobile devices [5], Jeff Brown, Zach Negrey, and I developed a system which displays live bus locations in a format that is suitable for the screen size of a smartphone. The native application was made for android based devices and also iOS devices.

Unfortunately, with the limited capabilities of smartphone devices, the native application is limited in its functionality. Though it displays the current location of each bus route perfectly on a map, it does not allow users to view bus route schedules in an intuitive and easy to read manner, nor does it give users the ability to get navigational instructions when utilizing the transit system. Lastly, any changes to the bus schedule require each application for all platforms to be manually updated as the route information is hard coded into the application itself.

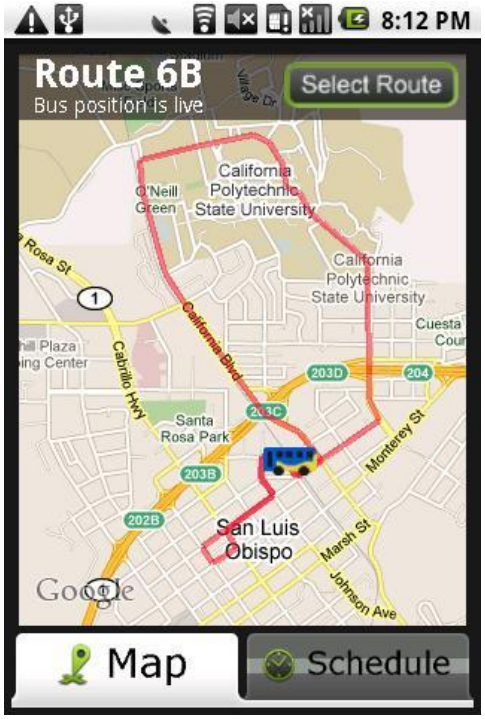


Fig. 2 SLO Bus Tracker, native Android application by Jeff Brown and Zach Negrey, displaying active location of Route 6B bus. To be implemented with the back-end server as part of SLOTTS

With the technological advances of smartphone wireless capabilities, the cost and time of sending data requests to an off-client server plus the cost and time of receiving results back from the server is small compared to the client devices doing computationally complex calculations with its limited CPU capabilities and power availabilities [5].

IV. PROBLEM STATEMENT

SLOTTS currently exists as a native application. The application is hard-coded with the bus schedule and requires manual updates when bus schedule changes. With limited capabilities of the smartphone hardware, aggregation of bus schedule information is limited, and the current application has no route navigation information.

An off-device server needs to be implemented to allow the above functionality to be added to SLOTTS. The native application will serve as the front-end client to SLOTTS while the off-device server will be the back-end to SLOTTS. The front-end client will send CPU / Memory intensive requests to the back-end server. The back-end server will respond back to the front-end client with the results.

In Figure 3, the current SLO Bus Tracker native android application shows the bus schedule in a simple long list without ability to search specific stop, route, or time information.

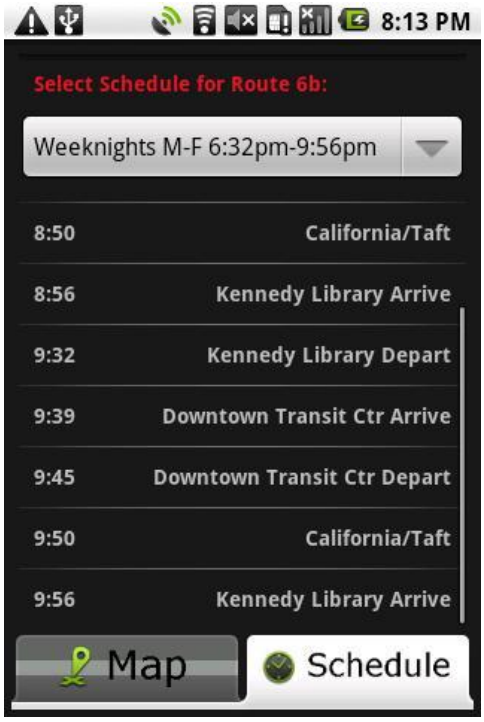


Fig. 3 SLO Bus Tracker, native Android application by Jeff Brown and Zach Negrey, displaying list of arrival times for route 6B. SLOTTS aims to improve the application by allowing for better aggregation and search of route, stop, time, and transfer information

V. ENGINEERING REQUIREMENTS

Below are the engineering requirements for the back-end server to be implemented as part of SLOTTS.

- Thorough database which can fully represent all information and data regarding the bus transit system
- Navigation guidance using the transit system based on client device location and client destination
- Single location where database entries can be updated easily with a Graphic User Interface
- Interface between the server and client devices where client can make database or navigation requests to the server.
- Server maintains its own database with updated bus location from the city server.
- If bus locations are invalid or the server is unable to contact the city server, the server is to predict where bus location is based on last known location and bus schedule.

VI. DESIGN PROCESS

The design process followed when developing the server are outlined Figure 4.

First, the problem with SLOTTS was identified. Upon identifying the problem, I brainstormed solutions which could be used to solve the problem. With multiple ideas, I finalized the engineering requirements of the solution. Next, I conducted research on multiple types of software technologies that could be used to run the server.

Most of the technologies researched were software which I had never used or learned prior. As such, most of the time spent on this project was learning how to use these technologies. These technologies included a web server from Apache, relational database using MySQL, and server-side web development from PHP. Additionally, research had to be performed on determining how the server would be able to communicate with the city server.

While research was being performed, the hardware was being designed and built. Soon after the server was physically built, the software researched, installed, and configured on the server.

Once the server was physically built and configured, software development began. The high level software architecture was first designed. From this design, the software code was written. Throughout development, collaboration with the front-end developers was maintained to ensure that the server would be compatible with the client-side application.

The last phase before implementation was testing. The server was tested first by doing queries to the interface using a computer web browser. This was to be followed by a modification of the client-side software to implement the server. However, due to time constraints, testing through client-side software was not reached and is planned to be conducted in the summer of 2011.

Following testing, the client-side software is to be redesigned to fully implement the server. This is planned to be done after the client-side testing is complete. Upon, final testing of the redesigned client-side application, the software will be released to the general public.

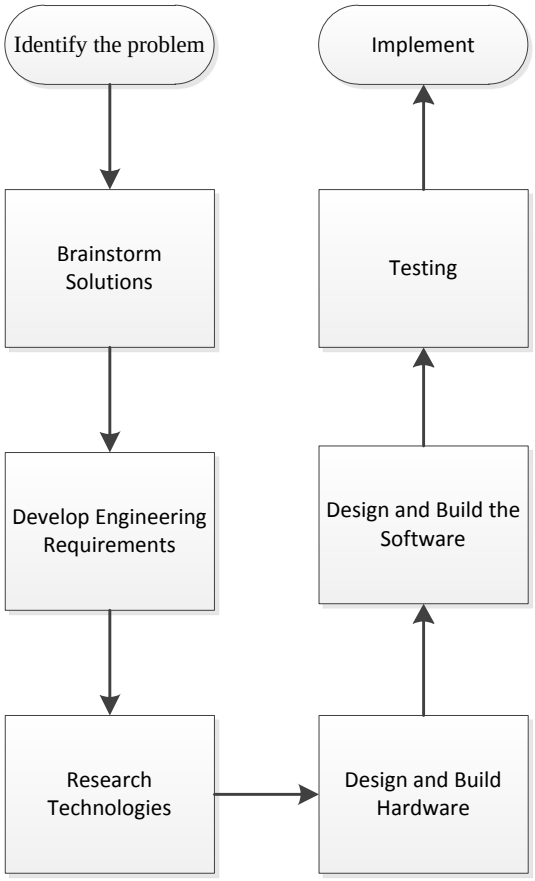


Fig. 4 Design Process Diagram

VII. WORK PLAN

The system from design to being built and programmed spanned 2 quarters or approximately 12 weeks. The original work schedule was split up into 5 primary development tasks. The following are the primary development tasks included:

Server Environment (ENV)

- Have server functional with operating system and software ready
    - Build Server Hardware (*ENV – 1*)
    - Install Server Operating System (*ENV – 2*)
    - Install LAMP Configuration Software (*ENV – 3*)
- Ensure Network is configured for server
    - Configure Router (*ENV – 4*)
    - Configure server networking and Apache (*ENV – 5*)

Database (DTB)

- Configure MySQL Database
    - Create the tables (*DTB – 1*)
    - Input transit system information into database (*DTB – 2*)

Java Back-End (JBE)

- Ensure java background processes can access and modify database (*JBE – 1*)
- Create java process which can modify database fields (*JBE – 2*)
- Create java process which can predict where bus is based on database information (*JBE – 3*)
- Update bus prediction process to check DRI servers for bus location (*JBE – 4*)

PHP Interface (PHP)

- Ensure php scripts are able to query the database (*PHP – 1*)
- Create php script which can send over individual route information (*PHP – 2*)
- Create php script which can be used by client in updating client data of routes (*PHP – 3*)
- Create php script to accept start point and end point location and output recommended route (*PHP – 4*)

Client App (APP)

- Help in interfacing Android app with server (CLA – 1)
- Help in adding route suggestion feature into Android app (CLA – 2).

Schedule of Work

The schedule of work incorporated the primary development tasks. Additionally, a testing period and time allotted to the creation of this document were added. The Gantt chart in Appendix A illustrates the schedule of work planned for this project.

VIII. SYSTEM OVERVIEW

The system consists of two primary elements, the front-end client application installed on individual smartphone devices and the back-end server. Figure 5 illustrates how these two elements are implemented together.

