Real-time Route Planning utilizing Mobile Pollution Detectors

Renji Luke Harold (671917)

Supervisor: Prof. Richard O. Sinnott
Melbourne School of Engineering

University of Melbourne

This report is submitted for

COMP90019 - Distributed Computing Project (25 credit points)
(Software Development Project)

Acknowledgement

My sincere acknowledgement goes to the project supervisor Professor Richard O Sinnott for his support and guidance throughout the project. I would also like to extend my acknowledgement to my project partner Md Akmal Hossain, technical consultant Dr. Marcos Nino-Ruiz and Jiajie Li for their continuous support with technical knowledge, which have been invaluable for completion of this project.

Declaration

I hereby declare that

- *This thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.*

- *Where necessary I have received clearance for this research from the University's Ethics Committee and have submitted all required data to the Department*

- *The thesis is 4736 words in length (excluding text in images, table, bibliographies and appendices).*

Renji Luke Harold

November 2015

Abstract

The report provides details regarding the project undertaken for replicating the existing AirCasting application for iOS, which currently is only available in Android. The details regarding the application user interface and the various user functionalities can be referred from Md Akmal Hossain's report. This report focuses on the the architecture employed for data management and storage. As an improvement to the current application, a functionality has been added to the application to enable the user to be able to identify the least polluted route between two points. This project provides implementation details on how this functionality was designed and incorporated into the application. The report concludes with the challenges faced during the course of the project and recommendations for future development.

Table of Contents

List of Figures

1.0 Introduction

1.1. Purpose of Project

The purpose of the project is to develop an iOS application that is capable of recording certain environmental data, which can be used to measure the level of air pollution level. The collected data can be used for various purposes such as sustainable development, urban planning, pollution identification, environment quality improvement etc. At present, a US based non-profit organization HabitatMap [2] offers a web-based platform named AirCasting to collect environmental pollution data using smart phone application and external sensor device. The existing AirCasting application is only operational within Android platform, which comprises of only around 53% of the smart phone users [5]. The objective of this project is to provide a solution to make the usage of AirCasting application more available to the general public. This will be especially beneficial to projects like Australian Urban Research Infrastructure Network (AURIN) and Clean Air and Urban Landscape (CAUL) initiatives. The application can also be beneficial to individuals, who are environmental enthusiasts, by providing location specific pollution information.

1.2. Scope of Project

The project scope has been defined based on the existing AirCasting platform and Android application. The first scope of the project is to develop an iOS application that replicates the existing AirCasting application for Android platform. Replication needs to include both operational and visual aspect of the existing application. As an additional feature, application should allow real time streaming (upload) of data to the remote AirCasting database. The second scope of the project is to develop a routing process based on A* algorithm which can recommend a route between source and destination which is the least polluted.

1.3. Report Outline

The report has been organized in following segments. Segment 2 titled “Background”, presents a brief overview of the existing projects where use of the application will be beneficial. It also includes brief description about the AirCasting platform and AirBeam sensor device used in this project. Segment 3 titled “Design Specification” describes the operation of the AirCasting server and current Android application architecture. Segment 4 titled “Design Implementation” contains the details of the data storage model and the implementation of the Clean Route functionality. Segment 5 titled “Challenges and Solutions” outlines the key challenges encountered during project development and solutions implemented to overcome them. At the end of the

report, possible future improvements for the application have been discussed in segment 6 titled “Future Directions” and project conclusions have been drawn in segment 7 titled “Conclusions”.

2.0 Background

2.1. AURIN

Australian Urban Research Infrastructure Network (AURIN) is a national collaboration among several Australian Government entities, National Research Infrastructure for Australia (NCRIS) and University of Melbourne to establish an e-research infrastructure to help better decision making for urban settlement and development.

Clean Air is one of the many different projects that comprise AURIN. It is an initiative to estimate the air quality of different localities. The data can allow authorities to take appropriate measures for polluted areas to improve living quality, help recommend individuals to travel within less polluted areas etc. [1]

2.2. CAUL

Clean Air and Urban Landscape (CAUL) is a research project funded by the Department of Environment and is supported by Green Building Council Australia (GBCA) with participation of University of Melbourne, RMIT University, University of Wollongong and University of Western Australia. The project focuses on environmental research within urban localities to help government and industry to take better decisions about future planning and development. [9] [10]

2.3. AirCasting

AirCasting is an open source online platform, which offers recording, mapping and sharing of environmental and health data through smart phone application and external sensor devices. The application communicates with external sensor device to record pollution level and visualizes the data within a web application named CrowdMap.
AirCasting platform is used among environmental research organizations, academic institutions and environmental enthusiasts who like to contribute towards green environment initiative.

2.4. AirBeam

AirBeam is a wearable monitoring device developed by HabitatMap [2] that contains Humidity, Particulate Matter and Temperature sensors. The device allows Bluetooth communication service to allow smart application to collect data detected by the sensors.

