University of the Witwatersrand

School of Electrical and Information Engineering

Private Bag 3, 2050, Johannesburg, South Africa

Taxi-cab Service Management System

Front-End:	*Student Number:*
Danielle Winter	563795
Frederick Nieuwoudt	386372
Back-End:	
Stephen Friedman	360938
Sello Molele	0604606X

11 April 2016

Abstract

Contents

1 INTRODUCTION

1.1 Project Overview

A software management system for a taxi company WitsCABS is designed and a basic prototype of the system has been implemented. Sections 2 and 3 address the front-end and back-end frameworks respectively with details for the Software Requirements Specification (SRS) and Software Design Specification (SDS) given for each section. In addition to this, the sections detail how the work was split into sprints according to an Agile, SCRUM framework and how the prototype systems were implemented.

1.2 Problem Statement

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 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 calls from customers and processes the new requests. WitsCABS requires the software application to organise their taxi-fleet, coordinate drivers and manage new job requests.

1.3 Project Objectives

- Create the software requirements specification
- Create the design documentation for front and back ends
- Sprint planning and retrospective for front and back ends
- Implementation of prototype modules for the front end interfaces
- Implementation of prototype back-end modules
- Implementation of an Agile management technique

1.4 Stakeholders

The project stakeholders are divided into Users, Developers, the Project Manager and the Scrum Master. The respective stakeholders and their roles are documented below.

1.4.1 Users

Users are categorised into:

- Drivers
- DCC agents
- Customers

Drivers operate the taxi-cabs, are assigned jobs and pick up and deliver customers. They have access to a driver-specific front-end interface of the WitsCABS application and are able to specify when they accept a job or are on their way as well as state when they have completed a job. Drivers have access to the WitsCABS application through a smart-phone. They also have access to GPS and map navigation and are under navigational tracking.

DCC agents operate the WitsCABS call centre and have computer access to a DCC-specific interface of the WitsCABS application. They have the ability to add new clients to the database and can view active jobs.

Customers phone in to the DCC and provide their details, current location and destination. They are assigned a driver, told a likely departure time and given a cost for their journey.

1.4.2 Developers

Developers of the WitsCABS application are responsible for the front and back end implementation as well as documentation for the system. The developers are:

- Frederick Nieuwoudt (front-end)

- Danielle Winter (front-end)
- Stephen Friedman (back-end)
- Sello Molele (back-end)

Frederick Nieuwoudt and Danielle Winter implemented and documented the prototype front end for the system using *angular.js* and *Twitter Bootstrap*. A pair programming approach was used for the implementation.

Stephen Friedman implemented the back end prototype framework and documentation. Sello Molele contributed to the documentation.

1.4.3 Project Manager

The project manager for the WitsCABS project is Danielle Winter. Their responsibilities include organising the development team and managing and integrating the documentation for the project.

1.4.4 Scrum Master

The Scrum Master for the project is Frederick Nieuwoudt. Their responsibilities include managing the sprint backlog and sprint retrospective.

2 PART A: FRONT-END

2.1 SRS

The software requirements specification quantifies the scope and success criteria for this project.

The required elements of the front-end of the project as discussed in the SRS that were successfully implemented into the demonstrable prototype include:

- The log-in page
- The driver active job view
- The dispatch agent data entry form
- The dispatch agent active jobs view

With the inclusion of these modules into the prototype allows for the demonstration of the basic work-flow set out for the users of the application.

The modules that have yet to be implemented in future sprints that have been laid out in the SRS include:

- Adding GPS functionality to the application

The major downfalls according to the SRS was the lack of test driven development. This was specified in the SRS as the testing plan to be followed.