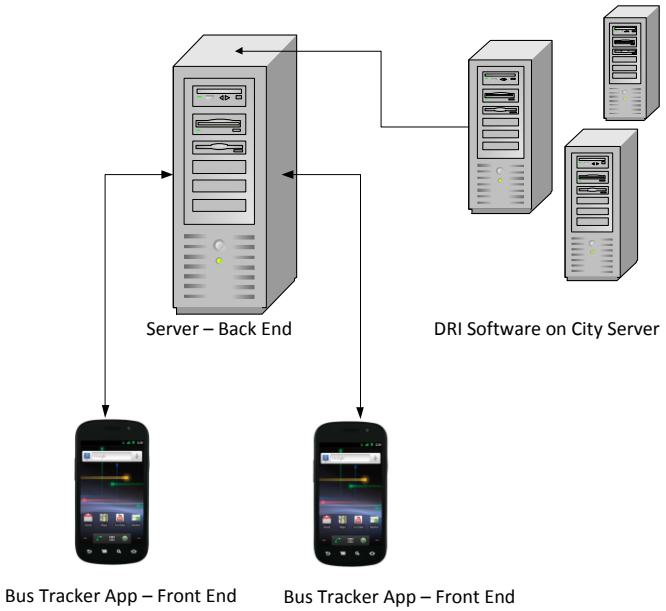


Fig. 5 High-Level System Overview of SLOTTS

IX. SERVER OVERVIEW

The server consists of the software and physical hardware. Additional components include connection to the front-end client and city server through the router and internet cloud. The software consists of the operating system and software technologies researched. Figure 6 illustrates in detail the server components and direction of communication between

the internal server and additional components. The server is setup in a Linux, Apache, MySQL, PHP configuration, also known as LAMP.

All of the software the server uses contains some form of open-source license. As such, the primary cost in developing and running the server is found in the purchase of hardware, internet connection, and power consumption.

The database to represent the transit system will be provided by MySQL. Communication between the server and the client devices will be provided by PHP engine going through the Apache Web Server. Java, with its built in capability to do http requests directly communicates with the city server. Both PHP and the Java portions use the same MySQL database.

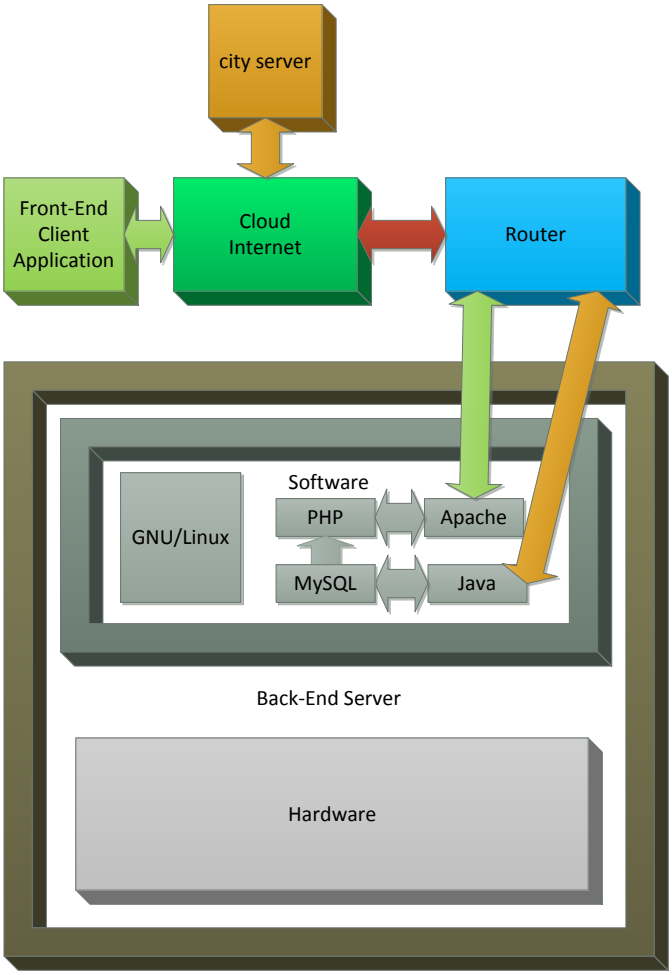


Fig. 6 Server Components Overview

X. SERVER HARDWARE

The server was hand built with custom hardware. Initially an old custom personal computer, built in the early 2000s, was used as the server. This server was obtained with no cost as it was donated. However, with limited CPU speed, hard drive space, and insufficient memory, the server performed poorly.

Several weeks after the original server was configured for SLOTTS, the server motherboard experienced hardware malfunction. With the first server rendered inoperable, I proceeded to build a new server for SLOTTS.

The second server was built with its mission in mind. The server would need a CPU capable of servicing multiple clients at a time. The second server built is suitable for the tasks it is to perform; however, it still suffers from low hard disk speed. With a low database size and low number of clients for initial testing, the negative effects of low hard disk speed is negligible.

A. Initial Server Hardware

The initial hardware consisted of the following:

- Intel Pentium IV dual core processor running at 1 gigahertz
- 1 gigabyte of ram, speed unknown
- 10 gigabyte hard disk, 5400 rpm

B. Final Server Hardware and Bill of Materials

TABLE 1
BILL of MATERIALS OF SERVER

Part Type	Make and Model	Cost
Motherboard	MSI 870A-G54	$99.00
CPU	AMD Athlon II X4 3.0 GHz	$99.00
Random Access Memory	Corsair XMS3 4GB DDR3 1600	$53.99
Hard Drive	Western Digital Caviar Green 1TB WD10EARS 5400RPM	$57.99
Total		**$309.98**

Additional components used with server and obtained with no costs include:

- Case
- Monitor
- Keyboard
- Linksys WRT160 Router

C. Internet Service Provider

The server is connected to the internet cloud using cable internet connection service provided by Charter

Communications. The service provides 8 megabits / second downlink and 2 megabits / second uplink speeds. This service is provided at a cost of $39.99 / month.

D. Future Hardware Plans

The current server setup is sufficient for a small amount of clients, around 100 or fewer clients. The biggest bottleneck is the internet connection. An uplink speed of 2MB/s can severely decrease effectiveness of using the server.

Future plans are to utilize cloud computing services such as those provided by the Amazon Elastic Compute Cloud (EC2). Amazon EC2 takes away the need to manage the hardware and internet connectivity of the server. With Amazon's large datacenters, bandwidth congestion as more clients connect to the server is negligible. However, this may be a more costly route as the number of clients increase as the service is based on the number of connected clients and usage time by clients per hour [6].

XI. SERVER CONFIGURATION AND SOFTWARE

The following describes the operating system and primary software used in the server. Steps to install and configure each component to work with the server are also described.

A. GNU / Linux

The operating system (OS) used is Ubuntu Server version 10.10, 32-bit. Ubuntu Server is GNU / Linux type OS based on the Debian distribution produced and maintained by Canonical Ltd. [7]. A GNU / Linux type OS was selected due to my familiarity with that type of OS and its known stability, which is essential if the front-end client will be reliant on it. Additionally, Ubuntu is an open source OS and therefore would result in no additional cost to the project. Ubuntu comes standard with software tools such as apt-get which makes it easy to maintain the OS. Maintaining this distribution is also made easier by being supported by a large online community. Due to its open-source license, support, stability, and proven reliability, Ubuntu is an ideal OS compared to alternative OS such as Windows Server.

Steps to Setup

- Obtain boot CD from Ubuntu website: http://www.ubuntu.com/download/ubuntu/download
- Used Ubuntu installer to install OS and format hard disk

B. Apache HTTP Server

The Apache HTTP Server is a freely-available source code which implements an HTTP web server. This web server allows the server to accept HTTP requests to php files on the server from client devices. Apache works with the PHP processor module, and transmits the generated contents of the file through HTTP to the client [8][9].

Steps to Setup

- Install Apache HTTP Server

```
$ sudo apt-get update
$ sudo apt-get install apache2
```

- Test the web server by going to
  http://<server IP address>
  Verify default welcome page

C. MySQL

MySQL is a relational database management system (RDBMS) that is freely distributed by Oracle. As MySQL is an RDBMS, it is an ideal database to be used to represent a bus transit system. Additionally, MySQL is well supported and proven to be stable and secure [10]. When using the InnoDB engine supplied by MySQL, the use of foreign keys and primary keys allow effective representation of a bus, stop, bus route, and time schedule. With the proven speed of MySQL, the server will be able to do complex searches on the database and give the front-end client the bus schedule data aggregation complexity it needs with little cost.

Steps to Setup

- Install MySQL

```
$ sudo apt-get update
$ sudo apt-get install mysql-sql
```

- Setup root username/password

```
$ mysql –u root
mysql> SET PASSWORD FOR 'root'@'localhost
       = PASSWORD('mypassword*');
```

D. PHP

PHP or PHP: Hypertext Preprocessor is a server-side scripting language. PHP supports several major databases including MySQL, the ability to parse text, the output of HTML, and is compatible with GNU/Linux and Apache. These mentioned features along with its open-source license, make PHP an ideal choice to be used as the interface between the client application and the server database [9].