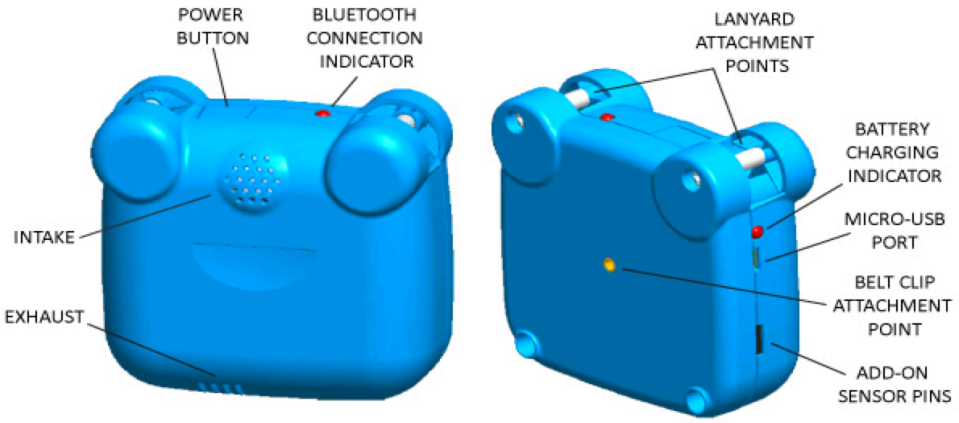

Figure 1. AirBeam Sensor Device

The device is battery powered and can be recharged via micro USB cable. It contains LED indicators for Bluetooth connection and battery charging status. It also includes additional connecting pins to add new sensors. Device hardware is Arduino Leonardo based and reprogrammable with Arduino IDE software.

3.0 Design Specification

3.1. System Architecture

The AirCasting system contains four key components: sensor device, smart phone application, remote database server and web application. Environmental data is detected via the sensor device, which is recorded in the smart phone application. Each recorded session is uploaded to the remote database server. Application can also download saved sessions from the server. The server also provides a web application to visualize each session data.

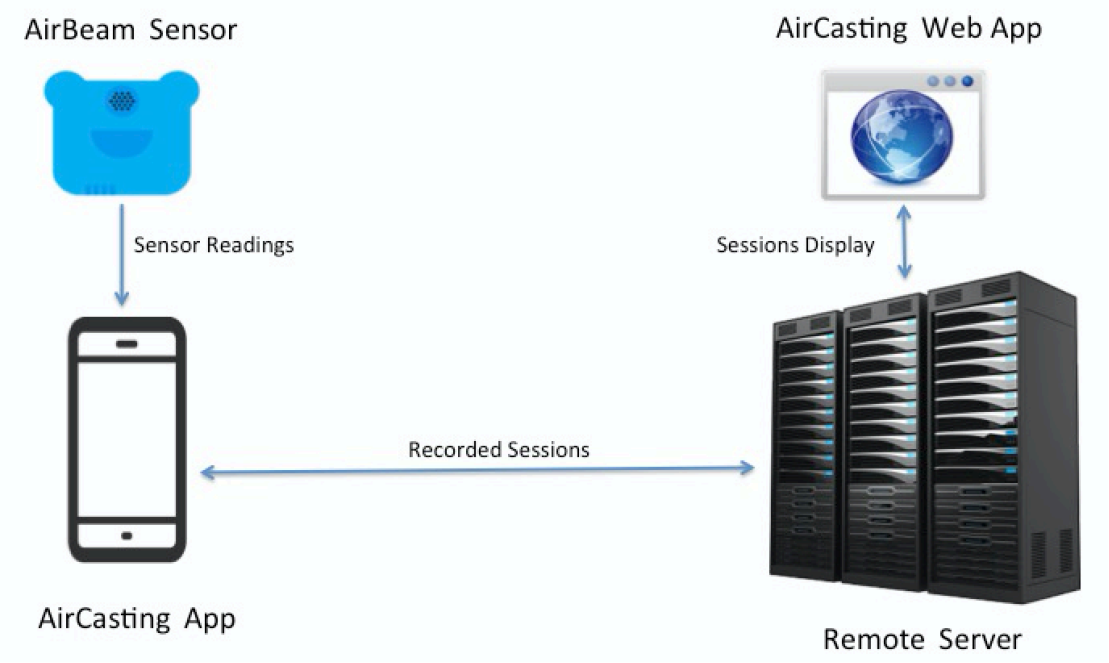

Figure 2. Current System Architecture

3.2. Data Storage Model

Data storage and management is a key aspect in this application, as the main purpose of the AirCasting application is to be able to collect data using the AirBeam device and represent it in a meaningful way for the users. In this section, I will be describing the architecture and workflow for data storage in the existing AirCasting application.

