FLEET OF TAXI SERVICE MANAGEMENT SYSTEM - SOFTWARE DESIGN SPECIFICATION

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. INTRODUCTION

1.1 Purpose of this Document

This Software Design Specification (SDS) details the design for the WitsCABS application. The back- and front-end interfaces are described separately. The front-end and back-end integration is also described. The remainder of Section 1 details the project scope and abbreviations and definitions used in the remainder of the SDS. Section 2 addresses the basic system architecture and Section 3 gives a more detailed description of how the system will be implemented.

1.2 Project Scope

The taxi fleet management system is a tool that provides a service where a user can request a taxi and the system will dispatch the closest available driver.

The system back-end will be required to store information about drivers, customers, and the current pricing scheme. The system will keep historical data regarding trips that have been taken for auditing and dispute resolution.

The system front end will be presented in the form of a web interface. This will provide a means for Dispatch Control Centre agents to input new customer's details and for drivers to manage their current job requests. The web interface will also provide a quoted price for the client.

For the purposes of this project, the entire system functionality is described in the SDS. However, only a few modules and components of the code are implemented as a prototype of the system in order to show basic functionality.

1.3 Definitions

1.3.1 Definitions

- WitsCABS: The name of the software application including front- and back-ends
- Driver: The taxi-cab drivers employed by the WitsCABS company to pick-up and deliver customers
- Customer: Clients of the company who request lifts

1.3.2 Abbreviations

- DCC: Dispatch Control Centre

2. SYSTEM ARCHITECTURE

2.1 Front-end Views

The front-end of the WitsCABS application is presented as two separate views in an HTML web-application. The front end is coded in angular.js using Twitter Bootstrap for a unified visual appearance and pop-up functionality. The respective interfaces are the DCC view and the Driver view. These views are linked through a common log-in screen view where the respective users select and log in to their interface.

2.1.1 DCC View The DCC view includes panels for a new customer form and an active job list. The "new customer form" panel has text boxes for the customer's name, current location, destination and cellphone number. A submit button allows the DCC agent to submit the entry to the back-end database, at which point, the distance and price for the job is calculated. The new job will be added to the "active jobs" panel in the DCC interface and the DCC will be able to see the price calculated for the customer.

2.1.2 Driver View The driver view allows the driver to see their assigned job, the customer name and contact details and their travel destination. In addition to this, two buttons are presented to the driver - one for "On the Way" when they leave the taxi rank and one for "Job Complete" when the customer has been delivered.

2.1.3 Log-in View The log-in page gives the user the ability to select whether they are a driver or a DCC agent and allows them to log in to their respective view. The log-in page should prompt users for a username and password and their entry should be authenticated through the back-end.

2.2 Back-end Implementation

The back-end of the WitsCABS application consists of two main parts. The part that directly interfaces with the front-end is an Apache web server which runs on Ubuntu and executes Python scripts in order to interact with the database and external APIs. The part of the back-end that stores the data is an SQLite database.

2.2.1 Web Server

The role of the web server in the WitsCABS application is primarily to act as an intermediary between the front-end and the database. It is also required to connect to external APIs in order to deliver functionality that would be too complex to implement in the application.

The three external APIs are Paym8, SMSPortal and Google Maps. The Paym8 API is used to charge customers for Trips that are taken using WitsCABS. The SMSPortal API is used to notify Customers about the status of their Driver. The Google Maps API is used to translate addresses in to GPS Coordinates, as well as route from one location to another and determine the real distance between two points.

2.2.2 Database

The role of the database in the WitsCABS application is to store data so that different users can interact with the same data without directly transmitting the data between each other and so that data can be kept for auditing and dispute resolution if need be.

For the database queries, either many simple queries could be implemented or few complex queries could be implemented. Complex queries were chosen as they take advantage of optimisation in SQL. Complex queries require more development and testing time, but once implemented and verified produce faster results than many simple queries.

3. DETAILED ARCHITECTURE AND FEATURES

3.1 Front-end Architecture

The front-end architecture is detailed below for the angular.js and HTML files. The front-end and back-end communicate through button pushes - firstly with the log-in page user authentication and then in the respective views.

3.1.1 angular.js File

- angular module - WitsCABS
- angular controller - JobController
- customer directive - jobs
- array of customers
  - customer name (string)
  - customer destination (string)
  - customer current location (string)

- customer phone number (string)
- distance (integer)
- price (integer)
- job active (boolean)
- driver assigned (boolean)

3.1.2 HTML File

When DCC agents submit a new customer in the form, the back-end database is updated and a taxi-driver is assigned to the new job. The driver interface will be updated. Drivers can state when a job is complete, at which point, the database will once again be updated and the respective views refreshed. Upon log-in details being entered, authentication is implemented through the back-end framework before users can be redirected to their respective pages.