2.2 SDS

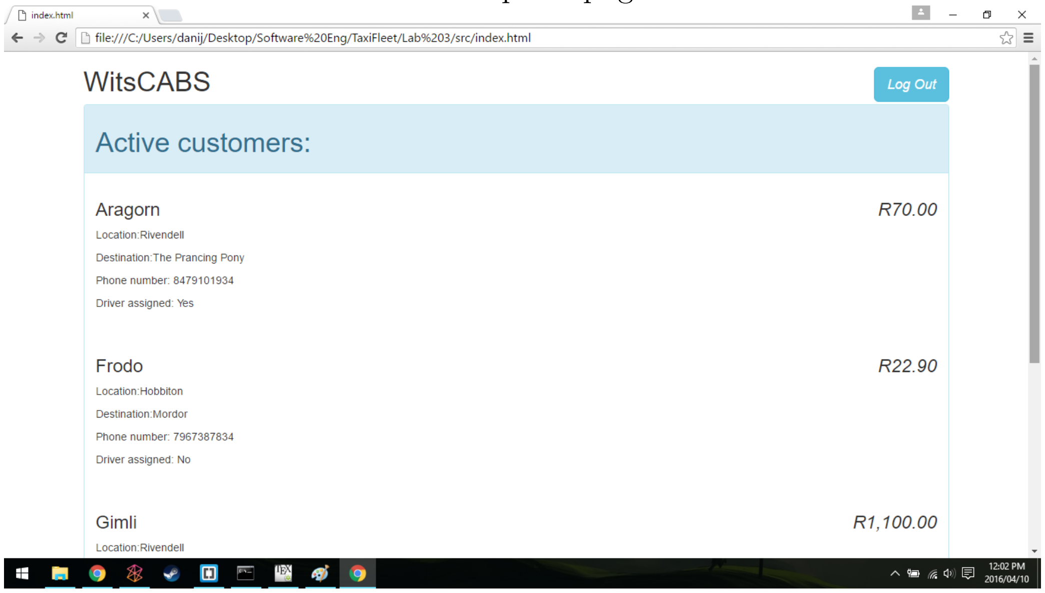
The Software Design Specification document details the different components of the front-end system and gives the relevant modules, controllers, directives and expressions used for the js file. The required components for the HTML code are described. All required data types are stated for mentioned variables.

The SDS document describes the base functionality of the system, but does not elaborate on any other features to be added at a later date. For example the automatic navigation update in the driver panel has not been elaborated upon. This is due to time constraints in implementing the project and the fact that these additional features were added at a later date. For future work, the SDS must be updated to include these modules.

2.3 Prototype Implementation

The basic functionality of the described system was implemented in angular.js and html code. The respective views for the different users of the product were created. Figure 1 and 2 show the DCC view for the system and Figure 3 shows the driver view. The log-in modal view is shown in Figure 4. These views were coded and implemented according to the requirements given in the SDS and SRS. The log-in modal was added as an additional feature for the system. These views satisfactorily represent the front-end product for the WitsCABS application and are a fair indicator of the final system functionality.

active panel.png


index.html
file:///C:/Users/danij/Desktop/Software%20Eng/TaxiFleet/Lab%203/src/index.html
Log Out
12:02 PM
2016/04/10

WitsCABS

Active customers:

Aragorn R70.00
Location:Rivendell
Destination:The Prancing Pony
Phone number: 8479101934
Driver assigned: Yes

Frodo R22.90
Location:Hobbiton
Destination:Mordor
Phone number: 7967387834
Driver assigned: No

Gimli R1,100.00
Location:Rivendell

Figure 1: DCC view with active customers panel.

The front-end is implemented using angular.js and Bootstrap3. Included customer form.png