As shown in Figure 2, the current architecture employed by the application is simple and consists of a local database and a remote database. The local database is local to the mobile device. Most Android devices use a SQLite database as their local storage. The remote database is a Rails server hosted on a remote machine. A MySQL database is hosted on this Rails Server for data storage. This database holds data that are posted from all the users of the application. The Rails server also hosts a website which facilitate users to view their data on a map called the CrowdMap [include link for site]. This website pulls data from the MySQL database. The website is publicly accessible for anyone to view the route taken by a user and the pollution on that route.

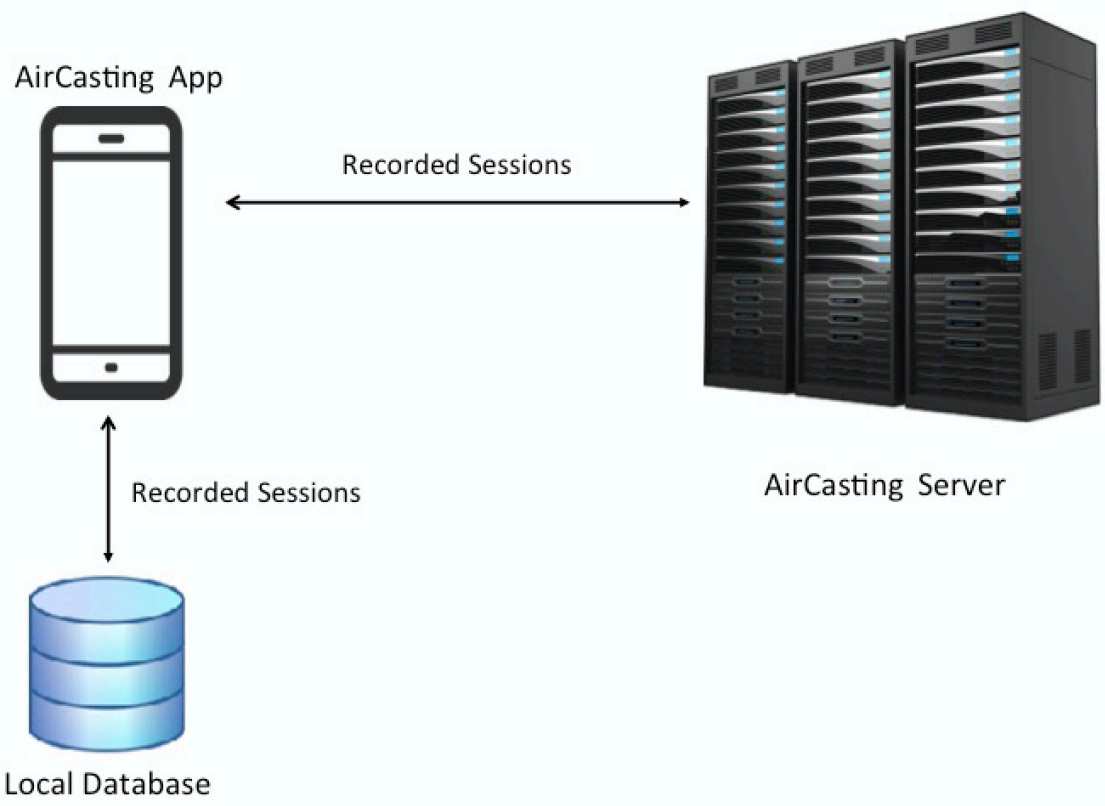

Figure 3. Current Database Architecture

The storage process is two stepped. When a user initiates recording on the device, all the pollution data that is being collected is first stored on the local database. After the recording is stopped, the user is presented with two options – either save the data only on the local database, or contribute the stored data to the remote AirCasting server which would be visible on the CrowdMap. If the user chooses the first option, the data remains only on the local database, and is accessible only by the owner of that data. If the user chooses the latter option, the data for that session is retrieved from the local database and is pushed to the remote AirCasting server. The data will be stored as a session posted by this particular user, which can be then viewed by the CrowdMap.

4.0 Design Implementation

4.1. Application Features

The application consists of three main views – Dashboard, Graph view and the Map view. The Dashboard is launched when the application starts up and displays the data that are currently being measured. The Graph view as the name suggests provides a graphical representation of the data being measured. And finally the Map view shows us the route we are taking while we are recording. It also plots the route taken for previously recorded sessions. For further details regarding the application

functionalities and design aspects please refer to Md Akmal Hossains report of the project.

4.2. Data Storage and Management

As explained earlier, the existing AirCasting application allows to store data locally or both on the local database and on the remote database. In order to store it on the remote database, the user has to explicitly choose the option "Save and Contribute" at the end of recording a session, other wise the data remains on the local database of the device. As an improvement to the existing workflow, we have incorporated an added functionality of being able to stream data in real time to the remote database. This feature pushes data to the remote database in real time, as the recording is in progress. This allows the user to not worry about explicitly pushing the data each time to the remote server.

