Location-aware Reminders

Final Year Dissertation
BSc Computer Science (Software Engineering)

Lewis Wilson – lw52@hw.ac.uk – H00270128

Supervisor: Phil Bartie

Second Reader: Stefano Padilla

April 22, 2021

Acknowledgements

I would like to dedicate the following section to the people who have helped and assisted me throughout this dissertation project and my family.

Dr Phil Barite, thank you very much for your excellent support and guidance throughout this dissertation project; I could not have asked for a better supervisor. The result of the application created would not have been nearly as successful without your knowledge and insight into spatial technologies and methods, which was invaluable.

Dr Stefano Padilla, for your great feedback and suggestions after the Deliverable 1 submission that helped shape this dissertation.

Mum, Dad and Claire, the seven years it has taken me to achieve this degree, have had many moments. When I faced challenges and obstacles throughout my time at college and university, you always supported and believed in me, through thick and thin, never giving up and constantly pushing me to do my best. Thank you for your support and always believing in me; I would never have made it this far without you.

Declaration

I, Lewis Wilson confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Lewis Wilson

Date: 22/04/2021

Abstract

Electronic reminders are commonly triggered based on time, but there is a desire for reminders to be more flexible. There are occasions where the time element is not as important as the location, for example, “remind me to buy milk the next time I am at a supermarket”. This reminder does not have a strict temporal filter, but prompting the user the next time they are near a store that sells milk would be helpful. To implement the ability to assign and trigger reminders at a generic location, such as supermarkets (e.g. Tesco, Sainsbury’s, Aldi = all supermarkets) an ontology has been considered.

This document details the development of a mobile application with an interface that allows users to create a reminder with a generic spatial location as the trigger. A literature review was conducted covering topics related to this project, such as existing applications and geospatial technology. The implementation was also discussed, from how the mobile application was built to how we used the user's location to trigger reminders. Finally, we evaluated the system to determine the success of the different geospatial techniques used to determine if a user is near a reminder location.

Table of Contents

1 Introduction 1
  1.1 Aims 1
  1.2 Objectives 2
  1.3 Document Summary 3
  1.4 Application Demonstration 3
2 Background 5
  2.1 Related Technology 5
    2.1.1 Positioning 5
    2.1.2 Geo-fences 8
  2.2 Existing Applications 9
    2.2.1 Temporal 10
    2.2.2 Temporal and Spatial 10
  2.3 Overview 12
  2.4 Spatial Data 12
    2.4.1 Digimap 13
    2.4.2 OpenStreetMap 13
  2.5 Data Storage 15
    2.5.1 PostgreSQL 15
    2.5.2 PostGIS 15
  2.6 Streaming and Visualising Map Data 16
    2.6.1 Vector and Raster Data 16
    2.6.2 Streaming Data 16
    2.6.3 Visualising Map Data 17
  2.7 Events 18
    2.7.1 Temporal Triggers 19
    2.7.2 Line of sight based triggers 19
    2.7.3 Notifications 20
  2.8 Places and Spaces 21
    2.8.1 Ontology 21
    2.8.2 Folksonomy 23
3 Implementation 25
  3.1 Version Control and Data Backup 25
  3.2 React Native Framework 25
  3.3 Expo Development Tools 26
  3.4 Application Frontend 27
  3.5 Application Backend 31
    3.5.1 Firebase Authentication and Cloud Firestore 31

3.5.2 Amazon Web Services API Gateway, Lambda and Relational Database 32
3.6 Datasets used 35
3.6.1 Ontology 35
3.6.2 Spatial indexes 36
3.7 Using location to trigger reminders 37
3.7.1 Noise in positioning data 37
3.7.2 Converting WGS84 to BNG 38
3.7.3 Euclidean distance 39
3.7.4 Network distance 41
4 Evaluation 46
4.1 Adjustments to the previously planned evaluation study 46
4.2 Evaluation of Functional Requirements 46
4.3 Evaluation of Non-Functional Requirements 47
4.4 Euclidean distance 48
4.5 Network distance 50
4.6 Comparing the Euclidean distance to the Network distance 52
5 Conclusions and Future work 54
5.1 Aims and Objectives Met 54
5.2 Future work 55
5.2.1 Conducting further testing 55
5.2.2 Sharing reminders between users 56
5.2.3 Improving the points of interest data 56
5.2.4 Improvements to the user interface 57
5.2.5 Triggering reminders when the application is in the background 58
5.2.6 React Native testing 58
6 Back matter 59
6.1 References 59
6.2 Appendix 63
6.2.1 Appendix A – Creating a lookup table linking POIs to the building they reside within 63
6.2.2 Appendix B – Euclidean distance implementation 64
6.2.3 Appendix C – Creating a table linking each POI to its closest edge 65
6.2.4 Appendix D – Function for finding the nearest edge to a user 66
6.2.5 Appendix E – Function for finding nearby POIs within the search range 67
6.2.6 Appendix F – Network distance implementation using pgr_dijkstraCost 68
6.2.7 Appendix G – Function for finding the geometry of the nearest edge to a user 69
6.2.8 Appendix H – Network distance implementation using pgr_withPointsCost 70
6.2.9 Appendix I – Unit Test example 71

1 Introduction

Mobile devices play a prominent role in our day-to-day lives and are commonly used to track reminders, e.g., food shopping lists or to exercise. However, current reminder implementations predominantly focus on the use of time to trigger a reminder. There is a desire to associate more than just time as the location of a reminder could be far more relevant; an example of such a reminder is “remind me to buy milk the next time I am at a supermarket”. Completing this reminder does not have a strict temporal deadline; instead, it must be completed at a specific location. It would be far more beneficial if the user receives a reminder when they are near a store that sells milk rather than focus being on a specific time.

This dissertation first reviews and critically evaluates literature relevant to creating such an application covering topics such as existing applications and related technology. A discussion about how the application was implemented is reviewed —discussing the modern mobile development technologies used, such as the React Native framework to create the mobile application. The geospatial components are then discussed, such as the API used by the application responsible for completing geospatial tasks on the cloud. These tasks include finding nearby points of interest and determining how far away the user is from them. Finally, an evaluation of the system was conducted through a systems test to determine the success of the different geospatial techniques employed to trigger reminders based on the user's location.

1.1 Aims

The aim of this dissertation is to create a mobile application that allows user to create reminders that have a location-based trigger associated with them. To trigger a reminder, we will use the user's location to determine if they are near a point of interest where they could complete a reminder, e.g. a supermarket is nearby. When users create a reminder, they should not detail the specific location where the reminder should trigger,e.g. Tesco Express on Princess Street. Instead, they should select a generic location type, i.e. food shopping, that considers all supermarkets equal (e.g. Tesco, Sainsbury's, Aldi = all supermarkets). The geospatial functionality portion of this implementation will be evaluated to determine the success and effectiveness of triggering reminders based on the user's location.

1.2 Objectives

A mobile application will be implemented for both major mobile operating systems, Android and iOS. The user will interact with the application through a user interface to create reminders with a location type as the reminder trigger. A database system will be implemented, allowing users to create accounts and store their reminders. The user interface will be designed with useability in mind, providing an experience that is easy to use and navigate. Features expected in reminder applications such as editing and deleting notes that have already created should be implemented.

Triggering reminders based on the user's location will require the application to run spatial queries on the database. An API will be implemented, allowing the mobile application to interact with the database remotely. The results returned from the API will allow us to determine if the user is near a point of interest and subsequently trigger reminders. The database will also need to store geospatial information to do this, such as information about points of interest (e.g., supermarkets and postboxes) and topographic data, detailing every coordinates position feature type (e.g. roads, buildings and paths).

The application will trigger reminders that have been associated with a generic location type, e.g. supermarkets at any location. An ontology will be implemented to provide this functionality, allowing us to consider all places of the same type as equal (e.g. Tesco, Sainsbury's, Aldi = all supermarkets). The implementation will be evaluated to determine the effectiveness of this solution.

1.3 Document Summary

Introduction: Summarising the aims and objectives of this document.

Background: Literature review of research conducted on related technology, existing applications, spatial data, data storage, streaming and visualising mapping data, events, places and spaces.

Implementation: Discussion of the implementation of the mobile application and the technologies and techniques used to implement the geospatial functionality.

Evaluation: An evaluation of the requirements implemented for the project and the geospatial functionality.

Conclusions and Future work: Concluding and summarizing the project, highlighting the objects met and finally, proposals for future work are suggested.

References: A listing of the references used throughout this document.

Appendix: A listing of the appendices used throughout this document.

1.4 Application Demonstration

Video clips of the application have been produced, each highlighting the critical functionalities of this proposed solution.

Video 1 – Demonstrates a user logging into the application and viewing currently active and completed reminders:

https://www.youtube.com/watch?v=vweMlVEIPpc

Video 2 – Demonstrates a user creating a reminder:

https://www.youtube.com/watch?v=Zg2aIn_HdqU

Video 3 – Demonstrates an example of a reminder being triggered spatially:

https://www.youtube.com/watch?v=sTqs4NlJo8A

Video 4 – Adjusting the search distance for reminders and showing how this has an impact on the nearby points of interest that the user finds:

https://www.youtube.com/watch?v=6D91wFN6oeo

2 Background

2.1 Related Technology

A mobile application that can prompt the user of location-based tasks requires many technologies, including geofence event triggering, spatial databases, and user interface design. The background to these and other similar systems are critically evaluated in the following sections.

2.1.1 Positioning

Accurately determining the user's location was critical to the success of the project. The different positioning technologies reviewed were the Global Navigation Satellite System (GNSS), WiFi and Cell Tower. There was an interest in determining the accuracy in different environments as well as their widespread availability.

Positioning using GNSS

Currently, there are multiple different GNSS systems in operation. These include Galileo, created by the European Union and the Global Positioning System (GPS) founded by the United States Government. GNSS provides a global system for obtaining high accuracy positioning data with excellent performance featuring worldwide coverage, low latency and exceptional availability (Langley, Teunissen and Montenbruck, 2017).

Outdoor Positioning using GNSS

GNSS dominates as the outdoor positioning technology used for mobile devices (Mautz, 2012). A study conducted by van Diggelen and Enge (2015) compared the accuracy of GPS in unobstructed line of sight (LOS) conditions. They had over one thousand participants from one hundred different countries; impressively, they recorded an average accuracy of 4.9 metres.

However, this is not the case in urban environments like cities, where GNSS systems provide inaccurate location estimations (Ben-Moshe *et al.*, 2011). The situation was described as an urban canyon with tall buildings and other obstacles blocking the receivers LOS to the navigation satellites. These signals were described as None-Line-of-Sight (NLOS) blocked signals and NLOS reflected signals (Nagai *et al.*, 2020).

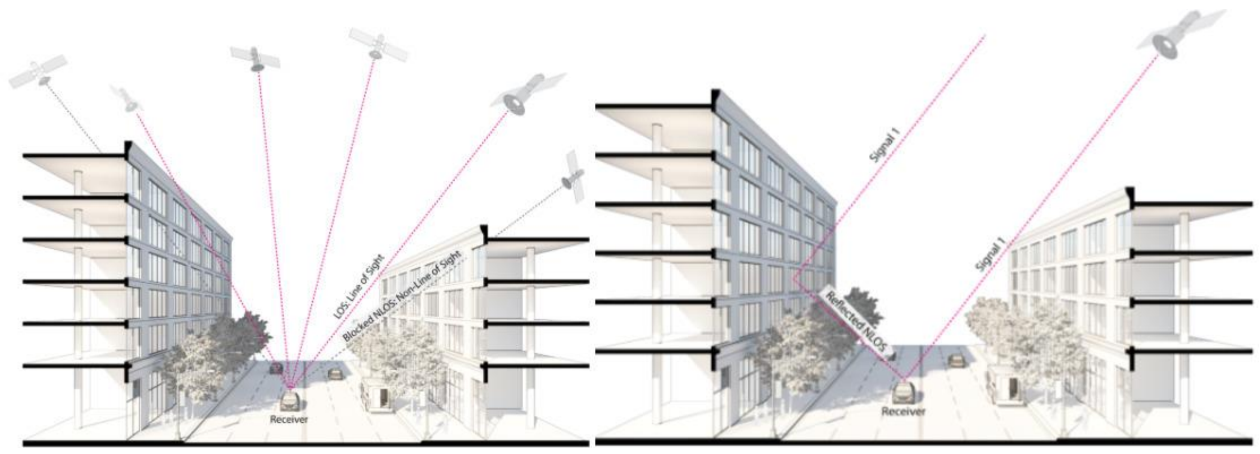


Figure 1: Three types of GNSS signal reception. The left part of the figure shows unobstructed LOS signals and NLOS blocked signals. The right part of the figure shows NLOS reflected signals (Nagai et al., 2020).

When using GNSS systems in urban canyons is multipath is the leading factor for poor performance. Multipath causes the signal to travel further compared to LOS signals. Thus, the signal strength is weaker and provides less accurate positioning data (Xie, Petovello and Basnayake, 2011). It occurs when the sensitive GNSS receivers read the indirect signals reflected from surrounding buildings, as shown in Figure 1 (Nagai *et al.*, 2020).

A study conducted by Groves, Wang and Ziebart (2012) recorded the performance of GNSS systems in urban canyons. They found the positional accuracy to have a margin of error up to 20 metres when trying to determine a person's location on a street (e.g. the wrong side of the street).

Indoor Positioning using GNSS

Indoor positioning is currently challenging using GNSS, as Mautz (2012) states and is an active research area. He documents some of the significant challenges faced when using GNSS indoors, which overlap directly with outdoor positioning:

- NLOS conditions always occur as the building blocks the signal path between the transmitter and the receiver.
- Multipath occurs as the signal is reflected due to the presence of objects or walls.

GNSS based systems require line of sight to satellites to operate accurately, and as discussed in outdoor positioning using GNSS, the accuracy is significantly impacted in dense urban areas. These issues are further compounded when using GNSS indoors, completely failing to provide accurate positioning data (Einsiedler, Radusch and Wolter, 2017).

WiFi Positioning

WiFi-based positioning relies on WiFi access points that continuously broadcast a signal to announce their existence and spread hundreds of metres in every direction. Due to the density of the access points in urban areas, signals commonly overlap, creating a reference system that can be used to determine the user's location. Users do not have to be connected to the WiFi networks to make use of this technology as it works by recording the unique MAC addresses and signal strengths of the access points (Zandbergen, 2009). A study by Mok and Retscher (2007) in an environment with a high density of access points found the median horizontal accuracy between 1 and 5m.

As Zandbergen (2009) states, WiFi positioning does have excellent performance indoors under the right conditions compared to GNSS systems like GPS. Battery life is also improved using WiFi positioning as it is less demanding than repeatedly checking GNSS. However, we cannot guarantee that this infrastructure is present for us to use.

Cell Tower Positioning

Cellular networks present a possible solution for determining the user’s location with almost worldwide coverage. Cellular networks are made up of cells, with each having at least one base station (cellular tower). Methods were available to track a user as they move through the network and determine which cell they are currently within. Commonly these cellular cells overlap, in which case the user connects to the cellular tower with the strongest signal strength. The most unsophisticated approach using cellular networks to determine the user’s location is to provide the known location of the base station. Some cell towers try to improve upon this by splitting the cell into sections and using directional antennas to narrow down the user's location (Zandbergen, 2009).

A study by Mohr, Edwards and McCarthy (2008) tested three different cellular operators in the United Kingdom. They found the accuracy of cellular positioning to have a median error of 246 metres in dense urban areas and 626 metres in rural settings.

Positioning conclusion

In this research project, we ruled out using cell tower positioning due to the low performance in urban and rural settings. WiFi positioning was not an appropriate choice either, as we cannot guarantee that the needed infrastructure will be present. GNSS in outdoor positioning is promising, though there are some issues with accuracy in urban locations. Solving the noisy urban data is a focal point of this research project. Due to the inaccuracy of using GNSS to determine indoor positioning, this project has focused solely on outdoor positioning using GNSS.

2.1.2 Geo-fences

Geo-fencing is a spatial function that is used to define real-world geographical locations as having boundaries. It creates a system that is context-aware of the user’s approximate position to a nearby point of interest (POI). Geo-fencing can be thought of as a ‘virtual fence’ around a particular POI that's shape can be circular or polygonal. When the user's device enters or exits a geo-fenced area, this can create a trigger that causes the device to send a notification, informing them to complete a task. Geo-fences work by continuously determining the user's position in relation to the predefined geo-fenced areas (Rahate and Shaikh, 2016).

It was essential to consider how notifications are triggered for geo-fenced locations. As Rahate and Shaikh (2016) state, there are multiple ways to trigger them, such as when the user enters and leaves the virtual area or trigger as the user moves throughout the geo-fenced location.
Rahimi, Maimaiti and Zincir-Heywood (2014) discuss that the geo-fencing system should monitor the time the user has spent at the same location known as the dwell time. This was so that the user is not repeatedly sent notifications; for example, if they are not moving or have not left the building, they are inside. It was crucial that the notifications the user receives are valuable and that we are not repeatedly notifying them of the same geofence trigger. Geofences were a focal point of the research project that helped deal with positional accuracy errors that could make it difficult for a user to intersect with a point of interest (POI) directly.

2.2 Existing Applications

In this section, existing reminders applications have been investigated, which fall into two categories: temporal or temporal and spatial. Temporal applications trigger reminders based on a time element that has been associated with a reminder. Temporal and spatial allow users to set a location for a reminder alongside a time for them to trigger. However, none of the applications with spatial capabilities let a user set a generic location type for a reminder to trigger, e.g. a supermarket; a specific location must always be provided.

2.2.1 Temporal

Evernote

Evernote is a popular note-taking application for both mobile and desktop devices. It provides the ability for a user to associate a temporal element to the reminder, allowing them to set a time for the reminder to trigger. However, this does not implement spatially based reminders, a key area of interest for this project. As shown on the Evernote forums, users are interested in this feature, with over a dozen threads requesting this (Evernote, 2018). Users commented, stating that such a feature would help remind them of a "shopping lists when they get to a particular store" or "reminding themselves to do something when they arrive at a client or place of business." The research and implementation highlighted in this document could have a far-reaching impact on both casual and business environments.

