



University of Glasgow | School of
Computing Science

Dynamic Noise and Pollution Campus Map

Petar Plamenov Yordanov

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — February 23, 2015

Abstract

With the increased need to monitor the environment in the recent years, people have developed multiple ways to observe and study fluctuations in nature. Temperature, air humidity, wind speed and air pressure are just some of the variables that are constantly monitored. We can find up to date information about these in the news or check them online at any time. However, in most cases the generation and collection of such environmental data is highly limited to specialised agencies such as forecasting and meteorologist companies working for television channels. There is nothing surprising so far, due to the fact that in order to acquire precise information, professional equipment that is carefully configured is required, or at least that was the case until recently. As technology advances on a daily basis, the sensors required to monitor the world around us have become more and more readily available.

This project aims to showcase valuable ways of putting such specialized technology into practice by utilizing an innovative sensor board, the Smart Citizen Kit. Some of the ways this goal can be achieved is by demonstrating practical ways of interacting with trustworthy environmental information (noise and air pollution), such as pairing the board with web services (facilitating the development process for mobile applications) and creating highly-customizable dynamic maps that are updated in real-time and can be used for route recommendation. Extensive evaluation consisting of a pilot and a final session with volunteers has been carried out to test the system as well as collect feedback and suggestions for future improvements.

This is an innovative project and there have been no previous attempts to create similar systems relying on the Smart Citizen Kit. Hopefully it will create a stable foundation for future development in the area.

Acknowledgements

I would like to thank Dr. Iadh Ounis and Mr. Richard McCreadie for supervising the project and helping with ideas as well as facilitating the functional requirement gathering procedures for the system. Many thanks to the Smart Citizen Kit co-founder Tomas Diez and his developer team from FabLab, Barcelona for the remote support and also to all volunteers who agreed to take part in the evaluation.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Contents

1	Introduction	1
1.1	Background	1
1.1.1	Smart Citizen Kit	1
1.1.2	Web Services	2
1.1.3	Mobile Web Applications	2
1.2	Aims	3
1.3	Motivations	3
2	Context	5
2.1	Related Work	5
3	Planning	8
3.1	Issue Analysis	8
3.1.1	Battery life	8
3.1.2	Sensor exposure	8
3.1.3	Data Spikes	9
3.1.4	Sensor accuracy (frequency of measurements taken)	9
3.1.5	Optimisation of the data space	9
3.2	Implementation Techniques	10
3.2.1	Agile Development Sprints	10
3.3	Requirements	10
3.3.1	Use Cases	10
3.3.2	User Stories	11
3.3.3	Functional, Non-Functional Requirements	11
3.3.4	Non-Functional Requirements	12
4	High-Level Design	14
4.1	Diagrams	14
4.2	Paper Prototypes	14
4.2.1	Server	14
4.2.2	Client	15

4.3	Technologies	15
4.3.1	Server	16
4.3.1.1	Web Application Framework	16
4.3.1.2	Asynchronous JavaScript Library	16
4.3.1.3	CSS Framework	17
4.3.1.4	DBMS	17
4.3.1.5	Map	17
4.3.2	Client	18
5	Implementation	19
5.1	Technologies Used	19
5.2	Prototype Iterations	20
5.2.1	Server	20
5.2.2	Client	20
5.3	Final Implementation	21
5.3.1	Android Client	21
5.3.1.1	SCK Configuration	21
5.3.1.2	Implementation Logic	22
5.3.2	Mobile Web Application	26
5.3.2.1	Back-end	26
5.3.2.2	Data Formatting	27
5.3.2.3	Logging	28
5.3.2.4	Implementation Logic	28
5.3.2.5	Front-end	36
6	Testing	43
6.1	Unit and Integration Testing	43
6.2	Performance	44
7	Evaluation	45
7.1	DB Benchmarks	45
7.2	Prototype Evaluation	47
7.3	Planning	47
7.4	Pilot Evaluation	48
7.5	Final Evaluation	48
8	Conclusion	49
8.1	Summary	49
8.2	Future Work	49

8.3	Lessons Learnt	49
Appendices		54
A	Appendices	55
A.1	Appendices	55
A.2	Resources	55
A.3	Progress Reports, Sprint Retrospectives	55
A.3.1	Progress Report 1	56
A.3.2	Progress Report 2	57
A.3.3	Progress Report 3	58
A.3.4	Progress Report 4	59
A.3.5	Progress Report 5	61
A.3.6	Progress Report 6	63
A.3.7	Progress Report 7	64
A.3.8	Progress Report 8	65
A.3.9	Progress Report 9	66
A.3.10	Progress Report 10	67
A.3.11	Semester 1 Retrospective Report	68
A.3.12	Progress Report 11	73
A.3.13	Progress Report 12	74
A.3.14	Progress Report 13	75
A.3.15	Progress Report 14	76
A.3.16	Progress Report 15	77
A.3.17	Progress Report 16	79
A.3.18	Progress Report 17	80
A.3.19	Progress Report 18	81
A.3.20	Progress Report 19	82
A.3.21	Progress Report 20	83
A.3.22	Progress Report 21	84
A.4	Evaluation Documentation	85
A.4.1	Consent Form	86
A.4.2	Introduction Script	87
A.4.3	Task Sheet	89
A.4.4	Debrief Script	91
A.4.5	Task Load Index Form	93
A.5	Requirements Gathering Session	94
A.6	Images	95

Chapter 1

Introduction

1.1 Background

In the recent years, knowing how to interpret environmental changes around us has become a more and more integral part of maintaining nature clean and unpolluted. Climate changes have been a common occurrence on many places around the globe, influenced by increased air pollution largely due to industrial activity and exhaust gases from motor vehicles. The advent of high performance electrical vehicles such as Elon Musks Tesla car model and Solar Citys solar panels have influenced the situation for the better [64] [63]. However, the significant transition from oil and petrol to sustainable energy will probably require at least a few more decades.

What this means is that we need to be able to monitor the environmental aspects affected by these factors. This will facilitate the observation and analysis of air and noise pollution in order to determine what effects they have on nature and how those detrimental effects can be reduced with alternative mechanisms such as solar panels and wind mills to facilitate the transition to a more environmentally-friendly future.

Aiming to enable every conscious citizen to participate in this process, the Fab Lab startup, located in Barcelona, Spain, have started the Smart Citizen initiative [60]. Tomaz Dies and his team have come up with an innovative piece of technology, the Smart Citizen Kit[61]. The project is still under development, but two versions of the new kit have been released to the general public, the first line being distributed among the first backers of the project who funded the Kickstarter campaign that made it all possible [59]. This kit represents a sensor board that is readily-available and uses open source hardware and software to monitor the environment.

The Smart Citizen Kit device is at the heart of this project as it aims to enable normal citizens to collect comprehensive environmental information so that it can be aggregated at a centralized server for dynamic visualizations at the fraction of the cost that would be involved if the same technology was purchased separately, possibly from different sources.

1.1.1 Smart Citizen Kit

The Smart Citizen Kit is a crowd-sourced project that started from a KickStarter campaign about two years ago (beginning of June, 2013, by Tomas Diez, Fab Lab director) [59]. Since then, 2 versions of the kit have been released, the first one being distributed to the 517 project backers. The second, version 1.1 improves the user experience as a whole and features a new kit firmware[60][58]. A version 2.0 of the API and web application is currently under development It is an open source platform, consisting of the following logical layers - a hardware device, a web service API (REST-based) and a web application.

The Smart Citizen Kit has the following sensors (figure 1.1a) that are used for data collection: solar, air humidity and pollution (CO and NO₂), temperature and noise sensor. It also constantly monitors the wi-fi network availability(figure 1.1b) in the area to maintain high connectivity rates as data is transmitted to the server every 60 seconds, unless the firmware is re-programmed. The hardware is built atop an Arduino Leonardo

screens as on regular desktop monitors. Mobile web applications might also include additional functionalities considering the fact that a large number of them have specifically been designed for access from mobile devices. Some examples of additional resources utilized include the GPS navigator, accelerometer, etc. These features improve the performance of the application as well as make it more user-interactive.

There has been an ongoing debate whether it is better to develop mobile web or native mobile applications[56]. Native web applications are easier to optimize for hand-held devices as there are a number of tools to facilitate the development process and there are frameworks in place to ensure that design and structure are consistent on all devices (Google Android SDK is an example [2]). What is more, they tend to perform faster than mobile web applications in a lot of cases. On the other hand, a carefully optimized mobile web application using a cloud hosting and a NoSQL database can successfully compete with a native app in terms of performance, as the latter still requires database access for core operations more often than not. However, native web applications have their drawbacks as well. They have more pre-requisites for end-users as they require the application code to be installed on the dedicated device, in the form of a client. This means that every time there is a new version of the software, for the changes to be synchronized, users need to make sure they install the latest update. Introduction of restructuring operations on a mobile web application is a much smoother transition as user feedback and performance can be tracked right away triggering further modifications if necessary, while with a native application this is not the case.

This project applies the client-server model (figure 1.2a) and implements a best-case scenario by utilizing the benefits introduced by native code clients (Android application in this case) and a responsive RESTful web server in the form of a mobile web application (figure 1.2c)). This aims to ensure end users get the best of both worlds by making use of the stable user interface provided by the client application and utilizing the processing power of a back-end server engine communicating directly with a NoSQL database instance.

1.2 Aims

The system aims to provide a responsive graphical user interface that will enable users to view environmental data collected by themselves and others. It seeks to provide up-to-date information no matter where the user is by pairing the Smart Citizen Kit with the GPS sensor of a mobile device - meaning that end-users will be able to walk around and benefit from the crowd-sourced database while at the same time contributing and generating more data that gets aggregated on the map right away. Users will be strongly encouraged to contribute to the process by collecting data in order to improve the overall quality of the mobile application. The application aims to primarily satisfy the needs of students and staff from the University of Glasgow. However, it could also be used by managers from the City Council to create a large-scale map of the city and identify the most polluted areas in order to take measures to reduce the level of toxic gases and potential noise disturbances. In other words, this project could serve as a small-scale proof of concept that would hopefully encourage funding and be used as a resource for future environment-related work.

The final product's goal is to improve the overall experience of students and staff who live and study on the premises of the Glasgow University Campus by helping them to identify areas that have higher air pollution levels (NO₂ and CO tracking) so that they can be avoided to improve health. It will also show the quietest places that are suitable for strolls or when studying for exams.

Based on the aggregated data, the system seeks to provide customizable route recommendations that can be based on different sorting criteria, such as air and noise pollution, route distance, accessibility and more.

1.3 Motivations

This project utilizes the precision and flexibility provided by the Smart Citizen Kit to enable everyone to contribute to the cause of monitoring our environment, keeping it clean and minimizing the detrimental effects resulting from human activity. Having 5 high-quality sensors on a single board provides a lot of opportunities

for application development in this area.

As explained in the opening paragraph, air and noise pollution are topics that are discussed more and more often nowadays. While factories and traffic cause noise pollution and indirectly affect us psychologically (by creating noise disturbances, etc) during their operation process, they also constantly emit toxic gases that negatively affect our health. A lot more can be said on this topic, but the question remains whether contemporary citizens can participate more actively in this process so that we can all take responsibility for the improvement of the situation. And a dynamic system that can aggregate and visualize environmental information on a single map, paired with the Smart Citizen Kit is the great foundation for the achievement of this goal. Furthermore, this data aggregation process can be used to extend the application and implement a route recommendation feature (based on pollution levels). In addition, as backed up by the section above 1.1.3, the system that this project aims to implement will make use of the 'thin client' programming concept to minimise the computational logic in the client application and rely on a server for the data processing operations. This will facilitate server updates without the strict necessity of introducing a new client version and require end-users to download and install it every time a new feature has been implemented, aiming to attract more potential users.

Chapter 2

Context

2.1 Related Work

There have been multiple systems that have aimed to achieve similar goals, whether collectively in a single project or in separate components. Below is a short description of related work along with the identification of key differences when compared to the aims of this project.