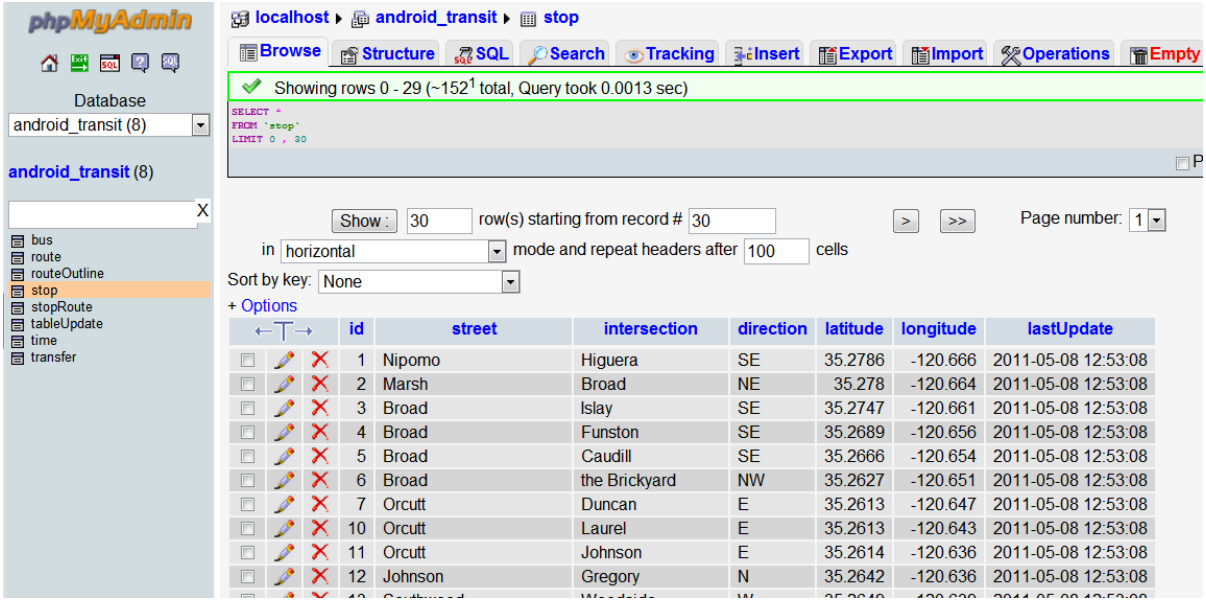


Fig. 7 Screenshot of phpMyAdmin with stop table of SLOTTS database being shown

Steps to Setup

- Install PHP5

```
$ sudo apt-get install php5
$ sudo apt-get libapache2-mod-php5
```

- Restart Apache

```
$ sudo /etc/init.d/apache2 restart
```

E. phpMyAdmin

phpMyAdmin is a web-tool which allows for easy administration of the MySQL database found on the server. This web-tool allows the use of a browser to create, delete, and modify databases found on the server [12]. This tool will allow city officials and maintainers of SLOTTS to easily update and modify transit schedule / information on the database with little technical knowledge of how to program in MySQL. As such, this fulfills the requirement of a single place to update the database and transit information on the server. Figure 7 demonstrates the capability of phpMyAdmin.

Steps to Setup

- Install phpMyAdmin

```
$ sudo apt-get install phpmyadmin
```

- Modify apache configuration file

```
Include the following line to
/etc/apache2/apche2.conf:
Include /etc/phpmyadmin/apache.conf
```

F. Java

Java is a programming language which is able to run on most computer platforms with little problems as a java virtual machine (JVM) runs the compiled code on top of the host operating system. With capability to do http requests while abstracting the process of making UNIX sockets, stability, and my personal experience with java, and strong XML parser, choosing java to update the database was an ideal choice [11].

- Install java JDK and JRE

```
$ sudo apt-get install sun-java6-bin
$ sudo apt-get sun-java6-jdk
$ sudo apt-get sun-java6-jre
```

E. Configuring the Router

The internet connection provided by Charter Communications was connected to the server through a router. Behind the router was a Network Translated Address (NAT), and the server was given a static private address. In order for outside connections to be made with the server, port 80 was set to automatically forward packets to the server's static IP address.

XII. DATABASE

The following section describes my experience in designing and creating the SLOTTS database. Additionally, the database design and schema is explained and outlined. Figure 8 illustrates the schema of the database

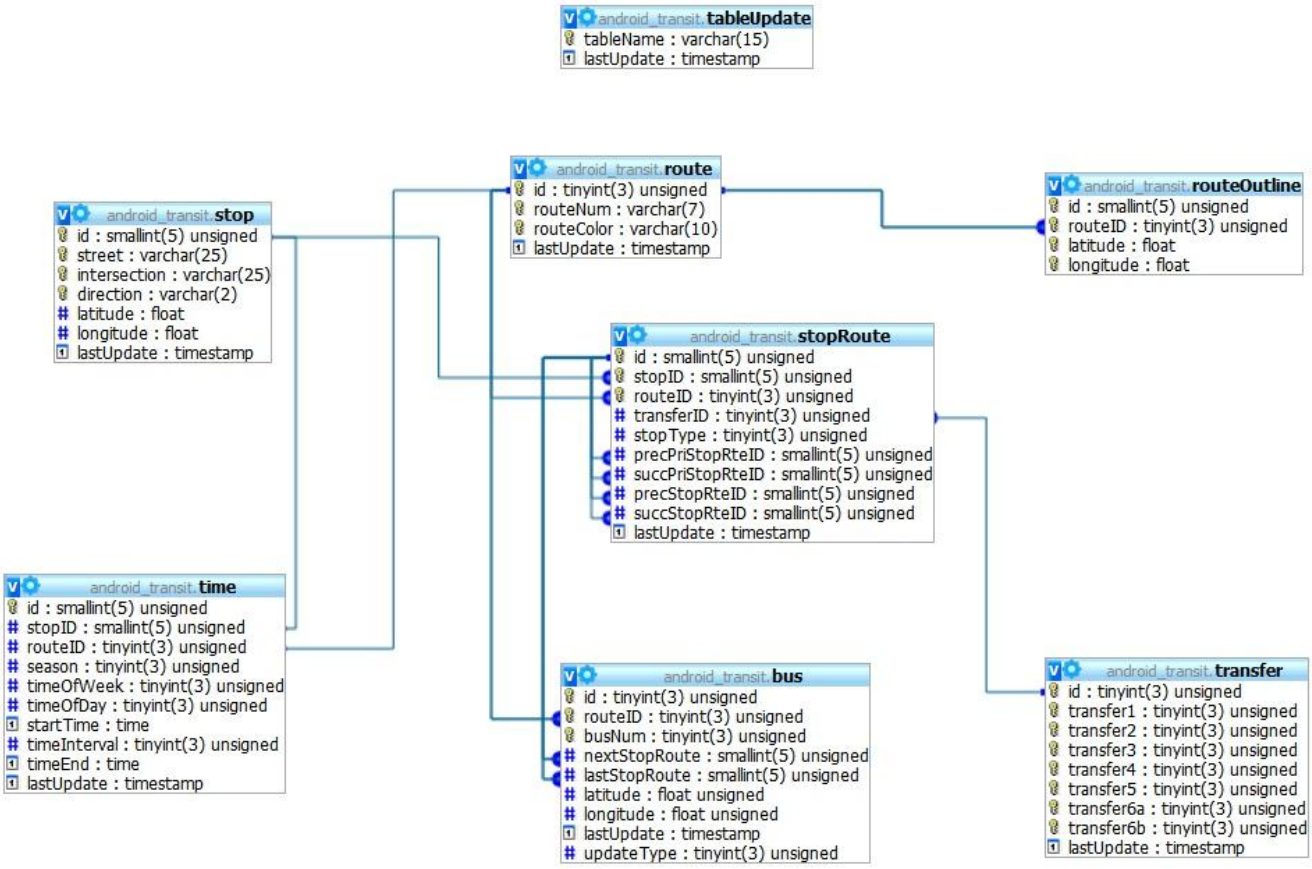


Fig. 8 SLOTTS Database Schema

Designing the Database

The database design needed to be able to describe the entire transit system while being able to be easily understood by others. Additionally, database queries needed to be efficient by being fast and not wasting time finding unnecessary data. With these requirements, a relational database design was chosen. With MySQL's innoDB engine's support of foreign and primary keys, linking entries in two different tables is easily done with error-checking, to avoid primary keys being linked with invalid foreign keys.

When designed, the following concepts were realized:

1. There may be one or two stops with the same name and street information.
2. If there are 2 stops with the same street and intersection name, they are on opposite sides of the street, not necessarily directly opposite to each other.
3. Certain stops have an interval time in which a certain bus from a particular stop will approach the bus.
4. The time interval schedule changes based on the day of the week, time of day, and season.
5. Most stops do not have a designated approach time and are in between stops which do have designated interval times of arrival for certain routes.
6. Some stops serve as designated transfer points between one stop to another stop.
7. Due to route guidance / direction features, the latitude and longitude of the stops would be needed.
8. The actual route outline needs to be recorded in order to be used for bus location prediction if GPS was unavailable.
9. There can be multiple busses running for a single route at the same time.
10. For better prediction tracking of busses, a bus' last updated coordinates need to be distinguished from an actual GPS update or a predicted coordinate.

11. To reduce database queries for updates, a separate table that maintains when a table is updated will be beneficial.
12. This database should be generic to any transit system in case of future portability to other transit systems in the future.