2.2.2 Temporal and Spatial

GeoNotes

GeoNotes was an application with the aim of connecting digital information to a physical location using a users mobile device (Espinoza *et al.*, 2001). It tried to do this in the same way that information is attached to post-its, graffiti or signs, with users having to be at the physical location of the note to access this information. As Espinoza *et al.* (2001) states, GeoNotes did not want the information for the system to all come from a central source as it could make the information seem formal or sterile. Instead, all users were encouraged to leave traces in the system, creating a richer, more natural user experience. Users could limit the visibility of their notes if they wanted to allow them only to be visible to friends and family. Although the current research project did not implement social features such as note sharing discussed in GeoNotes, this could be explored in future work.

Geonotes importantly demonstrated the ability for information to be linked to physical locations in the real world and events to be triggered when the user was near.

Google Keep

Google keep a note-taking application developed by Google, allowing the user to easily create versatile text and image-based notes such as checklists and drawings. A feature that was of interest to this project was the ability to associate a location-based reminder to the note. Spatial reminders are implemented by continually detecting the user's position using either GNSS or WiFi (Wang, Wang and Guo, 2019). If the user enters a place associated with a reminder, it will trigger and notify them of the note. Although this is limited to an exact predefined location, if you wanted to be reminded at a supermarket, you would have to specify the individual supermarket.

Apple Reminders

A note-taking application developed by Apple for their ecosystem is Reminders. It expands upon the functionality discussed in Google Keep allowing the user more control over location-based reminders. It allows the user (1) to define the size of the area for which the reminder should trigger (the geofence) and (2) specifics if the reminder should be triggered when the user enters or leaves the selected area (Apple, 2020). These features are shown in Figure 2 below:

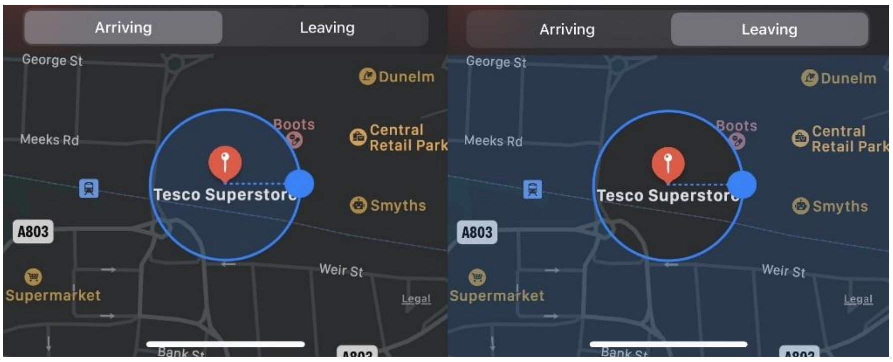


Figure 2: Showcasing user defined geo-fences and the different options with reminders on arrival and departure from the specficied location.

Existing Applications Conclusion

A downside in the approach adopted by both Google and Apple's reminder applications is that the user needs to highlight a specific trigger location rather than having the facility to attach a reminder to a generic location (e.g. park, shop). This was one of the focal points of this research project, allowing a more sophisticated reminder triggering system, i.e. a reminder to buy milk would trigger when at multiple different location types, e.g., a supermarket, corner shop or mini-mart.

2.3 Overview

An overview of how the four upcoming sections are connected is shown in Figure 3. The first section is about the data sets integral to the project, which allowed us to associate meaning with a user's location. Data storage was investigated through the use of a PostgreSQL + PostGIS database. To transfer data from the database to the application, we needed an interface to stream this; different networking streaming options were reviewed. Finally, visualising the data within the application is discussed by comparing different mapping libraries.

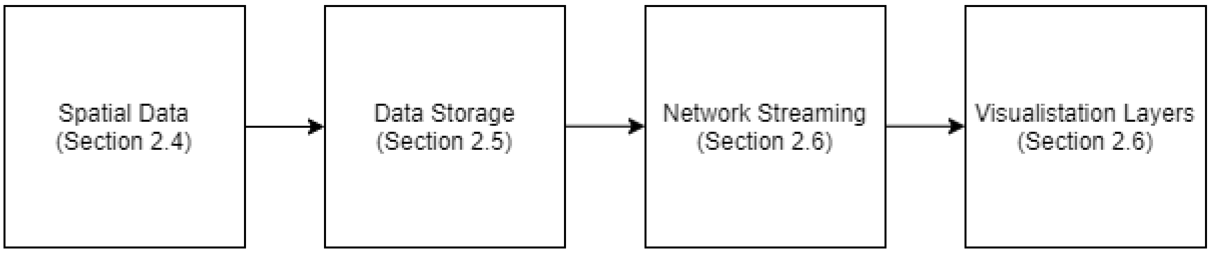

Spatial Data (Section 2.4)
Data Storage (Section 2.5)
Network Streaming (Section 2.6)
Visualistation Layers (Section 2.6)

Figure 3 Showcasing the upcoming sections and the connections between them.

2.4 Spatial Data

The quality of the location data used for this project directly influenced the usefulness of the system. Our knowledge of the real world had to be as accurate as possible as this is how we created the virtual world the user interacts with.

2.4.1 Digimap

Digimap is a service developed by The University of Edinburgh (2020). Digimap makes several Ordnance Survey (OS) data sets available to staff and students in higher education in the United Kingdom for free. The data sets provided offer some of the most accurate mapping data available in the country. The information page on Ordnance Survey data and GPS measurements states that Digimap uses the British National Grid (BNG) reference system when referring to coordinates (Digimap, 2020a). When working out with the BNG reference system, such as when using GPS (WGS84), it is advised that all coordinates are transformed to use the same reference system. It provides faster and more accurate results when the same coordinate reference system is used.

Below is an overview of the two datasets of interest for the project:

- The Points of Interest layer provided multiple attributes for each entry. These include the coordinates, a unique ID for each entry, a classification code and many other attributes (Digimap, 2020c). If the details are available, the business type and address are included. The POI classification is broken down into 3 levels of hierarchy. The most abstract level specifies 9 different types of POIs, such as retail or eating and drinking. This is then broken down further in the lower levels, for instance, specifying if eating and drinking is a cafe or a restaurant.
- The OS MasterMap (OSMM) topography layer provided complete polygonal coverage of the United Kingdom (Digimap, 2020b). For any coordinate position, we can determine the unique OSMM ID and attribute type it has been associated with it. Examples of these attribute types are roads, buildings or paths.

2.4.2 OpenStreetMap

OpenStreetMap (OSM) is a crowdsourced mapping service that provides access to a free editable map of the world (Haklay, 2010). Haklay comments that volunteered OSM data is accurate to about

6 metres on average compared to the position recorded by the Ordnance Survey data in the United Kingdom.

Users can suggest changes in a Wiki-like style with more than 6.25 billion nodes created by the 6.7 million registered users. This data can be downloaded for a whole region, but for this project, the OSM Scotland dataset was downloaded in the WGS84 coordinate system (OpenStreetMap, 2020c), which included two layers of particular interest:

- The OSM POI layer provided us with data similar to the Digimap POI layer (OpenStreetMap, 2020b). Although upon inspecting the POI data, it does not appear nearly as well maintained as the Digimap POI layer and is lacks local POIs.
- The OSM building layer provides us with polygons that have been created that annotate the entire shape of a building (OpenStreetMap, 2020a). These were very well documented and provided useful information about buildings such as the address and what is inside the building, for example, a shop or an office.

The Digimap service provided us access to Ordnance Survey data with a quality standard set by the UK government requiring the data to represent 99.6% of real-world features greater than 6 months old (Haklay, 2010). Whereas OSM has no deadlines or accuracy requirements, it is entirely user-driven, trying to provide the best possible mapping experiences for themselves and others.
Zhang (2018) discusses this idea of creating a data fusion using multiple sources of spatial data to try and improve the performance when detecting buildings. The data fusion they created was with the OSM building layer and satellite images. They found that the combination of data sources outperformed using just one source by 6%. We considered creating a data fusion using both OS and OSM data; however, we did not deem it worthwhile to create due to the low-performance improvement and limited time for development.

2.5 Data Storage

For this project, we needed a way to store the spatial data described in section 2.4 and be able to query it. A PostgreSQL database was investigated to do this with the added support of storing and querying spatial data through the PostGIS database extension.

2.5.1 PostgreSQL

PostgreSQL is an open-source “object-relational database management system.” It primarily supports the SQL standard and offers modern features such as complex queries and updatable views. As Regina and Leo (2011) state, it functions like an enterprise database handling terabytes of data with ease and complex SQL queries.

2.5.2 PostGIS

PostGIS (Geographic information system) is an extension for PostgreSQL databases that added support for storing spatial objects and implements new spatial functions (PostGIS, 2020a). It has two functions that were of particular interest for this project; the first is converting the user's location between spatial reference systems. To do this, it offers a function ST_Transform which takes a geometry point and the new reference system ID and returns the geometry converted to the new spatial system (PostGIS, 2020c). Finally, it offers a function ST_DWithin which takes two geometries and a precision value (the buffer around the two points) (PostGIS, 2020b). This function showcases how a geofence was implemented using PostGIS using the user's current location and the known location of POIs as the parameters to this function.

2.6 Streaming and Visualising Map Data

We reviewed the different ways spatial data can be represented, highlighting the key differences between vector and raster data. Interface solutions were evaluated that allowed us to transfer spatial data from the database to the application. Different libraries were then explored that allowed us to visualise this data within the mobile application.

2.6.1 Vector and Raster Data

The two primary types of spatial data in GIS are vector and raster. Vectors represent geographical features using points, lines and polygons (QGIS, 2020b). Points can be used to describe a specific point of interest, such as a mailbox or a lamppost. Line features describe roads and paths, and polygons represent boundary areas such as buildings or fields. Raster data is described as being similar to image data, consisting of a grid of pixels representing mapping data (QGIS, 2020a).

Figure 4 highlights the differences between Vector and Raster data. The first image (a) shows how data described in vectors scales exceptionally well. Image (b) shows mapping data in raster format, which will appear pixelated when resized, as shown in the image (c) (D'Roza and Bilchev, 2003).

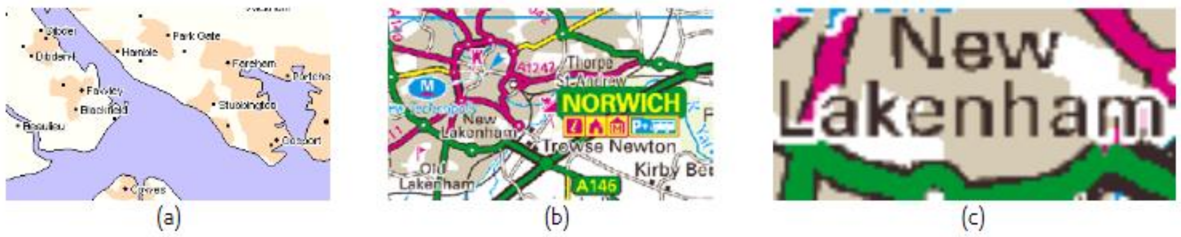


Figure 4: Map data as (a) vector, (b) raster and (c) pixelated raster after resizing (D'Roza and Bilchev, 2003).

2.6.2 Streaming Data

Middleware is software that sits between the database and the application that allows spatial data to flow freely between them. It reduces the strain on the application by offloading the retrieval of data from the database to the middleware. Middleware can also preprocess and aggregate the data using the database backend before retrieval (Maduako, 2012). The Open Geospatial Consortium (OGC) outline standards that aim to improve how data is shared across the Internet (Michaelis and Ames, 2008). Two standards were of interest for this research project:

- Web Map Service (WMS): Describes how a client requests map images, stating that map images are returned in a standard format (e.g. JPEG) and that no geographic information is attached.
- Web Feature Service (WFS): Describes how the client requests map vector data that has not been assembled into a standard image format like in WMS. Assembling it typically reduces the accuracy and quality of the data.

An example of a middleware that implements the above OGC standards is Pg_tileserv (2020). Pg_tileserv is a lightweight implementation, stripping away the majority of the overheads, relying purely upon HTTP and HTTPS requests; however, it is limited to work exclusively with GIS data. Upon receiving a request, SQL on the database is executed, returning a map tile (vector data). Map tiles represent a smaller chunk of the overall data set that can quickly be rendered by mapping libraries on the application side (Crunchy Data, 2020).

2.6.3 Visualising Map Data

Once the mapping data was streamed to the application, we needed a way to visualise this. Mapping libraries were be used to accomplish this; two possible solutions were reviewed below:

OpenLayers

OpenLayers is an open-source Javascript library for displaying dynamic maps on web pages (OpenLayers, 2020b). It provides a rich API with an extensive feature set supporting mapping features such as vector and tiled layers. OpenLayers is designed to easily integrate with GIS applications as it takes coordinates inputs as [longitude, latitude], matching how the data is stored in a PostGIS database (OpenLayers, 2020a).

Leaflet

Leaflet is a popular library for creating mobile-friendly dynamic maps. It has leaner functionality compared to OpenLayers, with missing features added through extensions (Leaflet, 2020). Leaflet is also written purely in JavaScript, is open source and is free to use. The community for Leaflet is also larger; for instance, searching “Leaflet StackOverflow” returns 314,000 results versus 64,900 results for OpenLayers.

There are other methods of visualising mapping data, such as Google or Bing Maps. These services only provide a limited feature set for free and modify their interfaces frequently, which can break implementations. They include some extra features like street view, though missing this was not detrimental to the project's success. Due to the fact, OpenLayers is designed with PostGIS data in mind and the fact it has built-in support for Pg_tileserv, for this research project, it was the starting point when visualising map data.

2.7 Events

Location-based services can be one of two types, either push or pull. In a push-type service, the user receives information from the system without explicitly requesting it. The user could have previously subscribed to this; for example, the user could have created a shopping list which is later retrieved when the user is inside a store. Examples of push-type services have been discussed, such as GeoNotes and Apple Reminders. Pull is the opposite; the user must explicitly request the information (Rahate and Shaikh, 2016). Thus the location reminder application will be a push-type service. Discussed below is a range of topics concerned with triggering the push events.

2.7.1 Temporal Triggers

As Chin (2011) states, it is far more convenient for a user to be reminded when they are at the location where they can complete a reminder rather than at a predetermined date or time. However, this will not always be possible as some things cannot just wait until we are naturally near a location to complete them. For this reason, the user should be able to set a time-based reminder alongside a location-based reminder.

It would also be essential to consider the time when triggering events using the user's location. Axelrod and Fitoussi (2014) discuss the idea of having “blocked time periods” in which events cannot be triggered. An example of this would be if a user wanted to buy a product from a shop, the user should only receive a notification reminding them if the shop is currently open.

2.7.2 Line of sight based triggers

Understanding what the user can currently see would be extremely powerful in deciding if an event should be triggered or not. Bartie *et al.* (2018) covered a system for tourists in Edinburgh to help them explore the city. The system successfully identified statues and buildings with high accuracy by modelling what the user could see in real-time using LiDAR data. Users could trigger a geo-fenced area and receive a notification about a reminder that they cannot see. An example of this could be if the user is on a bridge; they might not see what is below them or on a parallel road behind a row of shops.

There are different ways to measure the distance from a user's location to a point, e.g. the Euclidean distances (straight line) and the Network distance (roads between points). Due to the fact, the Euclidean distance measures in a straight line from point A to B causes the scenario discussed above with the user potentially receiving notifications for reminders they cannot see. An approach to solve was using a network distance, which measured the shortest path between the two locations using a network (Buczkowska, Coulombel and de Lapparent, 2019). However, this does have issues in urban areas due to multiple roads having access to the same POIs.

2.7.3 Notifications

Getting notifications right was extremely important, as Felt, Egelman and Wagner, (2012) state. If an application continually sends notifications that are not deemed valuable, this can lead to users becoming frustrated and ignoring them. In the worst-case scenario, this can lead to the user deleting the offending application. A study conducted by Sahami Shirazi *et al.* (2014) revealed that notifications from communication and reminder application like calendars are deemed more important to users than other categories of applications. This study confirms that the use of notifications should be used to alert users of reminders. Below a review is conducted into the effectiveness of different types of notifications commonly found in smartphones.

Push Notifications

Push notifications are textual notifications that appear like text messages displayed when the user's device is locked or unlocked. They could be used to display a reminder when a user enters a geofenced location. Alkhaldi *et al.* (2016) highlighted that push notifications are shown to increase users repeated use of an application. This also meant that push notifications had a positive impact on the increased use of application features. Users could be more likely to interact with location-based reminders if they are delivered through the use of push notifications; therefore, for this project, the use of push notifications was investigated.

2.7.3.1 Audio Notifications

Audio notifications were utilised in this project, audibly alerting a user when they enter a geo-fenced area associated with a reminder. Garzonis, Bevan and O’Neill (2008) discuss that generic audio notifications are virtually useless when informing the user about specific information like a reminder. To overcome this issue, they proposed using a unique sound for the reminder, allowing the user to associate a meaning to the sound; this was implemented in this project. An example of this is the sound of an old fashioned landline telephone ringing commonly associated with receiving a phone call on a modern smartphone.

Haptic Notifications

Finally, haptic notifications such as vibrations were implemented, alerting the user of a reminder when they enter a geo-fenced area. Wang, Millet and Smith (2016) discussed that vibration notifications are an essential component of mobile notifications systems, as they can be used to get the users attention when social or environmental factors preoccupy their attention. However, vibration notifications suffer from being generic; users cannot differentiate one vibration from another.

2.8 Places and Spaces

For this project, we needed a way to create a link that considered all generic location types the same, e.g. all supermarkets are considered the same for triggering reminders. An ontology is investigated to do this using the predefined classification from the OS POI layer discussed in section 2.4. A folksonomy is also explored, allowing users to search for reminder locations that other users have classified.

2.8.1 Ontology

When users enter the name of a place into a search engine, the results returned favour those with the original text in their name. Therefore users are potentially missing out on related places that are nearby or referred to by a different name (Jones *et al.*, 2002). For this project, it was essential to map between places and spaces, creating a connection between them to minimise the above scenario. Marmasse and Schmandt (2000) discussed that users find it much more valuable if they can identify a set of coordinates as meanings, for example, a shop or office. It would be compelling if the user could set a reminder to buy a product inside a supermarket instead of specifying the individual location. This mapping was created through the use of an ontology.