Ear Phone[73](figure 2.1a)

This is a product from the CSIRO organisation, short for Commonwealth Scientific and Industrial Research Organization, which is the Australian National Science Agency. The innovative idea behind this software tool is that it is a crowd-sourced way to collect and analyze information about noise pollution. It does not rely on any investment from people who would like to participate in the process in the sense that the final product is just open source software. Everyone can install it on their phone and it aims to conserve resources as much as possible, working in the background as long as your mobile device is on. There are some ethical constraints, that Rajib Rana's team claim to have successfully addressed. The application does not track your microphone input while you are talking using a smart artificial intelligence engine to differentiate between surrounding sounds (classified as noise) and normal conversations (that are disregarded by the engine). It also makes use of the phone's GPS and accelerometer sensors in order to determine the location of the device only when it is outdoor (as GPS coverage within buildings is not reliable and reduces battery life faster) and handheld (the orientation of the phone is used to determine this). All of the collected information is transferred to a centralized server once the phone is connected to a Wi-Fi network. A major drawback of the system is the fact that those operations affect the battery and could reduce the phone's usage time to a few hours.

The system described above differs from the 'Dynamic Noise and Pollution Campus Map' in terms of the way data is collected. The latter collects a much wider range of information and takes care of mobile device battery's life by utilizing a specially tailored technology for data collection - the Smart Citizen Kit. It has precise sound and air-pollution sensors, making the scope of the system much wider. What is more, the phone serves only as an intermediary and transfers information between the sensor board and a centralized server, increasing the potential time for application usage.

AQICN[47]

Air Quality Index CN(figure 2.1b) is an organization located in China, Beijing. It consists of a small team of system engineers, who has relied on the US Embassies in China for the initial provision of PM2.5 for many cities in Asia and more specifically China. These are precise sensors that enable accurate air pollution monitoring in PPM (parts per million). The project is constantly expanding, with more sensors being statically installed throughout cities. It spans the United Kingdom as well and there are hundreds of sensors on the territory of the islands. The highest concentration of tracking devices is in Scotland, around the area of Glasgow and Edinburgh(figure 2.1c), two of the cities taking significant participation in this environmental monitoring process ultimately aiming to reduce air pollution.

The main difference in what 'Dynamic Noise and Pollution Campus Map' aims to achieve is the degree of interaction with the end user. The 'AQICN' system is aimed primarily for scientist who can effectively analyse information retrieved from static sensors and act upon it, while the main goal of the project is to involve end users as much as possible, involving a portable sensor device, meaning that once pollution data is transmitted to the server, a user can see the effect it has had on the aggregated map right away. Additionally, the scope is different as 'AQICN' operates solely with air pollution sensors.

Dust Maps[71]

This is another interesting low-cost crowd-sourced system. It relies on a fine particle dust sensor in order to produce air pollution maps in different locations via grid representation. This is almost entirely the proprietary technology of KIT (the Karlsruhe Institute of Technology). The researchers there have utilized a number of off-the-shelf components, the main one being a fine dust sensor. This sensor's accuracy is in the range of one microgram per cubic meter, which is more than enough for smoke and coarser dust. However, it is not sufficient to trap fine dust (which could be improved). The sensor technology is paired with the camera of a mobile device, such as a smartphone and this allows for quick real-time data processing, followed by transmission to a processing server's back-end where the information gets aggregated on a map (figure 2.1e). An interesting approach to incentivize more people to participate in the data collection process is the offer of bonuses and prizes from the developers.

One of the main difference from the project described in this paper is that the prototype does not look very easy to use (figure 2.1d) and although the team have promised to improve the style and introduce a convenient clip to hold the sensor, it would still make using the smartphone relatively inconvenient as well as endanger the fine sensor when it is carried around. 'DNPCM' uses dedicated technology to monitor a user's surroundings and involves the prototype of a compact enclosure that can be attached to a backpack or carried in a pocket as long as good ventilation is provided. The Arduino board serving the sensor board processes additional variable information, which can be used to efficiently scale the system and improve its monitoring capabilities without changing the hardware used for data collection.

There are also some interesting papers that relate to the idea behind this project. Here is a little information in the form of an abstract about each of them:

The '**ExposureSense**'[72] project in Switzerland involves the deployment of a number of sensors such as CO and NO₂ sensors on public transport vehicles: trams, buses, etc. ExposureSense is a more advanced version which can also monitor people's daily activities and compute the levels of pollution they have been exposed to throughout their daily life. Another big change is the support it provides for additional sensor data collection by plugging new sensors to improve the quality of air data collection. This paper also describes a crowd-sourced system as it relies on smart phones and the voluntary participation of people. The project described in this paper does not involve noise pollution tracking.

The article '**Noisemap: multi-tier incentive mechanisms for participative urban sensing**'[69] describes the Noisemap project and how it achieved its success. As the name suggests, this pollution system tracks noise levels solely. Issues related to finding participants have been described as well as different ways to overcome them. One of the main hindrances when testing such systems is that the device battery life decreases dramatically due to the use of power to feed the sensor as well as the need of accurate GPS tracking and server data transmission. The authors of the paper claim that by using incentive schemes they managed to increase the number of participants from 402 to 3 357 and for about 2 months each of them had captured data in the average duration of 6 hours (the total measurements this experiment captured were more than 85 000 which is considered a successful outcome).

The main difference with 'DNPCM' is that the scopes of the projects differ and the paper described above focuses on the evaluation component while this paper regards system development and its testing and evaluation as equally important integral part for successful project outcome.

The paper '**Mobile Application for Noise Pollution Monitoring through Gamification Techniques**'[70] is different in the sense that it provides a more cost-efficient evaluation technique. The main idea of the mobile application that is described in the project is that noise pollution can be efficiently crowd-sourced if relying

on two main points - avoiding using external technology and relying on microphones present in any smartphone nowadays as well as using gamification techniques in order to gravitate more people to the data collection process.

The second point of the approach is significant as it removes the need to provide other incentives such as participant payments or prizes that can make the scaling of the process prohibitively expensive and not viable. The paper also reiterates the fact that local authorities can make incredible use of this crowd-sourced information to improve the quality of life for their citizens.

As with other papers, the scopes of the projects are different, as 'DNPCM' aims to analyse noise and air pollution.



(a) Ear Phone system architecture



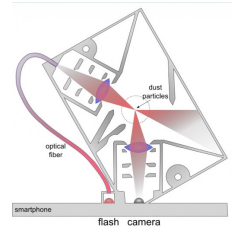
(b) AQICN Air Pollution Index Project



(c) Glasgow Dumbarton road air pollution



(d) Dust Maps Final Assembly View



(e) Dust Maps internal sensor structure

Figure 2.1: Similar projects that have implemented software engineering products.

Chapter 3

Planning

3.1 Issue Analysis

The achievement of the goals that this project has set as a target has been facilitated by leveraging an efficient mobile web application and a client application for data collection and processing. The client communication with the sensor board is the main source of environmental information. However, a number of concerns that have arisen at the early planning phase after some research were addressed at the start and possible resolutions were considered. Some of the main ones have been mentioned in the sections below:

3.1.1 Battery life

With the continuous use of GPS and accelerometer sensors there comes the risk of draining any smart phones battery at a very high rate. In order to address this concern in the final implementation, a step that was taken in the planning process was to use the thin-client design pattern. This involved extracting most of the computational logic to the server when designing its architecture from the early start. What this means is that when data is collected the mobile device will use only the bare minimum of resources required to locate the device, retrieve the sensor information and transmit to the centralized server for processing. This guarantees a much more battery-friendly flow of operation in this application. The frequency of updates was also a significant concern as it is directly tied to the accuracy of the system and inversely proportional to the battery life duration (as the frequency of measurements increases, the processing power needed increases and so the operational time of the device is reduced).

3.1.2 Sensor exposure

The air and noise sensors do not have protection as the board has been designed to be minimalistic in order to avoid additional space constraints and to allow users to experiment depending on their particular needs. In the aim of making sure that the SCK remains intact throughout the whole development and evaluation process, a prototype carriage case was produced (as described in section 7.2, chapter 7). This case uses the location of the key sensors used in this system to avoid the risk of microclimate development as much as possible by introducing ventilation holes. They ensure an improved air circulation and increase the quality of the information retrieved from the surrounding environment. For this system to function as expected and utilize the precise board sensor capabilities, it needs to ensure that the sensors have a good air ventilation and there are no barriers that present a risk to the accuracy of the data readings. Otherwise the data might become skewed and the performance rate would drop dramatically.

3.1.3 Data Spikes

Sensor readings sometimes generate data spikes which might be an unreliable representation of the surrounding environment (data formatting might influence this too). This might largely influence the statistical information for an area on campus. One way to reduce the impact of this is to use range modes when tracking information, meaning that if mode 1 is between 0 and 5 ppm, mode 2 - between 5 and 10 ppm, and so on, a difference between two consecutive data readings cannot span a difference of more than a single mode. In other words, data can be classified from mode 1 to mode 2, but the jump between 1 and mode 3 would classify such a data measurement as unreliable, and so it could be disregarded. Calculating the average value of data readings collected from the same location is another option. This is an operation that will limit the risk of losing valuable information that is collected by participants in the evaluation and will decrease the influence of data outliers on the final results. This approach has been employed when designing and implementing the system visualization modes as it is considered to be the favourable one in terms of effectiveness.

3.1.4 Sensor accuracy (frequency of measurements taken)

The environmental updates transmitted by the Smart Citizen Kit are sent every 60 seconds by default. However, when considering this time window in the context of this project it becomes clear that if the board is constantly moving this time interval might be too long. The software can be overridden so that environmental data measurements are taken every 30 seconds. Doing this more often (than 30 sec.) is not viable as transmission costs become too high in terms of charging power. Wi-Fi connection instabilities also cause update congestions meaning that, sometimes multiple updates are not timely transmitted to the server (as concluded after testing). Another consideration is to give the end user the option to increase this interval depending on the movement speed. For instance, if the device is static, a few minutes between updates might be sufficient, meaning that the SCK will still save data every 30 seconds, but the Android client will not request a new update until the user-specified temporal interval has expired.

3.1.5 Optimisation of the data space

The visualization of information on the map involves a working space constraint that needs to be carefully addressed to prevent extreme rendering times and guarantee system scalability as well as easy data browsing for end users. The maximum expected system data load is as follows:

- 10 days data generation, approx 5 hours/day (50 hrs)
- 10 evaluation participants (500 hrs total)
- 2 data readings/ min. (default settings)

$$500 \text{ h} = 500 * 3600 \text{ s} = 1\,800\,000 \text{ s}$$

$$\text{DR} = 1\,800\,000/30 = 60\,000 \text{ records}$$

To tackle the large data set visualization, some of the following approaches can be used:

- Data aggregation - this option will involve multiple readings being aggregated into a single one, which is then stored in the database or store all the information in the relevant collection and then aggregate it at run-time. This would involve grouping each 5 (for example) sensor data readings and finding their average $(dr1 + dr2 + dr3 + dr4 + dr5)/5$. It is important to note that considering each 5 data readings are grouped with a 30 sec. window, the system is going to be deriving a single data reading aggregation each 2,5 min. which, for the Glasgow University campus area, could be considered normal, taking into account walking speed.
- Changing the data post window (increase to reduce number of sensor readings) - least desirable, as the system requires more granularity and data accuracy.

The notes here have provided food for thought when choosing the solution mechanism that has been applied in the final implementation (5.3).

3.2 Implementation Techniques

The design structure of the system was a crucial part of the planning process. The client-server model is important for providing responsive behaviour and abstracting business logic to the server, reducing the size of the Android client as much as possible. Apart from having positive effect on the device's battery life, this is also more secure as the potential vulnerabilities are reduced and the client cannot potentially be exploited to hack the system.

Data storage was also an integral part considering the expected data load the system was designed to sustain. SQL ([55]) and NoSQL ([54] and [49] were researched) solutions were considered for this and after a series of tests to compare them, the relationless structure that Mongo DB provides proved to be much more efficient and suitable for the nature of this project.

3.2.1 Agile Development Sprints

In order to ensure that an optimized development process was used and a high standard was maintained, the Agile development practice (manifesto [45][46])(figure 3.1b) was employed. Two-week sprints were used to implement major system components and streamline the process. The progress made on each iteration was communicated and evaluated so that any required changes could be implemented in a short time span (figure 3.1a). Progress reports were produced on a weekly basis to present the newly implemented features and specify the ongoing work(appendices, A.3).

3.3 Requirements

The requirements for the system were thoroughly discussed with the project supervisors to make sure that project planning is consistent with the target end result (Q&A session, A.5).