Explanation of Tables

stop table

TABLE 2
STOP TABLE

Field Name	Field Type (size)	Description
id	smallint(5)	identification
street	varchar(25)	street that stop is on
intersection	varchar(25)	nearest intersection to stop if applicable
direction	varchar(2)	the direction of the flow of traffic on the side of the road which the stop is on
latitude	int(11)	latitude of stop
longitude	int(11)	longitude of stop
lastUpdated	timestamp	date/time entry last updated

The stop table contains information regarding a stop regardless of what route it follows, time schedule, or transfer options. By having the stop table focus solely on a particular stop, will allow for quicker queries when searching for stops near a particular location. This is useful in finding which stop is closest to a person when doing route guidance. If a bus schedule information is needed regarding a stop and its relationship with a route, the stopRoute table uses the id of the stop table to link the stop with a route, time information, and transfer information.

route

TABLE 3
ROUTE TABLE

Field Name	Field Type (size)	Description
id	tinyint(3)	identification
routeNum	varchar(25)	street that stop is on
routeColor	varchar(25)	the color of the route
lastUpdated	timestamp	date/time entry last updated

The route table contains information regarding a particular route. As each route has a designated name number, a field for this is included as well as the designated color of the route. To connect a route to a stop, the stopRoute table is used with route's id just as is used with the stop table.

transfer

TABLE 4
TRANSFER TABLE

Field Name	Field Type (size)	Description
id	tinyint(3)	identification
transfer1	tinyint(3)	signifies there is a transfer to route 1
transfer2	tinyint(3)	signifies there is a transfer to route 2
transfer3	tinyint(3)	signifies there is a transfer to route 3
transfer4	tinyint(3)	signifies there is a transfer to route 4
transfer5	tinyint(3)	signifies there is a transfer to route 5
transfer6a	tinyint(3)	signifies there is a transfer to route 6a
transfer6b	tinyint(3)	signifies there is a transfer to route 6b
lastUpdated	timestamp	date/time entry last updated

The transfer table is used in conjunction with the stopRoute table to identify if a particular stop serving a route is also a transfer point to another route. This is particularly useful for identifying stops which transfer to a different route, but with the different route transfer stop being a different stop that is nearby. Due to this table being so specific to the San Luis Obispo transit system, this table does break from concept (12). However, the table's simplicity and relatively low number of possible routes that a city can have make this limitation insignificant.

stopRoute

TABLE 5
STOP ROUTE TABLE

Field Name	Field Type (size)	Description
id	smallint(5)	identification
stopID	smallint(5)	foreign key to stop this entry applies
routeID	tinyint(3)	foreign key to route this entry applies
transferID	tinyint(3)	foreign key to transfer this entry applies
stopType	tinyint(3)	identifies the stopRoute as primary or minor
precPriStopRteID	smallint(5)	The preceding primary stopRoute to this stopRoute
succPriStopRteID	smallint(5)	The succeeding primary stopRoute to this stopRoute
precStopRteID	smallint(5)	The preceding primary stopRoute to this stopRoute
succStopRteID	smallint(5)	The succeeding minor stopRoute to this stopRoute
lastUpdated	timestamp	date/time entry last updated

The stopRoute table is what connects a stop to a route and a transfer option. Essentially, this table allows a single stop to be associated with multiple routes and transfer options. Additionally, the table tracks the order of the stops in a route by having a foreign key which points to itself. There are two types of stops, primary and minor. Primary stops are stops with designated arrival times as mentioned in concept (3). Meanwhile, minor stops are those without designated arrival times but are in between two stops which are primary. This is

primarily used in predicting what times a bus will arrive at a minor stop based on its current location.

time

TABLE 6
TIME TABLE

Field Name	Field Type (size)	Description
id	smallint(5)	identification
stopID	smallint(5)	foreign key to stop this entry applies
routeID	tinyint(3)	foreign key to route this entry applies
season	tinyint(3)	the season this time entry applies to ie: regular, holiday
timeOfWeek	tinyint(3)	the time of week this entry applies to ie: weekday, weekend
timeOfDay	tinyint(3)	the time of day this entry applies to ie: all day, day only
startTime	time	The first time a bus arrives at the stop from the specifies route
timeInterval	tinyint(3)	Time in between bus arrivals
timeEnd	time	The last time the bus will arrive at the stop
lastUpdated	timestamp	date/time entry last updated

The time table addresses concepts (4) and (9). By linking a particular time to a stop and route by their respective ids instead of linking the time table to the stopRoute table, multiple time entries can be made for a single stop and route combination. This is particularly useful when a stop / route combination has a time schedule for the normal season but a different time schedule during holidays.

tableUpdate

TABLE 7
TIME TABLE

Field Name	Field Type (size)	Description
tableName	varchar(15)	name of table
lastUpdated	timestamp	date/time entry last updated

The tableUpdate table addresses concept (11). Each time an entry is added / deleted / modified to any table or if a table structure is modified, the tableUpdate will update the timestamp of the entry with the table's name in tableName with the current timestamp.

routeOutline

TABLE 8
ROUTE OUTLINE TABLE

Field Name	Field Type (size)	Description
id	smallint(5)	identification
routeID	tinyint(3)	foreign key to route this entry applies
latitude	int(11)	latitude of stop
longitude	int(11)	longitude of stop
prevRouteOutline	smallint(5)	the previous routeOutline entry to this current routeOutline
nextRouteOutline	smallint(5)	the next routeOutline entry to this current routeOutline

The routeOutline table is used to keep track of the actual route a bus will follow and drive through when serving a particular route. This table is primarily used in predicting where a bus is on the map by determining how far the bus could have gone on the route based on its last known GPS coordinates, average speed, and time elapsed from last known GPS coordinates.

bus

TABLE 9
BUS TABLE

Field Name	Field Type (size)	Description
id	smallint(5)	identification
routeID	tinyint(3)	foreign key to route this entry applies
busNum	tinyint(3)	bus identifier
nextStopRoute	smallint(5)	the next primary stop bus is heading to
lastStopRoute	smallint(5)	the previous or current primary stop bus has passed over
latitude	int(11)	latitude of stop
longitude	int(11)	longitude of stop
lastUpdated	timestamp	date/time entry last updated
updateType	tinyint(3)	The type of update last performed, either gps or predicted

The bus table links a single bus to a route. This design also allows for multiple busses to be serving a single route. More importantly, the bus table contains the latitude and longitude coordinates of the bus. This coordinate can be from the bus' GPS or predicted as specified in updateType.

Inserting data into the database

The database is one of the most time consuming events of this project. These are due to two reasons, one being my inexperience with databases and the other is inputting data into the database. Most of the latitude and longitude information for each stop entry was obtained directly by using Google Maps and using the drop lat/lang tool which places latitude and longitude coordinates on the map where the cursor sits. This coordinate was then entered into the database.

For the most part, phpMyAdmin was the primary tool used in inserting / modifying data into the database. However, inserts on tables such as stopRoute would require left-joins which the GUI of phpMyAdmin was not capable of.

The following are commands used to insert data into the database. If an insert for a table is not shown, that is because phpMyAdmin's interface was used to add entries for those tables.

routeOutline

```
INSERT INTO routeOutline (routeID,
latitude, longitude)
VALUES ((SELECT id FROM route WHERE
routeNum = 'routeNumVal'), latVal,
longVal);
```

Example:

```
INSERT INTO routeOutline (routeID,
latitude, longitude)
VALUES ((SELECT id FROM route WHERE
routeNum = '5'), -25.000, 35000);
```

time

```
INSERT INTO time (stopID, routeID, season,
timeOfWeek, timeOfDay, startTime,
timeInterval, timeEnd)
VALUES ((SELECT id FROM stop WHERE street =
'streetVal'  &&  intersection =
'intersectionVal'  && direction =
'directionVal'),(SELECT id FROM route WHERE
routeNum = 'routeNumVal'), seasonVal,
timeOfWeekVal, timeofDayVal, startTimeVal,
timeIntervalVal, timeEndVal);
```

Example:

```
INSERT INTO time (stopID, routeID, season,
timeOfWeek, timeOfDay, startTime,
timeInterval, timeEnd)
VALUES ((SELECT id FROM stop WHERE street =
'Nipomo'  &&  intersection = 'Higuera'  &&
direction = 'SE'),(SELECT id FROM route
WHERE routeNum = '5'), 1, 2, 1, 0800, 25,
1500);
```

stopRoute

```
INSERT INTO stopRoute (stopID, routeID,
stopType)
VALUES((SELECT id FROM stop WHERE street =
'streetVal' && intersection =
'intersectionVal' && direction =
'directionVal'),(SELECT id FROM route WHERE
routeNum = 'routeNumVal'), stopTypeVal);
```

Example:

```
INSERT INTO stopRoute (stopID, routeID,
stopType)
VALUES((SELECT id FROM stop WHERE street =
'sname' && intersection = 'iname' &&
direction = 'NW'),(SELECT id FROM route
WHERE routeNum = '5'), 1);
```

XIII. CLIENT TO SERVER INTERFACE

In order for the client to be able to access the server database or to have it calculate directions, an interface needs to be in place to allow for this. The apache web server and php processor serve as the interface for the client to the server. In order for this to be effective, the data returned to the client needs to be formatted in a manner so that parsing text requires minimal work on the server end. Additionally, the php files need to be able to accept appropriate external variables so that the client can make specific requests. Avoiding generic requests will prevent the server from having to send data or information the client may already have. With data caps usually imposed by the service provider of smartphone users, reducing the amount of data that has to be sent is a must.

External / Get Variables

The client is able to initialize external or get variables by placing the variable values into the URL when doing a http request on the php file. For instance, if the client is to send a variable x with the value of 5 and variable y with the value of 6 to a php file named test.php on the root of the server, the URL would look like the following:

```
http://<ip address>/test.php?x=5&y=6
```

From the example above, external variables begin after the '?' character which is appended to the URL. Additional variables can be added, and variables are separated by the '&' character.

This ability of php is powerful and will be the primary method in how the client communicates its request to the server. Variables that will be set to the client are current location latitude / longitude points and destination latitude / longitude points, option flags, and specific table and route names when client syncs itself with the server.

Interface PHP Files

The php interface consists of two php files which clients can access by HTTP request. These two files are updateClient.php and getDirections.php.

updateClient.php

This file is used to update the client with any changes that occurred on the database. The full logical flow of updateClient.php is illustrated in a flowchart seen in Appendix B.

The following table describes the external / get variables of this php file.

TABLE 10
EXTERNAL VARIABLES OF UPDATECLIENT.PHP

Variable Name	Constant Values	dynamic values
table	unk, stop, stopRoute, route, transfer, bus, updateTable, time	
route	1, 2, 3, 4, 5, 6a, 6b	
timestamp		timestamp of client

If the table value is not set or is not appropriate, the server automatically returns the names of the database tables that were updated after the given timestamp. If a timestamp is not given, the server will look for changes made 2-days prior to the current timestamp. Alternatively, if the table name is given, it searches that table for entry changes after the given timestamp or 2-days after the current timestamp if not given. The route variable is primarily used in filtering stopRoute searches. If the table to be searched is table stopRoute and the route is set, the server will return stopRoute entries modified after the given timestamp or 2-days after the current server timestamp.