This ontology allowed the user to set a reminder as a type of location. The Ordnance Survey POI data set referred to in section 2.4 already contains a classification system (McClune, 2019). The documentation specifies information about the 3 different levels of classification:

- Level 1 comprised 9 groups and provided the broadest categorisation.
- Level 2 breaking down the 9 groups into 52 categories.
- Level 3 breaking down the 52 categories into over 616 classes.

This then allowed the following break down to happen in the discussed example of buying a product from a supermarket:

- Group: 09 = Retail
- Category: 47 = Food, drink and multi-item retail
- Class: 0819 = Supermarket chains

Marmasse and Schmandt (2000) discussed solutions like Apple Reminders and Google Keep, where users are having to search for the location on a map and then associate a reminder to it. He further stated that users do not want to spend their valued time searching on a map for the exact location of where they would like a reminder to trigger. This documentation provided an excellent starting point for implementing an ontology, a key area of interest for this research project.

2.8.2 Folksonomy

Folksonomy is a classification system that allows users to freely associate tags with a set of data to make these classified items easier to find for themselves or others in the future (Peters and Stock, 2007). Compared to ontologies, this classification data is created entirely by the users and not by a central source. As Peters and Stock (2007) state, Flicker currently uses a Folksonomy to classify images, with the most popular images appearing first in the search results. When a user searches for an image, the results shown are influenced by what other users deem to be appropriate.

Sinclair and Cardew-Hall (2008) found that tags created through folksonomies had a positive impact on users searching over large datasets:

- Users commented that it helped when they were unsure where to begin their search. They found the suggested tags helpful in then narrowing down their search.
- Users searching using suggested tags found it 'easier than thinking about query terms and then manually searching for them.

They concluded that solely searching using a folksonomy based recommendation system was not sufficient when searching for specific information. Users still preferred to use a traditional search method that they could manually refine.

These results were incredibly relevant to this research project. When users are creating reminders, recommendations of locations to complete these tasks could be provided based upon what previous users have selected. Of course, as discussed in the results from Sinclair and Cardew-Hall (2008), this recommendation might not always be correct, and the user should be able to input their own type of location as a reminder. Sommaruga, Rota and Catenazzi (2011) comment on how Folksonomies currently face challenges due to a lack of precision and ambiguity naturally occurring as people create them. To counteract these issues, a popularity/voting system could be considered similar to

Flicker. When users agree with a suggestion, this will increase the likelihood of the recommendation being provided to someone else and similarly disagreeing, decreasing the likelihood.

3 Implementation

3.1 Version Control and Data Backup

Throughout the development process, Git was used, providing version control functionalities. Git was used in conjunction with a GitHub repository to host the project and act as a backup to protect against local data loss. The repository was also automatically backed up using Dropbox in case of any issues arising with GitHub.

3.2 React Native Framework

The React Native framework was selected to implement the front end portion of this project. It had all of the components predefined and well documented that were needed to make the user interface, e.g. a list, buttons and a map. In addition, it also allowed us to create an application for Andriod and iOS using a single codebase which, for the most part, worked seamlessly, with just a few disparities between the different mobile operating systems.

There were some initial development difficulties whilst learning React Native due to its steep learning curve:

- It does not wait for asynchronous API calls to finish before rendering components, so initially, when working with APIs, lots of components were throwing null value exceptions. Two solutions to solve this problem were found: rendering a temporary skeleton component whilst the API call resolves or rendering the component null.
- Its use of state to re-render components automatically.

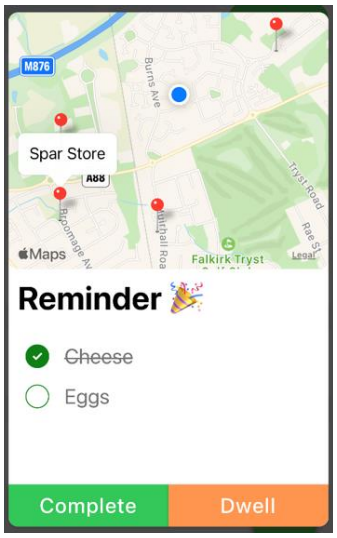


Figure 5 Map showing nearby POIs.

Rendering a component as null was used for the map inside the application; if the API did not find any nearby POIs, we did not want to render anything, e.g. a spinner to show a component is loading, leaving the user unable to use the application until a POI was found. However, returning null meant the user could still use the application, but when we found a nearby POI, the map component would instantly re-render, as shown in Figure 5.

3.3 Expo Development Tools

The Expo Development tools have been used to launch this project, handling all of the native dependency files for Android and iOS that we would have otherwise had to have managed ourselves. The use of the Expo development tools also made testing the application incredibly easy. Expo provides a mobile application for Android and iOS that allowed us to test the application natively. To do this, the command `expo start` is run from the root of the project directory, resulting in a QR code being generated in the terminal that can be scanned, downloading the application onto the phone. A webpage is also opened when this command is run, providing the same QR code and more configuration options such as production mode or control over the network connection options, e.g. Tunnel, LAN (default) and Local Network. Whenever an update is made to the codebase, this is instantly reflected on the mobile device providing an excellent development environment.

However, the flexibility and ease of using the Expo Development tools did not come without a cost and had limitations that impacted the final application. To use the background task scheduler, for instance, to fetch results from an API to find nearby POIs when the application is not in the forefront, a custom Expo Go build must be used. The Expo Go build does not currently support push notifications; thus, push notifications could not be implemented for the application. However, this was the only major hurdle when using Expo, and development would have been much slower without it, so the trade-off was undoubtedly worthwhile.

3.4 Application Frontend

Upon opening the application, the user is presented with the entry screen shown in Figure 6.

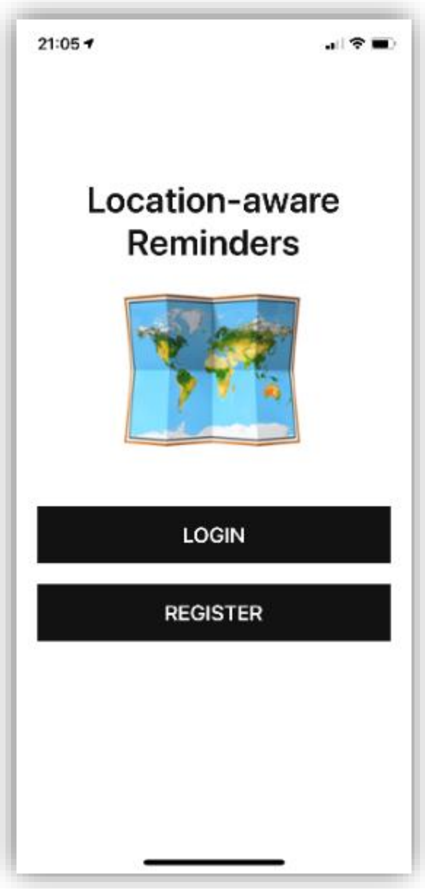


Figure 6 The entry screen of the application allowing the user to login or register.

The user can select to login or register when they are presented with the entry screen. When selecting to register, they are transferred to the account registration screen, as shown in Figure 7. To create an account, the user must enter their name, email address, confirm a password, and then finally click register. Upon completing registration, they are automatically transferred back to the entry screen, where they can then click login to be transferred to the login screen, shown in Figure 8.

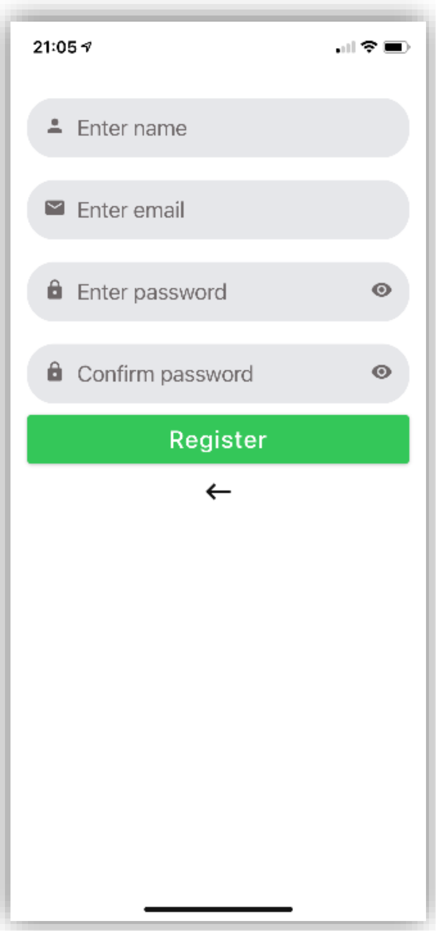


Figure 7 The registration screen.

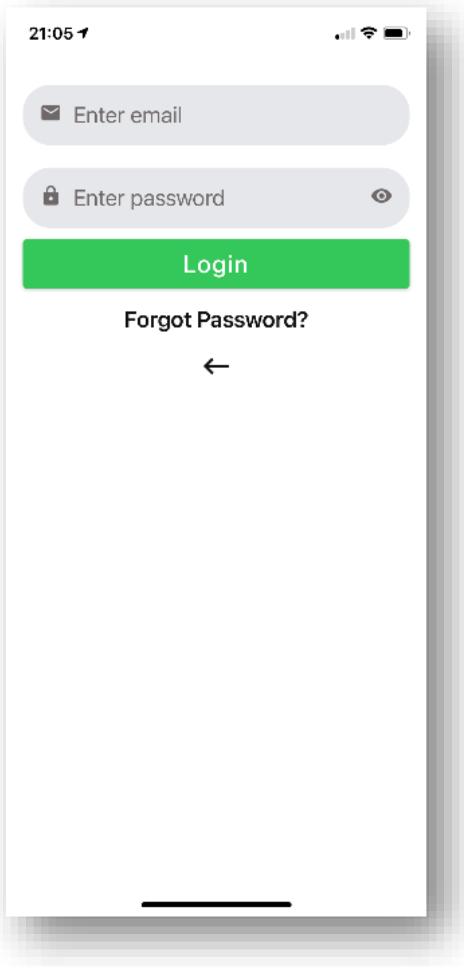


Figure 8 Login screen.

Once the user has logged into the application, they are presented with the home screen shown in Figure 9. It displays their currently active reminders and the ability to control how far they would like to search for POIs (close = 100 metres, standard = 500 metres and far = 1000 metres). Using the tab navigator at the bottom of the screen, the user can switch screens to view their completed reminders, as shown in Figure 10.

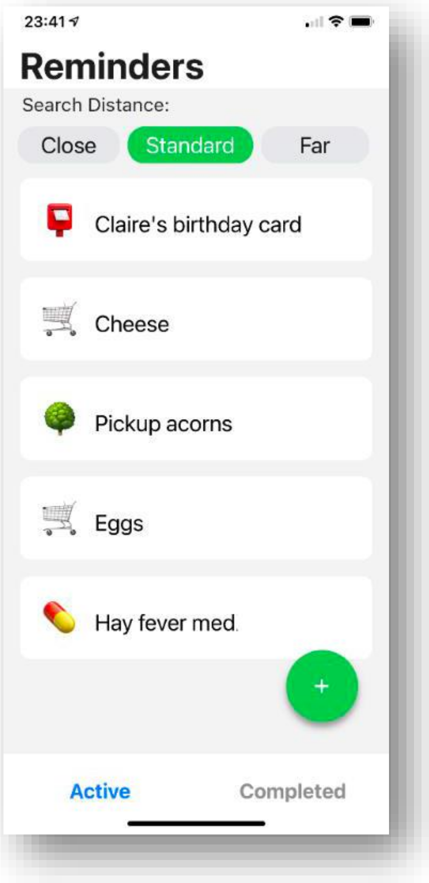


Figure 9 Home screen with active reminders, search distace and add new reminders button.

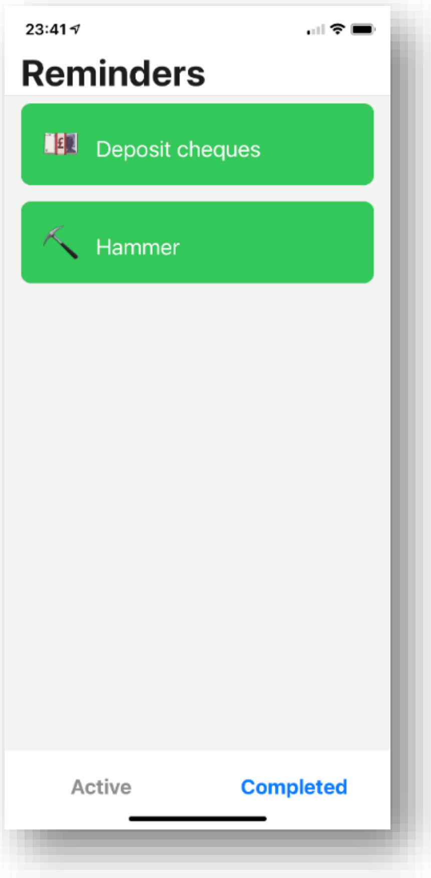


Figure 10 Homescreen showing completed reminders.

To add a new reminder, the user can click the green plus button shown on the home screen; they are then transferred to the create reminder screen shown in Figure 11. Creating a new reminder requires the message along with the location type of the reminder. The location type refers to the ontology class; for instance, if a food shopping reminder is created, this would trigger when the user walks past a supermarket, corner shop or mini-mart. Pressing the add reminder button creates the reminder and returns the user to the home screen. If the user presses the cancel button, this reminder is discarded, and the user is returned to the home screen.

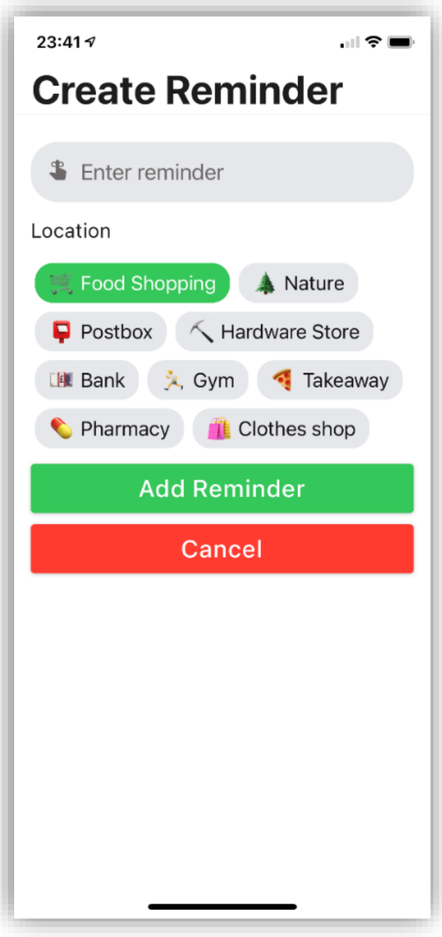


Figure 11 Create reminder screen.

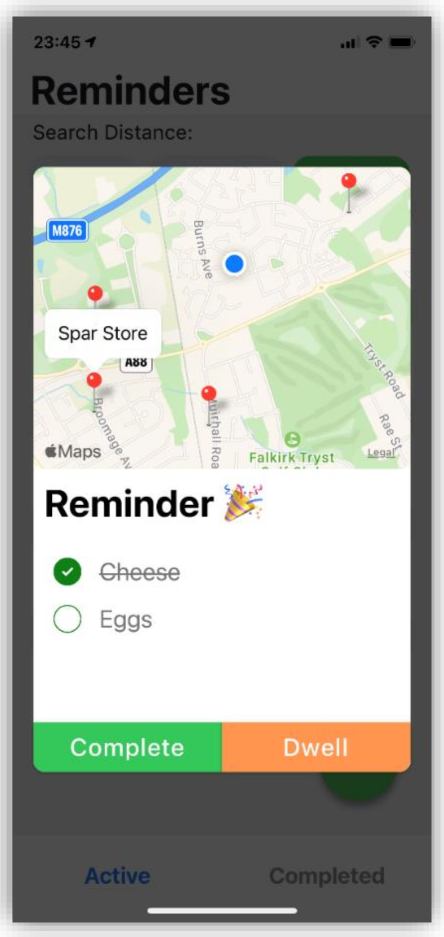


Figure 12 The modal showing the map of nearby POIs with the associated reminders below.

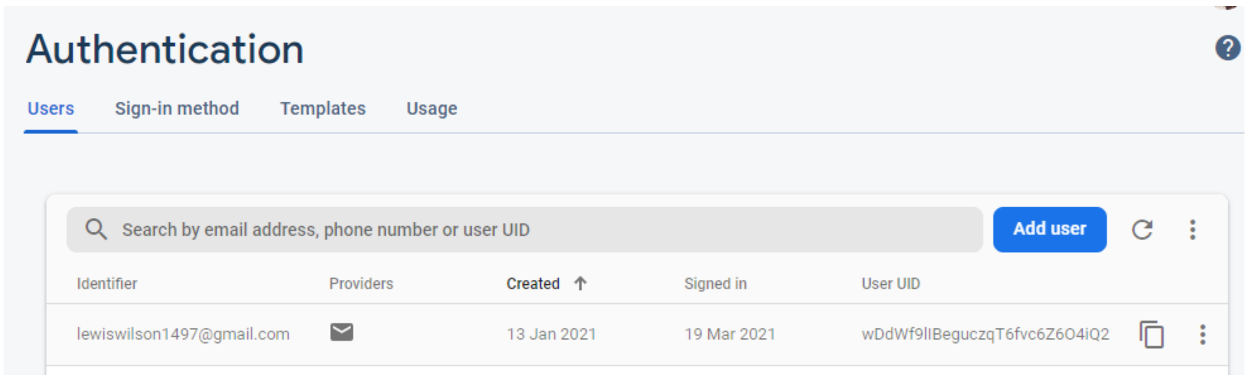
If the user crosses a geofence for a POI/s that they have an active reminder for, a modal appears (Figure 12), displaying a map showing the user's current location and pins representing the nearby POI/s. Below this, the user can see the reminders associated with these POI/s; they can mark reminders as complete by selecting the reminder, which scores through the text and displays a green tick. Then pressing the complete button will close the modal and move any completed reminders to the homepage's completed reminder tab. However, if the user cannot complete a reminder, they can press the complete button closing the modal. If the user wants to take a break, the dwell button can be pressed, pausing all reminders for a period of time. Until the user confirms the status of the reminders, the application will not retrigger the modal as it stops checking for nearby POI/s whilst this is open.

