CrowdSourcery: A Ubiquitous Crowdsourcing Route Planner

Daniel Coghlan
B.A.(Mod.) Business and Computing
Final Year Project April 2013
Supervisor: Dr Jonathan Dukes

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
# TABLE OF CONTENTS

**Chapter 1 - Introduction**  
1.1 General Introduction  
1.2 Project Objectives  
1.3 Readers Guide  

**Chapter 2 - Background**  
2.1 Chapter Introduction  
2.2 Introduction to Crowdsourcing  
2.2.1 – Definition and Background  
2.3 Existing Route Planning Solutions  
2.3.1 – Passive GPS Crowdsourcing  
2.3.2 – Ubiquitous Crowdsourcing  
2.3.3 – Induction Loops Vehicle Detection Systems  
2.4 Scope for Crowdsourcing in Route Planning  
2.5 Potential Problems of Crowdsourcing  
2.5.1 – Data Integrity  
2.5.2 – Real-Time Information Issues  
2.5.3 – Sparse of Excessive Information Issues  
2.6 Chapter Summary  

**Chapter 3 – Solution Design**  
3.1 Chapter Introduction  
3.2 Design Research  
3.3 Abstract Design Solution  
3.3.1 Information Types to Crowdsource  
3.3.2 Crowdsourcing Information System Model Decision  
3.4 Solution Design Features  
3.4.1 – A to B Optimal Route Calculator  
3.4.2 – Quality Control Filter  
3.4.3 – Interactive Mapping  
3.5 Performance Considerations  
3.5.1 – Scalability  
3.5.2 – Data Storage Model  
3.5.3 – Query Processing  
3.5.4 – Information Correctness  
3.6 Chapter Summary  

**Chapter 4 – Technology Stack**  
4.1 Chapter Introduction  
4.2 Languages  
4.2.1 – XML  
4.2.2 – Java  
4.2.3 – PHP
Acknowledgements

I would firstly like to thank my supervisor Jonathan Dukes for his suggestions, guidance, and support which was of great help throughout the year. I would also like to express my gratitude to those of you who assisted in the testing process.
Chapter 1 – Introduction

1.1 General Introduction

Urban traffic and the difficulties of commuting are among the most intractable problems that face cities across the world and are a major source of inefficiency, wasted fuel, and commuter frustration [7] [21]. With over 2.5 million cars now registered on Irish roads [8], such delays are becoming an increasing problem for Irish road users, particularly around the Dublin City Centre. Measuring these delays, and routing road users around them, is a crucial step towards improving the overall route planning process. While a significant number of applications have attempted to tackle this issue, many rely on hardware-intensive techniques such as motion sensors, GPS, and visual data from cameras [7]. In contrast to such approaches, this project proposes an alternative solution based on the use of smartphone technology and the notions of user interconnectivity and collective knowledge.

1.2 Project Objectives

The primary goal of this project is to develop an Android application that facilitates the collection and use of *user-driven or crowdsourced* information to improve the overall route planning process. Users will be provided with a platform to share their travel experiences in order to enhance the process of identifying and re-routing users around those congested or undesirable driving routes.

Ultimately, this project has three objective levels; (i) to allow users to *report* different kinds of homogeneous information regarding travel routes, (ii) to enable users to *view* information relating to travel routes, and (iii) to *use* this crowdsourced information as effectively as possible to calculate optimal routes for users. The real challenges emerge when addressing the third objective level; designing and implementing intelligent features which make use of the processed information in ways that are of value to the user. Without this crucial element, the application would essentially have no practical use.

The aim is to structure the application around a specific form of crowdsourcing known as *'ubiquitous crowdsourcing'*; in which contributed information is not limited to passively-generated sensor-readings from the device, but also includes proactively-generated users' opinions and perspectives, that are processed to offer real-time services to users [5]. Therefore, the application will deal with *dynamic and real-time information*, meaning that data integrity and timing issues are among the most important considerations of the project. Furthermore, rather than being limited solely to displaying information textually, the application should also include interactive mapping capabilities so as to illustrate as much useful information as possible to users through as many different platforms as possible. Back-
end objectives include implementing a data storage technology that is both efficient and scalable due to the large volumes of data that could potentially be handled by the application. It should ideally follow a RESTful approach to maximise interoperability, security, and above all else, efficiency.

Secondary objectives include conforming to the 'Best Practices' Android guide [18] to ensure optimal performance and designing a visually appealing user interface (UI).

Figure 1 – The Three Core Project Objectives:

1.3 Readers Guide

Chapter 2 – Background:

Chapter 2 explores the paradigm of crowdsourcing and how it can be deployed as a powerful problem-solving mechanism, particularly in the area of route planning.

Chapter 3 – Solution Design:

Chapter 3 provides a high-level analysis of the proposed solution design for the project. It first presents background re

search to existing design models for crowdsourcing information systems and then subsequently uses the findings of this research to outline a comprehensive solution design based on the objectives outlined in chapter 1.

Chapter 4 – Technology Stack:

Chapter 4 briefly introduces and describes the various programming languages and technologies used during the implementation process to provide the reader with a better understanding of the Technical Implementation chapter.

Chapter 5 – Technical Implementation:

Chapter 5 is the most detailed chapter, providing a low-level analysis of the steps taken to implement the various features of the project.
**Chapter 6 – Evaluation:**

Chapter 6 explores the testing framework that was used to evaluate the overall performance of the application. The observations and results are presented to the reader and are combined to derive a performance evaluation.

**Chapter 7 – Conclusion:**

In chapter 7, the author provides a personal conclusion to the project; what areas were successful and what problems were encountered. Planned future work relating to the application is also outlined and discussed in detail.
Chapter 2 – Background

2.1 Chapter Introduction

This chapter explores the paradigm of crowdsourcing. It firstly examines the underlying logic and principles of the concept, and then demonstrates how it can be deployed as a powerful problem-solving mechanism, particularly in the area of route planning.

2.2 Introduction to Crowdsourcing

2.2.1 Definition and Background:

"No one knows everything, everyone knows something, and all knowledge resides in humanity...new communications systems should provide members of a community with the means to coordinate their interactions within the same verbal universe of knowledge" - Pierre Lévy [1].

Crowdsourcing is an abstract term for a variety of approaches that harness the potential of large networks of people by issuing open calls for contribution to particular problems or tasks [2]. While no universally accepted definition of the concept currently exists, most usually converge around the notions of collective intelligence, public participation, collaborative problem-solving, and user-generated content (UGC) [3]. The underlying philosophy is that people rather than technology are the source of information and knowledge, and that people alone possess the ability to transform and use this knowledge. Technology is merely a function used to enable, and where possible, support such activities.

Crowdsourcing is not a new phenomenon, examples of such approaches can be identified in various shapes and forms throughout the centuries [4]. However, with the large-scale improvements in information technologies in recent decades, particularly with the emergence of web 2.0, the potential to contribute, harness, and diffuse collective intelligence and knowledge has dramatically increased; leading to the significant growth in popularity of web-based crowdsourcing information systems (for example Wikipedia, iMDb, TripAdvisor, Boards.ie etc). Employing information technology as a facilitator, crowdsourcing information systems are capable of processing potentially vast quantities of contributions from users in a fast, wide-reaching, and interactive manner [2]. It accommodates for a universal knowledge-base, rather than being restricted solely to the wisdom and opinions of experts or the readings of technology devices [1]. This provides a wide spectrum of alternative perspectives that can often diverge from the results of technology-based techniques, leading to unique and more comprehensive solutions.
2.3 Existing Route Planning Solutions

The real-time route planning process can be defined as follows: given the latest information about road delays and traffic congestion in a given area and a route query in the form of a (source, destination) pair, find the optimal route from the given source point to the given destination point that is estimated to have the shortest travel time at that moment in time [10]. The various techniques and technologies currently used to determine these optimal routes are described below.

