MS3: Smart Bus Application

Supervisors: Maria Spichkova, Margaret Hamilton
Team Member: Jiachen Yan, Ran Jing

This document is written by
Jiachen Yan - jiacheny2@gmail.com
and
Ran Jing - jingr1986@gmail.com
and last updated was on Mar 2015

Table of Content

1. Introduction

Smart bus service is currently running in Melbourne, Australia. There are nine lines, which are 703, 900, 901, 902, 903, 905, 906, 907 and 908. This project is to improve the smart bus service even further to increase the safety on the road, save the petrol of the bus, save the time and power.

There are three main functionaries of the prototype system. They are the system view, the passenger view and the driver view. The system view provides users a summary view of a whole system. Users can view all runs in one day of one direction of one bus line. Information of one run includes the total number of optional stops, the total number of booked optional stops, the number of passengers who booked the stops, and Google Map which shows the regular stops, booked optional stops and unbooked optional stops. The passenger view is shown after a passenger log in. The view allows the passenger to book a optional stops of one direction of one line at a certain time. The view can also show the booking history of the passenger. The driver view is shown after a driver log in. The view is able to show the next run which the driver need to work, and also all the shifts the driver has in the future.

2. Development Tools

2.1 Tools

Due to the unsuitability of RMIT server, the development was done mainly on the local machines. The main development tools are MAMP, Sequel Pro, and text editor, such as Sublime Text, Coda, Github.

MAMP

Sequel Pro

Sublime Text

Coda

GitHub

- **MAMP (free)** installs a local server environment in a matter of seconds on your Mac OS X computer, be it MacBook or iMac. Like similar packages from the Windows- and Linux-world, MAMP comes free of charge, and is easily installed. MAMP will not compromise any existing Apache installation already running on your system. You can install Apache, PHP and MySQL without starting a script or having to change any configuration files! Furthermore, if MAMP is no longer needed, just delete the MAMP folder and everything returns to its original state (i.e. MAMP does not modify any of the "normal" system).
- **Sequel Pro (free)** is a MySQL database management for Mac OS X. It has full MySQL support, such as full table management, table trigger, Create, rename, duplicate (with content) and delete databases, easy user management. The most important reason to use this software for this project is because the connection can be easily set up on Mac.
- Text Editor such as **Sublime Text (free)** or **Coda (need to purchase)** is used to write the code. They support the languages such as HTML, CSS, Javascript, jQuery, PHP etc.
- **GitHub** is a collaborative software development tool. It can sync and merge the codes automatically, which make the work much more efficient. It also has a GUI software for Mac and Windows.