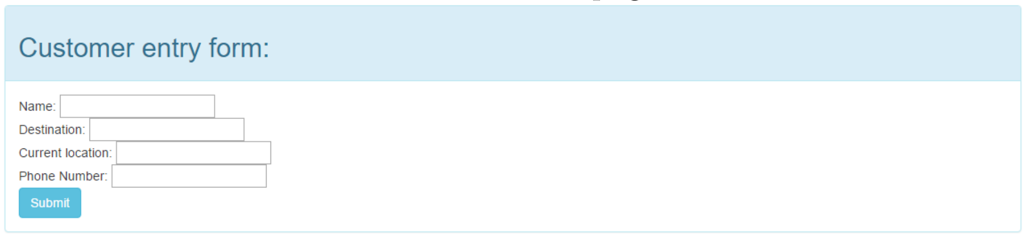
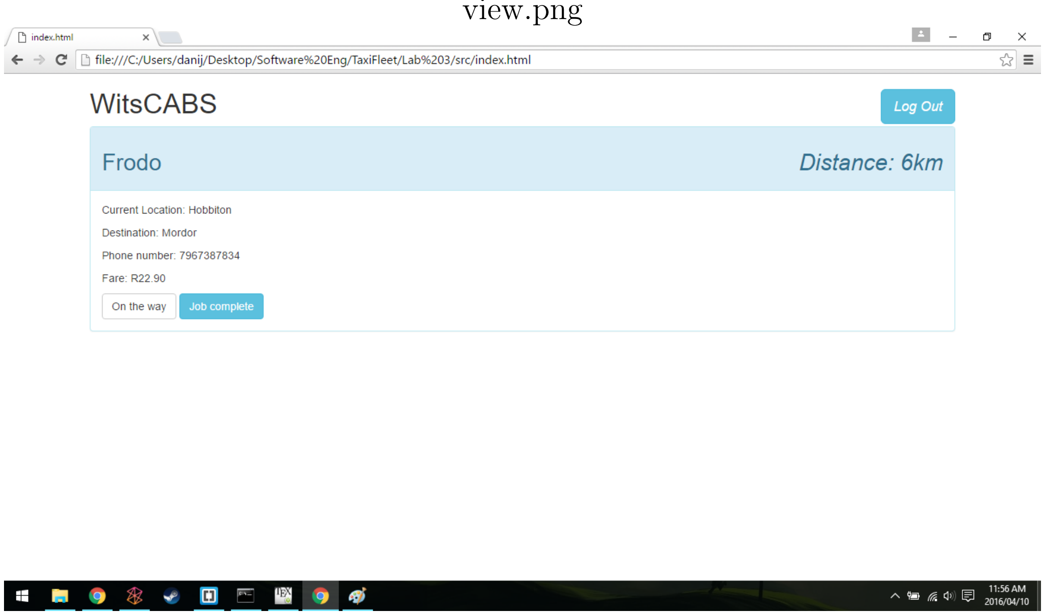


Figure 2: DCC view with new customer form.

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. 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. The work breakdown for the prototype implementation is given below:

- Frederick Nieuwoudt (Scrum Master):
  - Driver interface implementation
  - Login/log-off
  - Commenting DCC code
  - System aesthetics
  - System modifications
- Danielle Winter (project manager):
  - Driver interface commenting
  - Login/log-off commenting
  - DCC interface implementation
  - System aesthetics
  - System modifications

view.png


index.html
file:///C:/Users/danij/Desktop/Software%20Eng/TaxiFleet/Lab%203/src/index.html
WitsCABS
Log Out
On the way
Job complete
11:56 AM
2016/04/10

Frodo

Distance: 6km

Current Location: Hobbiton

Destination: Mordor

Phone number: 7967387834

Fare: R22.90

Figure 3: Driver interface view.

2.4 Product Backlog

Interface	Features	Expected Time (Days)
DCC	Active jobs panel	5
	Customer form panel	5
	Implement tabbed view (rather than panels)	1
	Automatic address completion using Google Maps API	5
Driver	Assigned job panel	5
	Buttons for "On the way" and "Job complete"	1
	Automatic update of navigation	5
General	Log-in page	5
	Log-in authentication	5
	Create basic interface framework	1
	Set up AngularJS customer directive	2
	Review user experience design (aesthetics using CSS)	5
	Set up google maps API	5