2.3.1 Passive GPS Crowdsourcing:

Google Directions provides one of the most widely used route planning services on the internet. It determines the optimal route between two given locations by considering a range of different factors such as geographic distance, average travelling speeds, and most relevantly, real-time traffic data. Interestingly, rather than using road sensors, Google collects this real-time traffic data using a form of passive crowdsourcing [9]. Any smart phone with Google Maps installed passively transmits the user's current location to the Google server via GPS, enabling them to determine the exact road that user is travelling on as well as the speed they are travelling at. When they combine this information with the information provided by other road users, across thousands of phones moving around a city at any given time, they can compute an accurate overview of live traffic conditions. This data is then processed and subsequently fed back to the directions service during the route calculating process in order to determine the shortest path.

Many other applications also use this approach of passively crowdsourcing GPS information to determine real-time route conditions. Waze, for example, is now the world's fastest growing community-based traffic and navigation application with over 30 million users.
world-wide [19]. Similar to Google Directions, Waze passively collects user travel information via GPS to build up a comprehensive database of real-time road data. However, unlike Google Directions, Waze also empowers users to proactively submit real-time road reports through ubiquitous crowdsourcing.

Figure 2.2:

2.3.2 Ubiquitous Crowdsourcing:

An increasing number of applications are now adopting the concept of ubiquitous crowdsourcing whereby, as previously mentioned, users are allowed to proactively submit real-time road reports in contrast to simply relying on passively collected GPS information. Waze is again a prime example of this approach where users can report situation-specific road information such as road closures, areas of heavy traffic, police checkpoints etc. In combining these two different crowdsourcing approaches, both ubiquitous and passive, Waze has been able to develop what is believed to be one of the largest and most reliable real-time traffic databases in the world [19]. Similarly, Roadify – a widely used public transport crowdsourcing application – has applied the concept of ubiquitous crowdsourcing to collect information regarding regional transit conditions from riders.

2.3.3 Induction Loop Vehicle Detector Systems:

An Induction Loop Detector System is a form of sensor technology that offers a reliable means of detecting the presence of moving objects at a given location [10]. Many government services implement these systems as a method of calculating traffic congestion which is then included in the computation of the optimal route between two given locations. Such systems consist of three main components: (i) a loop, (ii) a loop extension cable, and (iii) a detector, and works as follows:

The loop is a continuous run of wire that enters and exits from the same point and is buried under the traffic lane. The two ends of the loop wire are connected to the loop extension cable, which in turn connects to the vehicle detector. When a vehicle moves over the loop, a
magnetic frequency is resonated. The detector senses this emitted frequency and forces a normally open relay to close, which then triggers [10].

This approach has been proven to be the most accurate traffic delay and congestion measurement technique and is primarily used by city councils all over the world. However it is extremely expensive to implement and maintain and is therefore infeasible for a project such as this.

*Figure 2.3 [10]:*

![Induction Loop Vehicle Detection Process](image)

#### 2.4 Scope for Crowdsourcing in Route-Planning

"We recognise that the collective intelligence of communities is largely untapped by traditional public participation methods, which may result in less-than-ideal transit plans that neglect the needs of diverse constituencies" [1].

As can be seen with the example of Waze and its 30 million-strong user-base, the scope for crowdsourcing in the route-planning process through the use of smartphone technology is enormous. Smartphones and user-driven applications are becoming increasingly integrated into daily city life, and information can therefore be transferred among individuals more seamlessly than ever before. By empowering drivers to share travel information, route-planning applications can provide more accurate and reliable real-time information which can be used to improve optimal route calculations. Recent studies have reinforced this view, showing that 'connected commuting' with smartphones can, in most cases, provide valuable high-quality real-time data about traffic conditions [21]. Furthermore, these studies found that by applying ubiquitous crowdsourcing techniques to the general commuting and route-planning process, overall driver sentiment can be significantly increased. By developing a social platform for drivers to voice their opinions and share their travel-related frustrations, stress levels can be reduced considerably, leading to more pleasant and enjoyable commuting experiences.
2.5 Potential Problems of Crowdsourcing

Developing an open system which relies on the contributions of an anonymous user-base poses a number of interesting problems and difficulties. While the chosen solutions are explored in Chapters 3 and 5, some of the main problems are discussed below.

2.5.1 Data Integrity:

Although crowdsourcing has indeed empowered people to share their knowledge and wisdom through a broad range of platforms, it has also made it possible for people to contribute poor quality content. This substandard content can have significantly adverse effects on the internal operations of crowdsourcing information systems, leading to misinformed or inaccurate results and reduced trust in the system [5]. For this reason, data integrity and quality assurance are of fundamental importance for crowdsourcing applications in order to filter out such inaccurate or malicious content. The problem of substandard information quality can be divided into two separate elements reflecting both the source and the nature of the content in question; (i) data-pollution and (ii) data-accuracy [5].

(i) **Data-Pollution**: Because crowdsourcing information systems rely on openness to source and obtain information, virtually anyone with network access is allowed to submit contributions. A paradox may form if this same characteristic of openness begins to threaten the success and impact on the correctness of results. The notion of data-pollution refers to those malicious users who *deliberately* compromise the quality of results by submitting forged or false data, or interfere with the data.
Regardless of the motives, there exists a clear threat model in which these users can intentionally fabricate recorded data and corrupt the sensor's data. Based on this danger, a comprehensive quality control feature is needed to verify all submitted information and therefore minimise any incentive for future threats.

(ii) Data-Accuracy: Data-accuracy is a broader issue that refers to general inaccuracy of information provided by users. User contributions are ultimately based on personal experiences, opinions, and intrinsic attitudes. Therefore, information can unintentionally be biased or misinformed leading to unreliable results. As with data-pollution, there is a need for a quality control feature to filter out any reports with substandard information quality.

2.5.2 Real-Time Information Issues:

The dynamic nature of ubiquitous crowdsourcing, particularly in the area of route planning, means that real-time events by which users provide information about – such as road delays or car parking availability – may change following the user's initial report submission. If this were to occur, the processed data could be highly inaccurate, skewing the results set towards potentially flawed and misinformed route options. Therefore, in addition to the need for an information quality control feature as previously identified, there is also a need for a feature which eliminates any timing issues by enabling users to swiftly update or remove submitted information in such situations.

To illustrate this problem, consider the following example. A user of the application reports a road accident on a certain road, estimating delays of half an hour due to the closure of all three driving lanes. The application processes this information and provides an alternative route to other users who may wish to use that particular road to reach their intended location. However, shortly after the report is submitted, authorities safely re-opened two of the three lanes therefore cutting the estimated delay to just five minutes. While the delay itself is still in effect, its adverse consequences for the route planning process have been significantly reduced. Rather than deeming the information inaccurate and automatically removing it from the system as would be with case with the use of just an information quality control feature alone, users should be capable of updating this report to reflect the circumstances of the situation as accurately as possible.

2.5.3 Sparse or excessive information:

The quantity of information provided to the application from users will have a significant bearing on both effectiveness and performance. Sparsity of contributions by small crowds is a well-known challenge in web-based crowdsourcing information systems with severe impact on content quality [5]. Similarly, excessive amounts of information can lead to scalability-related problems in the form of longer response times due to higher client-server communication costs and more information processing. Therefore, concepts such as user
incentives and application scalability must be taken into consideration in designing and developing the application.

2.6 Chapter Summary

Having provided a background to the concept of crowdsourcing and its potential role in route planning problem, the next task is to design a system actually capable of using this harnessed information.
Chapter 3 – Solution Design

3.1 Chapter Introduction

In this chapter, a solution design for the project is formalised. To provide additional context, the reader is first presented with relevant background research into viable design models for general crowdsourcing information systems. The findings of this research are then used to assist the author in outlining a comprehensive solution design for the application.

3.2 Design Research

In approaching the general solution design, Geiger et al. [2] outline a useful framework for the design and implementation of crowdsourcing information systems (CIS). According to this framework, there exists four different 'archetypes' of CIS, distinguishable by (i) the type of information contributions sought by the application (homogeneous or heterogeneous), and (ii) the values sought from these information contributions (emergent or non-emergent);

(i) A system that seeks *homogeneous* contributions values all valid user contributions equally and is therefore oriented towards quantitative information processing. In contrast, a system seeking *heterogeneous* contributions values user contributions differently according to their individual aspects and qualities and is more suited to qualitative information processing in order to determine the content of greater quality [2].