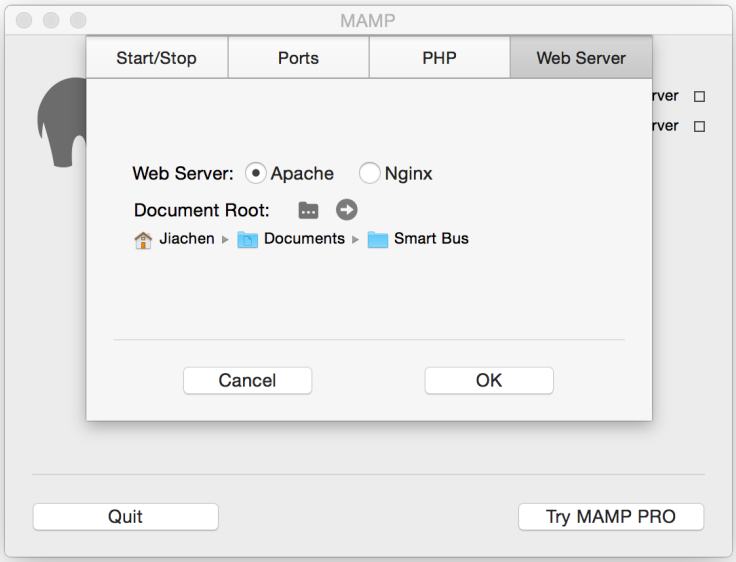
2.2 Set Up

2.2.1 GitHub

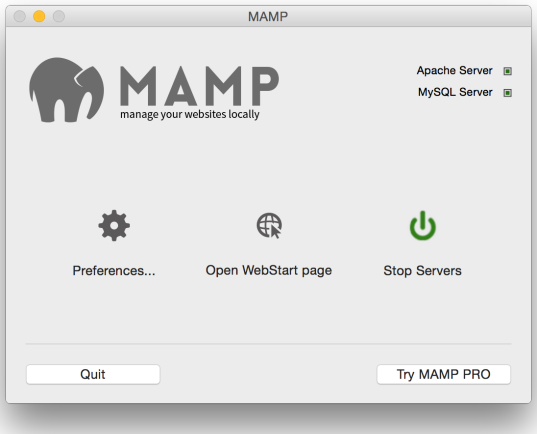
Go to https://github.com/jiacheny/smartbus and click ‘Clone in Desktop’ or ‘Download ZIP’ to get the code. And then to add the downloaded folder to GitHub for Mac or Windows.

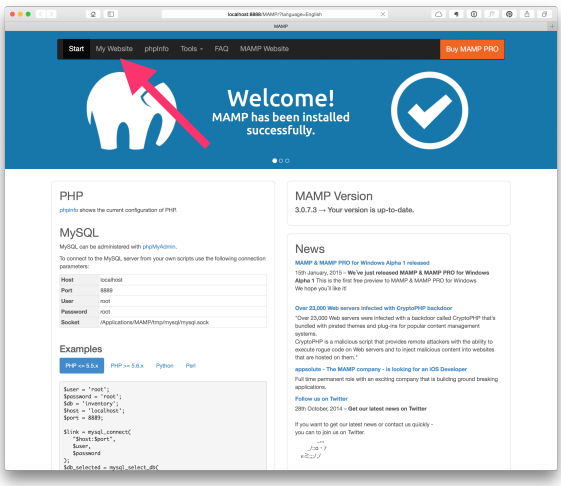
2.2.2 MAMP

Before ‘Start Servers’, click ‘Preferences’. In ‘Web Server’ tag, set the document root to the folder you downloaded from GitHub. The version of PHP can also be chosen. We used version 5.6.2.



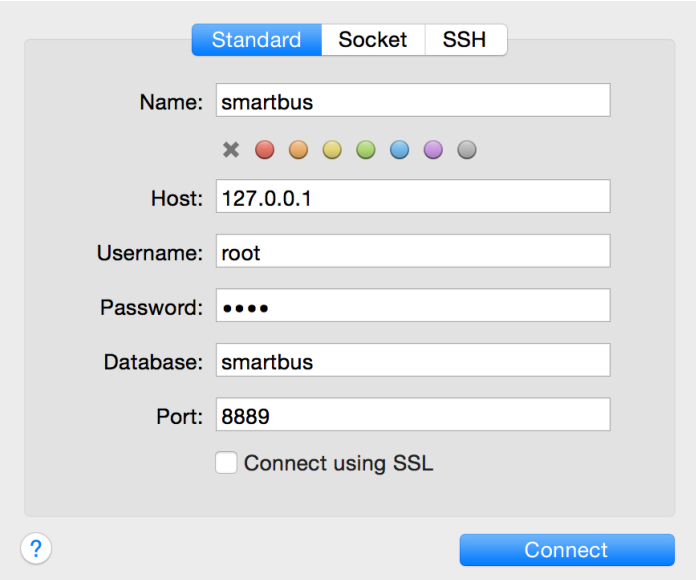
After setting up the ‘Document Root’, click ‘OK’ and then ‘Start Servers’. If successful, a successful webpage will be shown, and the project webpage can be view in ‘My Website’ tag, or the website can be accessed directly from http://localhost:8888/.





2.2.3 Sequel Pro

To view the database of the project, the connection to the server must be established first. The connection to MAMP is quite simple.



If you have own database running, host, username, password, port will be different. These information will be provided by the company.