Entries are delimited by the ';' character. Each field of the table is returned and each are separated by the ',' character within each entry. For instance an example valid request would be the following:

```
http://<ip address>/php/
updateClient.php?table=stop&
timestamp=2011-02-12
```

An example response from the server given this http request would be the following:

```
1,Nipomo,Higuera,SE,35.2786,-120.666;
2,Marsh,Broad,NE,35.278,-120.664;
3,Broad,Islay,SE,35.2747,-120.661;
4,Broad,Funston,SE,35.2689,-120.656;
5,Broad Duncan,E,35.2613,-120.647;
10,Orcutt,Laurel,E,35.2613,-120.643;
```

In this case, the fields returned are in the following format:

```
id, street, intersection, direction,
latitude, longitude;
```

getDirections.php

This file is used for the client to request directions from the server. The full logical flow of updateClient.php is illustrated in the flowcharts seen in Appendices C.1-C.4.

The following table describes the external / get variables of this php file.

TABLE 11
EXTERNAL VARIABLES OF GETDIRECTIONS.PHP

Variable Name	Constant Values	dynamic values
slat		latitude of client
slong		longitude of client
dlat		destination latitude
dlong		destination longitude
option	1 = shortest route with transfer, 2 = shortest route without transfer	

getDirections.php instantly checks to ensure that the external variables slat, slong, dlat, and dlong are set. Otherwise the php file returns nothing to the client and exits with failure on the server side. The option variable is not required; however, if it is not set, the server automatically defaults to finding the shortest route with transfer option.

Upon verifying that the required variables have been given, the server attempts to find the nearest stop to the client coordinates (slat, slong). This is done by setting up a query string, which will sort the distances of each stop from the client location by order of smallest distance to greater distance. Additionally, only the 2 stops which have the smallest amount of distance from the client is returned.

The core formula for finding the distance between the stop coordinates and client coordinates is the haversine formula which is able to determine distances between two coordinate points while considering the curvature of the earth [13]. The definition of the haversine formula is as follows:

For two points on a sphere (of radius R) with latitudes φ1 and φ2, latitude separation Δφ = φ1 − φ2, and longitude separation Δλ the distance d between the two points is:

$$\operatorname{haversin}\left(\frac{d}{R}\right) = \operatorname{haversin}(\Delta\phi) + \cos(\phi_1)\cos(\phi_2)\operatorname{haversin}(\Delta\lambda)$$

[13]

This translates into the following MySQL query:

```
SELECT *,
3956 * 2 * ASIN(SQRT(POWER(SIN((" . $lat .
" - abs(latitude)) * pi() / 180 / 2), 2) +
COS(" . $lat . " * pi() / 180) *
COS(abs(latitude) * pi() / 180) *
POWER(SIN((" . $long . "- longitude) * pi()
/ 180 / 2), 2)))
as distance
FROM stop
ORDER BY distance limit 2 [13]
```

Though this is a highly inefficient way to search for the closest stop to the client, the server is able to perform this query in under a millisecond due to the relatively small size of entries in the stop table.

This is done once again to find the closest stop to the destination coordinates. The returns of both queries are checked to ensure there exists a closest stop to the destination or client coordinates and is within the maximum distance. For instance, this would fail if a user is attempting to get directions to a bus stop while in the city of Los Angeles, CA. This is put in place to prevent abuse of the system.

Once done, options are checked if the nearest stop to client and nearest stop to destination are the same route. If they are not the same route and user specifies, no transfer option, then the 2nd nearest stops to both client and destination are checked for the same route. If there is no nearest start and destination stop that have the same route when no transfer is selected, then nothing is returned to the client and the server terminates with failure.

If transfer option is selected, and the two nearest stops are of different routes, a check is done to ensure that there is a transfer option en-route which will allow the user to transfer from the route from the first stop to the route of the last stop. If no transfer option is available, the other nearest stops are checked for transfer option. If there exists no possible transfer with any combination of the stops, nothing is returned to the client and the server terminates internally with failure.

An example of a valid http request to the server would be:

```
http://<ip address>/php/getDirections.php?
slat=35.25037&slong=-120.6439&
dlat=35.25774&-120.69157&
option=1
```

This is requesting directions from a client location near the stop with street name Broad, with intersection name of Marigold Shopping Center and a destination location near the stop with the street name Los Osos Valley Road and Laguna Village. For the start stop, the route is route 3 while the destination stop is route 4.

The server will then attempt to find if these two routes have a stop that contains a transfer option between the two routes. The database will indeed find that the stop with street name Downtown Transit Center is referenced by two stopRoutes which are of route 3 and route 4. Additionally, checking the transfer entries linked to these stopRoutes show that they indeed have a transfer option.

Therefore, this query would yield the following result:

```
START
1, walk, 100, SE;
2, board, 3, Broad, Marigold Center,
      35.2502, -120.644;
3, transfer, 4, Downtown Transit Center,
      NULL, 35.2829, -120.662;
4, exit, 4, LOVR, Laguna
      Village, 35.258, -120.629;
5, walk, 100, SE;
END
```

As seen from the above server response, the format of the response is:

```
START
step 1, walk, distance (ft), compass
      direction;
step 2, board, route, street, intersection,
      latitude, longitude;
step 3, transfer, route, street,
      intersection, latitude, longitude;
step 4, exit, route, street, intersection,
      latitude, longitude
step 5, walk, distance (ft), compass
      direction
END
```

Obtaining Turn Based Direction

To determine the actual directions for the walk portions of the navigation, the server queries the Google Server and uses the Google Maps API. The response from Google is parsed using the PHP xml parser. Two queries are done to G, one to determine walk navigation from client location to first stop and another to determine walk navigation from destination stop to destination location.

The Google Maps API user agreement states that the API is used in conjunction with Google maps. Because the server is being used solely by the client device running Google Maps within their application, the SLOTTS server is in compliance with the Google user agreement.

The following http request to the Google Maps API will yield directions from the client start location at 35.25037, -120.6439 to the first / start stop on Broad at Marigold Center:

```
"http://maps.googleapis.com/maps/api/directions/json?origin=" + 35.25037 + "," + -120.6439 + "&destination=" + 35.2502 + "," -120.644 +"&sensor=false" [14]
```

XIV. UPDATING CURRENT BUS LOCATION

The city of San Luis Obispo uses a server with software installed by Digital Recorder Incorporated in order to update bus locations and display bus locations on the city website. If the bus GPS is unavailable, the DRI software is able to do bus location prediction.

With permission from the city of San Luis Obispo and from DRI, our server and client devices are able to use the auto generated XML files generated by DRI software on the city server for SLOTTS. The server uses a java background process that every five seconds requests the city server for an updated XML file containing updated. Appendix D is a flowchart which illustrates the logical flow of the java background process.

On process startup, the process initializes a bus class for each bus serving a route. This bus class contains the bus number, route the bus is serving, latitude, longitude, last update time, and last update type (self-predicted or DRI). All bus objects are then updated using values found in the database. If the last updated time is more than 10 minutes old, the server sets the bus coordinates to where the bus normally would be if on time based on transit schedule found on the database.

Every five seconds, the java process does an http request to the city server. Using a Document Object Model (DOM) parser, the process parses the data, creates bus classes which contain bus number, bus route, bus latitude, bus longitude, and update type. If there is no response from the city server or the data received is not valid, the server predicts bus location based on where it should be based on transit schedule found in the database.

After the server's internal objects are updated, the server updates the bus entries in the database with the information found in its internal bus objects.

XV. TESTING

Due to time constraints faced. Complete testing was not able to be done where the client-app was to be incorporated with the SLOTTS server. The original testing plan consisted of three stages.

Internal Testing

The purpose of this test was to verify that the database, client interface, and java back-end process are functioning when accessed manually outside of the client front-end application. This would require doing several possible types of queries and modified on the server database. Accessing the php interface manually through a web browser and running the java process and evaluating the current state of the database to see if the java process is in fact updating the database. With nearly perfect uptime of the city server, simulating poor results or no connection from city server was performed by running the server unplugged from the router.

From this testing, the server proved capable and ready for the next phase of testing. All processes ran as expected. The server performed quickly at a satisfactory speed with potential to carry greater load and greater number of instantaneous clients.

Small Scale Client-Server Integrated Testing

This test would incorporate the front-end client device with a vertical prototype and the back-end server. Tests would be conducted to ensure that the client-app can indeed contact the server and that the server continues to behave as expected. The testing size would be small and the purpose is to test preliminary speed and functionality, not server capability to handle greater loads. As such, testing would be limited to developers and by invite only.

Due to time constraints, I was not able to work with the front-end developers in time to do this phase of testing. We expect to begin this phase of testing a few weeks after this report is written. It is expected to last 2-3 weeks.

Beta Release Testing

This test would incorporate a refined version of the front-end client that is suitable for public release. The primary reason for this test is to test the server and interfaces with increased load. Additionally, this would be the last test performed on the server before being integrated with a gold release of the front-end client app.

Due to the server and the client application still pending a small scale client-server integrated test, this test has not been conducted. Our team expects to reach this phase of testing in August or September of 2011.

XVI. CONCLUSION

The SLOTTS server is a critical element of SLOTTS. Its integration with a front-end client app will greatly increase the effectiveness of informing and motivating citizens and visitors of the city of San Luis Obispo to utilize the transit system and reduce traffic congestion and the carbon footprint it carries.

The creation of the SLOTTS server was no easy task. Going into this project I had no formal education or training in database concepts, MySQL, PHP, or how to manage a LAMP configured server. This project required me to research and learn the technologies used from the very beginning. Learning how to design a relational database, creating the database, and adding entries into the database was the most time consuming aspect of this project. Being that the server is heavily database reliant, a well-designed database was crucial to this project. It would be reasonable to say that this project is a database project as much as it is a server project.