3.3.1 Use Cases

The use cases for the project were considered to improve the structure planning process. The final software engineering product would enable final users to:

- discover quieter areas on campus (to be used for outdoor/ indoor studying /i.e. exam revision/ or relaxation for instance) (figure 3.1c)
- find the areas with least polluted air (for running, training of any form, healthy benefits or just for leisure activities in general) (figure 3.1d)
- identify routes suitable for strolls (around Kelvingrove park for example, which is in close proximity to the university campus) (figure 3.1e)
- help users who would like to meet new people in the cases when increased noise levels correspond to crowded venues
- could also be useful for Glasgow City Council and help them evaluate the technology and apply it to improve city life in general
- serve as a foundation for more similar university research/software engineering projects
- serve as a ground base and aid the further development of the innovative kit

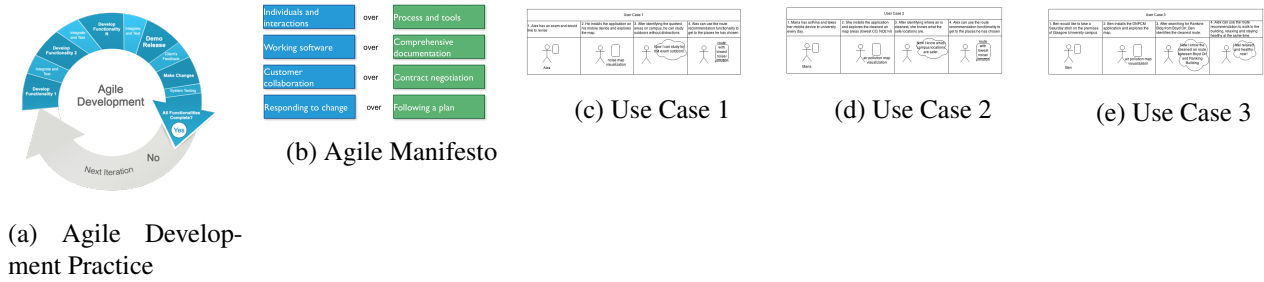


Figure 3.1: Project development practice (a,b). Use Cases (c,d,e).

3.3.2 User Stories

The ‘As a user, I want to’ template was used to devise a final list of user stories that were split into smaller chunks (issues) and assigned to backlogs (GitHub repository issues: [12]). Each Agile sprint had a corresponding backlog of issues that had to be implemented to consider the iteration successful. One of the main advantages of this template is that the features that need to be implemented can be prioritized very clearly and directly. Feedback from colleague students was very crucial in this process, as recording their thoughts on how such a product can be viable it was possible to construct a stable base of user requirements. The final result is presented in the following paragraphs. Below is a table showing the identified user stories:

User Requirements
‘As a user, I want to see the data visually in order to understand it easier. For example, with different colours, shapes, or styles in general.’
‘As a student, I want to be able to see the noise and pollution data for different locations around campus to check the air quality around the ‘Student Apartments’ building where I live or check popular routes in the area.’
‘As a user (student), I want this application to be user-friendly and intuitive so that I can use it to get the latest data on my mobile device /phone, tablet/ whenever I need to.’
‘As a user, I need to be able to see different data routes around Glasgow University campus in order to draw conclusions about different areas.’
‘As a user, I want to be able to choose from different visualization map styles in order to see correlations and data patterns better.’
‘As a user, I would like to be able to filter the data in order to display only routes/ locations that I prefer.’
‘As a student, I would enjoy being able to see aggregated data statistics such as most popular or quietest route in order to know where to go if I need to study for exams.’
‘As a user, I would like to be able to see a wider range of information on the campus map in order to compare different locations comprehensively.’

Table 3.1: Table representing requirements as user stories.

3.3.3 Functional, Non-Functional Requirements

The ‘MoSCoW’ requirements rating system has been used to prioritize the functional requirements.

Functional Requirements
MUST
1. Use the SCK 1.1 to generate sample data that can be used to create a prototype of the final product by simulating the mobile client and distributing the sensor readings across different latitude/longitude in the vicinity of the university campus.
2. Gathered data must be processed and visualized on the server as a dynamic map. Readings information must be aggregated to enable statistical evaluation.
3. SCK sensor readings must be paired with mobile device GPS data (latitude/ longitude) to establish positioning (geolocation).
4. The dynamically generated noise and air-pollution campus map must be accessible on the client carrier device. The server's UI must follow the mobile web application design standards and be compatible with mobile devices.
SHOULD
5. Gathered information should be used to generate different routes/ paths around the Glasgow University campus. This can later be used to generate statistical data and feed it back to the user /for instance, most popular paths, quiet and fresh air routes, etc./.
6. Multiple visualization styles should be implemented/ tested. Some examples include - heat style map, general gradient map, color-coded blocks/ grid elements, etc. This is necessary so that user feedback can be used to improve the functionality of the mobile web application.
COULD
7. When significant amounts of data have been acquired, some filters on this information could be applied. For example, sliders letting the user specify maximum/minimum values for noise or pollution and allowing them to combine those.
8. Aggregated data can be used to generate statistics (in a row-based textual manner for instance) based on different criteria. Some options: quietest to loudest route / location freshest to most polluted air route / location most popular to least popular route longest to shortest route
WOULD
9. Apart from the main information about noise and pollution presented to the users, the system can also show additional data from sensor readings (examples: time, SCK battery for convenient monitoring, etc.).

Table 3.2: Table representing functional requirements prioritized by using the MoSCoW rating model.

3.3.4 Non-Functional Requirements

Similarly, the non-functional requirements have been identified:

Non-Functional Requirements
'When using the application, I must be able to quickly determine what my exact location was when tracking a route.'
'I must be able to access the mobile web application on my device and learn how to use the graphical interface within an hour of usage.'
'I need to see data visualized on a map and be able to interpret the interface state with ease within an hour of using the application.'
'I need to be able to see my route on the map and I should learn how to track my location and movement within an hour of application usage.'
'I need to learn how to select multiple map/visualization styles after 1 hour of using the system.'
'A nice feature would be to be able to filter data. This needs to be intuitive so that I can use it with ease after 1 hour of system familiarization.'
'It would be nice to see some statistical data. I need to be able to interpret all the statistical data displayed and browse through it with no errors within 1 hour of application usage.'
'It would be nice to see temporal filtering as well, for example. This parameter should be easily accessible when browsing through the data.'

Table 3.3: Non-functional requirements to specify the criteria associated with expected system behaviour.

Chapter 4

High-Level Design

4.1 Diagrams

During the planning phase, the system was iteratively designed using structural diagrams to provide a solid base for the implementation phase of the project. A high-level framework diagram was prepared in order to identify the key system components and their interactions (figure 4.1a). A mobile web application diagram for the server was also designed to better illustrate the client-server communication using the application layers (application programming interface, API) (figure 4.1b). In order to define the key user-system interactions, a sequence diagram was generated (figure 4.1d). The modelling of the data source structure was presented in the form of an entity-relationship diagram (figure 4.1e). Another representation of the data store in a Compressed Chens (bubble) Notation is directly below:

```

1 [ User ] 1 ----- < has > ----- N [ Device ]
2 [ Device ] 1 ----- < provides > ----- N [ Data Reading ]
3 [ Data Reading ] M ----- < forms > ----- N [ Route ]
4 [ Device ] 1 ----- < has > ----- N [ Route ]
5

```

A class diagram was designed in the planning stage and has now been modified according to the final implementation (figure 4.2a, 4.2b).

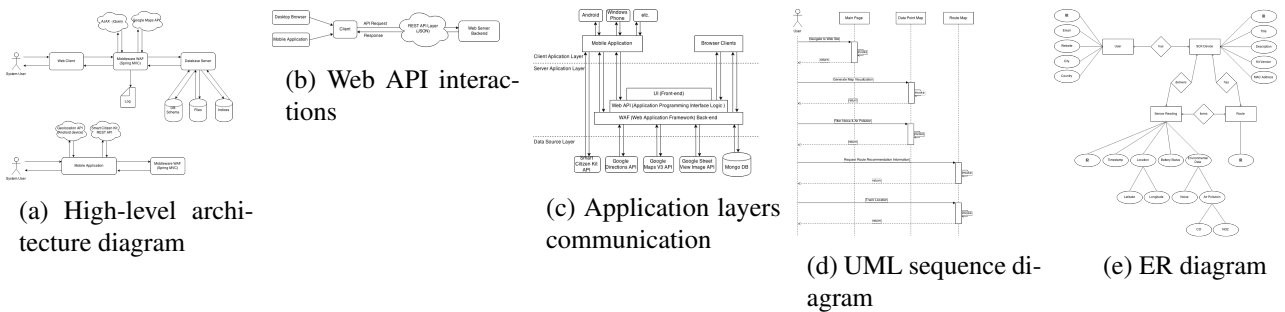


Figure 4.1: System structure diagrams

4.2 Paper Prototypes

4.2.1 Server

The user interface of the server application was designed via paper prototypes. This allowed to shape the basic framework that was used for prototype development in the initial Agile sprints. Below on figure 4.3 can be seen the main views and UI elements that were designed:

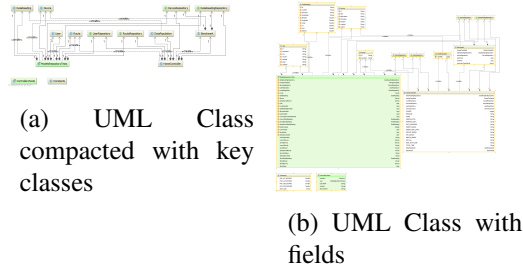


Figure 4.2: UML Class diagrams

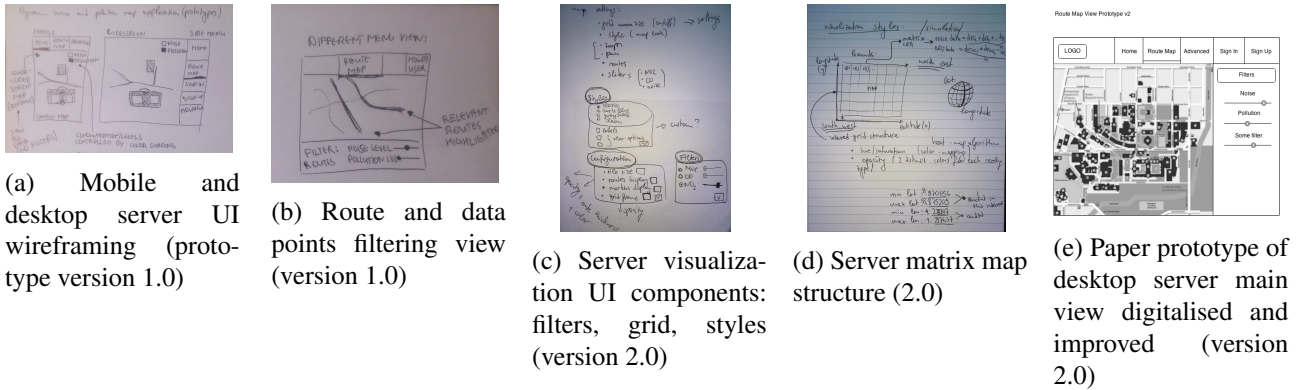


Figure 4.3: Server application UI paper prototyping

4.2.2 Client

The user interface of the client route tracking application was also designed with the help of wire-framing paper sketches and digital UI mock-ups. The main iterative versions of the client prototyping process can be inspected on figure 4.4 below (there is also a mock-up of the server on a mobile device to show how the client follows the same structural style for usability purposes).

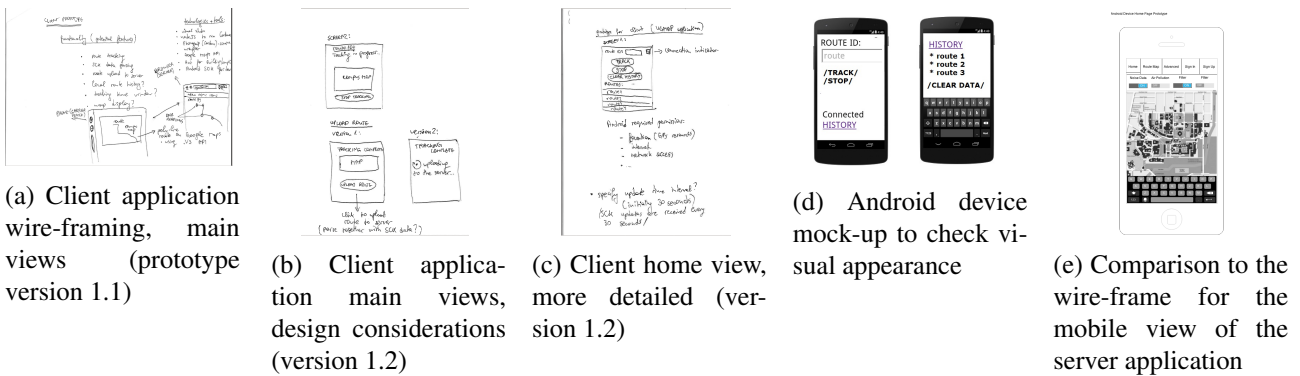


Figure 4.4: Client application UI paper prototyping

4.3 Technologies

The choice of technologies that were used in this project required careful considerations of different frameworks and components in order to ensure that the final product is as streamlined as possible. A table of the final choices is available in Chapter 'Implementation', section 5.1.