3.5 Application Backend

Due to the nature of this project being a mobile application, the project's backend needed to be constantly available; for this reason, self-hosting was disregarded, and cloud solutions explored. The implementation features two backends; Google's Firebase service responsible for user authentication and storing the user's reminders, and Amazon Web Services (AWS) to host the API and the Postgres database.

3.5.1 Firebase Authentication and Cloud Firestore

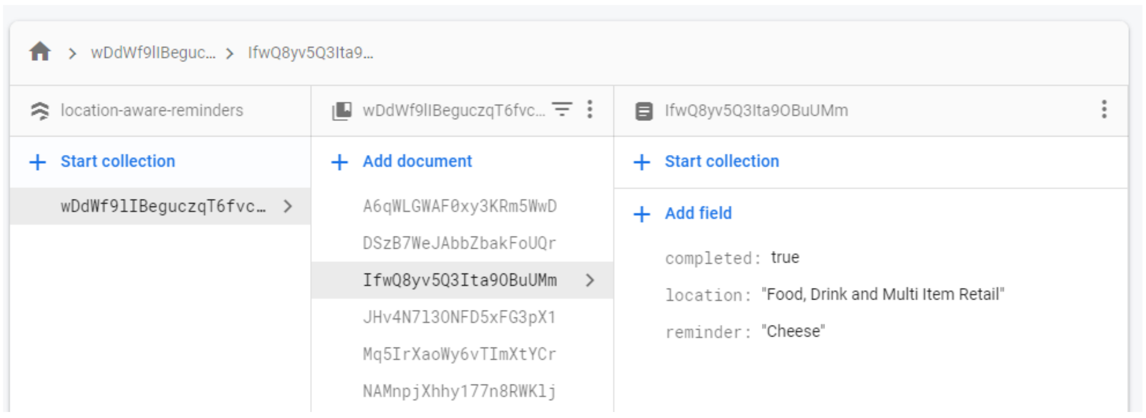
Firebase Authentication is responsible for handling user authentication within the application and was selected due to its rich feature set, built-in encryption and automatic password hashing. It allowed us to develop a fully-fledged account registration and login system for the application very quickly. In addition, it has all the modern features expected, such as email verification upon account registration and reset password functionality fully implemented out of the box.



Identifier	Providers	Created	Signed in	User UID
lewiswilson1497@gmail.com	✉	13 Jan 2021	19 Mar 2021	wDdWf9lIBeguczqT6fvc6Z6O4iQ2

Figure 13 Firebase Authentication showing a registered user.

The Cloud Firestore was used to store the reminders a user creates; it creates a unique collection for each user based on their unique user ID. For each reminder, a unique document is created, shown in Figure 14, detailing if the reminder has been completed, the reminder location (the ontology class) and the reminder text. Firestore implements listeners that fetch a copy of all of the documents whenever an update occurs; this allowed us to automatically re-render the home screen (Figure 9) whenever a user adds a new reminder or a reminder is completed (Figure 10).


wDdWf9lIBeguc... > IfwQ8yv5Q3Ita9...

location-aware-reminders | wDdWf9lIBeguczqT6fvc... | IfwQ8yv5Q3Ita9OBuUMm

+ Start collection | + Add document | + Start collection

wDdWf91IBeguczqT6fvc...

A6qWLGWAF0xy3KRm5WwD
DSzB7WeJAbbZbakFoUQr
IfwQ8yv5Q3Ita90BuUMm
JHv4N7130NFD5xFG3pX1
Mq5IrXaoWy6vTImXtYCr
NAMnpjXhhy177n8RWK1j

+ Add field

completed: true
location: "Food, Drink and Multi Item Retail"
reminder: "Cheese"

Figure 14 Firebase Cloud Firestore with a user's reminders.

3.5.2 Amazon Web Services API Gateway, Lambda and Relational Database

Amazon Web Services (AWS) provided us with the tools to create the backend for the application's geospatial functionality, hosting the API Gateway, the middleware between the database using Lambda, and finally hosting the PostgreSQL database using their Relational Database Service (RDS).

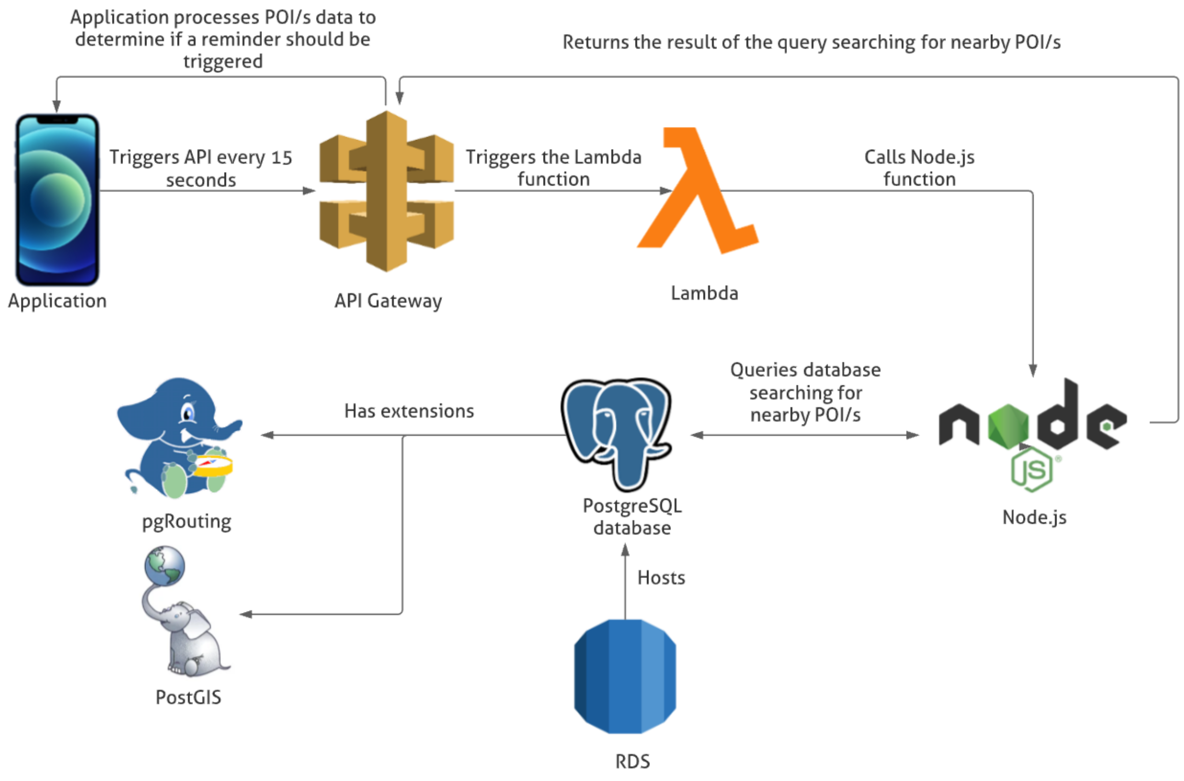
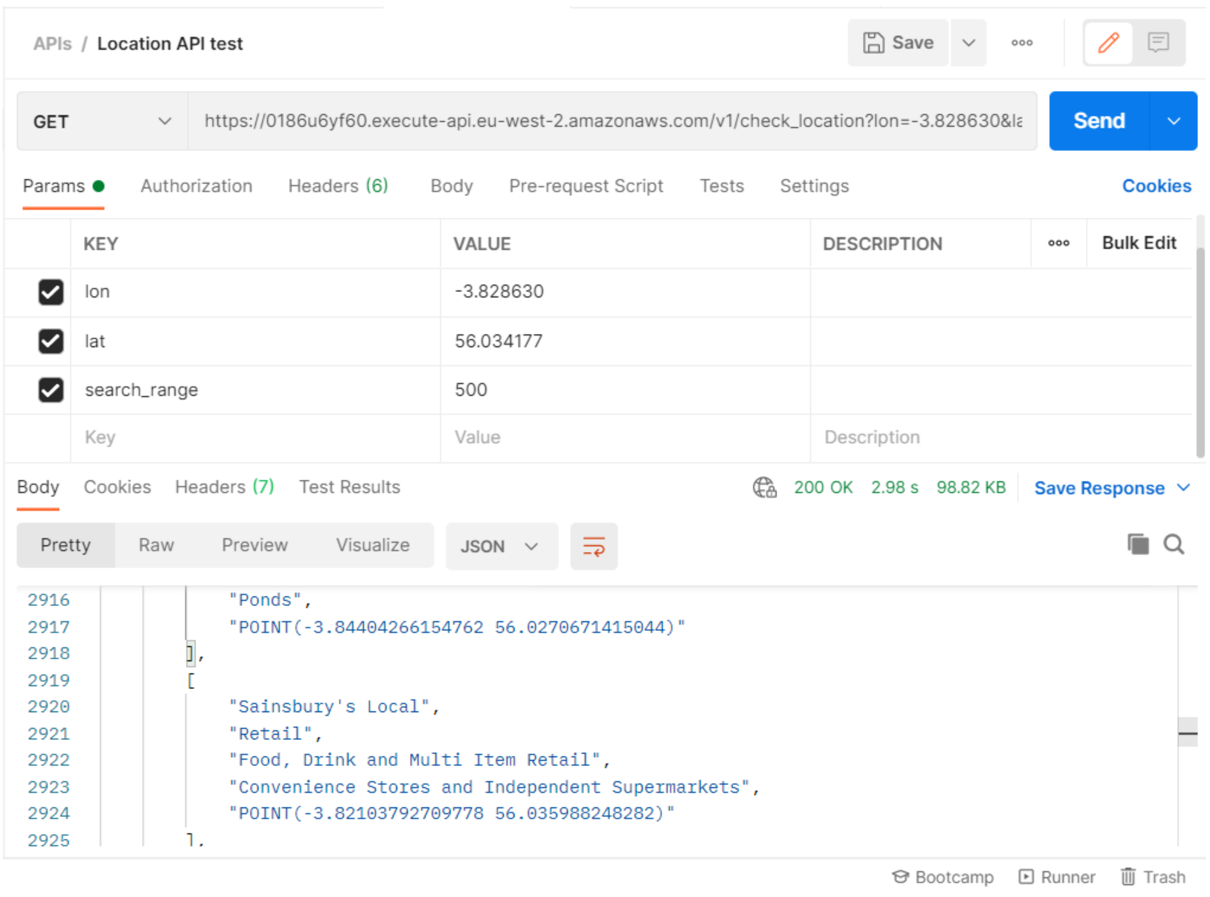

Application processes POI/s data to determine if a reminder should be triggered
Returns the result of the query searching for nearby POI/s
Triggers API every 15 seconds
Triggers the Lambda function
Calls Node.js function
Application
API Gateway
Lambda
Has extensions
Queries database searching for nearby POI/s
pgRouting
PostgreSQL database
Node.js
Hosts
PostGIS
RDS

Figure 15 Backend design architecture for the geospatial portion of the application.

The API Gateway was used to trigger the Lambda functions that query the database. Multiple APIs were implemented to find nearby POIs using different geospatial approaches. Each API requires three parameters, the user's latitude, longitude and search distance. An example of POI data returned from the API is shown in Figure 16, which was processed in the application to trigger reminders.



```
https://0186u6yf60.execute-api.eu-west-2.amazonaws.com/v1/check_location?lon=-3.828630&la
```

KEY	VALUE	DESCRIPTION
lon	-3.828630	
lat	56.034177	
search_range	500	
Key	Value	Description

200 OK 2.98 s 98.82 KB

```
            "Ponds",
            "POINT(-3.84404266154762 56.0270671415044)"
        ],
        [
            "Sainsbury's Local",
            "Retail",
            "Food, Drink and Multi Item Retail",
            "Convenience Stores and Independent Supermarkets",
            "POINT(-3.82103792709778 56.035988248282)"
        ],
```

Figure 16 Postman API testing software, showing an example of requesting POI data from the API using the provided location and search range parameters.

Lambda Functions allowed us to run code in the cloud when an event is triggered; in this case, the event was an API call. Node.js was selected as the middleware to query the database using the node-postgres library; an example of using the node-postgres library to run a query is shown in Figure 17.

```
const result = await client.query({ 'SELECT ...'
```

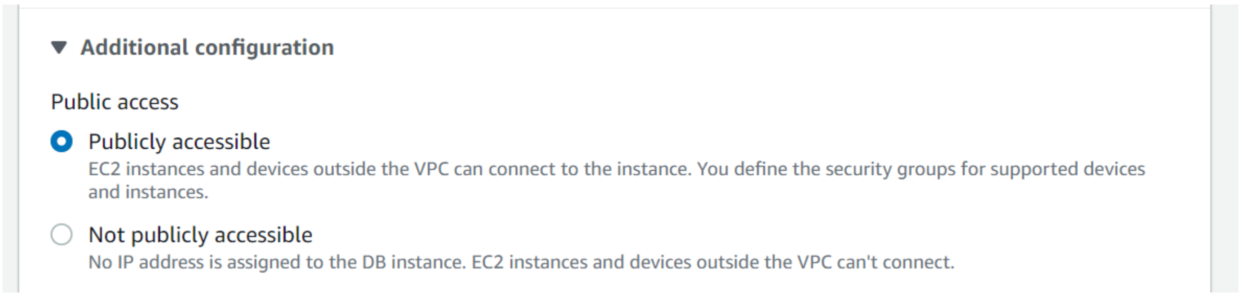
Figure 17 Storing the result of querying the database using the node-postgres library.

An event JSON object is created when an API is triggered, which can be interrogated, shown in Figure 18, to get the parameters passed to the API. These parameters represent the user's location and the current search distance which can then be used within queries.

```
const lon = event['params']['querystring']['lon'];
const lat = event['params']['querystring']['lat'];
const search_range = event['params']['querystring']['search_range'];
```

Figure 18 Storing the parameters from the API request.

Finally, RDS was used to host the PostgreSQL database, which comes preconfigured with a wide range of extensions preinstalled, such as PostGIS and pgRotuing. Security permissions were altered to access the database as, by default, no inbound or outbound traffic is permitted outwith the AWS Virtual Private Cloud (VPC). The database configurations were modified to allow connections from outside the VPC, shown in Figure 19. This database stores no personally identifiable data and is used strictly for the geospatial aspects of the project.



Additional configuration

Public access

[✓ Publicly accessible]
EC2 instances and devices outside the VPC can connect to the instance. You define the security groups for supported devices and instances.

Not publicly accessible
No IP address is assigned to the DB instance. EC2 instances and devices outside the VPC can't connect.

Figure 19 Updating AWS VPC public access settings.

3.6 Datasets used

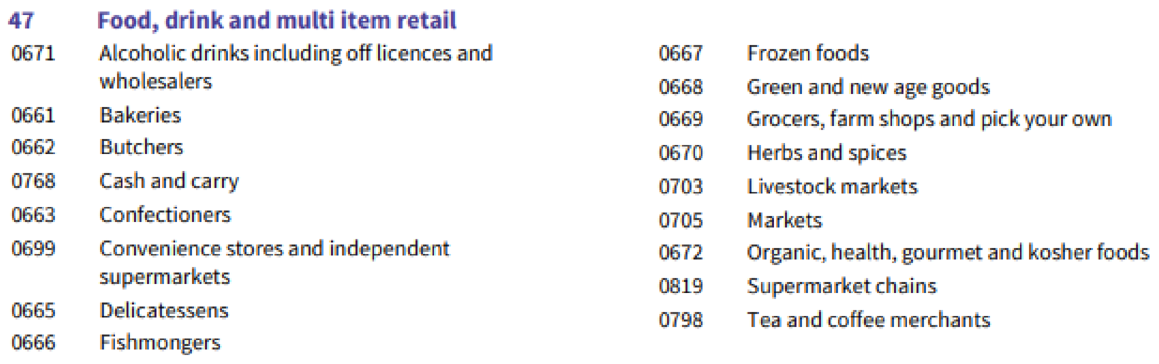
As discussed in section 2.4, the following data sets have been downloaded and used in this project:

- OS MasterMapTopography Layer, providing complete topographical coverage of Scotland.
- OS Points of Interest Layer, providing us with information about local POIs and a predefined classification system.

When downloading the POI layer, only a tiny cut out can be obtained at a time; due to development and testing being conducted in Falkirk, Scotland, only this section of POI data been downloaded.

3.6.1 Ontology

The OS POI layers predefined classification system was used as the ontology for the application, allowing us to consider all places of the same type equal, e.g. all supermarkets are the same. A small part of this classification is shown in Figure 20.



47	**Food, drink and multi item retail**		
0671	Alcoholic drinks including off licences and wholesalers	0667	Frozen foods
0661	Bakeries	0668	Green and new age goods
0662	Butchers	0669	Grocers, farm shops and pick your own
0768	Cash and carry	0670	Herbs and spices
0663	Confectioners	0703	Livestock markets
0699	Convenience stores and independent supermarkets	0705	Markets
0665	Delicatessens	0672	Organic, health, gourmet and kosher foods
0666	Fishmongers	0819	Supermarket chains
		0798	Tea and coffee merchants

Figure 20 Cutout of the Food, drink and multi item retail POI class defined by the OS.

All of the reminder location types featured in the application (Figure 21) have been linked with one of the predefined classes, e.g. food shopping was associated with the class shown in Figure 20: Food, drink and multi item retail. When a user creates a reminder, the application then starts checking if any of the nearby POIs found are of that class; if any are found, all related reminders are triggered.

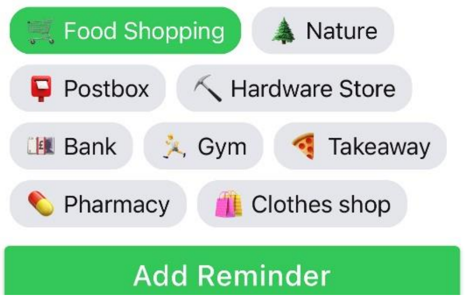
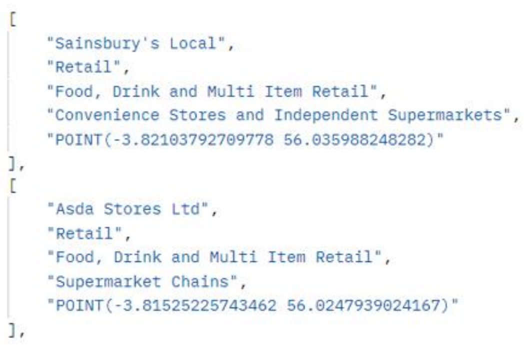


Figure 21 Reminder location types a user can select for a reminder.