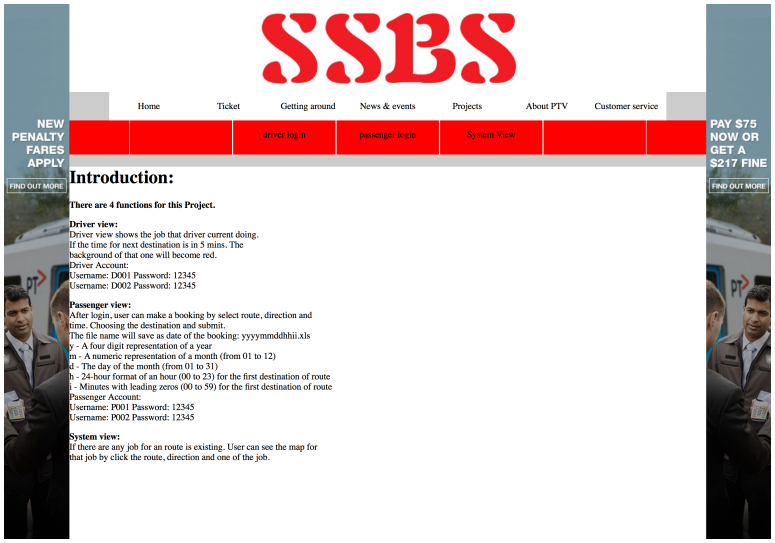
3. Database Design and Code Structure

(Please refer this section to appendix)

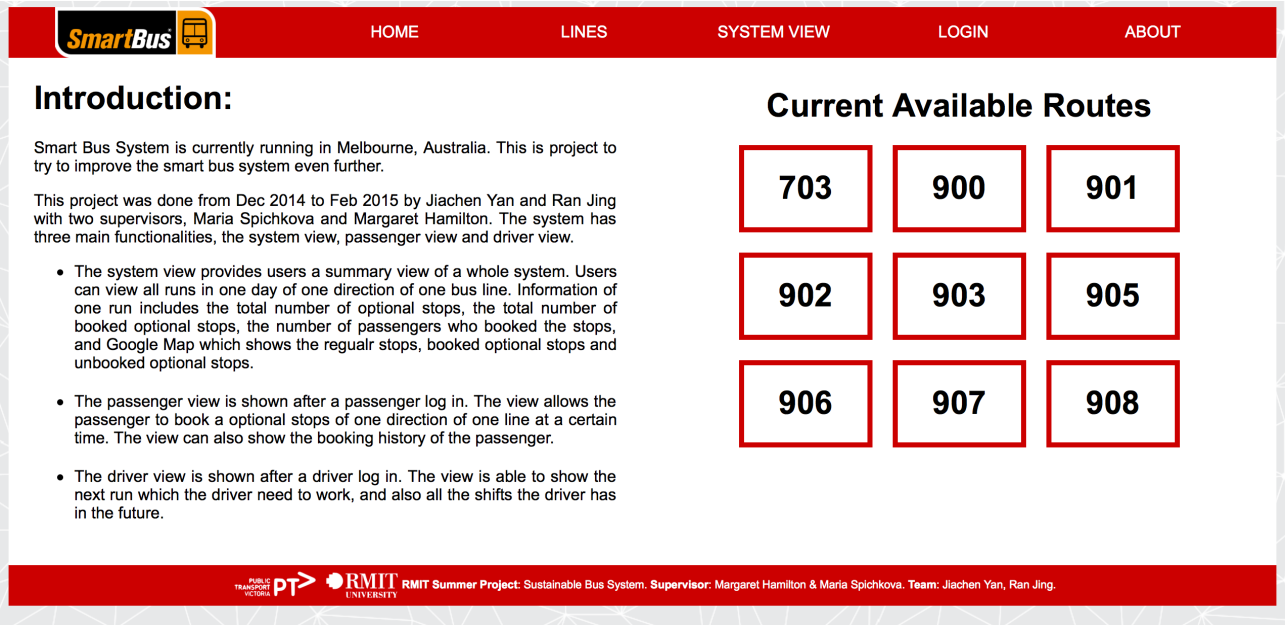
4. Discussion

- The previous work was based on the file system. It worked fine with the unchanged, small amount of the data. However, the file system was not very efficient to manage to huge amount of data and frequent changing data, especially the timetable data (approx. 1000 rows for one direction of one line in on day). Hence, we decided to move all the data to the database. During the move to database, almost all the php function files were re-wrote. The logic for every functionalities were also re-wrote, refined and deleted.

- A new web interface was designed. The usability of the new clean and neat interface is greatly improved, as all the unnecessary parts are removed.