4.3.1 Server

4.3.1.1 Web Application Framework

The design of the mobile web application involved 3 key requirements for the web application development framework that was going to be used - customizability, scalability and high modularity (in order to improve code maintenance and facilitate new component implementation). Consideration with regard to available documentation, online resources and potential future expansion of the project was also taken. The main web application frameworks that were compared were Spring MVC[42], Django[8] and Flask[20].

Customizability was an important factor from the start as the system built in this project required specific data manipulation and the chosen framework needed to conform to this in order to enable the successful server implementation. Starting from a structure complexity comparison, Flask[20] (Python-based) is the simplest of the three options, followed by Django (Python-based) and Spring MVC (highly-customizable). While Django[8] is a Python-based system and facilitates quick base application implementation providing a stable core that is sufficiently configurable, it still lacked the levels of customizability Spring MVC provides. Flask, being a Python-based micro-framework, appeared a very attractive option from the start as it is even further simplified than Django and significantly less complex than Spring MVC. However, its ORM support in the face of SQLAlchemy did not seem to be well documented. Although there were detailed instruction pages on the official website, code listings seemed to be lacking some specific sections meaning that building a highly-customized project using this framework would not have been as efficient as using the other two options. All in all, Spring MVC[42] was the dominant choice in this aspect. Flask's ORM support would also affect the scalability of the application in terms of the expected main data set growth rate (number of datapoints in the database).

Considering documentation and support resource availability online, Spring MVC and Django were favorites. Django was the more light-weight option. However, Spring MVC was very promising in terms of component scalability and modularity - high cohesion and low coupling. This meant that if future development involved multiple developers working together to expand the system server, Spring MVC would facilitate and smoothen this transition with the modularity levels it provides. Using a Python-based framework such as Django or Flask would reduce the levels of boilerplate code but it would be much less scalable in terms of functionality (a simple application implemented in Python involves much less source code; as more components get introduced in the system, however, Spring MVC as the Java-based option presented a much better alternative). After researching Spring's support for web service implementation, it became clear that it is very stable and particularly suitable for REST-service development (especially if using Spring's annotation-based configuration functionality).

Web Application Framework	Complexity	Customizability	ORM Support	Documentation	Maintainability
Flask	✓(very low)	✓	✗(low)	✗	✓
Django	✓(very low)	✗(relatively low)	✓(good)	✓(good doc.)	✓(good)
Spring MVC	✗(high)	✓(highly customizable)	✓(very good)	✓(good documentation)	✓(great)

Table 4.1: A comparison between web application frameworks (WAF).

4.3.1.2 Asynchronous JavaScript Library

In order to provide an immersive user experience a responsive user interface was required. Asynchronous JavaScript (the jQuery[31] JavaScript library) was used in order to reduce the number of calls to the database and facilitate visualization generation without the need of page refreshes. Apart from reducing the potential negative impact of Wi-Fi network bottlenecks, this also ensured timely response times and quick map rendering.

An alternative option to jQuery was considered, Zepto.js[44]. This is a new, open source library that aims to remove some of the constraints that jQuery imposes and its developers claim that it is essentially an improved modification of jQuery. A huge drawback for this library, however, was that the main source of support was the official website and it is not as widely used, hence there is much less online documentation for it in general.

4.3.1.3 CSS Framework

The choice of CSS framework was also very important. Two main tools were considered in this section - Twitter Bootstrap 3[43] and Boilerplate[7]. They both have advantages and disadvantages. Boilerplate allows developers to create a stable HTML5[28] base which can be used as a standing platform to create highly-customized templates. However, it does not include all of the necessary components to build a fully functional website. On the other hand, Bootstrap is able to provide the full package required to complete a fully-functional front-end structure and also is quite flexible at the same time. Furthermore, the server application that was required for this project had to be specifically optimized for mobile devices with smaller screens. The grid structure Bootstrap provides appeared to be very suitable for the needs of the mobile web application from the start as it has different categories of CSS classes based on the screen size, making the control of GUI appearance highly effective. These arguments influenced the choice of Bootstrap for styling the implementation of the server's front-end.

4.3.1.4 DBMS

In terms of database scalability, the maximum expected load (much higher than the evaluation numbers were, with pilot evaluation including 4 participants and final evaluation - 5) on the system server was considered. Assuming 10 days of data generation and 100 participants, with approximately 30 min./day data collection per participant, the total daily collection is 5 hours, and total number of data points is about 60 000.

In order to determine the most suitable DBMS to handle this data load comprehensive benchmark tests were carried out on an SQL and a NoSQL (MySQL[55] and Mongo DB[54] accordingly) database systems (expanded in section 7). Specific database structure in the face of ER modelling is a priority in an SQL database engine such as MySQL. On the other hand, the relationless structure of NoSQL makes it much more flexible and improves the efficiency of producing different views of the data. In summary, results have shown that in terms of data insertion and retrieval speed, Mongo DB far outperforms MySQL even when taking into consideration the efficient data-caching MySQL provides when utilizing prepared statements.

4.3.1.5 Map

The choice of the map engine was also crucial during the planning and design stages as this is one of the main components of the system's UI. Three highly functional options were considered after thorough research - Google Maps API[26], OpenStreetMap[38] and MapBox[33]. OpenStreetMap was an interesting option as after some research it turned out that the API it provides works very well with OpenLayers (an open source, high-performance library for map overlays and customizations). However, the drawback with this engine was that since it is user-generated, there was the risk that it is not 100% reliable (although it still provides effective routing services using different modes, such as - foot, bicycle, car, even horse; very similar to Google Maps[25] in this sense). On the other hand, while MapBox seemed much more reliable, hosting a huge map database on its servers, it turned out that the developers have extensively limited the features provided in the free version of the service. The free 'Starter' plan has a limit of 50, 000 views per month (which is not a huge drawback considering that some time will pass before the system starts generating such web traffic) and although it provides a list of predefined map styles, users cannot use custom styles (a significant drawback since one of the main criteria was the option to produce customized map views to highlight streets or satellite mode, especially for route recommendation). Google Maps JavaScript API v3 has a daily limit of 25 000 map loads (also more than enough) and additionally, each aspect of the map can be customized (examples - keyboard shortcuts, UI components - pan, map scale, zoom levels, street view, etc.). Another important advantage that tipped the scales in Google Maps' favor (it is

the final implementation choice) was the detailed documentation that is available online as it is currently the most used service in this category.

4.3.2 Client

In order to implement the data collection and route tracking Android client, a couple of alternative solutions were considered. One option was to implement the application using the native approach, utilizing an IDE such as Google’s Android Studio[22]. This is one of the most efficient ways to develop Android applications. However, native code is written in Java and the transferable Map API usage with the server was important as it provides more stable compatibility of the client and server system components. The server would be using Google Maps API v3 (built for JavaScript) while the client would have to use v2 (Java API implementation). What is more, writing native code reduces the portability of the final product to other mobile OS-s such as Windows Phone and Apple iOS.

An alternative solution that solves these disadvantages is the PhoneGap[39] open source framework for mobile application development using standardized web APIs. It uses the core Apache Cordova’s engine[4] and is thoroughly documented as well as widely used and highly recommended (as it became apparent after careful research). Additionally, the framework which integrates JavaScript in the functionality of the compiled sources with the help of the Node.js runtime environment has a build service supporting multiple OS-s (among them are Apple iOS, Google Android and Windows Phone). The advantages PhoneGap demonstrated (as listed above) made it the final choice for the client implementation. The two approaches were compared based on these criteria and Apache Phonegap was selected as the better choice for the needs of this system. The use of API v3 would be a great advantage as it means that it is compatible across the whole system (server and client application). Last but not least, PhoneGap is has a very good maintainability level. A summary of the comparison can be inspected below:

Client Im- plementation Approach	Lightweight	Customizability	Documentation	Maintainability	Cross- platform Compatibility
Native Android Development	✓(can be optimized very well)	✓(great configuration options)	✓(good)	✗(more difficult to maintain for a small-scale app)	✗(requires porting the whole implementation)
PhoneGap App. Container	✓(lightweight compiled source)	✓(great customizability)	✓(very well-documented)	✓(good doc.)	✓(great, allows cross-platform source compilation)

Table 4.2: A comparison between native mobile application development and using an application container such as Apache PhoneGap.

Apart from Node.js[37], other pre-requisites for Android development using PhoneGap included Android SDK[2], Microsoft Visual Studio[34], Apache Ant[3] (for project building) and ADB[1] (Android Debug Bridge - used for early emulator testing).

Since the jQuery library was the tool of choice for the development of the server component, this made its mobile version - jQuery Mobile [32] a prime candidate for the JavaScript operations required in the client implementation.

Chapter 5

Implementation

5.1 Technologies Used

The 'Planning' section 3 describes the software development tool considerations. Based on the arguments presented in that section, the final choices for the whole system have been presented below:

System Component	Implementation Tool Choice
Server	
Web Application Framework	Spring MVC [42] (as explained in section 4.3.1.1)
Logging	Apache Slf4 Java facade [6] to facilitate debugging and maintenance
Project Management	Apache Maven [5] for efficient project dependency management and maintenance (section 4.3.1)
Web Server	Eclipse's Jetty Web server and javax.servlet container [30]
JSP (JavaServer Pages) for front-end structuring	
JSON Parsing	Google's Gson JSON [24] object manipulation library
Asynchronous JavaScript	jQuery [31] JS library (section 4.3.1.2)
CSS Framework	Twitter Bootstrap 3.3 [43] (section 4.3.1.3)
DBMS	Mongo DB [54] (a NoSQL solution, chosen after comprehensive benchmarking 7)
Map Service	Google Maps (API v3[26], section 4.3.1.5)
Client	
Development Engine	Apache PhoneGap 3.0[39][4] (section 4.3.2)
SDK	Android Software Development Kit[2]
Testing & Debugging	Android ADB (debug bridge)[1] for device emulation
JavaScript Library	jQuery[32] mobile, specially designed for mobile device front-end elements manipulation
Project Management	Apache Ant[3] for building and compilation to source files (Apache Maven was an option as well, configuration requires the installation of more prerequisites)
Map Service	Google Maps (API v3[26] to maintain compatibility across the whole project (section 4.3.2)

Table 5.1: Final implementation tool choices.

5.2 Prototype Iterations

This section highlights some of the main steps involved in the first Agile iteration which was followed by a retrospective and feedback user sessions that fed into the final implementation.

5.2.1 Server

The first server iteration involved the implementation of the base visualization functionality. The user interface was highly limited in order to identify the most suitable plotting techniques for the map structures. A simulation was generated using static data (from a single location) previously generated by the Smart Citizen Kit device. More information about this first prototype is available below. As the client prototype had not been implemented at this stage, a simulation was created by generating random locations within the bounds of the map and assigning them to `dataReading` objects when parsing the data in `'parser.jsp'`. The utility function `'randomPosGen(lowLatBounds, highLatBounds, lowLonBounds, highLonBounds)'` was used in order to generate the random locations (a visualization of 500 data readings is shown in figure 5.1b). As displayed in 5.1b, the server is logging each mouseover event via a marker listener to aid debugging. Data readings were plotted on the map surface with different color-coding highlighting different variables as shown in figure 5.1c. The circle radius depends on the sensor reading value. The opacity/transparency of each circle is calculated using the formula $x\%/100 = \text{currentValue} / \text{maxValue}$ where `maxValue` is in the non-healthy interval for the specific variable and `currentValue` is the sensor reading value.