screen and pop up.png

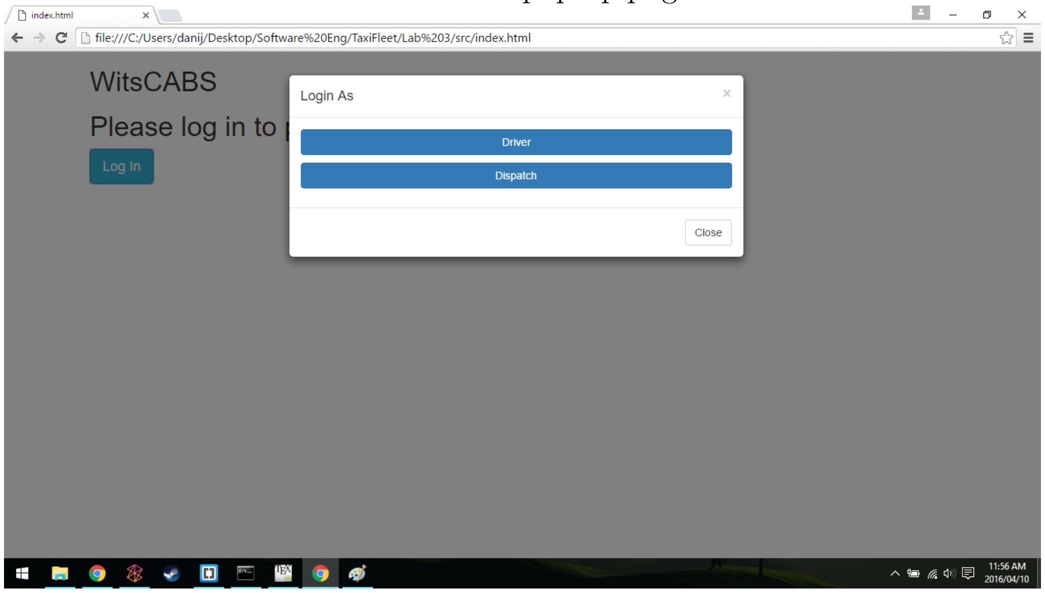


Figure 4: Log-in view with pop-up modal.

2.5 Sprint Planning

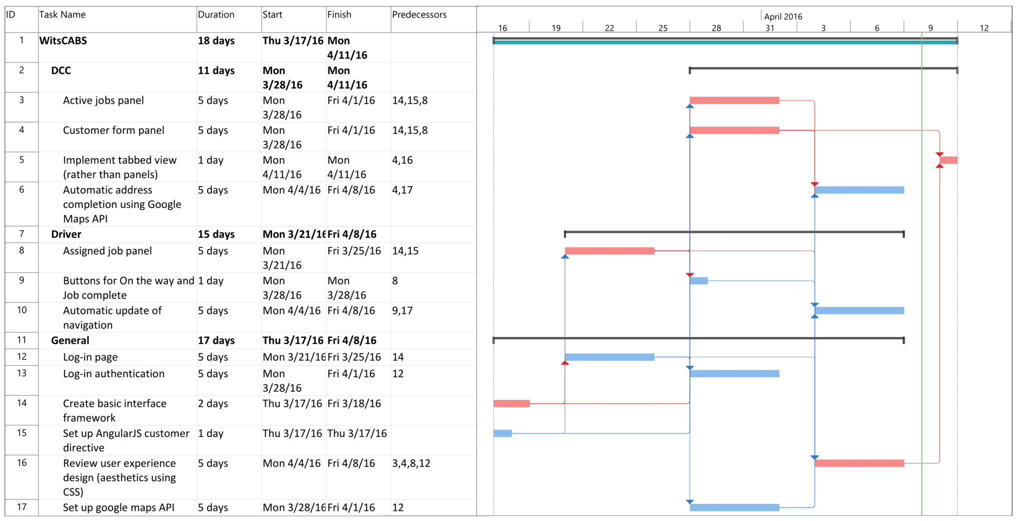
Sprint planning was undertaken to divide tasks in such a way that the most efficient path was taken.

In order to achieve this the task back log was represented on a gantt chart in order to visualize the product dependencies and which tasks could happen in parallel. The produced gantt chart is available in the figure 5 below.

The tasks were divided according to whether they were on the critical path or not. Creating the views for specific users was the most important for the success of the prototype. Tasks such and login authentication and the addition of Google maps functionality were not critical to the creation of a prototype that can visually demonstrate the final product.

2.6 Sprint Retrospective

Three sprints were completed to produce the presented front-end prototype. The features implemented in each sprint has been cataloged in the table be-



ID	Task Name	Duration	Start	Finish	Predecessors
1	**WitsCABS**	**18 days**	**Thu 3/17/16**	**Mon 4/11/16**	
2	**DCC**	**11 days**	**Mon 3/28/16**	**Mon 4/11/16**	
3	Active jobs panel	5 days	Mon 3/28/16	Fri 4/1/16	14,15,8
4	Customer form panel	5 days	Mon 3/28/16	Fri 4/1/16	14,15,8
5	Implement tabbed view (rather than panels)	1 day	Mon 4/11/16	Mon 4/11/16	4,16
6	Automatic address completion using Google Maps API	5 days	Mon 4/4/16	Fri 4/8/16	4,17
7	**Driver**	**15 days**	**Mon 3/21/16**	**Fri 4/8/16**	
8	Assigned job panel	5 days	Mon 3/21/16	Fri 3/25/16	14,15
9	Buttons for On the way and Job complete	1 day	Mon 3/28/16	Mon 3/28/16	8
10	Automatic update of navigation	5 days	Mon 4/4/16	Fri 4/8/16	9,17
11	**General**	**17 days**	**Thu 3/17/16**	**Fri 4/8/16**	
12	Log-in page	5 days	Mon 3/21/16	Fri 3/25/16	14
13	Log-in authentication	5 days	Mon 3/28/16	Fri 4/1/16	12
14	Create basic interface framework	2 days	Thu 3/17/16	Fri 3/18/16	
15	Set up AngularJS customer directive	1 day	Thu 3/17/16	Thu 3/17/16	
16	Review user experience design (aesthetics using CSS)	5 days	Mon 4/4/16	Fri 4/8/16	3,4,8,12
17	Set up google maps API	5 days	Mon 3/28/16	Fri 4/1/16	12