(ii) A system seeking a *non-emergent value* from its user contributions derives this value directly from some or all of the individual contributions in isolation. In such systems, an individual contribution delivers a fixed value, which is independent of other contributions. A system that seeks an *emergent* value from its user contributions, in contrast, can only derive this value from the entirety of contributions and the relationships between them. An individual contribution therefore only delivers value as part of the collection of contributions as a whole [2].

Using the above information, the four CIS archetypes can be identified as follows:

1. *Crowd Processing Systems* – Rely on large quantities of homogeneous contributions and seek non-emergent value that is derived directly from the individual contributions. All user contributions submitted to the system are deemed to contain the same quality of information and therefore provide the same value, however each contribution is considered individually in order to deliver a collective result.
2. *Crowd Rating Systems* – Rely on large quantities of homogeneous contributions but seek a collective value only from the entirety of these contributions. As the title suggests, this type of CIS is best used as a rating-based system whereby the overall rating of the given item is a function of all the collective ratings submitted by users.

3. *Crowd Solving Systems* – Seek non-emergent values that derive directly from the isolated values of their heterogeneous contributions. As the title again suggests, this type of CIS is typically used to solve more complex problems whereby each contribution has a unique solution that must be evaluated individually [2].

4. *Crowd Creation Systems* – Use heterogeneous contributions but seek a collective value that emerges only from the accumulation of contributions and their relationships. In informal terms, this form of CIS uses all information submitted to the system and relies on the qualitative characteristics of this information rather than quantitative to derive results. A practical example is Wikipedia whereby the knowledge contributed by users is used to provide a unified page of content [2].

Figure 3.1 – Four CIS Archetypes:

3.3 Abstract Solution Design

3.3.1 Information Types to Crowdsource:

In deciding on which type of crowdsourcing information system to implement for the solution design, it is first necessary to identify what kinds of information users should be allowed to contribute to the application. As outlined in Chapter 1, the primary objective of the project is to develop an application that will ultimately improve the route planning process. According to Thiagarajan et al. [7], one of the most important steps in achieving this is to measure and display road delays as accurately as possible, and to subsequently route
road users around them. Therefore, the main form of information sought by the application is that relating to *road delays*. However, rather than merely focusing on road delays alone, it was decided that the application should also incorporate complementary information regarding routes that could be of value to the user (ie nearby amenities, services etc). For the purpose of this project, a decision was made to include the two services deemed by the author to be most important to motorists in general route planning; *car park* information, and *petrol station* information.

**Figure 3.2:**

<table>
<thead>
<tr>
<th>Road Delays</th>
<th>Petrol Stations</th>
<th>Car Parks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Road Name</td>
<td>Road Name</td>
<td>Road Name</td>
</tr>
<tr>
<td>Delay Description</td>
<td>Station Name</td>
<td>Car Park Name</td>
</tr>
<tr>
<td>Delay Severity</td>
<td>Petrol Price</td>
<td>Rate Per Hour</td>
</tr>
<tr>
<td>Estimated Duration</td>
<td>Diesel Price</td>
<td>Space Availability</td>
</tr>
</tbody>
</table>

### 3.3.2 CIS Model Decision:

Linking back to the framework proposed by Geiger et al, the application will essentially seek homogenous information since users are limited to submitting generic kinds of information with equal content quality. Non-emergent value will be derived from this information as each contributed delay report is considered individually in order to calculate the quickest route for the user. Therefore, the application will be modelled on a *Crowd Processing Information System*, which as Figure illustrates 3.1, seeks homogeneous information and non-emergent information value.

**Figure 3.3 – Abstract Solution Design Logic:**

<table>
<thead>
<tr>
<th>Involvement</th>
<th>Data Wisdom</th>
<th>Contrib. Quality</th>
<th>Value Sought</th>
<th>Incentives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participatory</td>
<td>Collective</td>
<td>Homogeneous</td>
<td>Non-Emergent</td>
<td>Service</td>
</tr>
</tbody>
</table>
3.4 Solution Design Features

Having established a Crowdsourcing Information System for the application to be modelled upon, it is now possible to examine the various features included in the solution design.

3.4.1 A to B Optimal Route Calculator:

The primary feature of this application is an A to B route calculator between two given location values inputted by the user. This feature will calculate a list of viable routes between the two locations through the Google Directions API and then use a series of algorithms to determine the optimal route based on predicted distances, durations, and crowdsourced delay information submitted by users. The main goal is to combine the real-time road delay reports submitted to the application with the estimations returned by the Directions API in such a way that will help identify the optimal route, or rather the route with the shortest estimated delay time. These routes will then be displayed both on an interactive map to facilitate easy navigation and in a list view to provide additional information regarding the route. In addition, the feature will make use of location sensors such as GPS and network providers to detect and track the user's movement along the route.

Figure 3.4 provides a general overview of how the feature will work. Given a start location A and end location B, the Directions API will first return a list of viable route options between the two points, in this case Route 1 and Route 2. Based on its computations, route A is determined to be the optimal route by the Directions API as it has a shorter estimated duration of just 20 minutes under normal circumstances as opposed to Route 2's 30 minute estimated duration. However, this estimation does not take into account the various delays reported along each route and can therefore be seen as inaccurate. What this feature will do is calculate the estimated duration of all the reported road delays along each route and then combine them with the total estimated route duration computed by the Directions API to determine the new optimal route with the shortest total duration. Moving back to figure 3.4, route 2 is now determined to be the optimal route as it has a total estimated duration with delays of just 35 minutes in contrast to the 40 minute total estimated duration with delays of Route 1.
3.4.2 Quality Control Feature:

As highlighted in Chapter 2, data integrity is an area of considerable importance in crowdsourcing information systems. This solution design proposes a quality control filter based on two techniques; (i) user-based content verification, and (ii) contributor-mobility measurement.

(i) **User-Driven Content Verification:** Users will be provided with the ability to verify or rate submitted information reports, assigning positive ratings to accurate reports and negative ratings to those that are inaccurate or fabricated. Each report will be given an overall **credibility rating** between 0 and 100% corresponding to the verification ratings entered by users. To ensure that only accurate and verified road reports are stored in the system, the application will execute a PHP script at regular intervals, comparing each report rating with a pre-determined condition. Should any report credibility rating fail to satisfy this condition, that report is automatically removed from the system. An example of such a condition is provided below;

```php
if($rating < 20 && $number_votes > 10){
    mysql_query("DELETE FROM Database WHERE id = $id");
}
```

(ii) **Contributor-Mobility Measurement:** While user-driven content verification may succeed in significantly reducing the quantity of substandard or malicious quality content processed by the application, it still ultimately relies upon crowdsourced problem-solving. As a result, the scope for data-pollution still exists, albeit significantly reduced. The second quality-control technique aims to address this
issue by measuring user-mobility, or rather the distance between the contributor's current location and the location specified in the contributed report. Studies show that crowdsourcing participants with a smaller mobility are likely to offer more accurate and valuable contributions [5] and therefore, based on this research, the author believes that contributor-mobility can be used as a viable basis for quality filtering. This technique will use a location-sensor to determine latitude and longitude co-ordinates corresponding to the contributor's location and avail of reverse-geocoding (See Chapter 4) to obtain the co-ordinates of the source location of the submitted report. Using the Harversine mathematical formula, the application will then compute the distance between these two sets of co-ordinates in order to calculate the contributor's mobility rating. This rating can then be weighted and integrated with the existing credibility rating to develop an overall accuracy rating system.

In addition to these two techniques, users will also be able to update and remove content submitted to the system to preserve the accuracy of contributed information in light of constantly changing road conditions. This addresses the problems that emerge in dealing with real-time road information. Furthermore, users have the option to provide estimated durations to delay reports. Once expired, the application will automatically remove these reports from the system unless the estimation has been updated in the mean time. If an estimated duration is not supplied by the user, a default value will be assigned so as to ensure that stale delay reports do not overload the system.