After completing this process, extensive testing and evaluation of the whole client-server architecture was conducted (more details in the 'Evaluation'[7] and 'Testing' [6] chapter) and the opinion of potential users was requested so that the plan for the final implementation could be refined. This would include generating more visualization modes and implementing a user interface as well as a route generation functionality.

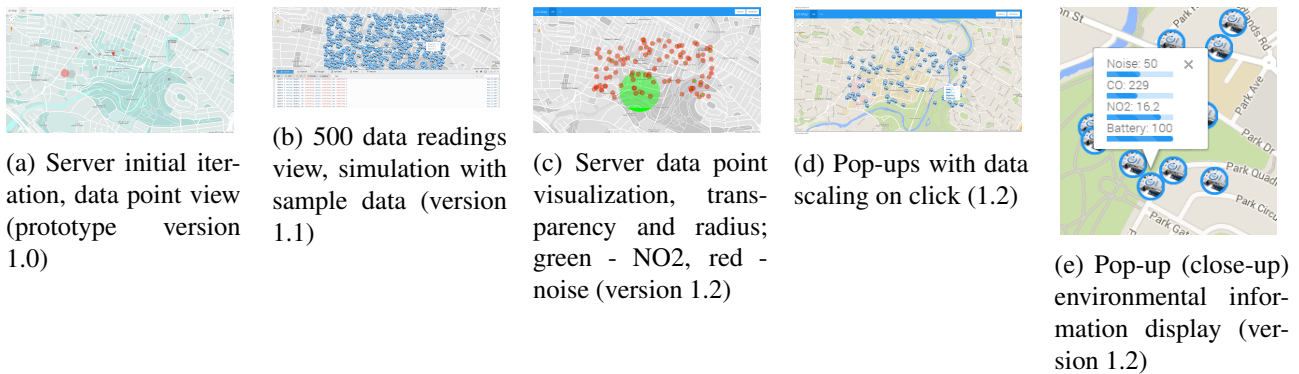


Figure 5.1: Server application UI paper prototyping

5.2.2 Client

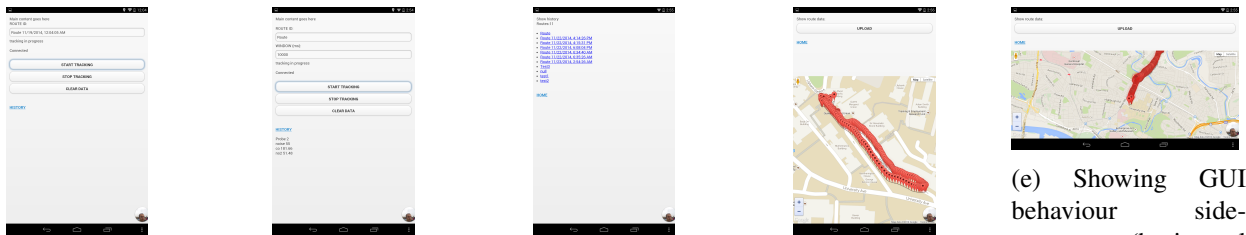
The client prototype was implemented iteratively, component by component. The first feature that was the device state checking - this needs to be done to ensure that the application is fired up without errors. The application needs Wi-Fi access and network state checking to determine device connectivity as well as GPS data access. Hence, the relevant Android permissions were enabled:

```

1 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
2 <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
3 <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
4 <uses-permission android:name="android.permission.INTERNET" />
5 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

```

Figure 5.2a below shows the main screen of the prototype. It has a field for route identifier - the ID can be generated automatically using the following format: 'Route_timestamp_', if the field is left blank. The main view has the basic route tracking buttons - 'Start' and 'Stop'. 'Clear Data' can be used to delete the local tracking history. Figure 5.2b shows an improved version, including a temporal window field (can be used to specify time window between updates in milliseconds) and sensor data feedback displayed in real-time below the main buttons. There is also a link to the history page. Once a user has completed recording a route, they can browse their history and access information from their previous route tracking sessions (figure 5.2c). The links in the list can be tapped/ clicked to access the route meta-data page displaying the geolocation and contextual information via Google Map API v3 pins/ 'Marker' and 'Polyline' objects (as shown on figures 5.2d and 5.2e).



(a) Client version 1.1 main screen

(b) Client main screen (version 1.2) - temporal window field added

(c) History page view, clickable routes (version 1.2)

(d) Route page view, data points (map markers) hold environmental data (v. 1.2)

(e) Showing GUI behaviour side-way (horizontal wide-screen mode)

Figure 5.2: Server application UI paper prototyping

5.3 Final Implementation

5.3.1 Android Client

5.3.1.1 SCK Configuration

The first step was the SCK device firmware update. The Smart Citizen Kit team had just released a new software version to drive the sensor to ensure that it matches the new API calls that they had previously introduced on their main server. There are multiple ways to do this - using the service on the official web page after installing the 'Codebender' extension on Mozilla Firefox and Arduino's drivers (this runs an applet which automatically uploads the latest firmware version /the SCK 1.1 is currently equipped with 0.9.0 [40]/ on the board), manually editing the configuration files in the firmware package and uploading them to the device or using the Arduino IDE in order to set the parameters via command-line commands. The third approach was chosen as it is the most efficient in the cases of firmware modification.

In order to optimize the process of collecting data and at the same time ensure that frequent sensor probing is carried out (addressing the points identified in 3.1, 'Planning' chapter 3) some important configuration measures need to be taken. One of the main aspects is the size of the temporal window between two consecutive updates. It needs to be small enough to allow for sufficiently frequent environmental data probing and at the same time ensure that the risk of network congestion (and data losses during Wi-Fi transmission) is reduced to a minimum. After significant testing with different window sizes (5, 10, 20, 30, 60 (default update time) seconds) it was concluded that the optimum choice is a time window of 30 seconds. This is much better than the default time imposed by the Smart Citizen Kit (60 sec.) and at the same time conserves device (SCK and carrier - tablet, smart phone) battery power as well as prevents network congestion. As described in the 'Issue Analysis' 3.1 section GPS tracking and sending updates over a Wi-Fi network are the primary potential bottlenecks of this system and they were eliminated in the initial implementation stage. The modification is highlighted in the code listing below:

```

1  /*SENSOR READINGS constants*/
2
3  #define DEFAULT_TIME_UPDATE 30 //time between two consecutive updates is 30 seconds
4  #define DEFAULT_MIN_UPDATES 1 //minimum number of updates before posting
5  #define POST_MAX 20 //max number of postings at a time

```

Before each configuration can be tested, after the board has been connected via USB, the Wi-Fly module needs to be activated using Arduino IDE's 'Serial Monitor': '\$\$\$' (using the *115200 baud* and *"No line return"* options). After this, the network parameters need to be set using the *"Carriage return"* option with the following commands - *"set wlan ssid SSID"* (save the Wi-Fi id), *"set wlan key KEY"* (password for WEP and WEP64) (the Glasgow University campus 'eduroam' Wi-Fi network uses this wireless encryption protocol), *'set wlan phrase PHRASE'* (WPA1 and WPA2), *"set wlan ext_antenna X"* (configure the device to use the external antenna using parameter '1'). The final command is *"set wlan auth 4"*. The code '4' is used for WPA2 as there appears to be a security issue when connecting to the university network and this made it necessary to use an alternative approach - set a smart phone as a mobile access point (acting as a Wi-Fi network provider). The device's MAC address can then be retrieved *"get mac"* and utilized to connect the sensor to the main server. In order to get back to the normal operational mode the *"exit"* command can be used.

The main server (<http://smarcitizen.me/> [61]) was used in the process as a monitoring tool to check how updates in transmission frequency affect the performance of the system. After the configuration of the device software is completed, the output it produces when probing its surroundings can be inspected using the 'Serial Monitor' in Arduino IDE. The format of the output is listed below:

```

1  Please, wait wifly sleep
2  *****
3  SCK Connected!!
4  updates = 13
5  Posted to Server!
6  Old connection active. Closing...
7  Temperature: 26920 C RAW
8  Humidity: 26216 % RAW
9  Light: 88.90 lx
10 Battery: 100.00 %
11 Solar Panel: 0 mV
12 Carbon Monoxide: 191.53 kOhm
13 Nitrogen Dioxide: 26.56 kOhm
14 Noise: 0 mV
15 Wifi Spots: 3
16 UTC: 2014-12-12 2:42:21
17 *****

```

5.3.1.2 Implementation Logic

The Android application implementation with the chosen tools requires the configuration of PhoneGap, Android SDK and ADB as well as Apache Ant.

After this process, the target application version needs to be specified using the Android SDK manager and configuring Node.js - the minimum Android OS version the client supports is 1.9 so that older devices can still install and run it without issues.

```

1  <!-- android_parser.js -->
2  <uses-sdk android:minSdkVersion="10" android:targetSdkVersion="19" ... />

```

A configuration 'xml' script for the application needs to be prepared to include the metadata when compiling the source code (the PhoneGap[39] build service was used for this task as it enables hydration/timely synchronization so that new changes become visible in the compiled files immediately/, facilitates quick extraction for Windows Phone[68], Android[48] and iOS[48], supports private application hosting as well as provides direct code pulling from GitHub). The final configuration script (based in 'config.xml') is structured as follows (with the relevant Android APIs included):

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- config.xml reference: https://build.phonegap.com/docs/config-xml -->
3  <widget xmlns = "http://www.w3.org/ns/widgets"
4  xmlns:gap = "http://phonegap.com/ns/1.0"
5  id = "UGMAP"
6  version = "1.0.0">
7  <!-- project name, description and icon settings -->
8  <name>Dynamic Noise And Pollution Campus Map</name>
9
10 <description>
11 Generate a dynamic campus map via the innovative Smart Citizen Kit, developed by Fab Lab, Barcelona.

```

```

12 </description>
13
14 <preference name="phonegap-version" value="3.0.0" />
15 <icon src="img/icon.png" />
16
17 <author href="http://ppyordanov.com" email="ppyordanov@yahoo.com">
18   Peter Yordanov
19 </author>
20 <!-- api libraries used -->
21 <feature name="http://api.phonegap.com/1.0/file"/> <!-- used for storing the route data locally on the carrier device; provides
22   access to carrier device media contents, used in route data retrieval -->
23 <feature name="http://api.phonegap.com/1.0/geolocation"/> <!-- used for accurate geolocation tracking using the device GPS -->
24 <feature name="http://api.phonegap.com/1.0/network"/> <!-- facilitates Wi-Fi network resource utilization for environmental data
   transmission to the server and network state checking (connected/not connected)-->
</widget>

```

The high-level structure of the client involves introducing a minimal GUI (in order to keep the 'thin client' development paradigm) with the key functionality needed for effective geo-location tracking and environmental data probing. Structural simplicity is crucial in this component.

A single template page has been used to design the main views. It has the following structure:

```

1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <title>UG Campus Map Application</title>
6   <meta name="description" content="Noise and Pollution Glasgow University Campus Map Application">
7
8   <!-- main scripts imported: jquery-1.6.4.min, jquery.mobile -->
9   <!-- css frameworks and styles - jquery.mobile, main style -->
10   ..
11   <script type="text/javascript" src="js/custom/utility.js"></script> <!-- utility functions for environmental data retrieval and
   parsing -->
12   <script type="text/javascript" src="js/custom/track.js"></script> <!-- main functions for server data transmission and user
   interface interactions control -->
13 </head>
14 <body onload="init();"></body> <!-- initialize network connection status check -->
15
16 <!-- HOME VIEW -->
17 <div data-role="page" id="home"> .. </div>
18
19 <!-- HISTORY VIEW -->
20 <div data-role="page" id="history"> .. </div>
21
22 <!-- ROUTE VIEW -->
23 <div data-role="page" id="route"> .. </div>
24
25 </body>
26 </html>

```

The home view (as shown in figure 5.3a) is the main page of the application. There are 5 key elements on this page:

- a 'Route ID' field - this is the route identifier that is automatically generated in case this field is left blank (it has the format "Route TS" where TS is the current time stamp in the form of a JavaScript Date() object)
- 'Time Window' - this field is used to specify the temporal window for environmental data probing (in milliseconds); the default and minimum value is 30 seconds to match the number of times the SCK board transmits updated to the device (which is one of the reasons for the minimum window constraint). Input in this field is validated - if a numerical value cannot be parsed or it is less than the minimum, the default value remains unchanged (defined as a global variable for consistency maintenance throughout the full program run-time cycle):