This feature was added as a setting on the Settings Menu. Under the option "Upload Data" on the Settings Menu, the user can choose either the "Stream" or "Record" option. The "Record" option is the existing functionality of providing the user the option to either store the data locally or to contribute it to the CrowdMap. If the "Stream" option is chosen, when the user starts and stops recording, the data has already been stored in real time on the remote server.

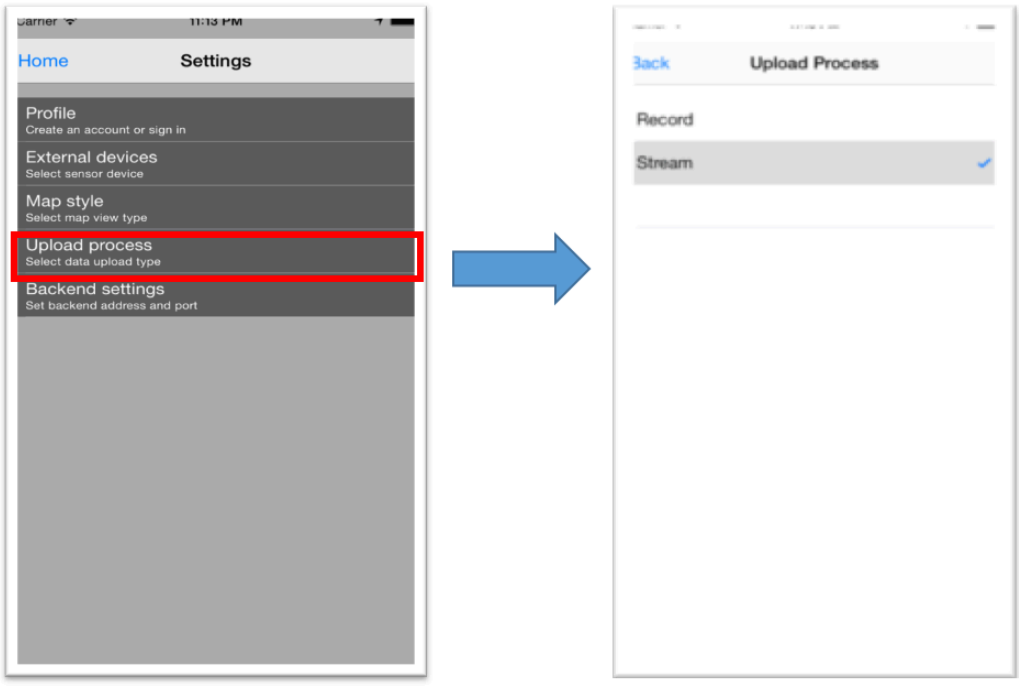

Figure 4. Menu for setting Upload Process

4.2.1. Local Database:

As mentioned earlier, the local database is a SQLite server. The schema of the local database has been created as an exact replica of the existing MySQL database on the Rails server. Given below are the database tables created for the application:

- deleted_sessions
- measurements
- notes
- regression
- sessions
- stream
- tagging
- tags
- users

These tables are created once when the application is installed and launched. All data stored on this database are retained until the application is uninstalled or deleted by the workflow of the application.

As mentioned earlier, for the "Record" option data is stored on this local database. When recording is initiated, the application starts a thread which invokes a function called 'updateMeasurements' every second. This function receives the pollution data being recorded by the application and stores it internally on the database. Along with pollution data, the user's current location is also stored in order to be able to represent the data on a map later on. Data is inserted into the SQLite database using an external framework called FMDB [8], which provides wrapper functions connect to a database, and perform all other DDL and DML statements like create, insert, update and delete.

Data points are collected every second, and longer the duration of the session being recorded, larger the amount of data being stored on the database. Hence if a user records several sessions with a very high duration, it will take a lot of storage space of the mobile device, and may cause risk of crashing if the device is running low on memory. This problem is solved by the "Streaming" option, as all the data is stored only on the remote server and not on the local database.

4.2.2. Remote Database:

The AirCasting application has employed a centralised database architecture for data storage and management. The Rails server hosts a MySQL database which is the central repository for all the AirCasting data contributed by all its users. The Rails Server has many REST services implemented in Ruby for storing, retrieving and

deleting sessions and users. The server is always kept in sync with the local database for a particular user. This means that a session being recorded will have a copy on the local database as well as on the remote database. This data redundancy has its advantages and its drawbacks. The advantages would be to safeguard against data corruption or a database crash for the remote server as that would mean that all the data is lost (due to the centralised architecture of the application). The drawbacks are extra usage of bandwidth, as the local and remote databases will always have to send and receive data in order to keep them in sync. Another drawback is as mentioned earlier, if a user has a huge number of sessions, with each session being of a very long duration, apart from the bandwidth being used, the amount of storage required on the local database would also be extensive.