Figure 5: Gantt chart showing all required tasks and their dependencies.

low.

Sprint	Features
1	Create basic interface framework Set up AngularJS customer directive
2	Driver assigned job panel Create default login page Create login modal
3	Add DCC active jobs panel Add DCC customer form panel

3 PART B: BACK-END

3.1 SRS

The Software Requirements Specification (SRS) defines the scope and success criteria of the WitsCABS application.

The SRS required the following functionality to be present in the WitsCABS application:

- DCC Agents must be able to view all active Trips
- DCC Agents must be able to capture Trips
- Drivers must be able to view their allocated Trip
- Drivers must be able to mark Trips as completed

The prototype produced for the back-end demonstrates a database capable of storing the information required for the above functionality. It does, however, only contain a script for demonstrating the extracting of information to view all Trips.

Further work required is to create scripts capable of extracting or writing any other information required as well as interfacing those scripts with the front-end.

The nature of the implementation that was lacking was that it was not test driven. This was specified in the SRS as the testing plan to be followed. Had it been, the script would have been created and the database created to fit the tests.

3.2 SDS

The Software Design Specification (SDS) document details the different parts of the back-end that are required, namely the Web Server for interfacing with the front-end and the Database for storing data.

Http interfaces are required for the front-end to interact with the Web Server. These are accessed through GET and POST calls from the front-end.

The required inputs and the supplied output of each call is defined in the SDS.

Once created, databases are difficult to alter without compromising the integrity of the data contained inside its tables, so a strict and robust definition of the database is required. The nature of each table, as well as the structure of each table is defined in the SDS.

The SDS lacks clear definition on how third party APIs will be accessed. This is due to time constraints in implementing the project and the fact that these additional features were added at a later date. For future work, the SDS must be updated to include these modules.

3.3 Prototype Implementation

The core of the back-end is the database, so it was decided that the demonstration should show the database structure and should contain at least one query. The chosen query was the GetActiveTrips query, as it is the single most complex query required due to the query requiring data from almost all tables.

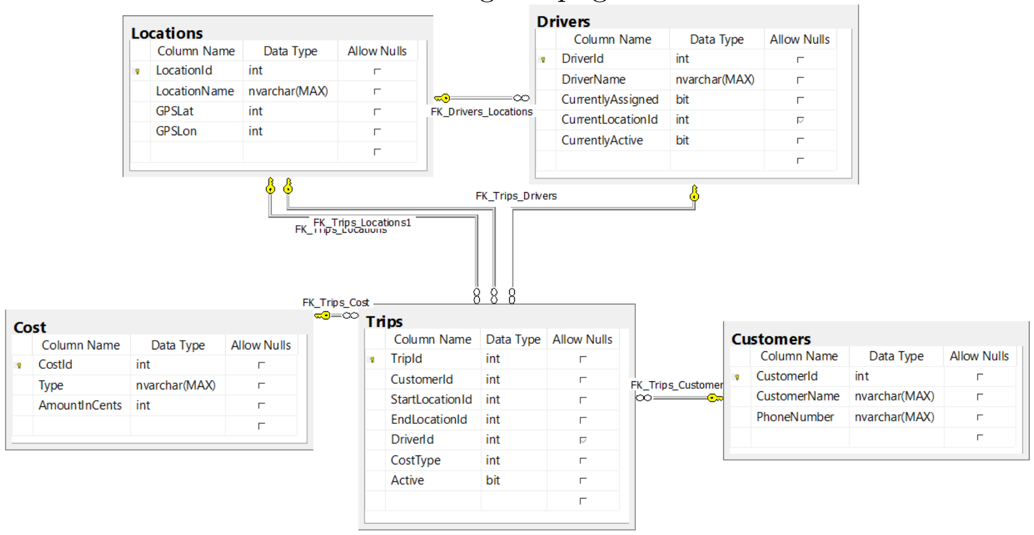
The database structure was created by using database diagrams in Microsoft SQL Management studio, a graphical interface that the developer had previous experience in using. This generated an MSSQL database, which was populated with sample data. A tool called SQLConverter was used to convert the MSSQL database into an SQLite database, from which a database creation script was generated. The database is shown in Figure 6.