Old index.php



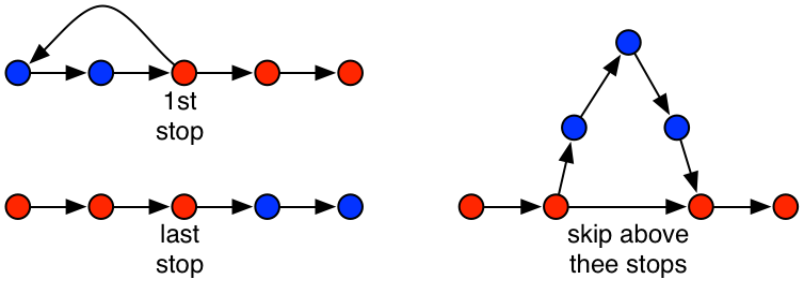
New index.php

- Previous work has two separate login, one for passengers and one for drivers. We merged these two login into one login to improve the usability of the website. Now passengers and drivers share same login page, and of course, they see different functionalities after login.

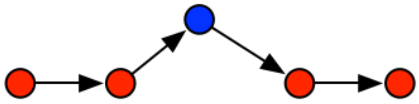
- When we created a new design, we did not take mobile view into an account. We realised this problem when we almost finished the new interface, so we decided to give up for mobile view due to the time constraint. For mobile view compatiable, bootstrap is recommended.

- Current database only has the data for Line 905 towards The Pines SC, so this is first priority to have all 9 lines data stored in the database.

- The following three situations have not been implemented yet.



- However, the following case works at the moment.



- We created a page called lines.php. We find this page can help you to decide a optional stop easily. Also this lines.php can be seen as a simple version of the system view.

- The another challenge is the change of the data from PTV. PTV does not give the notice when they change data. Once they change the data, api call may fail or return empty data, so it is important and necessary to find a way to auto detect and auto update the data in our database.

- When we try to insert the optional stops into one line, we do manually. The current workflow is:
    1. add optional stops into table ‘stopsOpt’ (manually)
    2. have all regular stops in order for one direction of one line in table ‘lineStopsOrder’ (data generated by code)
    3. copy these sorted data from ‘lineStopsOrder’ to ‘stopsInOrder’ (manually)
    4. insert the optional stops for that direction of the line and update order_id accordingly (manually)

  This process can definitely be improved by having a interface to insert a optional stop.

- ‘stopsInOrder’ had an attribute called ‘stop_id’. This ‘stop_id’ has both regular and optional stops. We use another attribute ‘regular’ to identify the type of stops. 1 is regular stop, and 0 is optional stop. We think this design is not a perfect way to store data, but due to the time constraint, we use this temporary design.