In this project, we have simplified the process of managing sessions. Along with that, instead of a Rails back end server, we have adopted CouchDB as our back end database. With regards to sessions, there is no more the process of syncing sessions between the local and remote database. The remote CouchDB server will act as the single point of storage for all data. Sessions are stored on the local database only when the "Record" setting is chosen. Once those sessions are uploaded to the remote server, they will be deleted from the local database. This tremendously reduces the amount of data being sent across between the server and the mobile device. It also reduces the load on the server as the server has to manage multiple requests of syncing for all the users registered on the AirCasting platform.

The reason for choosing another database other than replicating the existing Rails server was as a solution to a challenge faced during implementation. Details on this will be discussed in the Challenges section below [bookmark]. Among the databases available in the industry now, CouchDB seemed the most suitable for this project, as it provides the necessary REST services for inserting and querying the remote database from the mobile application.

For the purposes of this project, we have created a virtual machine on the NeCTAR cloud [7], and installed the CouchDB database on it [couchdb url]. In order to store information, we have created to databases – aircasting_users, aircasting_database. The former stores data regarding all the users that are registered on the AirCasting application. The latter database stores all the session data from all the users. Session data for different users are identified by user id tagged for every document stored. All documents for a particular session is identified by a UUID which is generated from the application each time a session is created.

Data is stored in JSON format as key-value pairs. When data needs to be pushed to the remote server the application constructs the JSON in the required format and uses CouchDB's REST services in order to store the data on the database.

Below given is an example of a parent document that holds the session ID for a particular session, and other high-level information. You can see here that this document also contains the username and user ID which tells us which user this session belongs to.


```
{
   "_id": "04e5e213-5dc2-4a3d-9871-a0cf4c01b965",
   "_rev": "2-0dad1a25907c36821fc8202038a7a0c7",
   "username": "maggies",
   "phone_model": "iPhone4",
   "photo_content_type": "jpeg",
   "user_id": "6ca3e986-bfcf-4b21-8330-5d506f72a062",
   "photo_file_name": "null",
   "sensor_package_name": "AirBeam",
   "text": "To University from Windsor by Maggie",
   "photo_file_size": "1mb",
   "updated_at": "2015-10-27 20:15:51.738004",
   "session_id": "04e5e213-5dc2-4a3d-9871-a0cf4c01b965",
   "os_version": "iOS8",
   "date": "2015-10-27 20:15:51.737987",
   "photo_updated_at": "2015-10-27 20:15:51.738007",
   "created_at": "2015-10-27 20:15:51.738001"
}
```


Given below is the structure of JSON created for a session data. Here you can see a simple JSON message structure that holds all the measurement data for a particular recording. Each message has its own ID, as well as an ancestor_id, which tells us which session this particular measurement belongs to.


```
{
   "_id": "000a0c6b-324a-4500-9982-c62e13867db6",
   "_rev": "1-10701e635aafe5e9d74f9ba575150a55",
   "created_at": "2015-10-27 20:14:55.304528",
   "measurements": {
       "decibel": {
           "unit_symbol": "dB",
           "measured_value": 118
       },
       "temperature": {
           "unit_symbol": "C",
           "measured_value": 30
       },
       "particulate_matter": {
           "unit_symbol": "ug/m3",
           "measured_value": 5
       },
       "humidity": {
```

```
            "unit_symbol": "%",
            "measured_value": 72
        }
    },
    "longitude": "144.956698945015",
    "latitude": "-37.80123305507",
    "ancestor_id": "d071e1f9-4f7e-4df4-b0c2-7437a2a0fc69"
}
```


Given below is an example of a JSON message constructed when a new user is created. Here we can see a user ID is assigned for each user (the _id field). It also contains the password in an encoded format, along with other information of the user collected when creating an account through the application. All user data are stored under the aircasting_users database.


```
{
    "_id": "3C3447F1-E4C9-415A-8536-0D18FD540391",
    "_rev": "1-48f0d14011a741baeccf0e5c14d449f1",
    "last_sign_in_at": "Oct 31, 2015, 3:25 PM ",
    "sign_in_count": 1,
    "created_at": "Oct 31, 2015, 3:25 PM",
    "reset_password_token": "null",
    "encrypted_password": "dGhleWtpbGxlZGtlbm55",
    "email": "kenny@southpark.com",
    "username": "kennym",
    "updated_at": "null",
    "current_sign_in_at": "Oct 31, 2015, 3:25 PM",
    "reset_password_sent_at": "null",
    "send_emails": "yes"
}
```


4.4. Clean Route

In the market, there are several applications that allows a user to find the shortest path between two places, the most predominant application being Google Maps. But from the perspective of an environmental enthusiast, a user may want to travel through a path that is the least polluted, rather than the quickest. Since the main purpose of this application is the collection of data regarding air quality, incorporating the functionality of being able to get the least polluted route seemed like a significant addition to the application that complements its existing purpose.

