ELEN4009 SOFTWARE ENGINEERING LABORATORY 3

Danielle Winter (563795), Frederick Nieuwoudt (386372), Stephen Friedman (360938), Sello Molele (0604606x)

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

1. PROJECT DESCRIPTION

The WitsCABS company owns a fleet of taxi-cabs and employs drivers for customer transportation within a city. The city is demarcated into service zones with a single taxi-stand in each service zone. Cab drivers will wait for clients at their closest taxi-stand. Each driver has a smartphone, embedded with a GPS, maps and navigational tracking and access to the WitsCabs web application.

A centralised Dispatch Control Centre (DCC) receives call from customers. Customers specify their name, current location, destination and contact number. The DCC agent has access to their own interface on the WitsCABS application and can enter new customer jobs into the system. The back-end calculates distance and cost for the job and assigns an available driver to the task. At this point the driver receives a notification of a new client and will proceed to collect and deliver the client. The driver tracking is monitored by the back-end application and an SMS notification is sent to the client when the driver is close by. When the job is completed, the driver will return to their taxi-stand.

The front-end component of the system has a single web application with two separate interfaces for the driver and the DCC respectively. An additional feature is the ability for the different users to log in and access their own interface. The driver interface gives information to the driver about current jobs (customer details) and has buttons for the driver to confirm when they have left to collect the client and when the task is complete.

The DCC interface has two panes - one for entering in new customers and one for viewing active jobs. The first pane comprises a form for the DCC agent to enter customer details. When they click the submit button, the entered information is sent to the back-end for distance and price calculations as well as driver assignment. The list of current jobs on the second pane in the DCC interface is then updated. When a driver completes a job, the job is removed from the active list.

2. GROUP RESPONSIBILITIES

2.1 Front-end

The front-end views for the driver and DCC interfaces were coded by Danielle Winter and Frederick Nieuwoudt.A pair programming method was used for implementing this system.

Frederick Nieuwoudt (Scrum Master):

- Driver interface implementation
- Login/log-off
- Commenting DCC code
- System aesthetics
- System modifications

Danielle Winter:

- Driver interface commenting
- Login/log-off commenting
- DCC interface implementation
- System aesthetics
- System modifications

2.2 Back-end

The back-end database was coded by Stephen Friedman. Compilation of research into Apache configuration was done by Sello Molele.

Stephen Friedman:

- Database design and construction
- Research into server configuration

Sello Molele:

- Verification of research
- Compilation of research

3. USING THE SOFTWARE

3.1 Front-end

The front-end is implemented using *angular.js* and *Bootstrap3*. Included in the HTML file are URL source links for the *angular.js* library, the *Bootstrap* stylesheets and a *jQuery* plugin. This means that the user merely needs to open the HTML file, which will run if the user has internet access. No additional programs need to be installed.

For coding, the front-end framework, the text editor *Brackets* was used.

3.2 Back-end

The back-end is implemented using an *Apache* web server and a *SQLite* database on Ubuntu. In the live version of the product, both *Apache* and *SQLite* will need to be installed for the back-end to function. The demonstration illustrates the behaviour of the database through the SQL Query for all active calls that have been logged and not yet completed.

To run the query, ensure that *SQLite* is installed. This can be done by running the command *sudo apt-get sqlite3* in the linux terminal. *SQLite* is started by running *sqlite3*. The demonstration data is initialised by running *.read CreateSampleDatabase.sql*. The script is called by running *.read GetActiveJobs.sql*. Note that SQLite is incapable of performing the square root function, so squared values involving distance are returned, to be square rooted before being displayed.

For coding, *SQLiteStudio* was used.