- DCC View
  - Headings

  - Active job panel
    - active customers list
    - driver assigned (boolean)
    - customer location, destination and phone number (paragraphs under each customer)

  - New customer form panel
    - customer name field
    - customer destination field
    - customer current location field
    - customer phone number field
    - submit button

- Driver View
  - Headings

  - Assigned job panel
    - customer details
    - "On the way" button
    - "Job complete" button

- Log-in Page
  - Log-in modal
    - driver log-in button - redirects to driver interface
    - DCC log-in button - redirects to DCC interface
    - close button

3.2 Back-end Architecture

3.2.1 Web Server

The Web Server functions primarily as an intermediary between the front-end and the database. It fulfils this role through a collection of interfaces that the front-end can access. The Interfaces are as follows:

- Login: This is to check that the User attempting to log in is a valid User and check which type of User it is, if successful.
  - Inputs:
    - UserName - A string containing the User's login name
    - PasswordHash - A string containing the User's hashed password
  - Output:
    - String containing either "Driver", "Agent", or "Rejected"
- GetActiveTrips: This is to retrieve a list of all Trips that are currently active.
  - Inputs:

    * None required
  - Output:
    * An array containing Trip objects
- CreateTrip: This is to store the information about a new trip when a DCC Agent captures it.
  - Inputs:
    * StartLocation - A string containing the friendly name of the location
    * EndLocation - A string containing the friendly name of the location
    * CustomerName - A string containing the Customer's name
    * ContactNumber - A string containing the Customer's cell phone number
    * CreditCardNumber - A string containing the Customer's credit card number
    * CreditCardType - A string containing the Customer's credit card type ("Visa" or "Mastercard" expected)
    * CreditCardCvv - A string containing the Customer's credit card CVV number
  - Output:
    * A boolean indicating if the Trip was created successfully or not
- GetDriverTrip:
  - Inputs:
    * DriverId - an integer indicating the system's internal id of the Driver
  - Output:
    * A Trip object
- SendNotification:
  - Inputs:
    * TripId - an integer indicating the system's internal id of the Trip
  - Output:
    * A boolean indicating if the Customer was notified successfully or not
- SetTripComplete:
  - Inputs:
    * TripId - an integer indicating the system's internal id of the Trip
  - Output:
    * A boolean indicating if the Trip was successfully marked as completed or not

3.2.2 Database

The database consists of the following tables:

- Users: This table contains information to verify a User when they log in. The users' username and hashed password are stored. The hash of the password is stored so that nobody that has access to the WitsCABS database can compromise the users' passwords.
- Agents: This table contains the real name of DCC agents and is used by the login process to determine if the user is a DCC agent.
- Drivers: This table contains the real name of Drivers and is used by the login process to determine if the user is a Driver. This table also keeps track of the status and location of the Driver so that the best driver is chosen for each Trip.
- Cost: This table contains rates that can apply to trips.
- Customers: This table contains details for Customers, including contact information for SMS notification and financial information in order to charge Customers for Trips that are taken.
- Locations: This table stores addresses against GPS coordinates, so that friendly names can used for informative purposes, while accurate GPS coordinates can be used for distance calculations.
- Trips: This table contains information relating to each Trip undertaken. It allows DCC agents to view all active Trips, it allows Drivers to be easily assigned to Trips and it allows for the retention of historical data for auditing or dispute resolution purposes.

3.2.3 Detailed Database Table Structure

Users:

Field	Type	Notes
UserId	Int	Primary Key, Auto-increment
UserName	Text	Login Name of User
PasswordHash	Text	Password, hashed to prevent unauthorised use

Agents:

Field	Type	Notes
AgentId	Int	Primary Key, Auto-increment
UserId	Int	Foreign Key
Name	Text	Real name of DCC agent

Drivers:

Field	Type	Notes
DriverId	Int	Primary Key, Auto-increment
UserId	Int	Foreign Key
Name	Text	Real name of Driver
CurrentLocationId	Int	Foreign Key
CurrentlyActive	Bit	Indicates if a driver is at work or not
CurrentlyAssigned	Bit	Indicates if driver is busy with a Trip

Cost:

Field	Type	Notes
CostId	Int	Primary Key, Auto-increment
Type	Text	Friendly description of Cost
AmountInCents	Int	Money is indicated to 2 decimal places, so to avoid rounding issues, work with money as an integer number of cents and divide by 100 before displaying in order to show an amount in Rands

Customers:

Field	Type	Notes
CustomerId	Int	Primary Key, Auto-increment
Name	Text	Real name of Customer
PhoneNumber	Text	Contact number of Customer
CreditCardNumber	Text	Financial information of Customer
CreditCardType	Text	Financial information of Customer
CVV	Text	Financial information of Customer

Location:

Field	Type	Notes
LocationId	Int	Primary Key, Auto-increment
LocationName	Text	Friendly name of the Location
GPSLat	Int	GPS Latitude, stored in seconds
GPSLon	Int	GPS Longitude, stored in seconds

Trip:

Field	Type	Notes
TripId	Int	Primary Key, Auto-increment
CustomerId	Int	Foreign Key
StartLocationId	Int	Foreign Key
EndLocationId	Int	Foreign Key
DriverId	Int	Foreign Key
CostId	Int	Foreign Key
Active	Bit	Indicates if the Trip is under-way or if it has been completed