To explain this functionality in more detail, using the AirCasting application, a user can enter a source (can be the user's current location) and a destination, and the application will plot on the map a route with the least pollution. Since the AirCasting

application collects data regarding pollution, it provides us with the necessary ingredients to be able to incorporate this functionality.

In order to be able to find the least polluted route, we need an algorithm that is capable of calculating this. There are several algorithms available in finding the shortest path between two points, the most commonly used one being the Djikstra's algorithm. But for our project we will be making use of the A* algorithm for finding the least polluted route.

4.4.1. A* Algorithm:

The A* algorithm is a much more efficient algorithm than the Djikstra's algorithm. Its logic is similar to how Djikstra is implemented, but it incorporates a heuristic to speed up the algorithm, thus making it efficient in finding the shortest path. The heuristic basically calculates which the shortest available cost from the current point to the destination irrespective of whether a path exists. Thus, the A* algorithm can be defined as a combination of the Djikstra's algorithm and the Greedy-Best-First-Search algorithm. [6]

This algorithm mainly works over two main lists. The first is called an open list, which is a sorted list of all possible paths adjacent to the current node, or in other words the neighbours of a node. The list is sorted based on the cost for each path to reach the goal. The second list is called a closed list, which contains the list of all nodes already traversed by the algorithm so that it does not traverse through them again. The algorithm traverses through the nodes in the open list. If the first item in the list is the destination node, then the algorithm stops as we have already found our shortest path, else it calculates the cost of neighbouring nodes to reach the goal and adds them to the open list, and the algorithm continues to traverse through the nodes on the open list until it reaches the final destination.

A node on the open list is chosen as the path based on its cost. In the A* algorithm, the cost is calculated using the function given below:

$$F(n) = G(n) + H(n)$$

Where:

G(n) – Cost of travelling from starting node to the current node

H(n) – Estimated cheapest cost to reach the destination from the current node (heuristic) [6]

4.4.2. Implementation:

A new view has been developed on the AirCasting application that provides an interface for the user to enter the source and destination addresses. On submitting the request, the application will query the database and calculate the least polluted route.

As mentioned earlier the A* algorithm is employed to find the least polluted route. Traditionally this algorithm is used to calculate the shortest path, as the cost calculated is the distance between points. In this scenario, the cost would be the combination of distance and the amount of pollution on a particular path. The reason for choosing a combination of both distance and pollution is because if we were to choose only pollution as the cost, in a worst case scenario, the path calculated by the algorithm could be a ridiculously long path, that even the most environmentally enthusiastic person may not be interested in choosing.

The A* algorithm is applied on a list of coordinates that are retrieved from the remote database. This list of coordinates to be retrieved is decided based on the source and destination entered by the user. To facilitate our testing, we have manually uploaded 17000 user points on to the database which represent routes from various parts of Melbourne. The steps for filtering the required coordinates from the database are described below:

1. First find the midpoint between the two points entered by the user. The midpoint can be found using the formula:

   Midpoint (lat, long) = (lat1 + lat2)/2, (long1 + long2)/2

   Where: *(lat, long)* - coordinates of the midpoint
   (lat1, long1) - coordinates of the source
   (lat2, long2) - coordinates of the destination

2. Calculate the distance between the midpoint and the source. Distance between two coordinates on the map was calculated using the function *distanceFromLocation()* which is provided by the CoreLocation framework available in Swift. This method takes an argument of type CLLocation and can be invoked as:

   distance = source.distanceFromLocation(destination)

   Where: *distance* – variable to store the distance in meters
   source – variable of type CLLocation holding the latitude and longitude of the source
   destination – variable of type CLLocation holding the latitude and longitude of the source

3. Using this distance as radius, and the coordinates of the midpoint as centre, find the list of coordinates that fall within the circle thus formed. This can be found by checking if a set of coordinates satisfy the equation:

$$(lat_m - lat)^2 + (long_m - long)^2 < radius^2$$

Where: lat_m – latitude of midpoint
lat – latitude of coordinate being checked if within circle
long – longitude of coordinate being checked if within circle
radius – radius of circle