3.4.3 Interactive Mapping:

A key feature of the application will be its interactive mapping capabilities, used to display the crowdsourced information to users. In addition to simply displaying routes and other complimentary information crowdsourced by the application, users will also be able to interact with the map in various ways such as viewing their current location, requesting directions to nearby petrol stations and car parks, and reporting road delays at specific locations.

3.5 Performance Considerations

3.5.1 Scalability:

Scalability is a central concept for applications which deal with large volumes of information. Poorly scaled applications will suffer from reduced performance, longer server response times, and overall inefficiency. While this application is oriented specifically towards route planning in the Dublin area, the possibility of having to handle demand on a far greater scale must be taken into account when designing the application.
3.5.2 Data Storage Model:

The decision on which data storage model to implement is one of considerable importance for any application based on the concept crowdsourcing. The choice of storage method can again have a substantial effect on both underlying performance and the potential scalability of the application and therefore must be considered carefully. The technology selected for this application will be explored in detail in the Technical Implementation Chapter.

3.5.3 Query Processing:

Query processing refers to how the application queries and processes the information stored on the server. Classical crowdsourcing applications are developed in either a centralised or decentralised manner [6]. Centralised approaches refer to the notion of 'query-shipping' whereby data is transferred from the application to an external server and all queries are evaluated at the server. In contrast, decentralised or 'data-shipping' methods send the queries to the device itself, where all computations are performed locally on the client-side.

Both approaches have their advantages; centralised methods can significantly reduce communication costs for high selectivity queries by providing the ability to exploit server resources when they are plentiful and can facilitate resource-poor client devices. On the other hand, decentralised approaches can exploit the resources (CPU, memory etc) of powerful client devices and reduce the cost of communication in the presence of locality or large query results [11]. The trade-off between the two techniques relate to the corresponding response times, communication costs, and potential scalability. Query-shipping is reliant on server performance for response and is less scalable with increasing client loads. Data-shipping on the other hand offers superior scalability, however communication costs are significantly greater and it is infeasible when faced with devices of lower-quality.

For the context of this application, a centralised three-tier client-server architecture is adopted in which the client, rather than sending queries directly to the server as would be the case using technologies such as JDBC, uses a middle-tier which is used to handle any interactions with the server. All queries are processed server-side and responses are returned back to the client, again via the middle business-logic tier. The benefits of implementing a three-tiered architecture over one of just two-tiers derive from the separation of the client tier from the business logic tier. This ensures that performance is maximised, communication times are minimised, and the internal query logic is well structured and easily maintained.
3.5.4 Information Correctness:

Correctness relates to the structure of the information contributed by users and is often overlooked by developers. Information contributed to the system should be 'correct' in the sense that important fields such as road names or delay descriptions cannot be left empty. The application should therefore have a filter in place to prohibit users from submitting empty fields that are deemed to be important for the functioning of the application. Furthermore, as the application is centred on the use of location-based values, the issue of non-existent or misspelt road names must be addressed. The application should again have mechanism in place to prevent users from submitting reports containing non-existent road names in order to ensure information credibility and accuracy. This mechanism should also be capable of identifying misspelt road names and correcting them to their intended values.

3.5 Chapter Summary

In summary, the solution design is modelled on a Crowd Processing System. The three types of homogeneous information sought by the application relates to to road delays, petrol stations, and car parks. The application will have a number of core features such as an A to B optimal route calculator, a quality control filter, and an interactive map, and it must also take into account a number of important considerations to ensure maximum performance. The following chapter introduces the various technologies used by the author in implementing the solution design.
Chapter 4 – Technology Stack

4.1 Chapter Introduction

Chapter 4 briefly introduces and describes the various programming languages, services, and technologies used to develop the application in order to provide the reader with a better understanding of the terms used in the Technical Implementation chapter.

4.2 Languages

4.2.1 XML:

XML was used to define the logical structure of the user interface (UI) element of the application. The Android SDK provides a comprehensive XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts [16].

4.2.2 Java:

Java provided functionality to the UI and was the core language used to develop the application. The Android SDK again provides the API libraries and developer tools necessary to build, test, and debug applications for Android.

4.2.3 PHP:

PHP was the server-side scripting language chosen to create the middle-tier web-service to handle interactions between the client and the server database.

4.2.4 SQL:

In order to query the database, SQL was used within the middle-tier PHP scripts.

4.3 Technologies Deployed

4.3.1 MySQL:

The information storage technology selected was a MySQL database hosted on an external server. Based on the requirements of the application, MySQL provides the most practical and efficient way of managing large quantities of user-driven information in a fast, reliable, and secure manner. Unlike the other methods of data-storage provided by Android, MySQL is
highly scalable supporting a theoretical limit of approximately 8 TB of data, and can be easily accessed through the use of various server-side scripting languages.

4.3.2 Google Maps Android API:

The Google Maps API is the most widely used mapping service for Android, which enables developers to add powerful interactive mapping capabilities to their applications [14]. The API provides an extensive external library, which automatically handles activities such as accessing the Google Maps servers, downloading data, displaying maps, and responding to map gestures. In addition, a number of highly useful functions and overlays are provided to developers, which came of great use in this project. These include:

- **Interactive Markers** – Markers were used to identify and display specific points of interest on the map such as locations of road delays, petrol stations, car parks. The OnMarkerClick function provides the ability to implement further functionality upon user interaction with the map.
- **Polylines** – Polylines are linear overlays of connected line segments on a map. A polyline object consists of multiple (latitude, longitude) co-ordinates, and creates a series of line segments that connect those locations in an ordered sequence. The Polyline class was relied upon to display the routes returned by the application's A to B route calculating feature.
- **LatLongBounds** – The LatLongBounds class enables the creation of bounded perimeters aligned with pairs of (South West, North East) latitude and longitude co-ordinates. For the purpose of scalability, this class was used to narrow down the amount of information handled by the application by creating a distance-based perimeter around the user's current location. Only those submitted reports with latitude-longitude points within the perimeter are retrieved and processed by the application.

4.3.3 Google Directions API:

The Google Directions API is a service that calculates directions between two given locations through the use of HTTP requests [13]. In addition to just directions alone, the API also provides other useful information regarding the journey routes such as the total estimated durations, total geodesic distances in kilometres, and encoded polyline points that can be used by the Google Maps API to plot the returned routes on a map. Many of the features implemented in the project, particularly the A to B route calculator, relied on the Directions API for functionality so it was therefore crucial to fully understand the restrictions and limitations of the service. A basic directions HTTP request takes the form:

http://maps.googleapis.com/maps/api/directions/json?origin=Start_Location&destination=End_Location&sensor=false
Responses – Once a HTTP request is sent to the server, a response is returned in JSON format. Therefore, in order to use the information returned by the API, a large amount of JSON parsing was required, which added an additional layer of complexity to the project (See Chapter 5 for more details).

Polyline Points: As mentioned previously, the Directions API returns a useful piece of information in its response in its 'overview_polyline' object in the form of the two end polyline points. Once decoded and parsed, developers can use the two unscrambled beginning and end points to plot a geodesic polyline along the route. The difficulty in this respect was researching and coming up with ways of decoding the scrambled data.

4.3.4 Google Geocoder:

The Google Geocoder is a powerful class provided by the Google Maps API that facilitates the handling of geocoding and reverse-geocoding [12]. Geocoding is the process of transforming a street address value or other description of a location into a (latitude, longitude) co-ordinate. Reverse-geocoding, in contrast, is the process of transforming a (latitude, longitude) co-ordinate into a single standardised address value. To complete the geocoding process, the Geocoder requires using a backend web-service to query the server that is not provided by the core Android framework. Since the application is centred around the use of road locations and mapping co-ordinates, the Geocoder service is central to the project. The various problems associated with using the Geocoder are explored in the Technical Implementation and Evaluation chapters.

4.3.5 Cron Job:

A cron job is a tool or process that developers use to schedule periodic executions of host commands or web applications on a server. Within this project, a cron job was set up on the server to execute specific PHP scripts to identify and filter out expired or inaccurate reports as determined by the estimated durations and accuracy ratings submitted by users.