The SQL Query was created in gedit and was designed to illustrate that the data needed to display the DCC Agent view could be retrieved from the database. As the database alone cannot query the Google Maps API, a crude x-y coordinate system was implemented to show that the database is capable of doing calculations. One limitation of SQLite is that is cannot perform square root calculations, so the calculations are left at a point where further maths could be done.

In order to run the demonstration, the user must ensure that SQLite is installed. This is done in Ubuntu by running the following in the terminal:

```
sudo apt-get sqlite3
```

Diagram.png



Locations

	Column Name	Data Type	Allow Nulls
🔑	LocationId	int	☐
	LocationName	nvarchar(MAX)	☐
	GPSLat	int	☐
	GPSLon	int	☐
			☐

Drivers

	Column Name	Data Type	Allow Nulls
🔑	DriverId	int	☐
	DriverName	nvarchar(MAX)	☐
	CurrentlyAssigned	bit	☐
	CurrentLocationId	int	✓
	CurrentlyActive	bit	☐
			☐

Cost

	Column Name	Data Type	Allow Nulls
🔑	CostId	int	☐
	Type	nvarchar(MAX)	☐
	AmountInCents	int	☐
			☐

Trips

	Column Name	Data Type	Allow Nulls
🔑	TripId	int	☐
	CustomerId	int	☐
	StartLocationId	int	☐
	EndLocationId	int	☐
	DriverId	int	✓
	CostType	int	☐
	Active	bit	☐
			☐

Customers

	Column Name	Data Type	Allow Nulls
🔑	CustomerId	int	☐
	CustomerName	nvarchar(MAX)	☐
	PhoneNumber	nvarchar(MAX)	☐
			☐

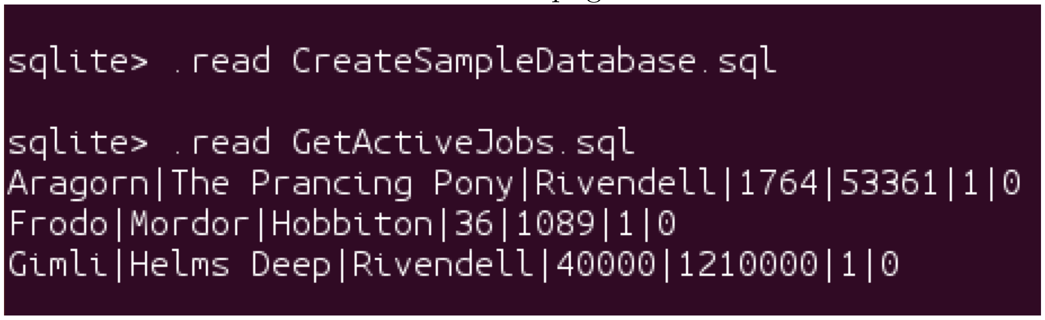
Figure 6: Database diagram of the prototype.

In order to import the database and view the demonstration output, the following must be run in the terminal:

```
sqlite3
.read CreateSampleDatabase.sql
.read GetActiveJobs.sql
```

The results of the above query are shown in Figure 7. The demonstration code was created and tested by Stephen Friedman.

Results.png



```
sqlite> .read CreateSampleDatabase.sql

sqlite> .read GetActiveJobs.sql
Aragorn|The Prancing Pony|Rivendell|1764|53361|1|0
Frodo|Mordor|Hobbiton|36|1089|1|0
Gimli|Helms Deep|Rivendell|40000|1210000|1|0
```

Figure 7: Database prototype query results.