```

1   var DEFAULT_WINDOW = 30 * 1000;
2   ..
3   var winVal = $("#window").val(); //input element id content retrieval via jQuery
4
5   //if field contents are empty, 'not a number' check is false and the value is greater/equal to 30 seconds, update the window value
6   if (winVal != '' && isNaN(winVal) == false && winVal >= 30000) {
7     WINDOW = parseInt(winVal);
8   }
9

```

- 'SCK Device API Code' - this is used to identify the Smart Citizen board that the client communicates with via the main SCK API to retrieve information about its surroundings. This means that the Android application can work with any SCK device and what is more, it can transmit updates remotely without the need to be present at the location (due to the fact that there is an intermediary server associated with the communication), making the data collection process even more streamlined. In case the API code provided by the user is invalid (an error response is received, that is) the application informs the user and falls back to using the default authentication code provided.
- a connection indicator - this is used to check whether the device is currently connected to a Wi-Fi network.

The implementation is based on the usage of the 'navigator' JavaScript object:

```
1  if ((navigator.network.connection.type === 'none' || navigator.network.connection.type === null ||
2     navigator.network.connection.type === 'unknown' )) {
3     $('#connection').html('Not Connected'); //update the status of the application
4     $('#start').prop('disabled', true).addClass('ui-disabled'); //disable the tracking button
5 }
6
```

- The 'Track' button is used for starting and stopping route tracking sessions. It triggers an event which outputs a 'flash' message to the user depending on whether the process starts or ends ('Tracking in progress!': figure 5.3b and 'Completed tracking and transmitted data to the server!': figure 5.3c accordingly on successful server response). This message is dynamically generated via using jQuery[31] and has a 'fade out' effect at 3000 ms timeout. While tracking is in progress, the user can inspect the number of probes that have been collected in the current session, the context variable levels, geo-location data as well as time stamp information. Route tracking relies on the use of the HTML5 geo-location API[21] which has a good support in most browsers (became available around the end of 2010 for Opera Mobile). The 'localize()' function pairs environmental information to geo-location data. It uses the 'navigator.geolocation.watchPosition()' call to retrieve the current location of the device, saving this in the 'location_id' variable. The function is also called with an interval 'trackIntervalId' (inherited by the value of the temporal window variable). These variables are later used to stop tracking the location data, clear the function call interval and store the data on the device prior to server transmission:

```
1  //track.js
2  //stop getting current location
3  clearInterval(trackIntervalId);
4  navigator.geolocation.clearWatch(location_id);
5
6  window.localStorage.setItem(route_id, JSON.stringify(route_data)); //store the route information
7
```

On successful location retrieval, the 'GEO_LOCATION' object is populated with the current lat/lon data and then passed to the 'generateData()' function to pair it with the values for noise, NO and CO2. Failures to connect to the server are logged to the console as error messages. 'trackIntervalId' ensures that the function is called every 'n' seconds while tracking. The timeout is set to 5 seconds to ensure high precision and the 'enableHighAccuracy' option parameter is enabled:

```
1  //utility.js
2  function localize() {
3      var options = {
4          enableHighAccuracy: true, //configure accuracy
5          timeout: 5000, //set request timeout
6          maximumAge: 0
7      };
8
9      var GEO_LOCATION = {}; // initialize the data object
10     var start_location_loaded = false; // a flag to check whether the initial location has been retrieved
11     location_id = navigator.geolocation.watchPosition(
12         //success
13         function (current_location) { //if successful, generate and transmit data
14             GEO_LOCATION.location = current_location;
15             if (start_location_loaded == false) {
16                 generateData(GEO_LOCATION.location);
17                 start_location_loaded = true;
18             }
19         },
20         //fail
21         function (error_message) {
22             console.log(error_message); //if the request fails, log the error message to the console
23         },
24         // options
25     );
26     trackIntervalId = setInterval(function () {
27         generateData(GEO_LOCATION.location); //set the interval using the value of WINDOW
28     }, WINDOW);
29 }
```

The SCK environmental data is retrieved by sending requests to the Smart Citizen Kit server's API [41]. Data is retrieved as JSON which is parsed to suit the data structure used by the server models and saved to the main 'context.data' array of associative array elements.

```
1  //track.js, utility.js
2  var SCK_API_CODE = "084122509ae13c389bf752915861249cff652249";
3  ..
4  var DATA_SOURCE = 'http://api.smartcitizen.me/v0.0.1/' + SCK_API_CODE + '/lastpost.json';
5  ..
6  function getSCKData(location, routeId) {
7      var sample_data = {
8          no2: "",
9          co: "",
10         noise: "",
11         battery: "",
12         latitude: "",
13         longitude: "",
14         routeId: "",
15         timestamp: "",

```

```

16     light: "",
17     hum: "",
18     temp: ""
19   };
20   var json_obj = getSCKReading();
21   .. //populate the sample_data associative array and add to the main data structure
22   context_data.push(sample_data);
23   ..
24 }

```

The procedure is similar when retrieving the user and device information once the tracking process is stopped before transmitting the information to the server. It is stored in associative arrays and returned in order to be added to the final parameterized object 'dataReadings':

```

1 //utility.js
2 //retrieve user and device information once a user stops tracking in order to send the data to the server
3 function getUserData() {
4   var form_data = {
5     me: {
6       id: "",
7       username: "",
8       city: "",
9       country: "",
10      website: "",
11      email: "",
12      created: ""
13    }
14  };
15  ..
16  function getDeviceData() {
17    var form_data = {
18      devices: [
19        {
20          id: "",
21          title: "",
22          description: "",
23          location: "",
24          created: "",
25          kit_version: ""
26        }
27      ]
28    };

```

When the 'Stop Tracking' button is pressed, the application saves the route in the (local) storage space of the device and transmits it in case the 'context_data' array is not empty. A local saving operation is performed so that a personal history could be kept on each device having the application installed instead of introducing the need to provide authentication services and download all of this data from the server every time. The history page displays the route information stored on the device in the form of a list of route id-s and their number (the final version is shown in figure 5.3d). There is a button 'Clear Route Data From Device' which allows users to remove the local data saved by this application.

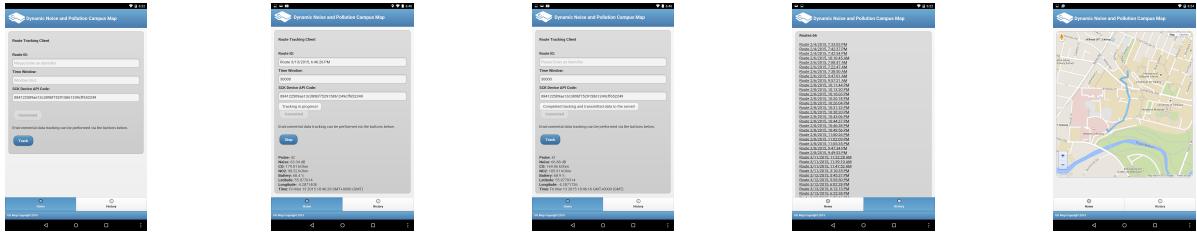
An associative array of aggregated data is created so that it can be parameterised by the Spring annotations service when it reaches the main server's end-point '<http://ugmap.me/addRoute>'.

```

1 //if there is data to be transmitted, send to server
2 if (context_data.length != 0) {
3   //create the parameterized dictionary object by stringifying the main arrays
4   var dataReadings = {context: JSON.stringify(context_data),
5     user: JSON.stringify(user),
6     device: JSON.stringify(device)};
7   console.log(dataReadings); //log the final object for debugging
8   //transmit the dataReadings object to the ugmap.me server
9   $.ajax({
10    type: 'POST',
11    url: "http://ugmap.me/addRoute",
12    data: dataReadings,
13    dataType: "json",
14    success: function (response) {
15      console.log(response);
16    }
17  });
18 }
19 //reset the key variables, ready for tracking again
20 route_id = null;
21 route_data = [];
22 context_data = [];

```

When a route from the list is clicked, the application showcases the route geo-location information plotted on a map on the route view page in order to allow users to inspect personal routes on their devices (as displayed in figure 5.3e).



(a) Final Android client main page (version 2.0) (b) Route tracking started - feedback user message; environmental data shown at the bottom (c) Route tracking completed, data transmission user feedback (d) History page, routes list (e) Single route page, displaying route trail

Figure 5.3: Android Client application final version 2.0

5.3.2 Mobile Web Application

The mobile web application has been built using Apache Maven[5] and uses the Jetty servlet container [30]. The DBMS of choice is Mongo DB[54] as it has proven to be a suitable option for the needs of this system (more information about this is available in section 7.1, chapter 7).

The JavaScript library of choice is jQuery[31] and the CSS framework that the mobile web application's front-end builds on top of is Twitter Bootstrap 3[43] (as mentioned in the chapters 34).

5.3.2.1 Back-end

The first step in the implementation of an MVC server is to design the model structure. After this process has been completed, the controllers can be designed in order to map the newly-created views to the system models. The key models in the architecture of this mobile web application are as follows:

- **DataReading** (primary) - this data model is used to store the environmental data; it is the basic data block in the implementation of the server. The main attributes are an identifying number - generated by Mongo DB's internal structures on insert, associated route and device identifiers, a timestamp, latitude/longitude to represent the geolocation and the environmental variables retrieved from the SCK sensor board accompanied by the current device battery level.
- **Route** (primary) - the route data model is used to serve as point of grouping data reading models into routes by associating them to different id-s. Each route also corresponds to a particular SCK device.
- **Device** (secondary) - this model stores general device meta data that retrieved from the Smart Citizen Kit API.
- **User** (secondary) - serves a purpose similar to the 'Device' model. The secondary data models are used on the server in order to show the source of environmental data that has been uploaded. In case the application is expanded to support user authentication (chapter 8, section 8.2), they can be used to store additional information attributes

These models are mapped to the relevant collections in the Mongo database using Spring MVC annotations:

```

1 @Document(collection = "DataReadings") //data readings collection
2 @Document(collection = "Routes") // collection for storing route documents
3 @Document(collection = "Devices") // devices collection
4 @Document(collection = "Users") //users collection

```

Repository classes in the form of interfaces extending 'MongoRepository<MODEL_TYPE, String>' are also used for object manipulation by means of key query methods implementation (as described in 7.1).

There is a single controller class to support the MVC paradigm - 'HomeController.java'. It contains the main REST API endpoint implementation. '/addRoute' accepts POST requests and attempts to retrieve

content parameters relevant to user and device meta-data as well as environmental information (data readings). With the help of Google's JSON Java library the parameterised objects are deserialized into the respective models and stored in the database using the 'MongoRepository' 'save()' method call on the autowired database collections in the class ('@Autowired' annotation). Accessor methods for these global repository variables have been implemented so that their instances can be accessed from the 'Benchmark.java' class. All of the operations are logged to the console.