In conclusion, the current state of the SLOTTS back-end server meets its engineering requirements. It solves the problem of computationally complex calculations being too intensive on limited mobile devices. By placing the burden of these CPU intensive calculations on the server, the front-end client has more resources to devote to a more aesthetically pleasing and functional GUI suitable to help riders navigate the transit system with ease.

ACKNOWLEDGEMENT

"Praise the bridge that carried you over."

-George Colman

This project was made with the culmination of knowledge and inspiration from the talented professors at California Polytechnic State University. Such noteworthy professors who have made a significant impact in my education and person as a whole include but are not limited to are John Seng, PhD; John Oliver, PhD; David Braun, PhD; Kurt Mammen, MS; Hasmik Gharibyan, PhD; Phillip Nico, PhD; David Janzen, PhD; Lynne Slivovsky, PhD; and Hugh Smith, PhD.

I also thank Christopher Lupo, PhD, for being my senior project advisor and helping me find solutions to problems encountered while working on the project. Success of this project would have not been possible without his support.

Additionally, I thank those individuals who have directly collaborated with me in the creation of SLOTTS. These individuals include Jeff Brown and Zach Negrey for the creation of the android and iOS front-end client application for SLOTTS; John Webster of the City of San Luis Obispo Transit and Dan Trujillo of Digital Recorder Inc. for giving us the necessary permissions and information to make this system possible.

Last but not least, I thank my family for the love, encouragement, and support to finish this project. I thank my parents, May and Alan Dimalanta, for their support and especially for funding the necessary hardware to construct this project. Lastly, I thank my fiancée, Pa Chia Vue, for her aide in inputting data into the project database and love which helped me overcome all difficulties faced throughout this project.

REFERENCES

[1] A. Cornejo (2010) Council Defers Bus Fare Hikes on Mustang Daily. [Online]. Available: http://sanluisobispo.com/2010/07/24 /1226949/council-defers-faire-hikes.html

[2] X. Jia, E. Sullivan, C. Nuworsoo, N. Hockaday, "EDAPTS Benefit / Cost Evaluation," California State Polytechnic University, Pomona, CA, California Polytechnic State University, San Luis Obispo, CA

[3] J. Webster. "RE: SLO Transit inquiry for Cal Poly senior project" Personal e-mail (Feb 1, 2011)

[4] (2011) The City of San Luis Obispo Transit Program Website. [Online]. Available: www.ci.san-luis-obispo.ca.us/publicworks/ transit.asp

[5] J. Jing, A. Helal, A. Elmagarmid, "Client-Server Computing in Mobile Environments" GTE Laboratories Inc., University of Florida, Purdue University

[6] (2011) Amazon Elastic Compute Cloud Overview. [Online]. Available: aws.amazon.com/ec2/

[7] (2011) Ubuntu Website. [Online]. Available: www.ubuntu.com/ ubuntu/

[8] (2011) About Apache HTTP Server Project. [Online]. Available: httpd.apache.org/ABOUT_APACHE.html

[9] (2011) What can PHP do? [Online]. Available: www.php.net/ manual/en/intro-whatcando.php

[10] A. Lentz, R. Schumacher (2010) Dispelling the Myths on MySQL Developer Zone Website. [Online]. Available: dev.mysql.com/tech -resouces/articles/dispelling-the-myths.html

[11] (2011) What is Java? [Online]. Available: http://www.java.com/en/download/faq/whatis_java.xml

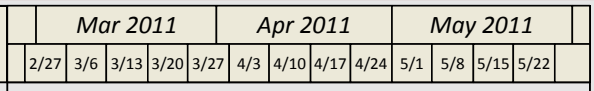
[12] (2011) About phpMyAdmin [Online]. Available: http://www.phpmyadmin.net/home_page/

[13] A. Rubin, (2006)"Geo/Spatial Search with MySQL" MySQL AB

[14] (2011) Google Maps Javascript API V3 Basics [Online]. Available: http://code.google.com/apis/maps/documentation/javascript/basics.html

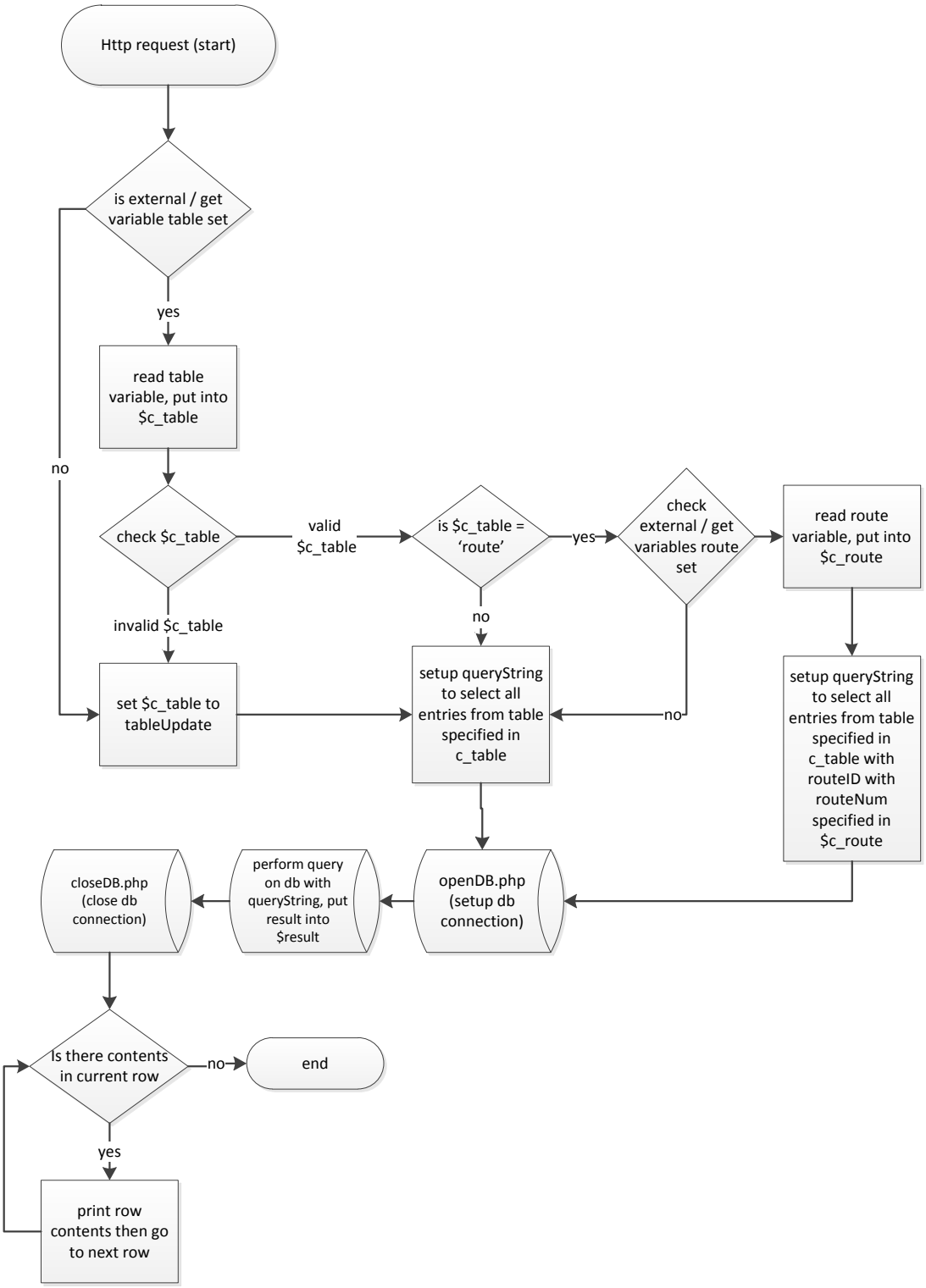
Appendix A – *Schedule of Work Gantt Chart*

	Task Name	Start	Finish	Duration
1	ENV-1	2/24/2011	2/25/2011	2d
2	ENV-2	2/28/2011	3/2/2011	3d
3	ENV-3	3/2/2011	3/10/2011	7d
4	ENV-4	3/2/2011	3/4/2011	3d
5	ENV-5	3/4/2011	3/14/2011	7d
6	DTB-1	3/21/2011	3/23/2011	3d
7	DTB-2	3/23/2011	3/31/2011	7d
8	JBE-1	3/21/2011	3/22/2011	2d
9	JBE-2	3/22/2011	3/30/2011	7d
10	JBE-3	3/30/2011	4/7/2011	7d
11	JBE-4	4/1/2011	4/28/2011	20d
12	PHP-1	3/21/2011	3/22/2011	2d
13	PHP-2	3/22/2011	3/30/2011	7d
14	PHP-3	3/30/2011	4/7/2011	7d
15	PHP-4	4/7/2011	4/26/2011	14d
16	CLA-1	4/15/2011	5/4/2011	14d
17	CLA-2	4/15/2011	5/4/2011	14d
18	TESTING	5/4/2011	6/3/2011	23d
19	DRAFT FINAL REPORT 1	5/6/2011	5/16/2011	7d
20	DRAFT FINAL REPORT 2	5/13/2011	5/23/2011	7d
21	DRAFT FINAL REPORT 3	5/20/2011	5/30/2011	7d
22	FINAL REPORT	5/27/2011	6/6/2011	7d