3.4 Product Backlog

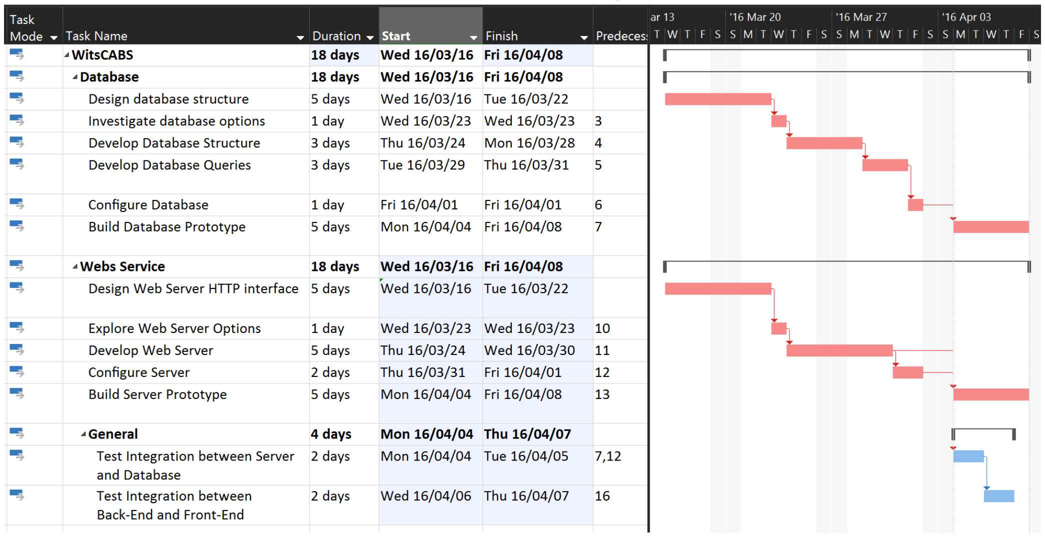
Component	Task	Expected Time (Days)
Database	Design database structure	5
	Investigate database options	1
	Develop Database Structure	3
	Develop Database Queries	3
	Configure Database	1
	Build Database Prototype	5
Web Server	Design Web Server HTTP interface	5
	Explore Web Server Options	1
	Develop Web Server	5
	Configure Server	2
	Build Server Prototype	5
General	Test Integration between Server and Database	2
	Test Integration between Back-End and Front-End	2

3.5 Sprint Planning

Sprint planning was done to ensure that tasks would be divided between the available three sprints. The tasks were placed on Gantt chart in order to figure out which tasks were critical and how tasks could be split into three sprints. The split is shown in Figure 8.

As both the Web Server and Database tasks are critical, and each task takes 18 days to complete, with two two hour testing tasks. The tasks were split as close to six hours each per sprint as possible.

End Gantt.png



Task Mode	Task Name	Duration	Start	Finish	Predecessors
	WitsCABS	**18 days**	**Wed 16/03/16**	**Fri 16/04/08**	
	Database	**18 days**	**Wed 16/03/16**	**Fri 16/04/08**	
	Design database structure	5 days	Wed 16/03/16	Tue 16/03/22	
	Investigate database options	1 day	Wed 16/03/23	Wed 16/03/23	3
	Develop Database Structure	3 days	Thu 16/03/24	Mon 16/03/28	4
	Develop Database Queries	3 days	Tue 16/03/29	Thu 16/03/31	5
	Configure Database	1 day	Fri 16/04/01	Fri 16/04/01	6
	Build Database Prototype	5 days	Mon 16/04/04	Fri 16/04/08	7
	Webs Service	**18 days**	**Wed 16/03/16**	**Fri 16/04/08**	
	Design Web Server HTTP interface	5 days	Wed 16/03/16	Tue 16/03/22	
	Explore Web Server Options	1 day	Wed 16/03/23	Wed 16/03/23	10
	Develop Web Server	5 days	Thu 16/03/24	Wed 16/03/30	11
	Configure Server	2 days	Thu 16/03/31	Fri 16/04/01	12
	Build Server Prototype	5 days	Mon 16/04/04	Fri 16/04/08	13
	General	**4 days**	**Mon 16/04/04**	**Thu 16/04/07**	
	Test Integration between Server and Database	2 days	Mon 16/04/04	Tue 16/04/05	7,12
	Test Integration between Back-End and Front-End	2 days	Wed 16/04/06	Thu 16/04/07	16

Figure 8: Gantt chart showing the Product Backlog. Critical Path tasks are in red.

3.6 Sprint Retrospective

Three sprints were completed, producing a back-end prototype. The split of which tasks were accomplished in each sprint are shown in the table below.

Sprint	Tasks
1	Design database structure Investigate database options
2	Develop Database Structure Develop Database Queries
3	Configure Database Build Database Prototype