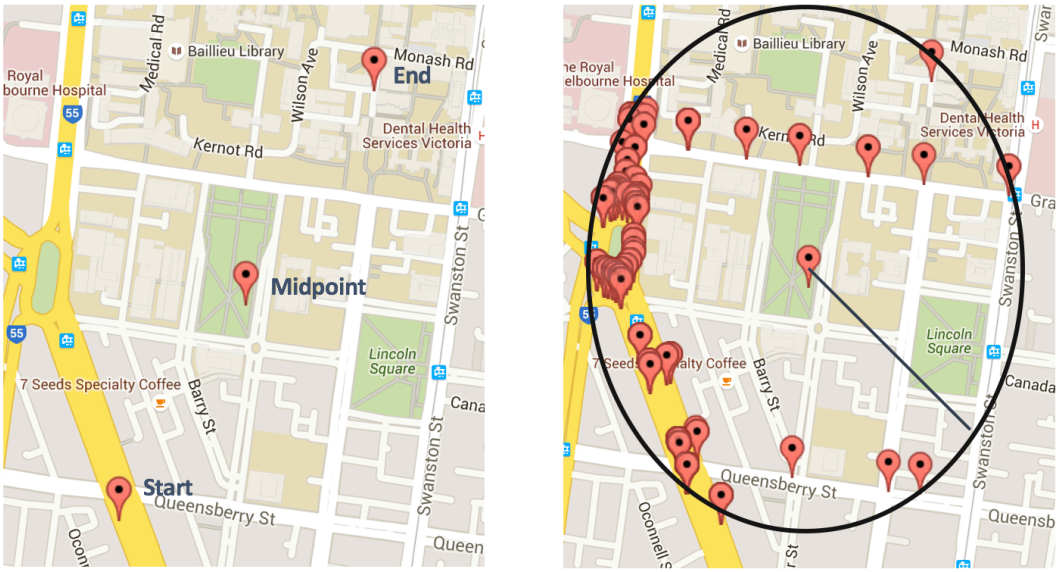

Figure 5. Map showing Start, End and Midpoint (Left), Map showing coordinates that fall inside circle of given radius (Right)

4. Using these values, the list of coordinates is queried from the database using CouchDB's REST Services. We have developed a map function called *find_latlong* that retrieves all coordinates and a list function called *filter_latlong* was applied to the output of the map function to filter it down to just the required coordinates. The list function contains the logic that validates whether a particular pair of coordinates satisfies the current range. The output of this step will be the required list of coordinates and the corresponding measurement values for each record.

5. Each record in this list contains the latitude, longitude, measurement value for decibel, temperature, humidity and particulate matter. In order to assign each record a pollution rating, which will contribute to the cost calculation in the A* algorithm, we find the maximum value for each measurement type from the entire list and divide the measured value of each measurement type for each record over

the maximum to get a percent. All four values are summed up and gives a pollution rating for a particular record which is added on the list.

6. Before the list of coordinates is fed into the algorithm, we need a way to identify a list of neighbours for each coordinate, as the algorithm depends on the list of neighbours for each coordinate. This can be achieved by iterating through the list, and for each coordinate on the list find the distance from it to every other coordinate and sort it by distance. The coordinates with the least distance would form the list of neighbours. But along with this we need one more validation. Even though the distance is less between two coordinates, it doesn't necessarily mean that it would be traversable between those two coordinates. In order to validate that it is, we make use of the MKDirections class provided by the MapKit framework which provides a route between two points. So for each list of coordinate shortlisted to be a neighbour, we should check whether a route exists to that point using this class.

7. Once the adjacency matrix is initialised, this is fed into the A* algorithm which has been implemented in Swift. In the class AStarRouting, there is a method called *calculateShortestPath()*, that takes the source and destination coordinates and the initialised matrix as arguments, processes it and finds the shortest path between the two points. The shortest path is again represented as a list of coordinates which is returned back to the MapViewController and the coordinates are annotated on the map which shows the user the shortest path from the source to destination.

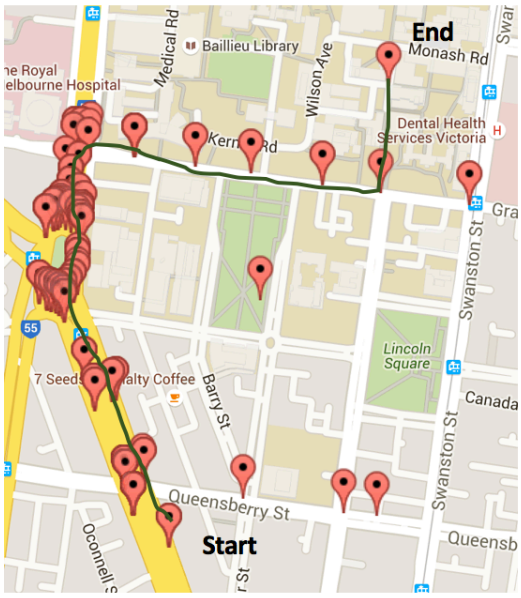

Figure 6. Least Polluted Route

5.0 Challenges and Solutions

5.1. Data Upload to Remote AirCasting Server

5.1.1. Challenge

In this project, as part of our efforts to try to replicate the existing architecture of the AirCasting application, we had replicated the backend AirCasting server on a remote machine on NeCTAR. We created a Rails server and configured a MySQL database with the schemas being the exact replica of the existing database. As part of our testing, we were able to successful post data to the server using REST services for creating a new user, and sign in as a particular user. But when we tried to post session data (i.e. data for a recording) to the database it kept failing.

As part of root cause analysis, we reverse engineered the REST requests and the data being posted to the server. We identified that the root cause was that the format with which Swift generates JSON was not compatible with the current implementation of services on the Rails server.