Figure 22 shows an example of two different POIs returned from the API, Sainsbury's and Asda, that would both trigger reminders associated with the food shopping reminder location—successfully treating different physical places, i.e. different supermarket chains, as equal for triggering reminders.

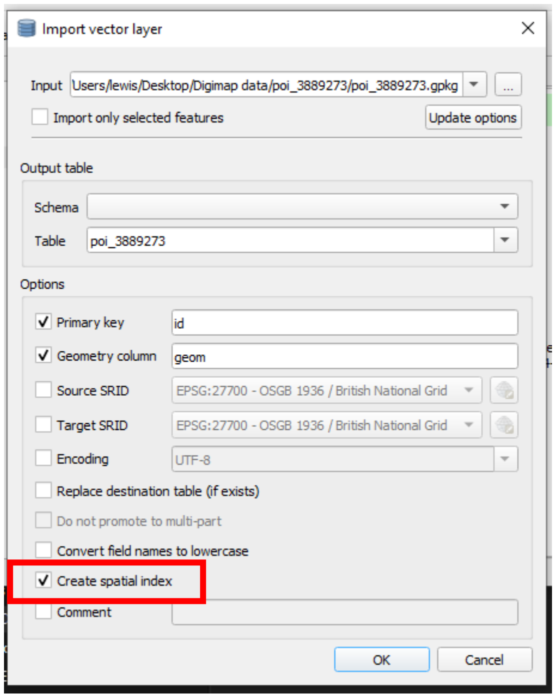


```
[
    "Sainsbury's Local",
    "Retail",
    "Food, Drink and Multi Item Retail",
    "Convenience Stores and Independent Supermarkets",
    "POINT(-3.82103792709778 56.035988248282)"
],
[
    "Asda Stores Ltd",
    "Retail",
    "Food, Drink and Multi Item Retail",
    "Supermarket Chains",
    "POINT(-3.81525225743462 56.0247939024167)"
],
```

Figure 22 API results for two different food shops featuring the same ontology class.

3.6.2 Spatial indexes

To upload the data layers to the database, we used the desktop application QGIS, which has a built-in database manager. When uploading the layers, it provides the option to create a spatial index, shown in Figure 23, which we opted to do, increasing the speed of data retrieval operations.



Import vector layer

Input Users/lewis/Desktop/Digimap data/poi_3889273/poi_3889273.gpkg

☐ Import only selected features

Output table

Schema

Table poi_3889273

Options

☑ Primary key: id

☑ Geometry column: geom

☐ Source SRID: EPSG:27700 - OSGB 1936 / British National Grid

☐ Target SRID: EPSG:27700 - OSGB 1936 / British National Grid

☐ Encoding: UTF-8

☐ Replace destination table (if exists)

☐ Do not promote to multi-part

☐ Convert field names to lowercase

☑ Create spatial index

☐ Comment

Figure 23 Using QGIS software to upload the POI layer to the database, and opting to creating a spatial index.

3.7 Using location to trigger reminders

To trigger reminders based on the user's location, we had to consider the location data's noise and different methods to calculate the user's distance from nearby POIs. The first method to calculate this distance was the Euclidean distance; this method draws a straight line between two points and calculates the distance; the orange line in Figure 24 is an example of this. The second method was to calculate the network distance using Djsiktras shortest path algorithm to calculate the distance/cost between two points. It uses a network made up of roads, paths and walkways to do this; the blue line in Figure 24 represents an example of the shortest path found that can then be used to calculate the distance between the points.

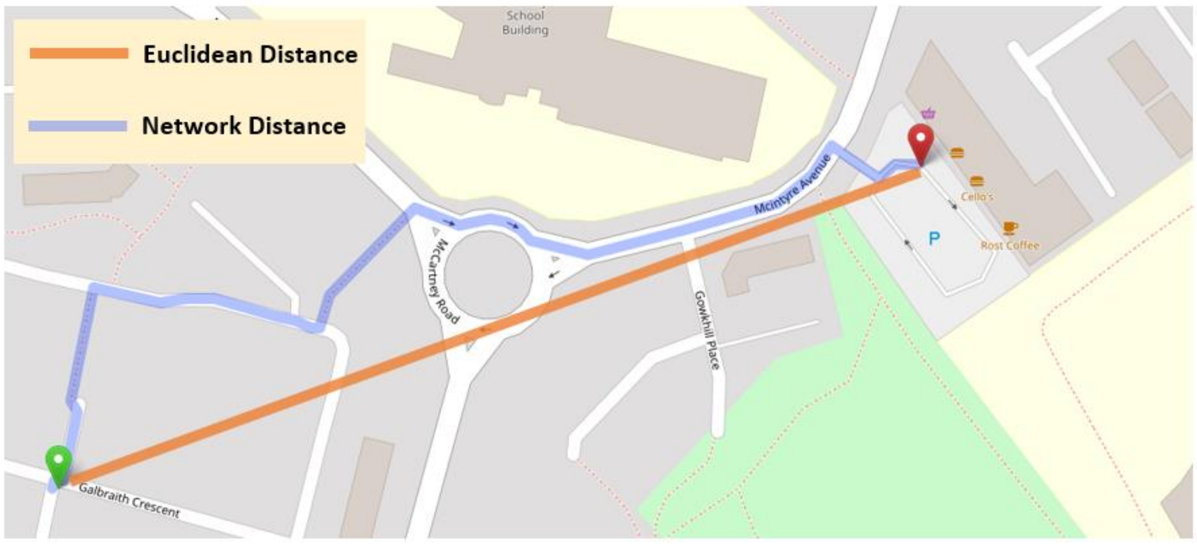


Figure 24 The Euclidean distance compared to the Network distance.

3.7.1 Noise in positioning data

When developing and testing the mobile application and from the literature review conducted in section 2.1.1, it quickly became apparent that a buffer would be required when working with GPS data as it is inherently very noisy. Figure 25 shows a recording of location data gathered using the mobile application Strava; we were standing stationary at the red point for 15 seconds. Due to this noise, it would be very unlikely for a user to intersect directly with a POI, causing a reminder to trigger; thus, a buffer was implemented to resolve this. The decision was made to create the buffer around the user, but we could have implemented it by creating a buffer around each POI. This approach requires us to store less information about each POI, which could be impactful if this project was to be expanded. An example of the buffer around the user can be seen in Figure 25.

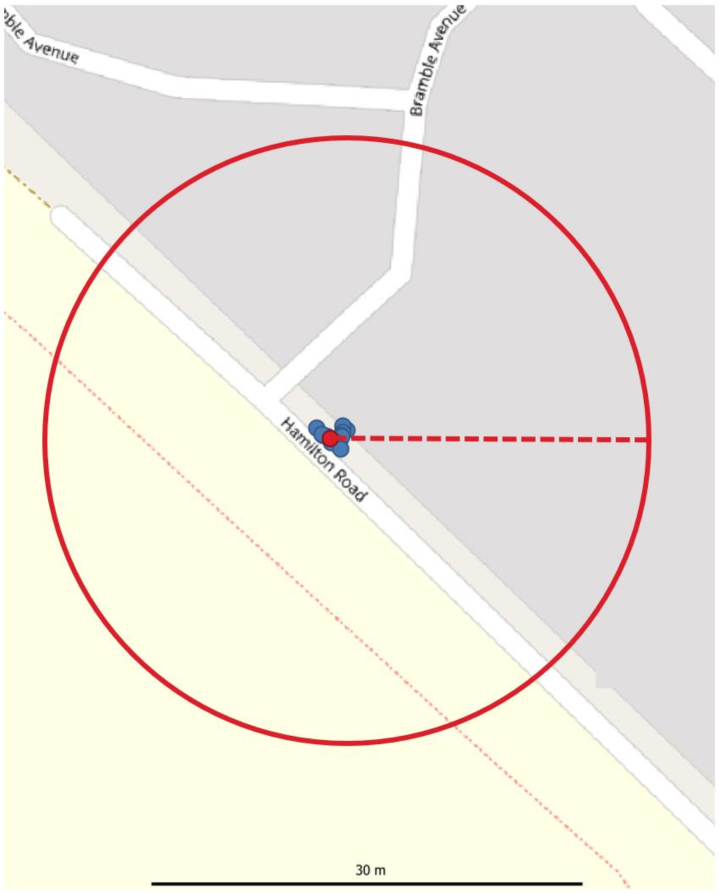


Figure 25 Standing stationary on the red point recording location data for 15 seconds, highlighting the noise in GPS data. Showing the use of a buffer around the user, making it easier to intersect with POIs and trigger reminders.

3.7.2 Converting WGS84 to BNG

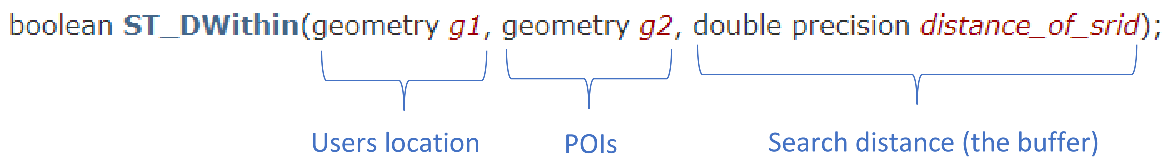
GPS was used to determine the user's location, which uses the spatial reference system WSG84 (4326). However, the OS POI and MasterMapTopography layers use the BNG reference system (27700), so we had to transform the GPS location to BNG to use it with these datasets. Whenever determining the user's location, the SQL command in Figure 26 was used to create a point representing their current location in BNG.

```
ST_Transform(ST_SetSRID(ST_MakePoint(longitude , latitude),4326),27700);
```

Figure 26 Transforming a users location from WGS84 (4326) to BNG (27700).

3.7.3 Euclidean distance

The Euclidean distance approach for finding nearby POIs works by creating a buffer all around the user (Figure 25), checking for intersections with nearby POIs. The PostGIS function ST_DWithin shown in Figure 27 allows us to implement this functionality.



```
boolean ST_DWithin(geometry g1, geometry g2, double precision distance_of_srid);
```

Figure 27 ST_DWithin function.

The function (ST_DWithin) returns true if the two geometries are within a given distance of each other. The query in Figure 26 was used to converts the user's position into a geometry point on the fly; it is then compared against the geometries of all the POIs, returning any POIs inclusively within the search distance parameter. The search distance is, in effect, the buffer; how far we are willing to search for POIs. However, Figure 28 shows POIs for a shopping complex, with the POI for Asda being placed in the middle of the store. In this case, the user might have to enter the building to trigger the reminder, which might mean the user would not be reminded until they are already inside the shop, which is not a helpful reminder.

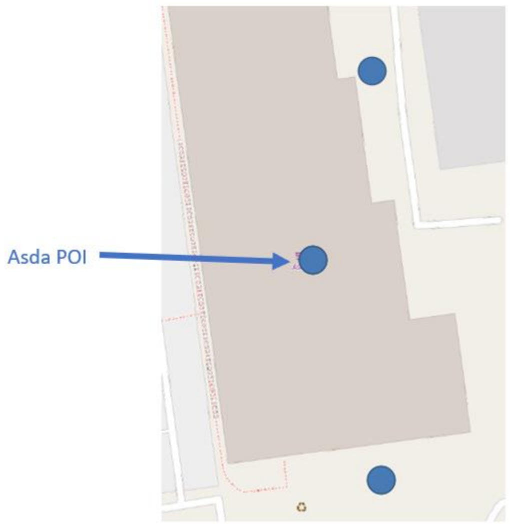


Figure 28 Shopping complex POIs.

A lookup table was created using a spatial join on the OS Points of Interest and OS MasterMapTopography layers to resolve this problem; this creates a link between POIs and the buildings they reside within. To do this, ST_Within was used shown in Figure 29, similar to ST_DWithin however it only returns true when a geometry is wholly within another. The query used to create the lookup table is shown in Appendix A.

$$\text{boolean } \textbf{ST\_Within}(\underbrace{\text{geometry } A}_{\text{POIs}}, \underbrace{\text{geometry } B}_{\text{Buildings}});$$

Figure 29 ST_Within function.

This lookup table allows us to intersect with buildings that we know contain POIs instead of just the single points shown in Figure 28. Thus, alleviating the potential issue of a user receiving a reminder after they have entered the building. Figure 30 shows an example of this if the user's buffer intersects anywhere with the building (the green polygon), a reminder could trigger.

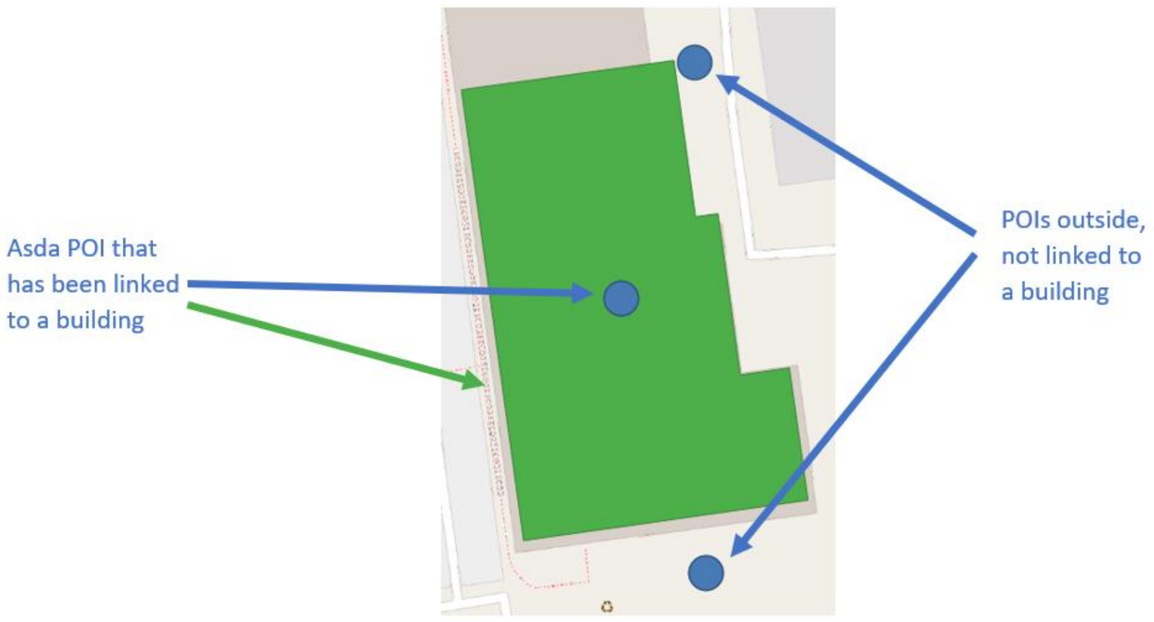
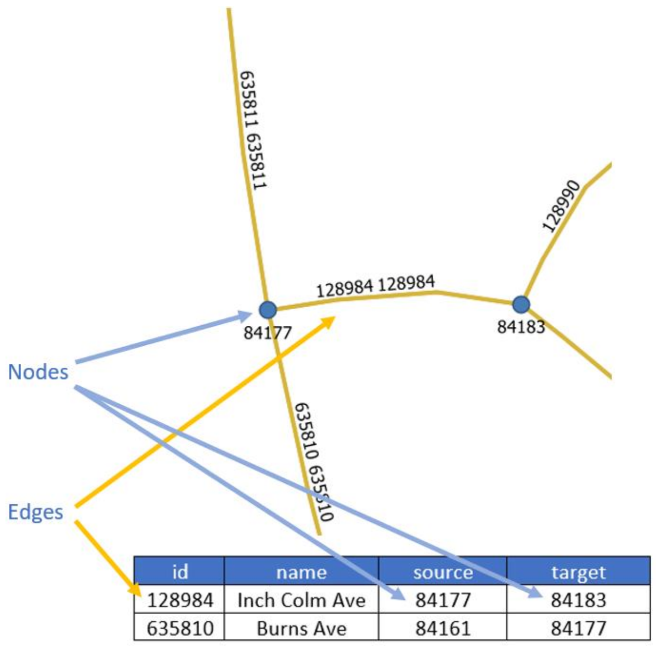

Asda POI that has been linked to a building
POIs outside, not linked to a building

Figure 30 The POI for Asda (shown in Figure 28) has been linked to the building it resides in through the use of a spatial join. Two POIs that do not reside in buildings are also shown.

However, as shown in Figure 30, not all POIs are located inside a building; when initially testing the Euclidean distance, reminders located outside of a building, e.g. a postbox, would never trigger as we were only checking for POIs located within buildings. We resolved this problem by having the algorithm check for intersections with the user's buffer for POIs located within a building and POIs located outside. Thereby, we can trigger reminders at the building level (e.g. Asda) and singular points (e.g. post boxes). The final implementation of the Euclidean distance can be seen in Appendix B.

3.7.4 Network distance

The second approach was to determine the distance (cost) between the user and POIs using a network made up of roads, paths and walkways. The pgRouting extension for PostgreSQL was used to do this, which provides geospatial routing algorithms such as Djisktras and A* shortest path. For this project, a database cutout of a network for central Scotland was obtained, but this can be built using a tool such as OSM2PO, which generates SQL files using OSM XML data, making it routable (Carsten Moeller, 2021). Figure 31 shows a partial cut out of this network.



id	name	source	target
128984	Inch Colm Ave	84177	84183
635810	Burns Ave	84161	84177

Figure 31 An overview showing a section of the Scotland network road edges and junction nodes. The table shows the source and target nodes that define each road edge, as well as the road name and ID.

The network comprises edges and nodes; each edge has two nodes, a source and a target, where the edge starts and finishes. Figure 31 shows that the Inch Colm Ave road edge connects to Burns Ave road edge using the source junction node of Inch Colm Ave and the target junction node of Burns Ave. However, this is entirely arbitrary as it is just how these edges have been drawn; these could have connected via source to source, source to target, etc. Thankfully, pgRouting automatically manages the connections between sources and targets for us.