4.3.6 Location-Sensors:

Location detection is a fundamental aspect of the project and enables the application to deliver more accurate and useful information to the user. The Android SDK provides a native
location framework that developers can use to determine a device's location and bearing. This framework offers three different location sources; GPS, network, and Wi-fi providers. Determining which provider to use and trust is a matter of accessing the trade-offs in accuracy, speed, and battery-efficiency [15].

Although GPS is the most accurate provider, it only works in exposed areas, is highly battery-intensive, and has a significantly slower response time. Android's network location provider, on the other hand, has a considerably faster response time and has less of a drainage affect on battery life, however its readings have less precision and can often return highly inaccurate readings during periods of weaker internet connectivity [15] [6]. To harness the advantages of both techniques, this application implements a hybrid model that incorporates both GPS and network location readings (See Technical Implementation Chapter).

Figure 4.1 – The Optimal Location Detecting Process [15]:

![Diagram showing the optimal location detecting process]
Chapter 5 – Technical Implementation

5.1 Chapter Introduction

This chapter provides an in depth analysis of the various steps taken in developing the project application. For structural purposes, this chapter is divided into two sections; back-end implementation and front-end implementation. These sections are further broken down into various sub-headings relating to the implementation of individual features relevant to the particular section.

5.2 Back-End Implementation

5.2.1 Three-Tier Client-Server Architecture:

As highlighted in the previous chapter, the information storage technology chosen for the project was a MySQL database hosted on an external server. Furthermore, the solution design outlined in chapter 3 proposes that a three-tiered client-server architecture should be implemented to connect to the server and query the database.

In order to develop the middle-tier to handle the interactions between the client (the application) and the server, an API was written using PHP which worked as an intermediary web-service. This web-service conformed to a RESTful architectural style that facilitated a call-response interaction between the application and the server through the use of HTTP requests. A high-level description of the architecture is provided below:

1. The client sends a HTTP request to the server to query the database.
2. The request connects to the server and executes the relevant PHP file corresponding to the URL of the request to act as the web-service.
3. The PHP file accesses the database and queries the database using SQL commands.
4. The web-service then returns a JSON response back to the client containing the relevant information from the database.
5. The application finally decodes and parses this response in order to display the information to users.
5.2.2 HTTP Requests:

To facilitate the sending and receiving of HTTP requests between client and server, a java-based function availing of various classes provided by the Apache Http Client bundled in with the Android SDK was used. The following arguments are taken in as a set of parameters; a URL corresponding to the location of the relevant PHP script on the server to act as the intermediary web-service, a HTTP type (POST / GET) determining the form of HTTP request to send, and a list of parameters containing the values to be processed by the web-service. This function is called every time the user requires access to the database. Information retrieved from the database is encoded to JSON format using PHP's useful json_encode function and is then returned to the client for parsing using APIs provided by the Android SDK.

```
// push single delay into final response array
array_push($response["delays"], $delaysArray);

// success
$response["success"] = 1;
$response["message"] = "Delays successfully retrieved";

// echoing JSON response
echo json_encode($response);
```

Figure 5.2 – Encoding Retrieved Information to JSON Format:

5.2.3 Database Architecture:

Within the MySQL database, three separate tables were created corresponding to the three types of information stored on the system. The included fields in each table relate to the structure of the specific type of information stored. Figures 5.2, 5.3, and 5.4 illustrate the layouts of each of the three tables:
5.3 Front-End Implementation

To analyse the front-end implementation in greater detail, the section is divided up into four sub-sections; three of which are related to the core objective levels outlined in the 'Project Objectives' section of Chapter 1, and the remaining sub-section describes the other important features implemented. These four sections are:

5.3.1 Facilitating Route Reporting
5.3.2 Displaying Route Information
5.3.3 Using the Crowdsourced Information
5.3.4 Other Important Features.
5.3.1 Facilitating Route Reporting:

It was decided by the author that the application should allow users to submit route reports through two separate approaches; 'manual reporting' and 'interactive map reporting'. The various steps taken in implementing these two approaches are outlined below. In addition, the issue of information correctness is also addressed in detail.

a. Interactive Map Reporting:

Description:

Interactive map reporting, as the title suggests, allows users to submit reports through an interactive map. Using the Google Maps API, users can interactively report information about a specific road by simply long-pressing the map at a desired point. Once pressed, a custom-made dialog interface is displayed on the screen prompting the user to enter additional details about the report. Once a report is submitted, a brief message informing the user of the submission is displayed on the screen and a marker overlay is then placed at the pressed location. The application asynchronously creates a new entry in the database using the information inputted by the user.

Implementation:

The first step in implementing this feature was to make use of the Google Maps API to load a new map activity. Using the OnMapLongClick method provided by the Maps API, the exact latitude and longitude of the pressed location could then be determined for later use by the feature. Once long-pressed, a custom-made dialog interface is inflated to the main UI view providing textboxes for users to enter further information. After the report is submitted, the MakeHttpRequest function is then called to send a HTTP request to the server to create a new entry in the database. If the application receives a successful response back from the server, the addMarker method, again provided by the Google Maps API, is called to plot a new marker at the related latitude and longitude coordinates.

To calculate the name of the road corresponding to the point pressed on the map by the user, the Google Geocoder was used to reverse-geocode the latitude and longitude co-ordinates of the location into a valid address, which could then be stored in the database.
b. Manual Reporting:

The alternative to interactive mapping is to submit reports manually using basic textboxes. To achieve this, a UI was designed consisting of multiple text fields and a submit button to enable users to enter report details and subsequently submit the report. Following a report submission, the application first geocodes the entered road name to derive an approximate latitude and longitude co-ordinate of that road. It then repeats the HTTP protocol procedure described in the interactive mapping approach to create a new entry in the database.

The problem with a purely textbox-based approach, however, is that it does not allow the user to input an exact location of interest. Instead, the Geocoder determines the approximate latitude and longitude co-ordinates to be the midpoint of the road entered by the user. This approximation can often vary greatly from the desired location of the user, resulting in what the author calls approximation variance, or rather a distance between the actual report location and the approximated report location computed by the Geocoder. In the context of road delay measurement, this is particularly problematic when considering longer roads such as motorways where the approximation variance could potentially be dozens if not hundreds of kilometres.
Implementing a feature that considers the user's exact location rather than the approximated road midpoint made this a more challenging task as it involved working with both location sensors and geocoding. Firstly, the location sensor is used to determine the exact latitude and longitude of the user's current location. To calculate the road name corresponding to that point, the current location co-ordinates are reverse-geocoded to a single address value. Rather than storing the approximated midpoint co-ordinates of the entered road as was the case previously, the new lat-long co-ordinates are instead inserted into the database.

Although this is a somewhat resource-intensive approach, the author believes that the advantages that flow in form of superior data accuracy far outweigh the disadvantages of slightly increased response times. Furthermore, this feature will be optional meaning users with weaker internet connections can continue to enter road names manually rather than using this approach.

**Figure 5.7:**

**c. Information Correctness:**

Chapter 3 introduces the concept of 'information correctness' whereby all information reported to the system should be complete and free of errors such as non-existent road names or empty fields. A number of solutions are adopted by the application to prevent users from entering incorrect information.
Empty Text Fields:

The first solution is a straightforward method that checks whether the user has entered text in the required textbox after pressing the submit button. If the required text field is left empty (ie road_name.getText().length() <= 0), the user will be prompted to review and re-submit the report form, otherwise the report is processed and a HTTP request is sent to the server to create a new database entry.

Non-existent Road Names:

The issue of non-existent roads is one of the most pressing for the project and therefore required a lot more thought. The implemented solution relies on the Geocoder; reverse-geocoding each submitted road name at submission time for a latitude-longitude co-ordinate result. If the reverse-geocoding request returns a valid location, then it can be deduced that the road exists and the report is subsequently submitted to the system using the road name returned by the Geocoder to ensure naming consistency.

The main benefit of using the Geocoder to verify submitted road names, as already highlighted, is that it significantly reduces the scope for harmful data-pollution by preventing users from submitting false of fabricated location entries. A standardised road-naming mechanism is also put in place by using the returned geocode values rather than the values actually inputting by the user, which is crucial for the A to B optimal-route calculating feature. In addition, the Geocoder is quite flexible and can correct misspelt road names (within a certain range) and identify road abbreviations.