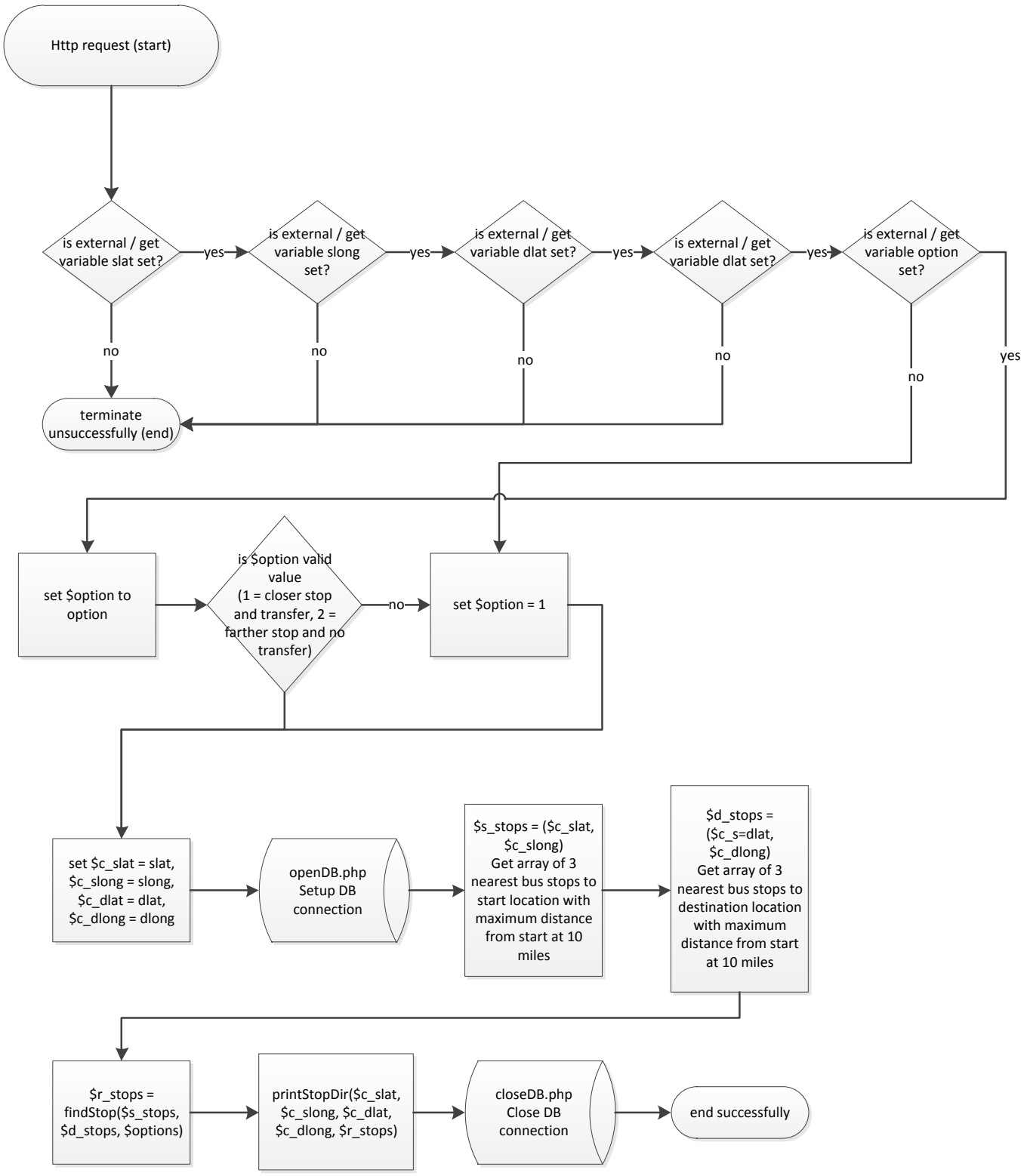


For task name definitions, see VII. Work Plan

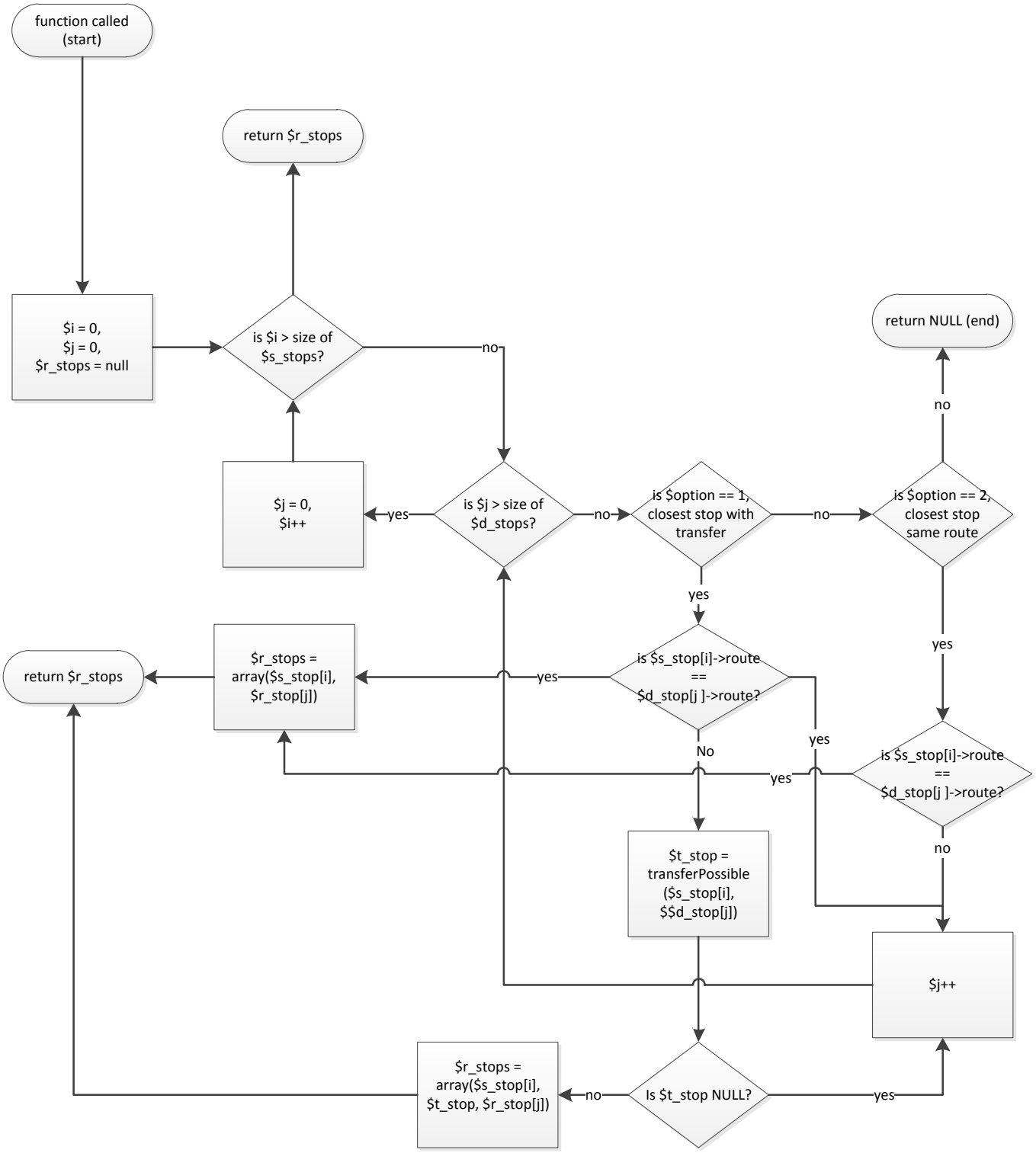
Appendix B – *updateClient.php Flowchart Diagram*



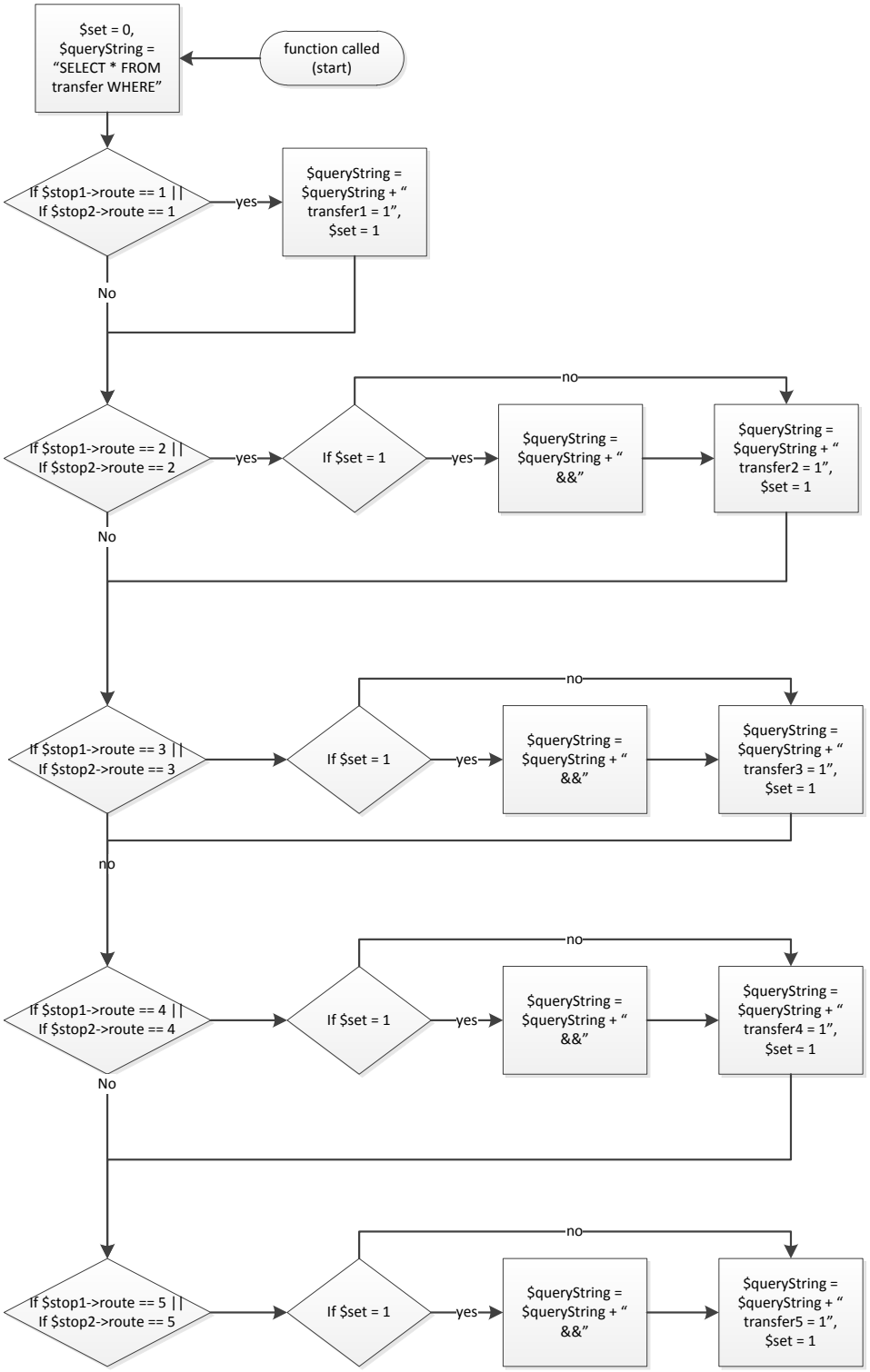
Appendix C.1 – *getDirections.php (main) Flowchart Diagram*