As we are interested in the shortest path between the user and the POI, Dijisktras shortest path algorithm was initially investigated; an example of routing between two edges is shown in Figure 32. It should be noted that in the implementation, we used the cost versions of these functions, e.g. pgr_dijkstraCost, as we are interested in the cost, not in visualising or using the paths which are shown purely, for example.

```
SELECT * FROM pgr_dijkstra(
  'SELECT id, source, target, cost, reverse_cost FROM scotland_subset',
  128426, -- starting point
  539111 -- end point
);
```

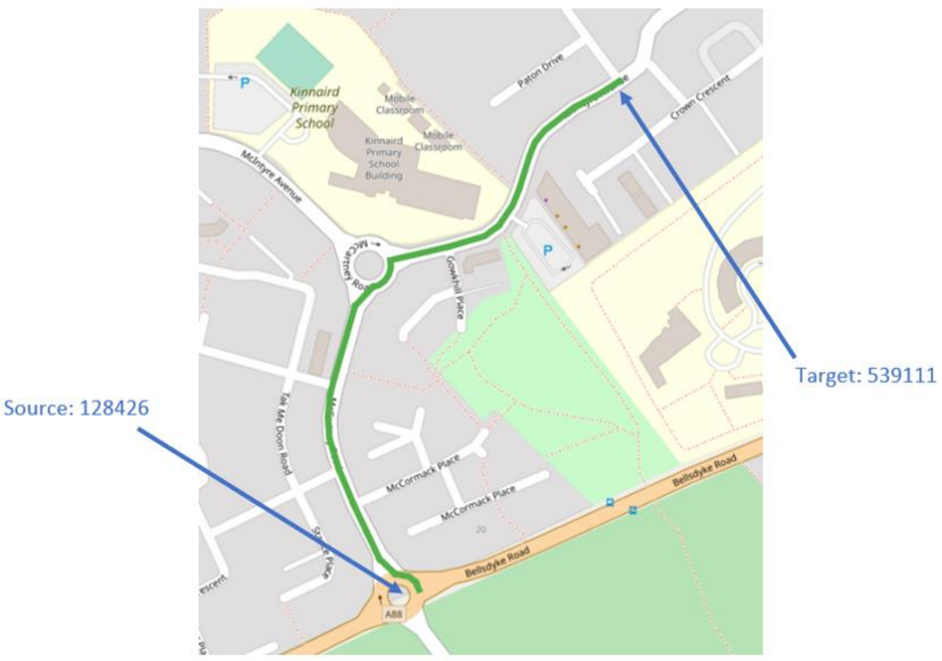
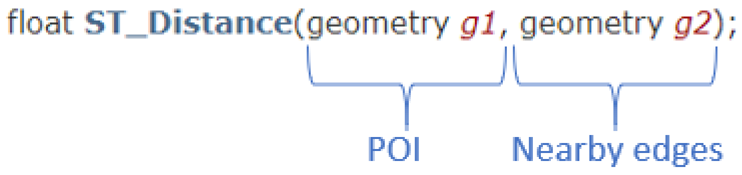


Figure 32 pgr_dijkstra function and visualising the shortest path between source 128426 and target 539111.

However, before we could calculate the user's distance from a POI using the network, some processing had to be done; the POI data is disjoint from the routing data, so a link had to be created from each POI to its nearest edge. To do this, we found all of the edges within 500 metres of each POI using ST_DWithin (Figure 27). These edges were then ordered using ST_Distance, shown in Figure 33, which returns the minimum distance between two geometries, resulting in finding the closest edge to each POI. The SQL command used to do this is shown in Appendix C.



```
float ST_Distance(geometry g1, geometry g2);
```

Figure 33 ST_Distance function.

Two custom functions were created, the first allowing us to snap the user on the fly to the nearest edge as described above for POIs, shown in Appendix D. The second function finds all nearby POIs that we would like to conduct a network search on, shown in Appendix E; the function for finding nearby POIs uses the Euclidean distance approach. Using these two functions, we can then calculate from the users nearest edge the network cost to travel to the nearest edge of a POI. The full implementation using pgr_dijsktraCost can be seen in Appendix F. However, an issue with this approach is highlighted in Figure 34, showing the end of a path from a user to a POI. It is highly unlikely that the target or source of an edge will end/begin precisely where a POI is located. Therefore extra distance will be included in the cost as the user would only need to travel partially along the edge to reach a POI.

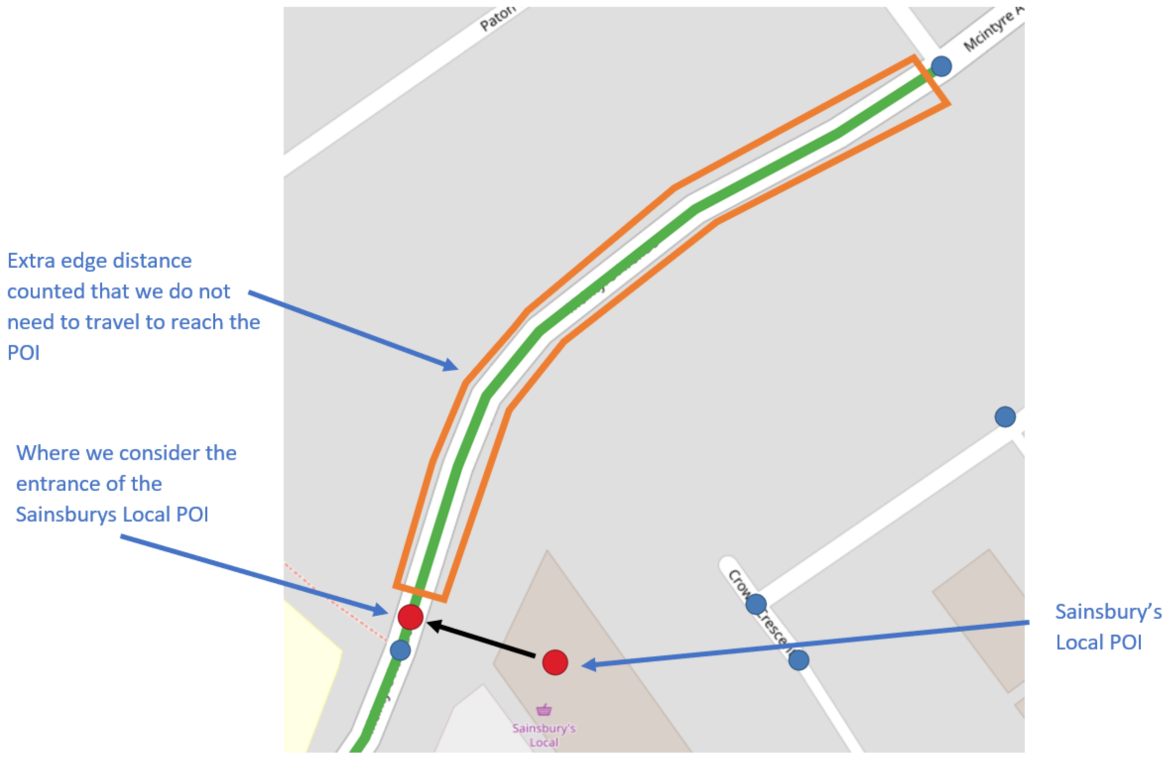
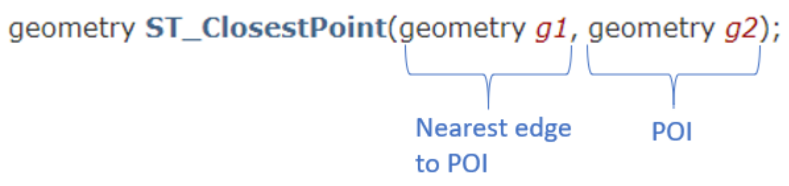

Extra edge distance counted that we do not need to travel to reach the POI
Where we consider the entrance of the Sainsburys Local POI
Sainsbury's Local POI
Paton
Mcintyre A
Sainsbury's Local

Figure 34 Highlighting the extra edge distance included when using pgr_dijsktraCost at the end of a path, showing the result of running ST_ClosestPoint on Sainsbury's Local POI, which is where we consider the entrance of the POI to be.

The same is also true when snapping the user onto the nearest edge; it is unlikely the user is precisely at the target or the source of an edge. A proposed function, pgr_withPoints, was found in pgRouting, allowing us to find the shortest path using Djskitrats shortest path algorithm from/to arbitrary points on the network. The pgr_withPoints function requires an extra piece of information to do this; we needed to find the nearest point on the closest edge to each POI. The function ST_ClosetPoint provides this functionality returning the point on geometry g1 (the nearest edge to the POI) closest to geometry g2 (the POI), shown in Figure 35.



```
geometry ST_ClosestPoint(geometry g1, geometry g2);
```

Figure 35 ST_ClosestPoint function.

We do not know precisely where the POI entrance is, so the closet point is a guess assuming the entrance would be accessible from the nearest edge; an example of this is shown in Figure 34 for the Sainsbury's Local POI. We also have to do the same calculation to find the nearest point on the closest edge to the user on the fly; a helper function to find the geometry of the nearest edge is shown in Appendix G. We can now use pgr_withPointsCost to determine the cost from the user's exact position on an edge to where we consider POI entrances to be. The complete pgr_withPointsCost implementation is shown in Appendix H. This implementation works first by finding all nearby POI/s within the maximum search range using the Euclidean distance. It then calculates the network distance to each POI, and if it is less than the maximum search range, it is fed back to the application. These POIs are then filtered based on all active reminder locations types; if a match is found, all associated reminders are triggered, showing the POI name and location.

4 Evaluation

An evaluation was conducted, reviewing the functional and non-functional components of the database systems and the application. The geospatial functionality was also evaluated by comparing the Ecludiean and Network distance approaches for finding nearby POIs.

4.1 Adjustments to the previously planned evaluation study

Due to the current restrictions in place by the Scottish Government in response to the COVID-19 pandemic, when completing this evaluation, having actual users test the system and provide feedback was not feasible. Thus, a system test was conducted focusing on testing and evaluating the performance of the different approaches of triggering reminders based on the user's location. All testing data was collected on foot due to the pandemic; ideally, location data would have been collected using multiple sources such as public transport, but this was deemed an unnecessary risk.

4.2 Evaluation of Functional Requirements

Below, the implementation status of the functional requirements for the project outlined in Deliverable 1 are shown:

Table 1 Database System Functional Requirements (M=Must, S=Should, C=Could, W=Would).

ID	Requirement	Priority	Implemented
DFR1	The database system must be able to store the location data sets	M	Yes
DFR2	The database must be able to run spatial queries	M	Yes
DFR3	The database must be able to store account information	M	Yes
DFR4	The database must store the related reminder information that is associated with each account	M	Yes

Table 2 Mobile Application Functional Requirements (M=Must, S=Should, C=Could, W=Would).

ID	Requirement	Priority	Implemented
MFR1	The application must support the ability for multiple users to create accounts	M	Yes

MFR2	The user must be able to delete their account	M	Yes
MFR3	(Logged out) The user must be able to create an account	M	Yes
MFR4	(Logged out) The user must be able to login into their account using the credentials provided when creating their account	M	Yes
MFR5	(Logged in) The application must be able to track the user's location	M	Yes
MFR6	(Logged in) The user must be able to create reminder entries with a location-based trigger	M	Yes
MFR7	(Logged in) The users should be able to delete any reminder entries created	S	No
MFR8	(Logged in) The user should be able to edit reminder entries that have been created	S	No
MFR9	(Logged in) The application must notify the user if they are near a POI that has been associated with a reminder through the use of text, haptic or audio notifications	M	Yes
MFR10	(Logged in) The user should be able to configure the notification types (text, haptic, or audio) they receive for a reminder	S	No
MFR11	(Logged in) The application should show the location of the nearby POIs on a map when a reminder is triggered	S	Yes
MFR12	(Logged in) The user should be able to control the search distance for POIs	S	Yes
MFR13	(Logged in) The application could have a feature to facilitate sharing reminder entries	C	No

4.3 Evaluation of Non-Functional Requirements

As discussed in section 4.1, due to us not evaluating the application with users, we cannot comment on some of the non-functional requirements about the application's qualities:

Table 3 Non-Functional Requirements (M=Must, S=Should, C=Could, W=Would).

ID	Requirement	Priority	Implemented
NFR1	The application should be easy to use and navigate	S	N/A
NFR2	The application should have an attractive user interface	S	N/A
NFR3	The application should have a responsive UI that works across most modern mobile devices	S	Yes
NFR4	Application data must be stored securely using encryption for personal information and the user's location	M	Yes
NFR5	Users data should be regularly backed up in the case of system failure so they do not lose their data	S	Yes
NFR6	The application should be battery aware and attempt to minimize battery use	S	No

4.4 Euclidean distance

An evaluation into triggering reminders using the Euclidean distance was conducted; this was the first approach implemented to find nearby POIs to trigger reminders and is the most naïve of the two implemented. The first example, Figure 36, shows a user walking in front of a shopping promenade and then their buffer intersecting a nearby building containing a POI, successfully triggering a reminder.

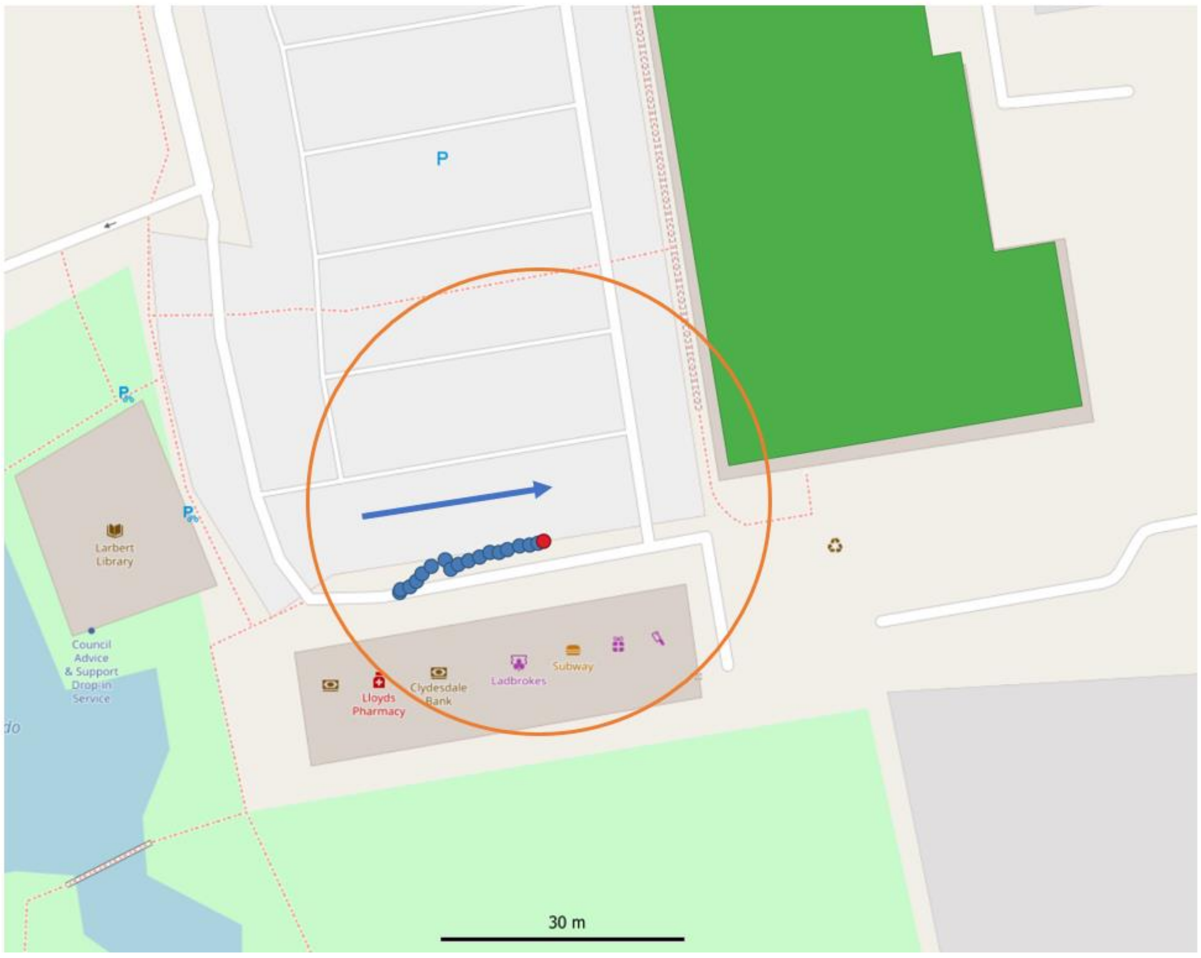


Figure 36 The green polygon is the building a POI has been mapped to, the orange circle is the buffer around the user. The blue points and arrow are the user walking and the direction of travel. The red circle is where the user is when their buffer intersects with the polygon, and a reminder is triggered.

The second example, shown in Figure 37, highlights one of the critical issues in using the Euclidean distance for finding nearby POIs. The user's buffer has intersected with a POI they are interested in, but they are walking down a road where it is impossible to reach this POI. To reach the POI, they would have to turn around and travel much further; thus, this reminder is not valuable to the user.