Unfortunately however, with the advantages come the disadvantages, which must also be taken into account. Firstly, the process of using the Geocoder every time a user submits a report is a resource-intensive task and will most likely lead to slower response times due to the need for interaction between the back-end Google web-service and the server. For users with wi-fi or 3G connections the effects will be minimal. However, for those users with slower connections the negative effects can be significant. In addition to longer response times, a weak internet connection can often lead to network connection errors between the Geocoder web-service and the Google server causing the Geocoder to return null values. This is particularly problematic as the application will therefore deem the corresponding road entered by the user to be non-existent.

While these issues relating to the use of the Geocoder can undoubtedly compromise the performance and efficiency of the application, the author again believes that the need for precise data-accuracy and a standardised road-naming system is too great for these features to be overlooked.
A final problematic area in relation to information reporting is internet connectivity. The majority of the application's features require back-end interaction with remote servers, either to access the MySQL database or to send requests to Google servers via the Maps, Directions, or Geocoding APIs. For this reason, it is important to ensure that the user has an adequate internet connection to accommodate such interactions, otherwise the application would simply crash.

Therefore, a function was written to detect whether the user is connected to the internet. This required using the ConnectivityManager class provided by the Android SDK, which determines the connectivity state of the user's device by querying the active network when called. If no connection is detected, a dialog box, similar to those in figure 5.8, is displayed on the screen prompting the user to connect to a network.

### 5.3.2 Displaying Route Information

Route information is displayed to users via the same two approaches used above in information reporting; interactive map display and text-based display. A number of additional features are provided to users, which required a lot more effort to design and implement.

#### a. Interactive Map Display:

**Description:**

The primary way of viewing submitted route reports is through an interactive map. Once loaded, markers are placed on the map at the geographic latitude and longitude co-ordinate of the reported entries retrieved from the database. Users can view additional information relating to a given report (such as delay descriptions, estimated durations, petrol prices etc) by simply pressing the corresponding marker on the map. Furthermore, a 'Get Directions to Here' option is provided upon the pressing of petrol station or car park markers, which computes and displays the quickest route between the users current location and the point at
which the marker is placed. Users can navigate between the three different report types by using a navigator provided at the top of the UI.

Figure 5.9:

Implementation:

The implementation of this feature again required heavy use of the Google Maps API. To load the markers on the map, a HTTP request is first sent using the MakeHttpRequest function to retrieve the relevant information; most importantly the latitude and longitude co-ordinates of each relevant report in the database. A loop then iterates through the array list containing the retrieved co-ordinates, calling the addMarker function to plot a marker at each point.

However, to ensure optimal efficiency and scalability, only those reports with latitude-longitude co-ordinates within an approximation of five square kilometres of the user's current location are retrieved from the database and plotted on the map. This avoids any issues relating to scalability, such as long response times and high communication costs between the client and the server, which could potentially emerge by retrieving every submitted report.

Invoking this required firstly detecting the users current location using the location sensor. An approximate five-kilometre perimeter is then created around this location using the LatLngBounds class provided by the Maps API. Using a straightforward SQL WHERE clause, the back-end PHP script can then determine which reports are located inside or outside the square perimeter based on the four corner co-ordinates. Only those reports that satisfy the clause are returned to the client to be processed by the application. To allow users to view all submitted reports on the map, a 'View all delays'
option is provided in the activity settings menu, however this must be explicitly selected by the user to be called.

**WHERE Clause Logic:**

```
WHERE latitude > y1 AND latitude < y2 AND longitude > x1 AND longitude < x2
```

The 'Get Directions to Here' option was significantly more difficult to implement as it involved working with the Google Directions API and JSON. While a more in-depth analysis of the steps taken to calculate and display route directions will be provided further on in section 5.3.3 (*Using the Crowdsourced Information*), a high-level description is as follows:

(i) Firstly, a HTTP request containing the (latitude, longitude) co-ordinates of the pressed marker is sent to the Google server via the Directions API.
(ii) The Directions API then returns a response in JSON format containing information relating to the requested route.
(iii) The application parses this JSON response to determine the overall route path.
(iv) A Polyline is drawn on the map between the two points decoded from the 'overview_polyline' object using the addPolyline function provided by the Maps API (See figure 5.5).

**b. Text-Based Display:**

**Description:**

With text-based display, reported information is displayed to users in a list view format rather than on a map. Users can easily scroll through the reported delays in the list to view any information relating to a particular road. In addition, users can select reports within the list to view them in further detail in a new screen activity.

**Figure 5.10:**

![Text-Based Information Display](image)
**Implementation:**

Two XML layout files were first created representing the main UI activity and the UI for the list view in which the retrieved route information was to be passed into. After the main activity is loaded, a HTTP request is sent to the server to retrieve all the relevant reports from the database. Once retrieved, the values are then passed into the list view UI using the ListAdapter class provided by the Android SDK.

To facilitate user interaction in the form of list item selections, a separate XML layout was designed for the new activity. The relevant information corresponding to the selected item is then passed to this new activity via an intent. In addition to simply displaying this information, the new activity provides users with the ability to use the quality control feature outlined in chapter 3 to update, remove, or verify reports. If used, another intent is invoked to register any changes to the content. This intent is then called back to the main activity, updating the originally pressed list view item accordingly.

**5.3.3 Using the Crowdsourced Information**

This section deals specifically with the A-to-B optimal route-planning feature, providing an in-depth analysis of the various steps taken and technologies used for its implementation.

**A-to-B Optimal Route Calculator:**

**Description:**

The A-to-B optimal route-calculating feature is designed to calculate and display the quickest route between user-specified start and end locations, based on estimates provided by the Google Directions API and the delay reports submitted by users. Once a start and end location have been entered, the Directions API calculates a list of the most viable routes between the two locations. Using the information returned in the response, the application then computes whether any of the returned routes contain any of the reported road delays in the application. If so, the overall estimated route duration is updated to reflect the estimated delay times corresponding to the matched delay reports and is subsequently displayed to the user in a list view. The route with the lowest estimated duration is deemed to be the optimal route.

Users can select a particular route from the list of computed routes to view the corresponding directions, either textually in a list view or visually through the Google Maps API (See Figure 5.11).
Implementation:

This was the most complex, and therefore time-consuming, feature to implement in the project. The process is divided up into four separate steps in order to provide greater detail and structure to the analysis.

Step 1 – Sending the Directions Request: When a user requests directions, a HTTP request is first sent to the Google server using the Directions API. The relevant start and end locations entered by the user are passed into the URL of the request as follows:


Step 2 – Parsing the Response: As described in chapter 4, responses from the Google Directions API are returned in JSON format. The Android SDK provides a comprehensive JSON handler class, which is used to parse the returned response so that the information can be processed by the application. The relevant information relating to each returned route is then stored by the application and passed into a list view for display to users. A typical directions response consists of three arrays of JSON objects representing three different layers of information detail. A breakdown of these layers is provided below:

- **Routes Array** – The routes array is the highest-level layer of the response, containing a list of the most viable routes between the two locations entered by the user (maximum of three routes).
- **Legs Array** – The Legs array provides an overview of a particular route in the routes array. Information such as the initial start and end co-ordinates, the total estimated
route duration, the total estimated route distance, and the encoded Polyline points are all stored in the legs array.

- **Steps Array** – The steps array is the lowest-level element of the API response, consisting of all the roads along a particular route in the routes array as well as their respective travel information. Important details such as the start and end location coordinates and the related driving instructions of each individual road are contained in this array.

**Figure 5.12 – Extract of a Sample JSON Directions Response:**

**Step 3 - Calculating the Optimal Route:** Having computed and parsed the list of viable routes with the Directions API, the application then incorporates the road delay reports submitted by users to determine the optimal route. This is done by comparing the roads along each route with the delayed roads reported in the database. If a pair of road names match, the estimated delay time corresponding to the matched delay report is retrieved and is added to the overall estimated route time. The various steps taken in implementing this are outlined below:
1. **Retrieving the Delayed Road-Names and Delay Times:** This simply involved getting all the reported delay road names and estimated delay times from the database using a straight-forward HTTP request. The returned values are stored in an array list of hashmaps to be used for later comparison.