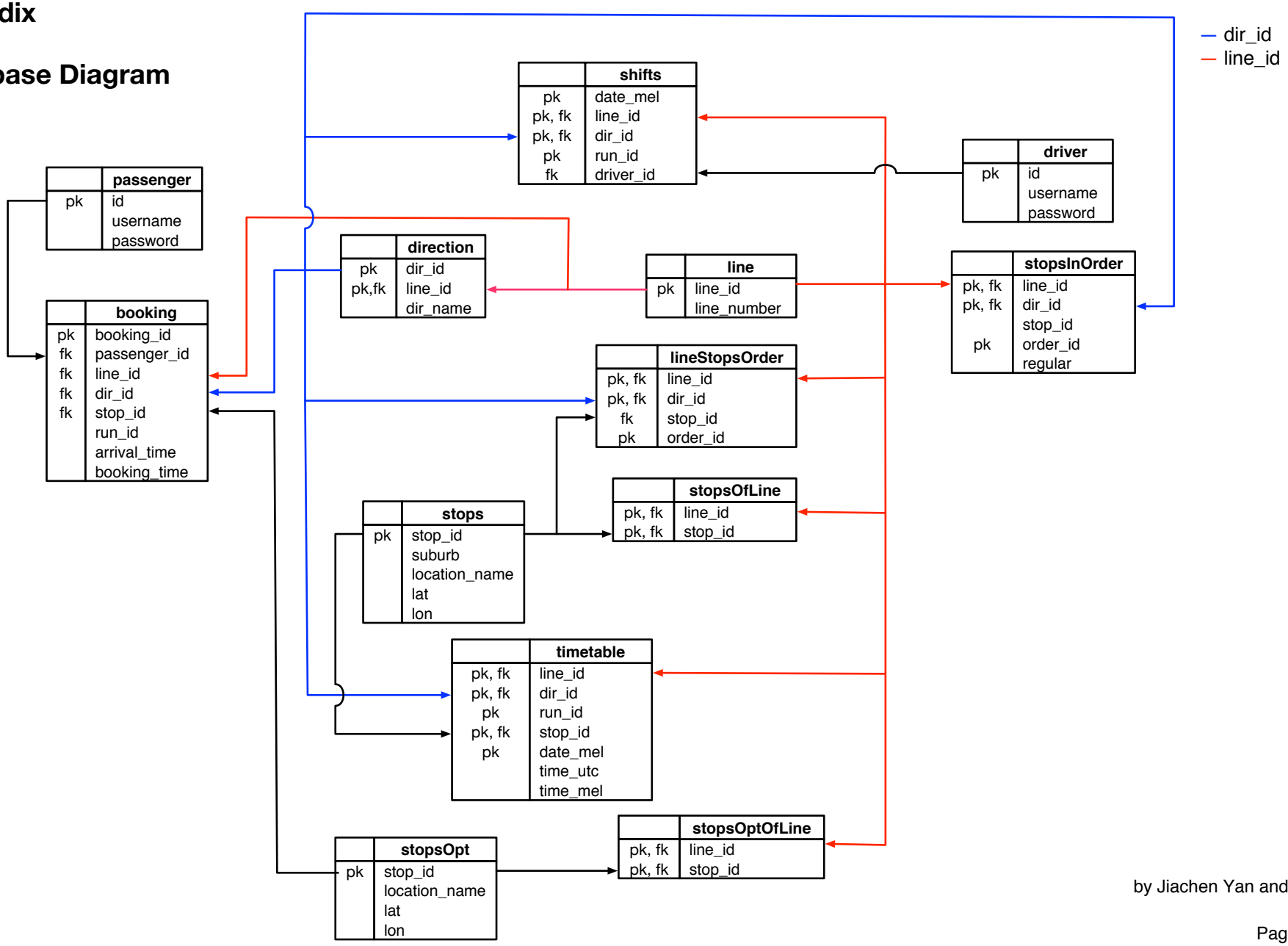
5. Future Improvements

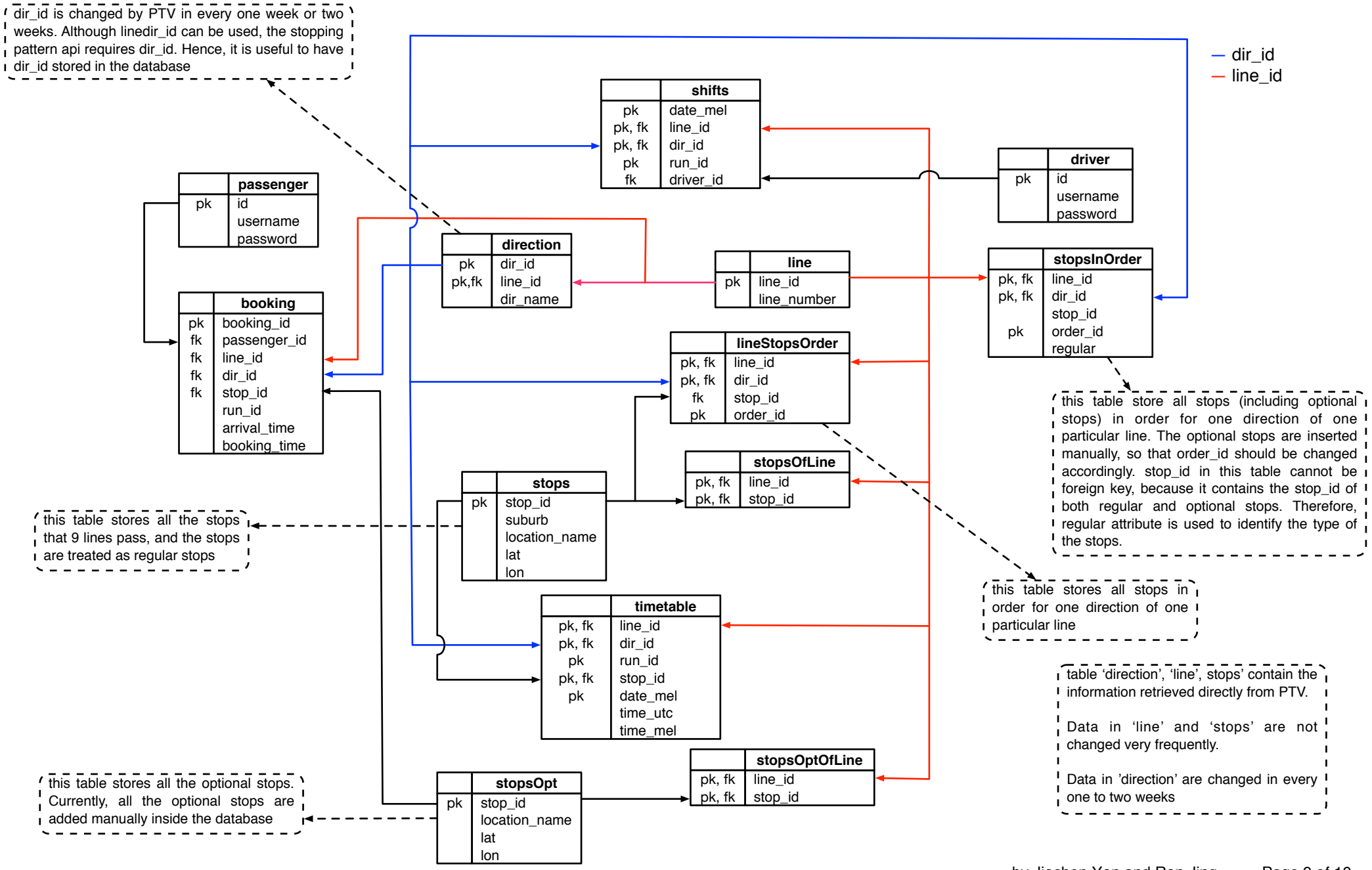
The following functionalities were not planned or not completed during the summer project period. However, they are potentially useful.