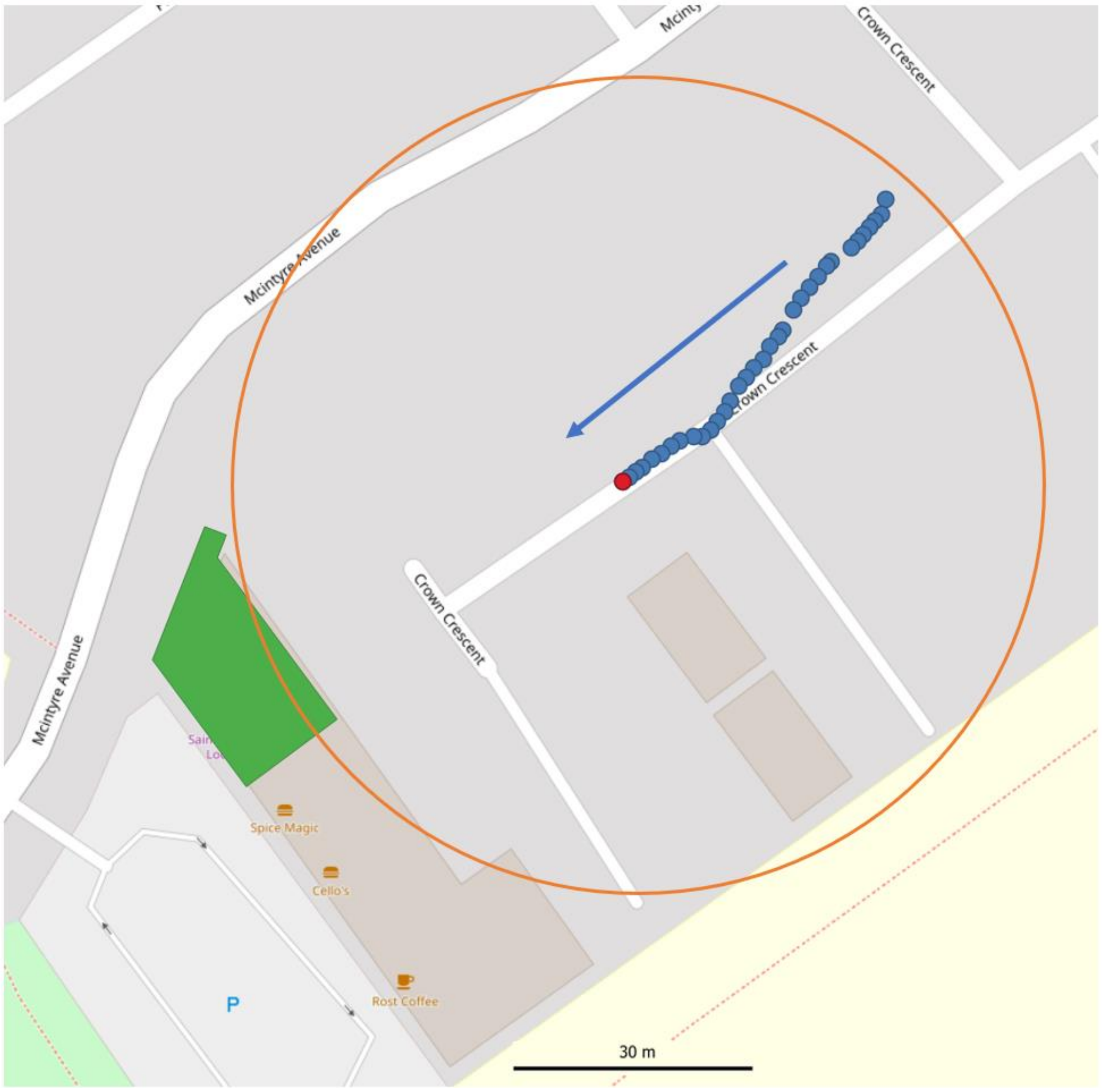


Figure 37 The green polygon is the building a POI has been mapped to, the orange circle is the buffer around the user. The blue points and arrow are the user walking and the direction of travel. The red circle is where the user is when their buffer intersects with the polygon, and a reminder is triggered.

For each of the nine reminder location types (food, nature, postbox, hardware store, etc.), real-world location data was collected using the Strava mobile application. Strava allowed us to export this data as a GPS exchange format file (GPX), an XML file that contains the recorded location data that we used to emulate being at the physical location without having to be there physically. Unit tests were created using the Jest Testing framework that used this recorded GPX data programmatically. These were used throughout the development and evaluation, ensuring reminders triggered correctly, checking for false positives, and ensuring that real-world GPS noise was included in testing, allowing us to refine the algorithm, ensuring that it copes with this noise. An example of a unit test can be seen in Appendix I.

4.5 Network distance

The second approach implemented for finding nearby POIs to trigger reminders was using a network distance, routing from the closest edge to the user to the edge nearest the POI. It provided us with a cost (in metres) for the user to travel to nearby POIs. The first method of calculating the cost was using pgr_dijskstraCost, but this does not consider where the user or the POI is along the edge. Figure 38 shows an example of visualising the shortest path used to calculate the cost using pgr_dijskstraCost between a user (starting at the roundabout) and Sainsbury's Local POI (the blue polygon). The second method, pgr_withPointsCost, allowed us to path between arbitrary points on the network; an example of this is the blue point shown in Figure 38, where we consider the entrance of the Sainsbury's local POI to be.

Figure 38 is one of the worst-case scenarios for pgr_dijsktraCost as we consider the entrance of the POI to be at the start of a new road edge (highlighted in red). As it does not know where the POI is located on the nearest road edge, it will count the entire distance of the last edge of the path, which substantially impacts the accuracy of the cost. Whereas pgr_withPointsCost only counts the distance along the road edge to the POI entrance (the blue point) and does not include the extra distance. The extra distance could also occur on the nearest edge to the user, but we consider the user to be at the same location in this example.

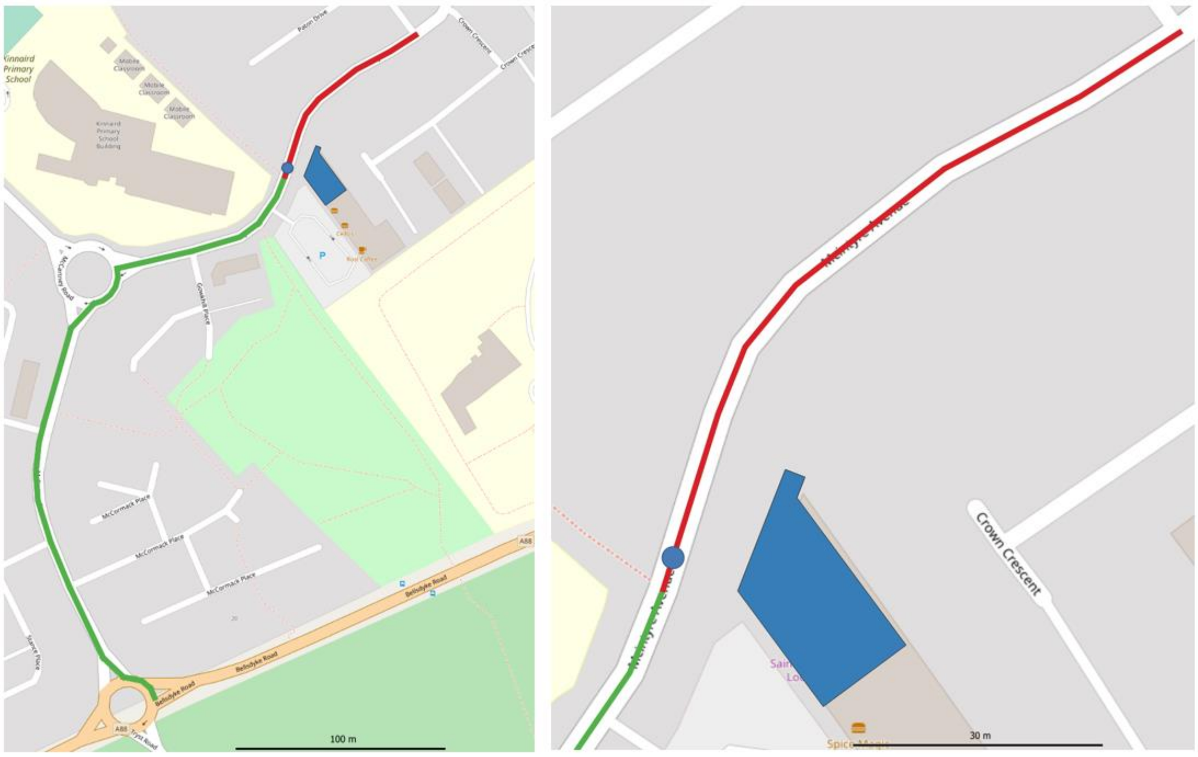


Figure 38 The left image shows the shortest path used by pgr_dijskstraCost to determine the cost from the roundabout edge to the edge nearest Sainsbury's Local POI (the red edge). The right image zooms in on this path, detailing when using pgr_dijskstraCost, the extra distance along the final edge that would not need to be travelled to reach the POI. The blue point is where we consider the entrance of Sainsbury's Local POI to be.

The cost of calculating the path shown in Figure 38 is in Table 4 below for both algorithms, pgr_dijskstraCost calculating the full path and pgr_withPoinstCost routing only to the entrance of the POI (the blue point):

Table 4 Cost of the network path shown in Figure 38 using both network algorithms.

pgr_dijskstraCost (metres)	pgr_withPointsCost (metres)
503.41m	448.97m

The results show that using pgr_withPointsCost results in a cost reduction of 54.44 metres or nearly 11.5% compared to pgr_ dijskstraCost. This substantial cost difference could lead to a user missing many nearby POIs. It should also be noted it is improbable a user will be precisely at the target or source of an edge, so the real-world results would be even better. However, this approach is still problematic as we consider the entrance of the POI to be the closest point on the nearest edge. The user could be reminded about POIs within their search range but cannot access them from this location, e.g. they are behind the building of the POI, a similar situation as Figure 37.

4.6 Comparing the Euclidean distance to the Network distance

To further evaluate the performance of the two approaches for finding nearby POIs, we have selected ten POIs and starting edges nearby to each POI randomly. Due to the unknown location of the user in relation to the starting edge, we have randomly selected their starting position to be the target of each edge. Table 5 shows the results of calculating the cost from the target of the edge to the POI using network and Euclidean distance approaches:

Table 5 Comparing the results of the cost between ten random POIs and edges using the Euclidean and network distances.

ID	Edge Name	POI Name	pgr_dijkstraCost (metres)	pgr_withPoints Cost (metres)	Euclidean Distance (metres)
1	Torlea Place	Spar Store	698.92m	594.67m	334.56m
2	Wallace Place	Davids Kitchen	221.51m	178m	173.17m
3	St Crispin's Place	Voseba Bakery & Cafe	266.44m	249.02m	164.83m
4	James Street	The Plough	436.01m	412.09m	168.95m
5	Dock Road	The Dundas	332.33m	303.89m	180.66m
6	Weir Street	EE in Argos	753.27m	697.08m	139.75m
7	Livingstone Drive	DJMC Joinery	810.82m	731.83m	550.75m
8	Harley Court	Gem Clean Services	462.78m	374.90m	218.39m
9	Alma Street	Firestorm	217.34m	185.93m	179.29m
10	Burnbank Road	Gambero Rosso	321.04m	306.93m	184.83m

The results in Table 5 show that the network distance algorithm pgr_withPointsCost outperforms pgr_ dijkstraCost as expected as we omit the extra unnecessary distance from edges in the cost calculation. However, comparing the Euclidean distance to pgr_withPointsCost, a more significant difference is apparent; the Euclidean distance, on average, is significantly less than using the network to calculate the cost to each POI. For example, in entry six of Table 5, the cost calculated using pgr_withPointsCost to EE in Argos starting at Weir Street is 697.08m, compared to the Euclidean distance of 139.75m. In this case, the user is located physically nearby to the POI but would have to travel much further to access it. Entry six is an extreme example, but generally, as shown in Table 5, the Euclidean distance vastly underestimates the distance compared to the actual cost (pgr_withPointsCost). Thus, solely using the Euclidean distance to find nearby POIs can not be recommended.

5 Conclusions and Future work

In conclusion, this dissertation project has achieved a fully functioning location-based reminder mobile application, from the initial planning stage to delivery in only seven months. This project was the most significant and challenging piece of work ever personally undertaken; creating a mobile application for Android & iOS, a fully functioning API using modern cloud technologies and learning a wide array of spatial technologies and techniques. Discussed below are the aims and objectives met; future work has also been proposed providing suggestions that could improve this project.

5.1 Aims and Objectives Met

The objective of evaluating the completed application through an evaluation study could not be completed due to the previously mentioned Covid-19 pandemic; however, an alternative evaluation of the application was completed undertaking a systems test of this project's spatial components. Real-world data was collected that could be later replayed, allowing us to develop and test the system's spatial components. The system was also tested using non-recorded real-world data. Sadly, this alternative evaluation does not evaluate the user interface's effectiveness, which is a significant portion of this project; however, videos have been created to demonstrate the user interface's key features and functionality, which links to can be found in section 1.4.

A single database was proposed to store user accounts, user reminders, and location data; however, the actual solution uses two separate databases and an authentication service. Firebase Authentication was used for registering and authenticating user accounts, Firebase Cloud Store was used for storing users reminders, and a PostgreSQL database to store the geospatial data. The PostgreSQL database uses the PostGIS extension, allowing us to store the geospatial data on POIs, with the OS POI data obtained from Digimap. Another extension, pgRotuing, a geospatial network routing library, was used alongside PostGIS. We used functions from both extensions in spatial queries to determine if a user has entered or is nearby a location associated with a reminder. An API was implemented to run these spatial queries on the PostgreSQL database remotely from the mobile application.

The system also meets the objective of implementing an ontology with a classification system that considers all types of generic places equal, e.g. all supermarkets are the same; thus, allowing a user to create reminders for generic location types. Regrettably, the useability features such as editing and deleting reminders could not be implemented in this initial version of the project due to the React Native front end framework's steep learning curve.

5.2 Future work

Throughout working on this project, it was always understood that not all of the requirements would be implemented during the application's initial development. Discussed below is future work that is recommended to add to the current implementation significantly.

5.2.1 Conducting further testing

Due to the previously mentioned pandemic, it has not been possible to have users psychically test the application due to Government restrictions. However, it would be advisable to conduct a system useability study evaluation of the application in the future, in which the results could be used to inform and influence future design changes.

It was also not possible to assess the accuracy of the system's geospatial functionality using different modes of transportation such as busses or cycling. Testing these other means of transportation would allow us to evaluate the current implementation further and potentially discover potential implementation issues.

5.2.2 Sharing reminders between users

The ability for users to share reminders could be implemented. For example, if a couple could contribute to the same reminder list, if either person was near a POI to complete a shared reminder, they could do it for the other person or vice versa. This feature could be implemented with some adjustments to the Firebase Cloud Firestore; an interface would need to be created, allowing users to add other members to a group where they could then share the same unique ID, thus accessing the same set of reminders.

5.2.3 Improving the points of interest data

The POI data from the OS Points of Interest Layer is generally very well maintained; however, some mistakes have been spotted when testing the application, such as spelling mistakes and miss categorised POIs. This data is only accurate within six months; therefore, some of the POIs no longer exist, e.g. a business has shut down. Time should be spent going through this data and fixing these issues as the quality directly impacts how helpful the user's reminders are. The more accurate this data is, the better the application's performance is; however, to fix this data requires knowledge of the local area, perhaps even being a community member to recognise a POI no longer exists.

A solution to resolve the potential lack of local knowledge could be to resolve these problems within the application; when a user receives a reminder and notices a problem, i.e. the POI has a spelling mistake or has been miscategorised, they could be offered the ability to correct this error. These updates could then be fed back to the OS to improve the quality of data for others.

In the network distance implementation, we guess where the entrance of a POI is, as discussed in section 3.7.4. For some occurrences, this will not be correct, i.e. it will consider the entrance to a shop to be behind the building. This data is called micro geography and is an active research area with many large companies working on this problem, such as the OS and Google, to find doors in buildings. In the future, this data could potentially be available from the OS and be used similarly to how POI data was in this project.

5.2.4 Improvements to the user interface

The improvements to the user interface below are what we would have wanted to implement if more time was available during the application development sprint:

- The user interface does not feature two of the should have requirements, the ability to edit and delete reminders. To implement this, when a user selects a reminder on the home screen, buttons could appear, allowing the user to edit or delete a reminder.
- The application does not feature any accessibility settings; an accessibility page could be implemented, allowing the user to customise the user interface, e.g. change the background colour, enable high contrast text, and adjust the font size.
- Further customization controls regarding how a user receives notifications for a reminder could be implemented, allowing them to select all or pick and choose between text, haptic or audio notifications.
- A setting could be implemented, allowing the user to customise the dwell time; currently, this is a fixed time.
- Currently, a user can select between three predefined reminder search ranges on the home screen for all reminders. An improvement to this would be when a user creates a reminder; it allows them to set a predefined or custom search distance for each reminder. Users might be willing to travel further out of their way for a specific reminder but not for others. We tried to implement this feature, but this was not feasible due to design decisions earlier regarding the API.
- The home screen could be improved by allowing the users to search for reminders and customise the order they appear in; currently, it is alphabetic, but the ordering could be customised, e.g. reminder types or time added.

5.2.5 Triggering reminders when the application is in the background

A feature that was investigated during development but could sadly not be implemented into this version of the project was searching for nearby POIs whilst the application is not in the forefront. Currently, users must have the application running in the foreground for reminders to trigger, meaning they cannot do other tasks with their phone or lock it to preserve battery usage. Unfortunately, not having this feature significantly limits the real-world usage of this version of the application. Background reminders could not be implemented because the Expo Background TaskManager disables the use of notifications. POIs could be found in the background, but we had no way to notify the user. However, it is possible to do this; a more experienced developer could 'eject' from Expo, losing the benefits of the development tools but allowing you complete control over background task scheduling and notifications. Ejecting from Expo is a substantial undertaking resulting in managing the native dependency files for iOS and Android and the manual linking of packages.

5.2.6 React Native testing

Due to the lack of experience and the initial steep learning curve when using React Native, unit tests were not written for the components. However, this was investigated, and testing frameworks such as Jest support the testing of React Native out of the box. This work sadly could not be prioritised due to a lack of time in the application building sprint. The use of units tests would significantly add to the project, ensuring changes could be made in confidence and testing that the changes did not affect other components, which was a common occurrence.

6 Back matter

6.1 References

Alkhaldi, G., Hamilton, F. L., Lau, R., Webster, R., Michie, S. and Murray, E. (2016) 'The Effectiveness of Prompts to Promote Engagement With Digital Interventions: A Systematic Review', *Journal of Medical Internet Research*, 18(1), p. e6. doi: 10.2196/jmir.4790.

Apple (2020) *Add dates or locations to reminders on Mac*, *Apple Support*. Available at: https://support.apple.com/en-gb/guide/reminders/remnd4b206fb/7.0/mac/11.0 (Accessed: 17 November 2020).