2. **Getting Names of Each Road Along Each Route:** Because the response returned by the Directions API does not explicitly provide the names of the roads along each particular route, the task of computing these names was made significantly more difficult. However, the response does provide some useful information in the form of the start and end co-ordinates of each individual road. Using the Geocoder, these co-ordinates are reverse-geocoded into single road addresses, which are then stored in array lists corresponding to each returned route. Again, while this is quite a resource-intensive approach having to geocode the co-ordinates of every road along each returned route, it is the only possible way of computing the names of the individual roads. Furthermore, it is also advantageous as it converts the route road-names to the same standardised format as the reported delay road names, which is essential for the comparison process.

3. **Comparing Road Names:** The task of writing an algorithm that compares the delayed road-names with the names of the roads along each route was more complicated than expected. It requires first looping through three different arrays and comparing each delayed road-name with all of the road-names along each returned route option. If a match is found, the estimated delay time corresponding to the matched delay is added to a counter and the inner loop then breaks to ensure that duplicate road-names are not matched. The returned counters are then added to the respective total estimated route durations to evaluate the optimal route.

**Figure 5.9 – Algorithm to Compare Road-Names:**

```java
for (int i = 0; i < routesArray.length(); i++) {
    int counted = 0;
    for (int j = 0; j < delaysSeverityList.size(); j++) {
        for (int d = 0; d < roadNames.get(i).size(); d++) {
            if (delaysSeverityList.get(j).get(TAG_ROAD)
                .equalsIgnoreCase(roadNames.get(i).get(d))) {
                counted += Integer.parseInt(delaysSeverityList
                    .get(j).get(TAG_SEVERITY));
                break;
            }
        }
    }
}
```

**Step 4 – Displaying the Calculated Routes:** Displaying the route on a map requires first decoding the overview _polyline_ object points returned in the Directions API response from ASCII characters to human-readable text. A widely used design pattern was utilised to complete this task, and what is returned following its implementation is a list of latitude-longitude co-ordinates corresponding to each road along the route. Using the addPolyline function provided by the Google Maps API, a Polyline can then be plotted along the route using the decoded list of co-ordinates (See Figure 5.7).
5.3.4 Other Important Features:

a. Location Sensor:

*Implementation:*

As mentioned in chapter 4, the chosen location sensor model uses a hybrid approach, combining updates from both GPS and network providers.

*GPS Provider:* Because GPS provides the most accurate location readings, the sensor first checks whether the GPS setting is enabled on the device and then, if enabled, determines whether a GPS location update can be received from the provider. The sensor is set up to request GPS updates every second to ensure as accurate a reading as possible.
**Network Provider:** If the GPS provider is unavailable for updates (i.e., the user is indoors), the sensor instead requests updates from the network provider using the same methodology taken for GPS update requests. If both the GPS and network providers are unavailable, a null location is returned and an error is displayed on screen to the user. However, if a valid location is returned, a marker is plotted on the map corresponding to the latitude and longitude coordinates of that returned location.

**Maximising Battery Life:** To ensure that battery life is not drained by constant background updates, additional functionality is included to ensure that update requests are stopped if the user temporarily tabs out of the application (i.e., making or receiving phone-calls, text messages etc). Only when the user re-entered the application will the sensor request new location updates.

```java
protected void onPause() {
    super.onPause();
    locationManager.removeUpdates(this);
}
```

### b. Accuracy Rating System:

**Implementation:**

It was decided by the author to develop an accuracy rating system based solely on user-driven accuracy verification, meaning that the concept of user-mobility measurement was abandoned. As described in Chapter 3, the user-driven accuracy rating system provides the ability to assign positive and negative ratings to delay reports to establish an overall report accuracy rating, using the simple formula:

\[
\text{Total Number of Ratings} \div \text{Number of Positive Ratings}
\]
c. Cron Job:

**Implementation:**

Setting up the cron job to automatically remove expired or inaccurate reports from the database involved first writing a PHP script to specify the conditions under which reports should be deleted. It was decided that reports should be removed based on two factors:

- Time Elapsed
- Report Accuracy

**Time Elapsed:** Users are provided with the option to enter an *estimated duration* for every delay report submitted to the system. Following a delay report submission, the cron job then periodically compares the current time of executing the PHP file with the time at which that delay report was submitted. If this figure exceeds the provided estimated duration and the estimation has not been updated in the mean time, the report is removed from the database.

If an estimated duration is not provided for the delay report, the cron job simply removes the corresponding database entry after an arbitrary period of 2 hours.

```php
if($estimatedDuration > 0) {
    if($timeElapsed >= $estimatedDuration) {
        mysql_query("DELETE FROM Test2 WHERE pid = $id");
    }
}
```

**Report Accuracy:** As highlighted earlier, the application enables users to validate submitted reports using the quality control filter. Based on the derived report accuracy ratings, the cron job can identify any reports that surpass an inaccurate level to warrant removal. The condition logic is that if a certain number of users deem a report to be inaccurate, that report is too unreliable to be used by the application. For example, if a delay report has a total accuracy rating of under 30% after 10 user verification ratings, it is automatically deleted from the system.

```php
if($estimatedDuration == 0) {
    if($timeElapsed >= $twoHours) {
        mysql_query("DELETE FROM Test2 WHERE pid = $id");
    }
}
```

```
if($accuracyRating <= 30 && $number_of_ratings >= 10) {
    mysql_query("DELETE FROM Test2 WHERE pid = $id");
}
```

The cron job was set up on the server to execute this PHP script every minute to guarantee the integrity of every report stored in the system.
Chapter 6 - Evaluation

6.1 Chapter Introduction

In this chapter, a framework for testing the performance and effectiveness of the application is introduced. In applying this framework, a project evaluation if the project is reached and presented to the reader.

6.2 Testing

Without a comprehensive user-base, testing the application's overall effectiveness was inherently difficult. Regardless, various usage tests were carried out to evaluate the performance of the individual features of the application. The logic assumed was that if all the individual features worked as intended, the application itself would be suitable for its intended use as a crowdsourcing route planner.

6.2.1 Test Model:

Due to time constraints, testing was mainly based on the author's general observations and experiences of using the application over a four-day period. During this timeframe, various usage tests were conducted under two different conditions based on internet connectivity strength:

1. **Strong Connectivity** – Wi-fi, 3G, and HSDPA Connections
2. **Weak Connectivity** – 2G and Edge (GPRS) Connections

These tests were conducted using varying quantities of information, ranging from small (up to 20 submitted reports) to moderate (over submitted 50 reports) amounts. Aspects of the application tested were location sensor accuracy, client-server response times, Geocoder performance, and the A to B route calculating feature.

6.3 Results and Observations

6.3.1 Strong-Connectivity Usage Tests

*Client-Server Response Times:* Observed client-server response times were generally instantaneous with strong-connectivity testing. Regardless of the quantity of information tested, response times were minimal, varying by split seconds rather than seconds.
Location Sensor Accuracy: Readings returned from the location sensors were, by in large, highly accurate. Any major problems were attributable to errors from the Geocoder in converting the returned location co-ordinates into road names rather than the readings themselves (Discussed Further On). Even when GPS updates were not available, the network provider proved to be reliable in its readings due to the quick internet connection.

Geocoder Performance: Most observed problems were associated with the Geocoder. In a small number of cases, the Geocoder returned the local town name rather than the expected road name. For example, in one specific case the address 'Malahide, Dublin' was returned instead of the expected value 'Malahide Road'. This inadvertently led to inaccurate location evaluations since the Geocoder is needed to convert the latitude and longitude co-ordinates returned by the location sensor into an address value. Rather than returning the name of the road the user was actually on, the Geocoder determined the user's position to be the midpoint of the returned town, which was often kilometres away. On a few occasions, test results also reported the Geocoder returning road names in Irish rather than English ('Stráid Úi Chonaill' rather than 'O Connell Street'). This is particularly problematic because the Android TextView Class does not recognise special characters (such as á) and instead displays a null value. Although these issues are an undoubted cause for concern, their occurrences were rare, observed in just a small percentage of tests.