Task	Estimated Time	Comments
Import all data from PTV	2 or 3 days	All 9 lines data and Route 445 data. As PTV keeps changing the data, the correctness of data should be verified carefully.
Cancel a booked stop	One day	
Change one regular stop to an optional stop, or one optional stop to a regular stop at a particular time.	Less than a week	To add this functionality to the system, each stop of one direction of one line should associate with one time attributes. The database design may need to be changed.
System works if PTV changes API data.	Less than a week	
An interface for adding addition optional stops (for system view and passenger view)	Less than two weeks	Database design might be changed, as optional stops might have pre-set and customised (by passengers) attributes.
Mobile website	Less than two weeks	
Build an app.	Cannot estimated	

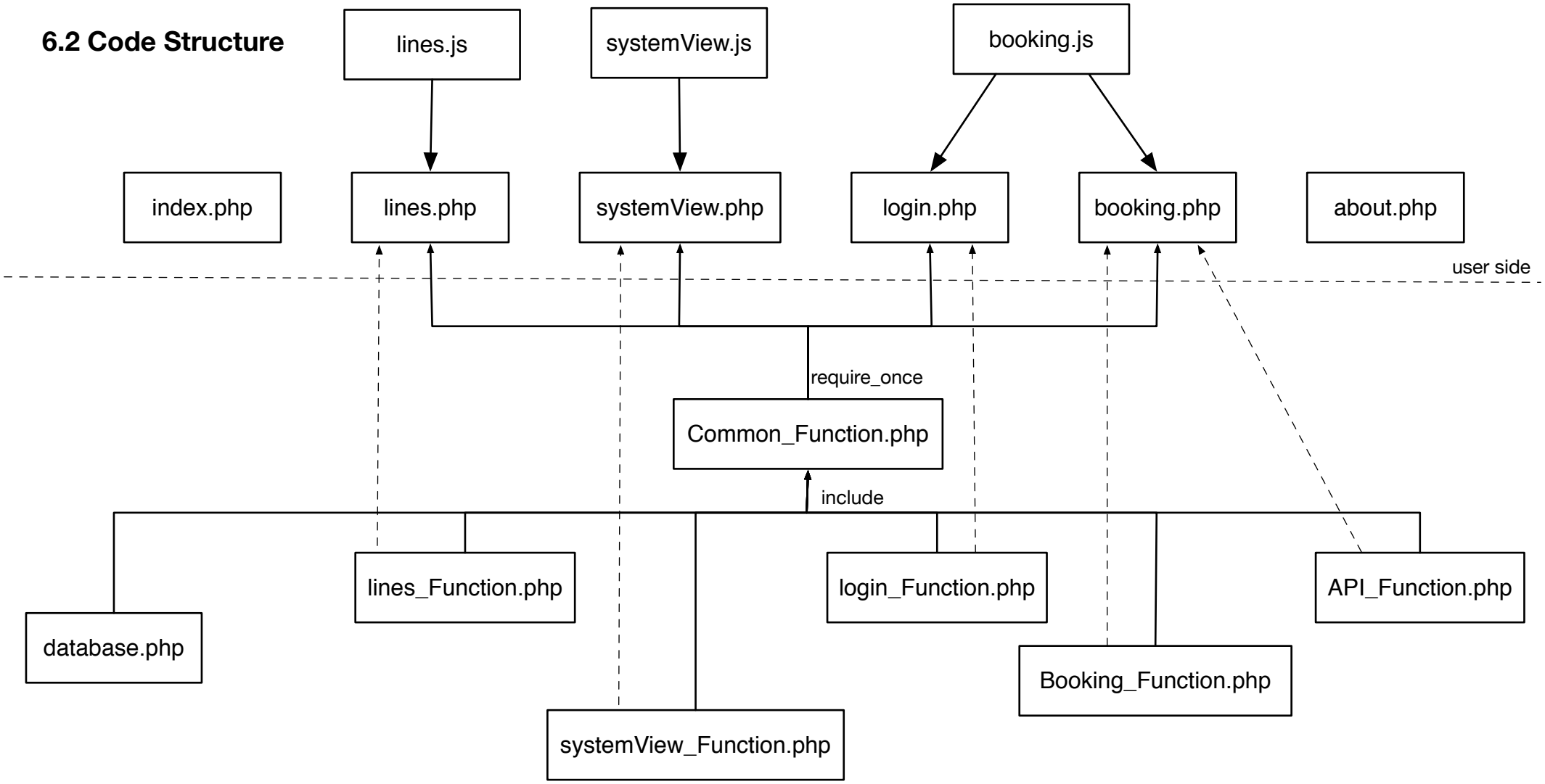
6. Appendix

6.1 Database Diagram




dir_id is changed by PTV in every one week or two weeks. Although linedir_id can be used, the stopping pattern api requires dir_id. Hence, it is useful to have dir_id stored in the database
dir_id
line_id
shifts
pk date_mel
pk, fk line_id
pk, fk dir_id
pk run_id
fk driver_id
driver
pk id
username
password
passenger
pk id
username
password
direction
pk dir_id
pk,fk line_id
dir_name
line
pk line_id
line_number
stopsInOrder
pk, fk line_id
pk, fk dir_id
stop_id
pk order_id
regular
booking
pk booking_id
fk passenger_id
fk line_id
fk dir_id
fk stop_id
run_id
arrival_time
booking_time
lineStopsOrder
pk, fk line_id
pk, fk dir_id
fk stop_id
pk order_id
this table store all stops (including optional stops) in order for one direction of one particular line. The optional stops are inserted manually, so that order_id should be changed accordingly. stop_id in this table cannot be foreign key, because it contains the stop_id of both regular and optional stops. Therefore, regular attribute is used to identify the type of the stops.
stopsOfLine
pk, fk line_id
pk, fk stop_id
stops
pk stop_id
suburb
location_name
lat
lon
this table stores all the stops that 9 lines pass, and the stops are treated as regular stops
this table stores all stops in order for one direction of one particular line
timetable
pk, fk line_id
pk, fk dir_id
pk run_id
pk, fk stop_id
pk date_mel
time_utc
time_mel
table ‘direction’, ‘line’, stops’ contain the information retrieved directly from PTV.
Data in ‘line’ and ‘stops’ are not changed very frequently.
Data in ’direction’ are changed in every one to two weeks
stopsOptOfLine
pk, fk line_id
pk, fk stop_id
this table stores all the optional stops. Currently, all the optional stops are added manually inside the database
stopsOpt
pk stop_id
location_name
lat
lon

6.2 Code Structure



Hopefully, this simple informal diagram can let you understand the code structure of this project easily. General structure is not changed too much since this project is passed to us. However, the logic to deal with data is now to use the database instead a file system. By doing so, we re-wrote almost all the php files, and have three new js files.

Some CSS info:
The tables, the buttons, grids layout etc are styled by using pure css. For more information: http://purecss.io
The font icon is used from Font Awesome. For more information: http://fortawesome.github.io/Font-Awesome/