Axelrod, E. and Fitoussi, H. (2014) 'Conditional location-based reminders'. Available at: https://patents.google.com/patent/US8700709B2/en (Accessed: 8 November 2020).

Bartie, P., Mackaness, W., Lemon, O., Dalmas, T., Janarthanam, S., Hill, R., Dickinson, A. and Liu, X. (2018) 'A dialogue based mobile virtual assistant for tourists: The SpaceBook Project', *Computers, Environment and Urban Systems*, 67, pp. 110–123. doi: 10.1016/j.compenvurbsys.2017.09.010.

Ben-Moshe, B., Elkin, E., Levi, H. and Weissman, A. (2011) 'Improving Accuracy of GNSS Devices in Urban Canyons', in *CCCG*. *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, CCCG 2011*.

Buczkowska, S., Coulombel, N. and de Lapparent, M. (2019) 'A comparison of Euclidean Distance, Travel Times, and Network Distances in Location Choice Mixture Models', *Networks and Spatial Economics*, 19(4), pp. 1215–1248. doi: 10.1007/s11067-018-9439-5.

Carsten Moeller (2021) *osm2po - openstreetmap converter and routing engine for java*. Available at: http://osm2po.de/ (Accessed: 16 March 2021).

Chin, P. G. (2011) 'Smart reminders'. Available at: https://patents.google.com/patent/US7925525B2/en (Accessed: 8 November 2020).

Crunchy Data (2020) *About pg_tileserv*. Available at: https://access.crunchydata.com/documentation/pg_tileserv/1.0.3/introduction/ (Accessed: 7 November 2020).

van Diggelen, F. and Enge, P. (2015) 'The World's first GPS MOOC and Worldwide Laboratory using Smartphones', in. *Proceedings of the 28th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2015)*, pp. 361–369. Available at: http://www.ion.org/publications/abstract.cfm?jp=p&articleID=13079 (Accessed: 20 October 2020).

Digimap (2020a) *Ordnance Survey data and GPS measurements*. Available at: https://digimap.edina.ac.uk/webhelp/os/data_information/os_data_issues/os_data_and_gps.htm (Accessed: 22 October 2020).

Digimap (2020b) *OS MasterMap Topography Layer*. Available at: https://digimap.edina.ac.uk/webhelp/os/osdigimaphelp.htm#data_information/os_products/mastermap_topo.htm (Accessed: 22 October 2020).

Digimap (2020c) *Points of Interest*. Available at: https://digimap.edina.ac.uk/webhelp/os/osdigimaphelp.htm#data_information/os_products/points_of_interest.htm (Accessed: 22 October 2020).

D'Roza, T. and Bilchev, G. (2003) 'An Overview of Location-Based Services', *BT Technology Journal*, 21(1), pp. 20–27. doi: 10.1023/A:1022491825047.

Einsiedler, J., Radusch, I. and Wolter, K. (2017) 'Vehicle indoor positioning: A survey', in *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*. *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, pp. 1–6. doi: 10.1109/WPNC.2017.8250068.

Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E. and Bylund, M. (2001) 'GeoNotes: Social and Navigational Aspects of Location-Based Information Systems', in Abowd, G. D., Brumitt, B., and Shafer, S. (eds) *Ubicomp 2001: Ubiquitous Computing*. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science), pp. 2–17. doi: 10.1007/3-540-45427-6_2.

Evernote (2018) *Need to add location based reminders*, *Evernote User Forum*. Available at: https://discussion.evernote.com/forums/topic/111234-need-to-add-location-based-reminders/ (Accessed: 2 November 2020).

Felt, A. P., Egelman, S. and Wagner, D. (2012) 'I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns', in *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*. New York, NY, USA: Association for Computing Machinery (SPSM '12), pp. 33–44. doi: 10.1145/2381934.2381943.

Garzonis, S., Bevan, C. and O'Neill, E. (2008) 'Mobile service audio notifications: intuitive semantics and noises', in *Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat*. New York, NY, USA: Association for Computing Machinery (OZCHI '08), pp. 156–163. doi: 10.1145/1517744.1517793.

Groves, P., Wang, L. and Ziebart, M. (2012) 'Shadow Matching Improved GNSS Accuracy in Urban Canyons', *GPS World (Magazine)*, 23.

Haklay, M. (2010) 'How Good is Volunteered Geographical Information? A Comparative Study of OpenStreetMap and Ordnance Survey Datasets', *Environment and Planning B: Planning and Design*, 37(4), pp. 682–703. doi: 10.1068/b35097.

Jones, C. B., Purves, R., Ruas, A., Sanderson, M., Sester, M., van Kreveld, M. and Weibel, R. (2002) 'Spatial information retrieval and geographical ontologies an overview of the SPIRIT project', in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: Association for Computing Machinery (SIGIR '02), pp. 387–388. doi: 10.1145/564376.564457.

Langley, R. B., Teunissen, P. J. G. and Montenbruck, O. (2017) 'Introduction to GNSS', in Teunissen, P. J. G. and Montenbruck, O. (eds) *Springer Handbook of Global Navigation Satellite Systems*. Cham: Springer International Publishing (Springer Handbooks), pp. 3–23. doi: 10.1007/978-3-319-42928-1_1.

Leaflet (2020) *Leaflet — an open-source JavaScript library for interactive maps*. Available at: https://leafletjs.com/ (Accessed: 5 November 2020).

Maduako, I. (2012) 'PostGIS-Based Heterogeneous Sensor Database Framework for the Sensor Observation Service', *Geoinformatics FCE CTU*, 8, pp. 55–72. doi: 10.14311/gi.8.4.

Marmasse, N. and Schmandt, C. (2000) ‘Location-Aware Information Delivery withComMotion’, in Thomas, P. and Gellersen, H.-W. (eds) *Handheld and Ubiquitous Computing*. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science), pp. 157–171. doi: 10.1007/3-540-39959-3_12.

Mautz, R. (2012) ‘Indoor positioning technologies’, p. 1 Band. doi: 10.3929/ETHZ-A-007313554.

McClune, A. (2019) ‘points-of-interest-user-guide’, p. 33.

Michaelis, C. D. and Ames, D. P. (2008) ‘Web Feature Service (WFS) and Web Map Service (WMS)’, in Shekhar, S. and Xiong, H. (eds) *Encyclopedia of GIS*. Boston, MA: Springer US, pp. 1259–1261. doi: 10.1007/978-0-387-35973-1_1480.

Mok, E. and Retscher, G. (2007) ‘Location determination using WiFi fingerprinting versus WiFi trilateration’, *Journal of Location Based Services*, 1(2), pp. 145–159. doi: 10.1080/17489720701781905.

Nagai, K., Fasoro, T., Spenko, M., Henderson, R. and Pervan, B. (2020) ‘Evaluating GNSS Navigation Availability in 3-D Mapped Urban Environments’, in *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*. *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 639–646. doi: 10.1109/PLANS46316.2020.9109929.

OpenLayers (2020a) *OpenLayers - Frequently Asked Questions (FAQ)*. Available at: https://openlayers.org/en/latest/doc/faq.html (Accessed: 5 November 2020).

OpenLayers (2020b) *OpenLayers - Welcome*. Available at: https://openlayers.org/ (Accessed: 5 November 2020).

OpenStreetMap (2020a) *Buildings - OpenStreetMap Wiki*. Available at: https://wiki.openstreetmap.org/wiki/Buildings (Accessed: 8 November 2020).

OpenStreetMap (2020b) *Points of interest - OpenStreetMap Wiki*. Available at: https://wiki.openstreetmap.org/wiki/Points_of_interest (Accessed: 8 November 2020).

OpenStreetMap (2020c) *Stats - OpenStreetMap Wiki*. Available at: https://wiki.openstreetmap.org/wiki/Stats (Accessed: 8 November 2020).

Peters, I. and Stock, W. G. (2007) ‘Folksonomy and information retrieval’, *Proceedings of the American Society for Information Science and Technology*, 44(1), pp. 1–28. doi: 10.1002/meet.1450440226.

pg_tileserv (2020) *CrunchyData/pg_tileserv*. Available at: https://github.com/CrunchyData/pg_tileserv (Accessed: 7 November 2020).

PostGIS (2020a) *PostGIS — Spatial and Geographic Objects for PostgreSQL*. Available at: https://postgis.net/ (Accessed: 6 November 2020).

PostGIS (2020b) *ST_DWithin*. Available at: https://postgis.net/docs/ST_DWithin.html (Accessed: 9 December 2020).

PostGIS (2020c) *ST_Transform*. Available at: https://postgis.net/docs/ST_Transform.html (Accessed: 22 October 2020).

QGIS (2020a) *Raster Data — QGIS Documentation documentation*. Available at: https://docs.qgis.org/3.4/en/docs/gentle_gis_introduction/raster_data.html (Accessed: 10 December 2020).

QGIS (2020b) *Vector Data — QGIS Documentation documentation*. Available at: https://docs.qgis.org/3.4/en/docs/gentle_gis_introduction/vector_data.html (Accessed: 10 December 2020).

Rahate, S. W. and Shaikh, D. M. Z. (2016) 'Geo-fencing Infrastructure: Location Based Service', 03(11), p. 4.

Rahimi, H., Maimaiti, T. and Zincir-Heywood, A. N. (2014) 'A case study for a secure and robust geo-fencing and access control framework', in *2014 IEEE Network Operations and Management Symposium (NOMS). 2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–8. doi: 10.1109/NOMS.2014.6838325.

Sahami Shirazi, A., Henze, N., Dingler, T., Pielot, M., Weber, D. and Schmidt, A. (2014) 'Large-scale assessment of mobile notifications', in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery (CHI '14), pp. 3055–3064. doi: 10.1145/2556288.2557189.

Sinclair, J. and Cardew-Hall, M. (2008) 'The folksonomy tag cloud: when is it useful?', *Journal of Information Science*, 34(1), pp. 15–29. doi: 10.1177/0165551506078083.

Sommaruga, L., Rota, P. and Catenazzi, N. (2011) '"Tagsonomy": Easy Access to Web Sites through a Combination of Taxonomy and Folksonomy', in Mugellini, E., Szczepaniak, P. S., Pettenati, M. C., and Sokhn, M. (eds) *Advances in Intelligent Web Mastering – 3*. Berlin, Heidelberg: Springer (Advances in Intelligent and Soft Computing), pp. 61–71. doi: 10.1007/978-3-642-18029-3_7.

The University of Edinburgh (2020) *Digimap*. Available at: https://digimap.edina.ac.uk/os (Accessed: 3 November 2020).

Wang, J., Wang, D. and Guo, B. (2019) 'A low-power sensor polling for aggregated-task context on mobile devices', *Future Generation Computer Systems*, 98, pp. 362–371. doi: 10.1016/j.future.2019.02.027.

Wang, Y., Millet, B. and Smith, J. L. (2016) 'Designing wearable vibrotactile notifications for information communication', *International Journal of Human-Computer Studies*, 89, pp. 24–34. doi: 10.1016/j.ijhcs.2016.01.004.

Xie, P., Petovello, M. G. and Basnayake, C. (2011) 'Multipath Signal Assessment in the High Sensitivity Receivers for Vehicular Applications', p. 13.

Zandbergen, P. A. (2009) 'Accuracy of iPhone Locations: A Comparison of Assisted GPS, WiFi and Cellular Positioning', *Transactions in GIS*, 13(s1), pp. 5–25. doi: 10.1111/j.1467-9671.2009.01152.x.

Zhang, G. (2018) *Fusion of Multi-temporal Multispectral Images and OpenStreetMap Data for the Classification of Local Climate Zones*. German Aerospace Center. Available at: https://www.researchgate.net/publication/336641480_Fusion_of_Multi-temporal_Multispectral_Images_and_OpenStreetMap_Data_for_the_Classification_of_Local_Climate_Zones.

6.2 Appendix

6.2.1 Appendix A – Creating a lookup table linking POIs to the building they reside within

```
CREATE TABLE
  poi_building_lookup
AS
SELECT
  poi_table.id as POIID,
  osmm_buildings.id as BUILDINGID
FROM
  osmm_buildings
JOIN
  poi_table ON ST_WITHIN (poi_table.geom, osmm_buildings.geom);
```

6.2.2 Appendix B – Euclidean distance implementation

```
SELECT
  poi_table.name,
  poi_table.groupname,
  poi_table.categoryname,
  poi_table.classname,
  ST_AsText(poi_table.geomwgs84)
FROM
  osmm_buildings
  JOIN poi_building_lookup ON osmm_buildings.id=poi_building_lookup.buildingid
  JOIN poi_table ON poi_building_lookup.poiid=poi_table.id
WHERE
  ST_DWithin(osmm_buildings.geom, ST_TRANSFORM(ST_SETSRID(ST_MAKEPOINT (' +
  lon + ',' +  lat + '),4326),27700), ' + search_range + ')
OR
  ST_DWithin(poi_table.geom, ST_TRANSFORM(ST_SETSRID(ST_MAKEPOINT (' + lon + '
  ,' +  lat + '),4326),27700), ' + search_range + ');
```

6.2.3 Appendix C – Creating a table linking each POI to its closest edge

```
CREATE TABLE
  poi_to_nearest_edge
AS
SELECT DISTINCT
  *
FROM (
  SELECT
    a.id as ptid,
    target,
    source,
    a.name as poi_name,
    b.id as rdid,
    a.geom as ptgeom,
    b.geom as rdgeom,
    st_distance(a.geom,b.geom) as dst,
    min(st_distance(a.geom,b.geom)) OVER (PARTITION BY a.id) as mindst
FROM
  poi_table a
JOIN
  scotland_subset b on ST_DWITHIN (a.geom,b.geom,500)) as n
WHERE
  n.dst=n.mindst
ORDER BY
  ptid ASC;
```

6.2.4 Appendix D – Function for finding the nearest edge to a user

```
CREATE OR REPLACE FUNCTION nearest_edge_to_user(lon NUMERIC,lat NUMERIC)
      RETURNS INTEGER AS $target$
          DECLARE target INTEGER;
      BEGIN
        SELECT
          scotland_subset.target ID INTO target
        FROM
          scotland_subset
        ORDER BY
          ST_DISTANCE(geom, ST_Transform(ST_SetSRID(ST_MakePoint
          (lon ,lat ),4326),27700))
        LIMIT 1; -- We only want the closet edge
        RETURN target;
      END;
 $target$ LANGUAGE plpgsql;
```

6.2.5 Appendix E – Function for finding nearby POIs within the search range

```
CREATE OR REPLACE FUNCTION nearby_pois(lon NUMERIC, lat NUMERIC, search_range
NUMERIC)
      RETURNS INTEGER[] AS $nearbyPOIS$
          DECLARE nearbyPOIS INTEGER[];
      BEGIN
        SELECT
          array( SELECT
                    poi_to_nearest_edge.target ID INTO nearbyPOIS
                  FROM
                    poi_to_nearest_edge
                  WHERE
                    ST_DWITHIN(ptgeom, ST_TRANSFORM(
                      ST_SESTRID(ST_MAKEPOINT(lon,lat),4326),27700)
                      ,search_range)
                );
          RETURN nearbyPOIS;
      END;
 $nearbyPOIS$ LANGUAGE plpgsql;
```

6.2.6 Appendix F – Network distance implementation using pgr_dijkstraCost

This query makes use of the functions from Appendix D and Appendix E.

```
SELECT
  *
FROM
  pgr_dijkstraCost(
       'SELECT
          id,
          source,
          target,
          ST_Length(geomwgs::GEOGRAPHY) AS cost,
          ST_LENGTH(geomwgs::GEOGRAPHY) AS reverse_cost
        FROM
          scotland_subset',
       nearest_edge_to_user(' +   lon + ',' +  lat + '),
       nearby_pois(' +   lon + ',' +  lat + ', '+ search_range +'))
WHERE
  cost < ' + search range + ';
```

6.2.7 Appendix G – Function for finding the geometry of the nearest edge to a user

```
CREATE OR REPLACE FUNCTION nearest_edge_geom_to_user(lon NUMERIC,lat NUMERIC)
      RETURNS GEOMETRY AS $geom$
          DECLARE geom GEOMETRY;
      BEGIN
        SELECT
          scotland_subset.geom ID INTO geom
        FROM
          scotland_subset
        ORDER BY
          ST_DISTANCE(geom, ST_Transform(ST_SetSRID(ST_MakePoint
          (lon ,lat ),4326),27700))
        LIMIT 1; -- We only want the closet edge
        RETURN geom;
      END;
 $target$ LANGUAGE plpgsql;
```

6.2.8 Appendix H – Network distance implementation using pgr_withPointsCost

This query makes use of the functions from Appendix E and Appendix G.

```
SELECT
  *
FROM
  pgr_withPointsCost(
    'SELECT
      id,
      source,
      target,
      cost,
      reverse_cost
    FROM
      scotland_subset
    ORDER BY id',
    'SELECT
      pid,
      edge_id,
      fraction
    FROM
     poi_to_nearest_road',
    ST_ClosetPoint(
      nearest_edge_geom_to_user(' +   lon + ',' +  lat + '),
      ST_TRANSFORM(ST_SESTRID(ST_MAKEPOINT(' + lon + ',' + lat + ')
      ,4326),27700)),
    nearby_pois(' +   lon + ',' +  lat + ', '+ search_range +'))
WHERE
  cost < ' + search range + ';
```

6.2.9 Appendix I – Unit Test example

```
test('Walking directly in front of Asda (Food, Drink and Multi Item Retail rem
inder)', async () => {

  let data = await load_data();
  var json = parser.xml2json(data);
  var coord_array = json.gpx.trk.trkseg.trkpt;

  for (const coordPair of coord_array){
      let result = await fetch('https://0186u6yf60.execute-api.eu-west
      2.amazonaws.com/v1/check_location?lon=' + coordPair.lon + '&lat=' + coo
      rdPair.lat + '&search_range=1000');

      let POIs = await result.json();

      try {
        if(POIs.body.flat().includes('Asda Stores Ltd')){
          expect(POIs.body).toEqual(
            expect.arrayContaining([
              "Asda Stores Ltd",
              "Retail",
              "Food, Drink and Multi Item Retail",
              "Supermarket Chains",
              "POINT(-3.81525225743462 56.0247939024167)"]),
            );
        }
    } catch (error) {
      console.log(error);
    }
  }
});
```