A to B Route Planner: Tests conducted on the A to B route calculator demonstrated acceptable response times in the majority of cases, where 'acceptable' is deemed to be anywhere between zero to five seconds. These response times were marginally longer depending on the increasing of two factors: (i) the number of roads along each returned route, and (ii) the number of delays reported in the system. This is firstly due to the additional geocoding that is required to determine the extra road names along each route. It can also be attributed to the additional comparisons that the feature must make between the delayed road names and the road name along each of the returned routes.

6.3.2 Weak-Connectivity Usage Tests

Client-Server Response Times: As was expected, there was a noticeable increase in observed client-server response times. However, these times were still relatively fast and fell well within what the author considered an acceptable time (between 0 – 5 seconds).

Location Sensor Accuracy: When GPS was unavailable, the observed readings of the location sensor were less reliable due to the need for internet-based network updates. In some cases, the sensor failed to detect location changes for up to a minute. In addition, readings were often inaccurate by anywhere up to an approximation of half a kilometre (ie. a road or two off).

Geocoder Performance: In a considerable amount of observations, the Geocoder returned null road name values due to its failure to connect to the back-end web-service. Furthermore, the problem of returning local town names rather than expected road names was again
observed, however this time on a far greater scale. Overall, the Geocoder proved to be highly volatile and unreliable with weak internet connections.

**A to B Route Planner:** Because of the A to B route planner's reliance on the Geocoder and the Directions API, the observed response times were dramatically increased, and in a considerable number of cases failed to return a response altogether. During periods of weak internet connectivity, tests results indicated that this feature is effectively useless.

### 6.4 Evaluation

Based on the results of the testing process, it can be deduced that application performance is mainly a function of two factors; internet connectivity strength and information quantity. Weaker internet connectivity tests saw performance levels decrease significantly in the majority of the areas tested, most notably in Geocoder performance and the A to B route planning feature. However, this is not considered to be a significant issue as all the major mobile networks now offer 3G internet connectivity as standard and furthermore, by the end of 2013 4G mobile broadband is expected to be introduced across the country [20]. These tests were conducted more to assess the worst-case scenario and the related problems could potentially occur in such situations. Results of strong connectivity usage tests demonstrated positive and reliable performance in all tested areas. While a few problems were observed, particularly with the Geocoder, they did not represent the general results of the sample set. An increase in the quantity of information handled by the application revealed marginally longer response times, however the observed impact was minimal and is not considered to be a major cause for concern.
Chapter 7 - Conclusion

7.1 Author's Comments

The aim of this project was to develop a crowdsourcing application that enabled users to report route information, view reported route information, and use this reported information to improve the general route planning process. Based on this, I believe the project has been quite successful. The application provides a simple yet fully functional crowdsourcing platform which delivers results that are of positive value to the user in the route planning process. Users can report and view information with relative ease through the interactive mapping feature and can avail of the A to B route calculating feature to make informed decisions on route options.

Having had limited previous experience with Android Application Development and back-end databases there was a considerable learning curve involved, which made it difficult to implement everything that I would have liked. Regardless, I am fully satisfied with the finished project and had it not been for time constraints, even more features could have been added to the application (See Future Work). I gained a lot of new technical skills and knowledge that will be of great benefit in future projects.

7.2 Issues and Problems

Although deemed an overall success, the application is not without its flaws. The main issues and problems are highlighted in this section.

7.2.1 Scalability:

An important issue concerning the application relates to scalability. As highlighted in chapter 6, little testing was done to evaluate the performance of the application in handling large amounts of information. While observed client-server response times were generally quick with smaller quantities of data, as the scale of information begins to increase, the strain placed on the server could potentially have an adverse effect on response times and overall performance. Although various measures were put in place to ensure that the application was as efficient and scalable as possible, such as using of the LatLngBounds class and WHERE clauses to narrow down the data handled by the system and implementing a three-tier client-server architecture, there could still be room for further improvements in this regard.

Based on test observations, the A-to-B route calculator feature is particularly exposed to scalability-related problems. As the quantity of delay reports submitted to the database increases, the feature will be required to compare more and more route names. Again, this
will lead to considerably longer communication costs and CPU processing tasks, which will reduce the overall performance of the application.

7.2.2 Reliance on the Geocoder:

A second and more pressing problem is the application's over-reliance on the Google Geocoder. The Geocoder is used in a large majority of the features in the application and has been proven by the usage tests conducted in chapter 6 to be quite unpredictable. The various problems associated with the Geocoder are described below.

Resource-Intensive: The processes of geocoding and reverse-geocoding involve sending HTTP request from the client to the Google server every time they are required. Obviously, this will have a negative effect on the overall performance and efficiency of the application, particularly as the user-base begins to increase. The route-calculating feature in particular relies heavily on the Geocoder to determine the names of the roads along the routes returned by the Google Directions API. The longer the route, the more HTTP requests that need to be sent to the server, meaning even longer response times for the user. This is consistent with observed test results, although more noticeable during periods of weaker internet connectivity.

General Flaws and Bugs: The Geocoder is known to have a number of bugs which can be extremely problematic for the functioning of the application. The most common problem, which has been experience on more than one occasion, is its failure to detect the back-end service needed to query the Google server. In such situations, the Geocoder will throw an IOException and return an empty list of values, which essentially renders the application useless. No reports can be submitted to the system because the Geocoder is needed to verify every entered road-name, and the optimal route calculator will throw an exception when attempting to geocode the Directions API response. While resolving the issue requires simply re-booting the Android device, this is obviously not feasible when dealing with a larger number of users. Unfortunately, this is a problem with the Google back-end server and is out of the author's control. To prevent the application from recognising these null values as actual road locations, a helper method isPresent() provided by the API is used to first detect whether the back-end service is available and then, if not, alert the user accordingly.

Misinterpreted Road Names: The Geocoder can sometimes misinterpret road names and return a location that corresponds to a different road with the same name, particularly when the road name is quite generic such as Main Street or Coast Road for example. While the application currently relies on the user to provide additional details about the road such as the corresponding town or area (ie 'Main Street, Malahide' rather than just 'Main Street'), this is not an ideal solution and in the majority of cases will not be done.

Future work aims to resolve this issue by simply changing the Geocoder settings to return a list of all the possible locations rather than just the assumed location. Using this returned list,
the application will prompt the user to manually select the intended address from the returned list as can be seen in various other applications such as Google Maps.

Figure 7.1:

Weak-Connections: During spells of weak internet connectivity, the Geocoder can sometimes fail to connect to the back-end web-service, again resulting in a null list of values being returned to the client. Other observed problems associated with the Geocoder and weak connectivity include road-names being returned in Irish and nearby town names being returned rather than the intended road, as were all observed in the testing stage.

7.3 Future Work

7.3.1 Incorporating GPS Traffic Patterns

The main focus of future work will be on incorporating real-time traffic patterns through the passive crowdsourcing of GPS travel information from users. As seen with services such as Google Directions and Waze, traffic data passively crowdsourced from smart phone GPS signals can provide an endless stream of valuable information regarding delays and traffic conditions. In combining this data with the ubiquitously collected road reports submitted by users, the application can then build a more comprehensive and complete model for route planning and optimal route calculation.

7.3.2 Pattern Learning Features

Fundamental to the GPS crowdsourcing feature outlined above is the development of intelligent features that can actually process and use this collected data. Recent studies suggest that route-planning applications should use historical analysis of traffic patterns and commuter sentiment to provide route guidance [21]. An interesting challenge, therefore, will be to develop features that can use the harnessed daily traffic patterns to learn about the routes which users are travelling on. The application could then recommend optimal routes to users based on internal traffic predictions rather than just reported real-time information.
alone. In addition, the application could identify roads with heavy traffic congestion based on external factors such as the time of day or the day of the week.


http://support.google.com/maps/bin/answer.py?hl=en&answer=2549020&topic=1687356&ctx=topic


https://developers.google.com/maps/documentation/directions/