5.1.2. Solution

As a solution to this issue, there were two options. The first would be to re-write the current implementation of the services used to parse the incoming requests to suit the Swift JSON format. But any change to this service would render it incapable of catering to requests from the Android version of the application. Also this may also mean that it may be required to re-write other services which are used to handle other requests like deleting sessions etc.

The second option, and the option that was chosen as the solution is to use a CouchDB database as the back end storage server. This solution replaces the current Rails server database entirely. One of the many reasons that we chose CouchDB as our choice was the ease with which it can be installed and set up. And also incorporating it with the iOS application was seamless as CouchDB provides its own REST services for storing and retrieving data.

6.0 Future Directions

6.1. Distributed Remote Server

With the Clean Air and the CAUL project continuing its research in environmental science the amount of data being collected is increasing rapidly. The complexity of the data being stored and the complexities of the queries being run on the database would also increase eventually. Hence implementing a distributed server architecture as opposed to the current centralised architecture is a key area of improvement that needs to be considered for future development. Implementing a distributed architecture also facilitate scalability and overcomes the issue of a single point of failure. The data can be distributed across multiple nodes to reduce the load on one server, as well as we can store replicas of data on several servers to be capable of recovering from crashes.

6.2. Spatial Server

At present, all the data are stored on one centralised database in JSON format, as the storage database being used is CouchDB. The routing algorithm is being run on data being queried from the database on the client side. This is imposing a fair bit of load on the client, especially in the case where the routes requested is fairly long, as the number of coordinates that needs to be processed would be fairly large. As part of future development, incorporating a spatial database like PostGIS to hold the data required for routing would help off load the routing calculation from the client to the server. This would optimise the routing process as PostGIS is specialised in handling geo-spatial data and at the same time reduce the load on the client.

7.0 Conclusion

In this report, the iOS version of AirCasting application has been presented. The project has been targeted for replication of Android version of the application. The project has been planned with two scopes. First scope of the project refers to the development of User Interface of the application and key functionalities of the android equivalent. As an additional feature, real time data upload/streaming has been implemented within the application. Within the second scope of the project, comes the new functionality for calculating a route between a source and destination address based on the pollution level.

In conclusion of the project, key user interface features and functionalities have been completed within the project timeframe. Due to resource and time constraints, the scope of Clean Route functionality had to be revised to a modified version of A* algorithm.

8.0 Online Repositories

The project source code is available on GitHub via the below link:
https://github.com/MIT-FYP/AirCasting-iOS

The CouchDB server which is hosted on Nectar is accessible via the below link:
http://115.146.85.75:5984/_utils/

A video demo of the application is available on Youtube via the below link:
https://youtu.be/aCOzQRbdEGk

9.0 References

[1] '*About AURIN*', Australian Urban Research Infrastructure Network, viewed 8th August 2015, url <http://aurin.org.au/about/>

[2] '*AirCasting*', Habitat Map, viewed 5th August 2015, url <http://aircasting.org/>

[3] '*AirBeam: Share & Improve Your Air*, Habitat Map, viewed 6th August 2015, url <https://www.kickstarter.com/projects/741031201/airbeam-share-and-improve-your-air>

[4] '*AirCasting Application*', Google Play, viewed 2nd August 2015, url <https://play.google.com/store/apps/details?id=pl.llp.aircasting&hl=en>

[5] comScore, Inc,. 'Comscore Reports May 2015 U.S. Smartphone Subscriber Market Share'. N.p., 2015. Web. 3 Nov. 2015. <http://www.comscore.com/Insights/Market-Rankings/comScore-Reports-May-2015-US-Smartphone-Subscriber-Market-Share>

[6] Theory.stanford.edu,. 'Introduction To A*'. N.p., 2015. Web. 3 Nov. 2015. <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

[7] Nectar,. 'Nectar Cloud - Nectar'. N.p., 2015. Web. 3 Nov. 2015. <http://nectar.org.au/research-cloud/>

[8] GitHub,. 'Ccgus/Fmdb'. N.p., 2015. Web. 3 Nov. 2015. <https://github.com/ccgus/fmdb>

[9] workshop, Clean. 'Clean Air And Urban Landscapes (CAUL) Hub - Information Session & Workshop - Events - Green Building Council Australia (GBCA)'. *Green Building Council of Australia*. N.p., 2015. Web. 3 Nov. 2015. <http://www.gbca.org.au/events.asp?eventid=33297&source=course-event-calendar>

[10] The CAUL Hub,. 'The CAUL Hub'. N.p., 2015. Web. 3 Nov. 2015. <http://urbannesp.com/>

[11] Smartcitizen.me,. 'Smart Citizen : Citizen Science Platform For Participatory Processes Of The People In The Cities'. N.p., 2015. Web. 3 Nov. 2015. <https://smartcitizen.me/>