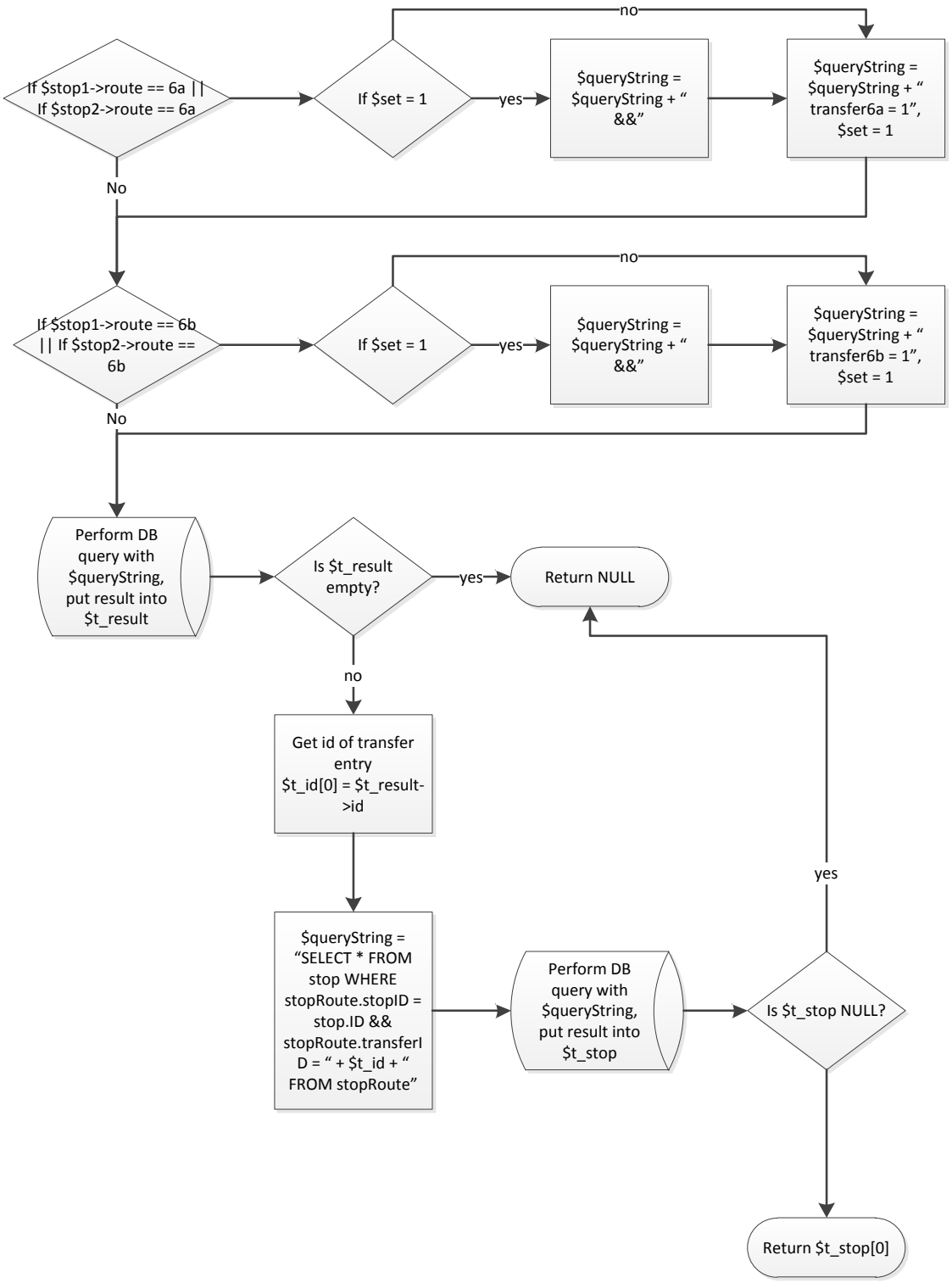
Appendix C.2 – *getDirections.php (findStop($s_stops, $d_stops, $option)) Flowchart Diagram*



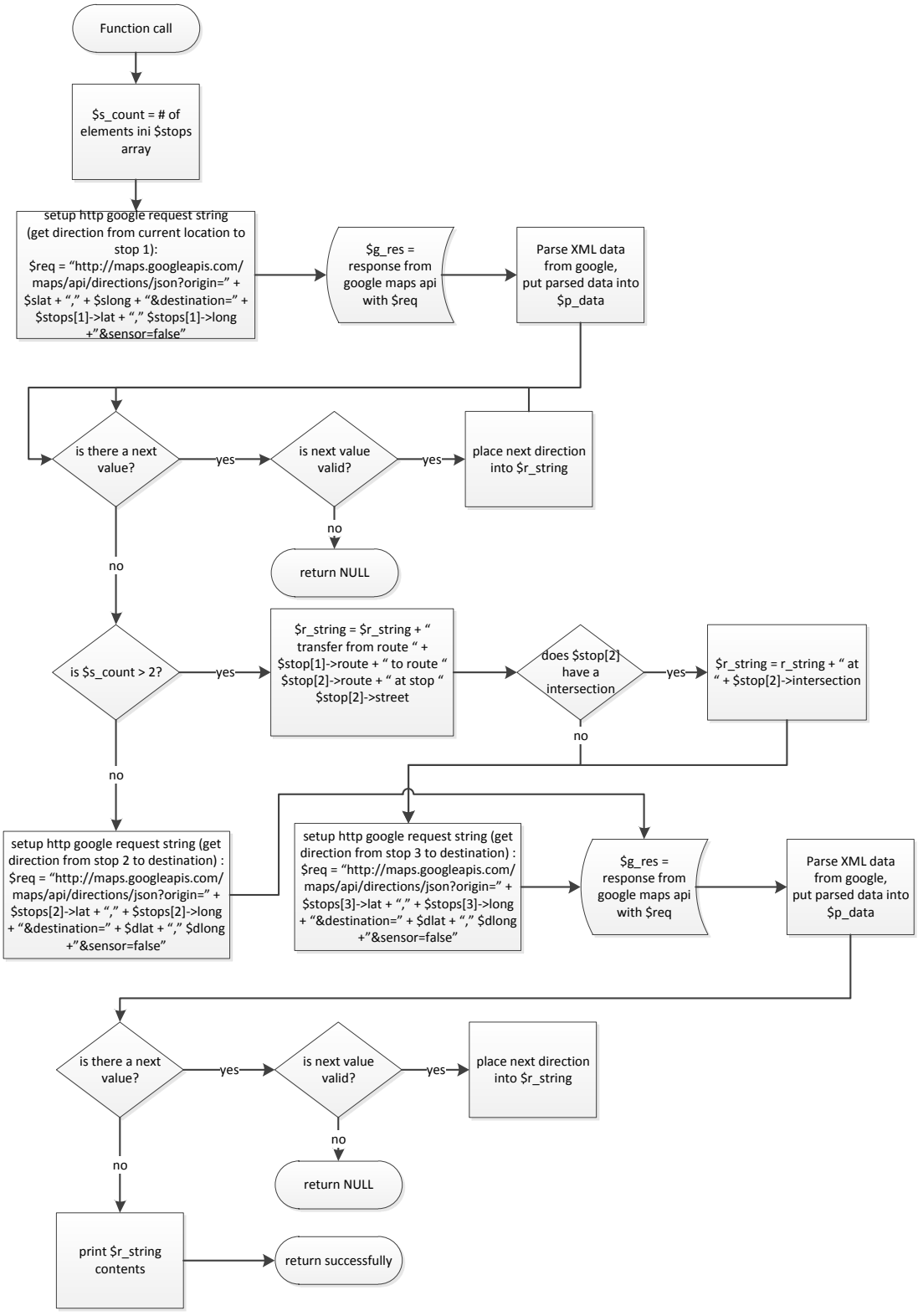
Appendix C.3.1 – *getDirections.php (transferPossible($stop1, $stop2)) Flowchart Diagram*



Appendix C.3.2 – *updateDirections.php (transferPossible($stop1, $stop2)) Flowchart Diagram Continued*



Appendix C.4 – *getDirections.php (printStopDir ($slat, $slong, $dlat, $dlong, $stops)) Flowchart Diagram*



Appendix D – *updateBus.java Flowchart Diagram*