```

1 @RequestMapping(value = "/addRoute", method = RequestMethod.POST)
2 public
3 @ResponseBody
4 String addRoute(@RequestParam("context") String json, @RequestParam("user") String userJSON, @RequestParam("device") String
5     deviceJSON) {
6     /* - parse the JSON strings into models using the Gson library
7        - save/update user and device data, create a new route and store data readings, logging the whole process
8     */
9     return "success";
10 }

```

The 'Home' view constructs a 'ModelMap' structure (internally implemented as a 'LinkedHashMap') of all data readings, routes, devices and users. This structure is returned to the main 'home.jsp' template for JavaScript parsing and processing:

```

1 @RequestMapping(method = RequestMethod.GET)
2 public String home(ModelMap model) {
3     List<DataReading> dataReadings = dataReadingRepository.findAll();
4     ..
5     model.addAttribute("dataReadingModels", dataReadings);
6     ..
7     LOGGER.info("Data Readings: " + dataReadings.size());
8     ..
9     return "home";
10 }

```

The spring MVC bean parameter configuration file ('mvc-dispatcher-servlet.xml') contains the system configuration information in order to enable server request handling. The data source bean component has been set in this file (as specified in Chapter 7, 7.1 section). The Spring context request dispatching process to the main controller as well as the resource, repository and view bean configuration is performed in this file:

```

1 <!-- mvc-dispatcher-servlet.xml -->
2 <context:component-scan base-package="com.springapp.mvc"/> <!-- base package specification -->
3 <context:annotation-config/>
4 <mongo:repositories base-package="com.springapp.mvc.repositories"/> <!-- wire the repository classes package -->
5 <mvc:resources mapping="/resources/**" location="/resources/">
6 <!-- enable annotation-based request handling -->
7 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" <!-- configure views directory structure -->
8     <property name="prefix" value="/WEB-INF/pages/">
9     <property name="suffix" value=".jsp"/>
10 </bean>
11 ..

```

5.3.2.2 Data Formatting

Apart from the noise variable, which is available directly in decibels from the sensor board, this is not the case when processing humidity, solar power, CO and NO2. The level values for these variables are represented by the actual sensor resistance levels in k (also known as RS). After contacting the co-founders of the product from FabLab, Barcelona, it became clear, that they have programmed the firmware in this way on purpose. They are currently conducting thorough testing to improve the way the base value (R0) for their sensors. 'R0' and 'RS' can be used to convert the raw resistance information to ppm (parts per million) using the following equation:

$$PPM_VALUE = RS/R0$$

The current base value of 'R0' has been set to 75 k (as recommended by FabLab's laboratory experts after contacting them to request more information about variable formatting). In order to facilitate the implementation of this data format conversion, an important decision needed to be made. One option was to modify the C++ source files of the SCK firmware and re-upload it to the sensor. This would ensure that the values are identical across all of the system components. However, this approach had one major drawback - it would introduce additional modifications to the core of the sensor board driving software and would make transitions to newer firmware versions unnecessarily inefficient. In order to avoid this potential drawback, another decision was taken. All of the conversion logic was transferred to the data parsing module responsible for loading database information from the database into RAM, ready for processing. This requires more processing in terms of the

number of executed operations compared to performing the conversion on the Android client prior to transmission to the server where the data is directly inserted in the database. However, it is much more flexible as it facilitates changes of the base 'R0' value. This means that once the constant is changed from 75 k to a different value, changes will immediately be reflected on the front end without the need to modify the contents of the database.

5.3.2.3 Logging

Logging has also been used in combination with the overridden 'toString()' method implementations of the main objects in order to provide prompt user feedback in the console while the server is running (figure 5.4a and 5.4b). This procedure facilitates the debugging process and is crucial for all server implementation stages. The Apache Log4J logging library for Java has been used to satisfy this requirement. 'toString()' methods have been optimized and rely on a standardized structure across all models:

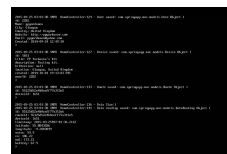
```

1 public String toString() {
2     StringBuilder string = new StringBuilder(); //use a string builder for improved efficiency
3     string.append(this.getClass().getName() + " Object {" + Constants.NEW_LINE); //get the class and object name
4     .. //add model attribute information
5     string.append("} " + Constants.NEW_LINE);
6     return string.toString(); //return the result
7 }

```



(a) Logging data set statistics (users, devices, routes and data readings) on page load



(b) Logging Android client transmitted data on REST endpoint request

Figure 5.4: Logging messages on the server console for system inspection.

Instead of logging to the console, this process can be performed in an external text file. This, however requires writing access to the target directory and log size allocation limit and is not as efficient as the console logging option which has been chosen to serve this system. The configuration can be seen in the following code listing:

```

1 # root logging configuration - log4j.properties
2 # logging works at 'INFO' level, highlighting the progress of the application at coarse-grained level to facilitate testing and
3 # evaluation
4 log4j.rootLogger=INFO, stdout, file # configure the root logger
5 # console output formatting section
6 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
7 log4j.appender.stdout.Target=System.out
8 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
9 log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

```

5.3.2.4 Implementation Logic

This section contains the description of the main server component's logic (the asynchronous JavaScript processing and how it ties to the relevant UI views). It includes thorough information about the algorithms used in the visualization implementation.

A bounded map surface was enforced in order to optimize working space. This was devised via using a 'google.maps.LatLngBounds()' object from API v3. Its constructor creates a rectangle from the points at its south-west and north-east corners. These corners are represented by 'google.maps.LatLng()' objects which have been declared in 'constants.js' under the 'utility' package as global variables. An event listener is attached to the main map object so that each time the center point changes the previous state is stored and if the new central point is not within the bounds of the imposed frame border, the map is rendered from the last 'stable' location using the 'panTo()' function.

```

1  var frameBorder = new google.maps.LatLngBounds(
2    new google.maps.LatLng(minLatBounds, minLonBounds),
3    new google.maps.LatLng(maxLatBounds, maxLonBounds)
4  ); //create a new bounds object to limit map surface panning
5  var lastCenter = map.getCenter(); //retrieve the current central point
6  google.maps.event.addListener(map, 'center_changed', function () {
7    if (frameBorder.contains(map.getCenter())) { //once the center changes, check if within the allowed bounds, if so, update the
      current central point
8      lastCenter = map.getCenter();
9      return;
10   }
11   map.panTo(lastCenter); //otherwise pan to the last 'stable' location
12 });

```

In order to implement the style switching and map controls configuration functionality, 'StyledMapType' objects were created. All of them are stored in the 'map_styles.js' script in an array data structure for efficient referencing. Those objects have a composite array structure - they comprise of 'stylers' arrays of associative array elements and 'featureType' attributes. The 'stylers' array contains formatting options of type 'MapTypeStyler' which can be applied to map features. Attributes such as 'hue', 'lightness', 'saturation', 'gamma', 'weight' etc. can be applied in this array. For instance, the 'color' value can be specified using an RGB hex string or modified by combining the HSL attributes. The 'featureType' and 'elementType' attributes are defined before 'stylers' and used to specify the components for which the style array will be used. The associative array objects can be aggregated to build more complex style arrays that are later integrated into 'StyledMapType' objects and can be applied to any 'google.maps.Map' object by using the 'maptypes.set('style', sMap)' and 'setMapTypeId('style')' functions. All of these objects follow the same structure:

```

1  //utility/constants.js
2  ..
3  var centerLat = 55.872912;
4  var centerLon = -4.289657;
5  var center = new google.maps.LatLng(centerLat, centerLon);
6  //map/map.js
7  ..
8  var styleArray = [
9    {
10     featureType: "road",
11     elementType: "geometry",
12     stylers: [ .. ]
13   }, .. ]; //style properties including color codes are placed in this section
14  var sMap = new google.maps.StyledMapType(styleArray, {name: "Styled Map"}); //define a new 'StyledMapType'
15  ..
16  var myOptions = { //add comments explaining map style here
17    zoom: 16, // set the default zoom level
18    minZoom: 15, // specify the minimum and maximum allowed zoom levels
19    maxZoom: 18,
20    center: center, //set the map center point
21    mapTypeId: google.maps.MapTypeId.ROADMAP, //configure the initial MapTypeId
22    disableDefaultUI: false, // configure the UI controls
23    mapTypeControl: false, // disable map type control
24    panControl: true, //enable the pan control user option
25    panControlOptions: {
26      position: google.maps.ControlPosition.TOP_LEFT //locate the pan buttons at the top left map surface area
27    },
28    zoomControl: true, // enable zoom control
29    zoomControlOptions: {
30      style: google.maps.ZoomControlStyle.LARGE, // large buttons, optimized for smaller screens
31      position: google.maps.ControlPosition.TOP_LEFT // locate top left below the panControlOptions object
32    },
33    scaleControl: true, // enable the scale control to let users inspect grid map
34    streetViewControl: true // enable street view control
35  }; //define the map options
36  var map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions); //create the main map object using an options
37  // associative array
38  map.maptypes.set('mapStyle', sMap); //add the new map style to the 'mapTypes'
39  map.setMapTypeId('mapStyle'); // use the mutator function to change the map style

```

Once the map object has been initialized, the map is dynamically populated with the entities retrieved from the main view (the 'home()' method in 'HomeController.java', 'controllers' package) using Google Maps API v3. The file 'parse.jsp' is used as a parser to transfer the entities to associated array data structures using JavaScript. At this time variable formatting occurs using the conversion equation described in 'Data Formatting', 5.3.2.2. In the 'markers_routes' script, map 'Markers' are generated for each individual data reading element, while routes are represented by 'Polyline' objects:

```

1  //markers_routes.js
2  function populateMap() {
3    for (var i = 0; i < routes.length; i++) {
4      .. // process data readings and render routes, generating 'InfoWindow()' objects showing the metadata for each
5    }
6  }
7  ..
8  function addPopUp(marker, content, trigger) { .. }
9  function generateMarker(dataReading, visible, map) { .. }
10 function generateRoute(newRoute, noiseAVG, coAVG, no2AVG, distance, duration, score, id) { .. }
11 function identifyValueRange() { .. }
12 function progressEvaluate(value, min, max) { .. }
13 function generatePopUpContent(noise, co, no2, battery, typeData, routeDistance, routeDuration, score, id, dataCount) { .. }

```

The signatures of the key functions that are called in the dynamic map population process (within the main loop in 'populateMap()') have been listed in the code snippet above. 'addPopUp()' is used to bind 'Polyline' and 'Marker' objects to their 'InfoWindow()' 'open' event on tap/click (object references are passed as input parameters). The 'generateMarker()' and 'generateRoute()' functions are used in order to create the markers and routes and call the rest of the functions. 'identifyValueRange()' is key as it is used to determine the variable ranges (the minimum and maximum values are used to derive the range) and facilitate the scaling that is available in the pop-ups associated with the map objects. In order to facilitate the filtering operations associated with object timestamps, 'progressEvaluate()' relies on the information retrieved by 'identifyValueRange()' in order to generate HTML5[28] progress bar elements and dynamically insert them into the pop-ups as seen below. 'generatePopUpContent()' is the general function for populating 'InfoWindow' objects bound to data readings and routes. The number of input parameters is necessary to facilitate the needs of multiple calling sources and improves the overall component reusability levels. The 'typeData' input parameter is crucial for this function as it determines the type of content that is going to be generated when it is called: 'null' is used to generate the meta-data contents for a data reading, '>0' is processed as a route and any negative integers including zero are an indication that the data for a map matrix cell has been passed in. The values of parameters 'noise', 'co' and 'no2' are used in the three cases, 'dataCount' is used for the routes and grid map rendering, while 'battery' can be inspected only for individual data readings. 'id', 'routeDistance' and 'routeDuration' are used only for routes. The 'score' parameter represents the overall pollution index - a resulting score from an evaluation algorithm that is used by the route generation engine in the route recommendation process.

The mobile application can also be used from a desktop machine. In order to enable the user to manipulate the map controls, such as the pan buttons, zoom, scale and street view control, enabling or disabling shortcut options controls have also been included to the user interface - keyboard shortcuts (enabling the use of the arrow keys for map panning and the "+" and "-" keys or mouse scroll wheel for zooming in and out accordingly). This functionality is implemented by calling 'toggleMapControls()' every time the appropriate view element is activated by a tap/click. Depending on the checkbox selection, the appropriate elements are rendered. In the 'Styles' dropdown menu, there is a collective checkbox for disabling/ enabling all UI controls. All of the options have been enabled by default. Below is a code snippet for the pan control element from the view page 'styles.jsp':

```

1 //styles.jsp
2 <label>
3   <input type="checkbox" name="styleOpts" value="panControl" checked> Pan Control <!-- pan control element with the 'panControl' value,
4     part of the 'MapOptions' object settings for map configuration, API v3 -->
5 </label>
6 ..

```

Once the state of the input elements changes, an event that renders the new map canvas state is triggered resulting in the appropriate controls being enabled or disabled:

```

1 //user_interface.js
2 function toggleMapControls() {
3   $("input[name='styleOpts']").each(function () {
4     var value = (this.checked ? true : false); // convert the 1/0 structure to standard boolean true/false
5     var type = $(this).attr("value");
6     map.set(type, value); // enable or disable the option based on the state of the elements in the view
7   });
8 } ..

```

Modes

The 'Modes' architecture follows some important principles. An accordion drop-down menu has been used to represent the mode options. Modes represent dynamically generated JavaScript (API v3) map overlays and they can be shown or hidden via a checkbox at the top left corner of each configuration options group. These options are disabled unless the main 'Show' checkbox is not selected. This ensures that the back-end processing engine is not going to apply more changes than necessary when generating the final overlay as only instructions associated with the visible mode components are executed. Specific input elements have been used to facilitate the process of rendering custom visualizations depending on the selected type:

```

1 //user_interface.js
2 function retrieveModes() {
3     // determine the modes that will be visualized
4     return selectedModes;
5 }
6 function renderMap(modes) { .. } //render the mode visualization depending on the result from 'retrieveModes()'

```

Data readings can be shown/ hidden and the meta-data 'on-click' (tap) event triggering the appearance of the meta-data pop-up can be enabled/ disabled. This is achieved by using the 'set()' function applied to each marker in the global markers data set 'POINT_DATA'. When setting the new value of the 'visible' marker attribute, the state of the 'Disable Information' checkbox available on the front-end UI is used. When processing the main markers data structure, the total count of data readings is retrieved and displayed in the 'Data Points' tab for statistical purposes.

The configuration settings for routes are similar as the 2 checkboxes serve the same purpose as in the previous described mode. The total route number is also available. The 'Polyline' objects' thickness (in pixels) as well as the transparency (in percentage) can be controlled via using the input fields. This is implemented via using the 'strokeWeight' and 'strokeOpacity' attributes of the 'Polyline' object accordingly:

```

1 // controls.js
2 ..
3 var thickness = parseInt($("#thicknessRoutes").val()); // retrieve the values from the input fields
4 var opacity = parseInt($("#opacityRoutes").val()) / 100;
5 ROUTE_DATA.forEach(function (entry) { //for each route in the main data structure
6     if (thickness !== NaN && thickness > 0) { // validate the thickness input
7         entry["route"].set("strokeWeight", thickness); //update the attribute
8     }
9     if (opacity !== NaN && (opacity > 0 && opacity <= 1)) { // validate opacity stroke, must be between 0 and 1.0 after division
10        if a number ('NaN' indicated not a number)
11        entry["route"].set("strokeOpacity", opacity); //update the attribute
12    }
13    });

```

The implementation of the 'Heat Map' visualization is based on color-coding the data reading points using a radial gradient that is green at its periphery and turns into red in the core (using a green-yellow-red HSL color-scheme). Map areas where orange/red is the dominant color are most accurate in terms of aggregated average data values. The heat map is initialized using the function 'generateHeatMap()' in the 'heat_map.js' script under the 'map' directory. It relies on a simplified data structure in the form of a plain array which stores 'LatLng' objects to represent the geolocation (latitude/ longitude) of each marker. The values of the main variables (noise and air pollutants - CO, NO2) are not required for this visualization. The array structure is parsed into an 'MVCArray' that can be processed by the 'visualization' map object using the 'HeatmapLayer()' function with an associative array of configuration options. Based on data point clustering, this function generates the final visualization as a color-coded overlay atop the map surface when it needs to be rendered. On page load, the data is only pre-processed and stored in the global 'HEAT_MAP' object to improve performance when displayed using the 'setMap()' function with the main 'Map' object as an input parameter:

```

1 //heat_map.js
2 function generateHeatMap() {
3     var points = new google.maps.MVCArray(locationARR); // create an MVCArray using the plain locationARR as input
4     HEAT_MAP = new google.maps.visualization.HeatmapLayer({
5         data: points, //use 'points' as a data source
6         radius: 50, //set the default radius size
7         map: null // do not render on the map
8     }); // pre-generate the heat map visualization and store in HEAT_MAP
9     //HEAT_MAP.setMap(map); when this instruction is executed at a later stage during UI interactions, the heat map will be rendered/
10    become visible

```

The grid map visualization structure is the most important mode as it is the primary component in the mobile web application and the base data source when producing route recommendations. The 'Show/Hide' checkbox has been implemented in this tab as well (when unchecked settings cannot be altered). The grid data structure is stored in the global 'GRID' object. It relies on the 'google.maps.geometry' library (libraries[52] are included in the main page source code as a 'googleapis.com' [26] URL parameter: '&libraries=geometry,visualization') and more specifically, the 'spherical' object and the 'computeOffset' function. The only constraints imposed in this process are the grid surface size and the cell width/ height (in meters). The grid surface size is determined by a north-west and a south-east point (variables 'northWestStart' to represent the first point; 'minLatBounds' and 'maxLonBounds' used for the bounding box to frame the map working space represent the limiting south-east point). The map grows from west to east being generated

on a row-by-row basis until a violation of the checks for geolocation containment within the frame occurs. The default matrix cell size is 50 by 50 meters and this can be controlled using the 'Tile Size' input field on the UI.

```

1 //grid_map.js
2 function generateGrid(tileSize) {
3   //set east and south variables as 50 m offsets from northWestStart
4   for (var heightTiles = 0; heightTiles < heightTilesN; heightTiles++) {
5     //in each outer loop iteration, move the points to the south by 'tileSizeMeters'*'heightTiles' where 'heightTiles' is the number
6     //of rows that have already been rendered; execute while within latitude bounds
7     newEast = google.maps.geometry.spherical.computeOffset(east, heightTiles * tileSizeMeters, southAngleDegrees);
8     newSouth = google.maps.geometry.spherical.computeOffset(south, heightTiles * tileSizeMeters, southAngleDegrees);
9
10    for (var widthTiles = 0; widthTiles < widthTilesN; widthTiles++) {
11      var tile = new google.maps.Rectangle();
12      //set the default tile options
13      var tileDATA = {tile: tile, noiseAVG: {sum: 0}, coAVG: {sum: 0}, no2AVG: {sum: 0}, count: 0}; //initialize the tile as a
14      //composite object
15      GRID.push(tileDATA); //add the tile data to the grid object
16      //in each inner loop iteration, move the points to the east by the tile size in meters; execute while in bounds
17      newEast = google.maps.geometry.spherical.computeOffset(newEast, tileSizeMeters, eastAngleDegrees);
18      var newSouth = google.maps.geometry.spherical.computeOffset(newSouth, tileSizeMeters, eastAngleDegrees);
19    }
20  }
21 }

```

Each row is constructed, starting from the location furthest to the west, generating the first cell, computing 50 m. offset to the east from both previous north-west and south-east points ('newEast' and 'newSouth'). In other words, the algorithm is sliding the points while within the allowed bounds. It generates a composite object 'tileDATA' for each cell in order to store the color-coding, the number of data points corresponding to the cell surface and the total sum (used for finding the average values when plotting the data). Pre-computation of these values is not executed to allow for immediate reflection of changes when the data-set changes. 'InfoWindow' object storing the meta-data for each cell are also generated in this function. When applying the settings, the map is aggregated and the appropriate coloring is rendered for each cell in the function 'toggleGrid()' if the 'Show/Hide' checkbox is selected. Checks parse the selected variable from the radio button group as well as the outline and fill transparency (outline display is optional) values from the input fields and validate them. If a custom tile size has been specified, the matrix is rendered and aggregated again using the functions 'generateGrid(tileSize)' and 'updateGridAggregation()'. The grid map exploration progress is also evaluated in this function based on the populated cells and the total number of cells in the matrix using the function 'mapExplorationProgress()' to determine the status in percentage. There are two types of scaling. Relative scaling uses the following value range: min = (minimum dataset value), max = (maximum dataset value). Absolute scaling uses a range determined by the norms the government imposes and hence the grid map has green hue coloring in every cell. The raw values are then scaled based on the range used and depending on the gradient choice (a RGB or HSL model) the color-codings range from red-green and red-yellow-green accordingly. The second option is better when performing a comparative analysis of the cells.

The functions, facilitating the color-coding feature using different gradients, have been implemented in the 'utility.js' class. 'convertToRGB()' utilizes the 'rgb()' (red-green-blue scheme) formatting to generate a color-coding, where the red-green transition is facilitated by scaling the values for red and green in percentages. The value for blue is not used here. The process is identical when generating an hsl (hue-saturation-lightness) color-coding. Saturation and lightness keep their constant values, while the input parameter 'n' is scaled between 0 and 120 degrees to achieve the red-yellow-green transition.

```

1 //utility.js
2 function convertToRGB(n) {
3   var B = 0; //blue coding is a constant; neutral effect to the final color
4   var R = Math.floor((255 * n) / 100); // scale the values for red and green
5   var G = Math.floor((255 * (100 - n)) / 100);
6   var RGB = "rgb(" + R + ", " + G + ", " + B + ")"; // encode final result and return
7   return RGB;
8 }
9 function convertToHSL(n) {
10  var H = (1 - n / 100) * 120; //scale only color hue between 0 and 120 degrees
11  var HSL = "hsl(" + H + ", 100%, 50%)"; // encode the final color, using constant values for saturation and lightness, then return
12  return HSL;
13 }

```

The indexed grid structure facilitates the observation of a single variable on the map surface at any given time. When using the 'Relative' scaling option, it helps to identify the areas where the variable values are lowest and highest accordingly. However, in order to make it possible to compare multiple variables, the 'Point Visualization' mode has been implemented. It plots the datapoints for each variable type as a circle overlay with a different opacity and radius size. This data is represented by using the 'google.maps.Circle' object and

different color-codings, chosen by the user and is stored in the 'POINT_VISUALIZATION' global array. The circle 'fillColor' attribute is loaded depending on the choice of the input elements with names 'noiseColor', 'coColor' and 'no2Color' on the 'modes.jsp' page. The radius size is determined by the radius radio button group in a similar fashion with the aid of the 'rangePercentage()' function that has been used for the variable scaling in the server system:

```

1 //constants.js
2 function rangePercentage(value, min, max) {
3   return (value - min) / (max - min) * 100; //scale any value to percentage based on a range it falls within (determined by the 'min'
4     and 'max' input parameters )
5 }
6 //point_vis.js
7 function generatePointVis(dataReading, visible, map, num) {
8   var noiseRadius = rangePercentage(dataReading.noise, minNoise, maxNoise) / 10;
9   ..
10  var pollutionOptions = {
11    .. //configure the options, including initial radius size, transparency and fill color
12  };
13  var noiseCircle = new google.maps.Circle(pollutionOptions);
14  ..
15  var element = {noiseCircle: noiseCircle, coCircle: coCircle, no2Circle: no2Circle, noise: noiseRadius, co: coRadius, no2: no2Radius};
16  POINT_VISUALIZATION.push(element); // add the composite object to the global data structure

```

Transparency is set via using the 'fillOpacity' attribute which is divided by 100 to retrieve a value between 0 and 1 as this is the documented operational range. Depending on whether the 'Show/Hide' option has been selected, the 'visible' attribute's value is changed. When the page is loaded, the 'generatePointVis()' function is called in order to perform the data pre-processing. Once this configuration is loaded, the 'togglePointVis()' function is called to visualize the points and it changes the radius size and color fillings accordingly.

Filters The implementation of the filter functionality relies on range jQuery sliders. The front-end elements are dynamically modified depending on the UI interactions. The code snippet below shows the key logic when the map is rendered. It is important to note that filters apply to the data readings, routes (applied to the overall average variable values), the heat map data points, as well as the 'Point Visualization' color-coded plot.

```

1 //constants_global.js
2 var baseStep = 0.001;
3 //map.js
4 ..
5 $("#noise").slider({
6   orientation: "horizontal", //specify orientation
7   range: true, //enable range
8   min: Math.floor(minNoise), // identify specific value range
9   max: Math.ceil(maxNoise),
10  values: [ Math.floor(minNoise), Math.floor(maxNoise) ], // set the initial slider handle positions
11  step: baseStep, //define the slider step, a constant used for each slider
12  slide: function (event, ui) {
13    .. //change the values on slider handles while sliding
14  },
15  stop: function (event, ui) {
16    .. //retrieve the values from slider handles once movement stops
17    renderData(); //render the data on the map
18  }
19 });
20 minRangeNoise = minNoise; // set the default minimum value
21 maxRangeNoise = maxNoise; // set the default maximum value
22 ..
23 function renderData() { //based on the filtering criteria, iterate over the main marker storage object 'POINT_DATA' and route object
24   'ROUTE_DATA', then visualize only the data points that fall within the range
25   locationARR = []; //reset the object used for heatMap generation in order to reflect changes for this visualization as well
26   ..

```

The sliders for CO, NO2 and temporal filtering have been implemented identically. However, when using timestamps as a filtering criterion, the step is '1' based on the number of days in the range which is displayed directly above the slider in the format shown in the following example: 'Time (days between 14-January-2015 and 13-March-2015)'. The size of the resulting data set is displayed for each filter in the format: '(557 data readings)'.

In order to implement the route generation mode, the Google Directions API[23] as well as the indexed grid structure have been used. There is a form-style structure on the user interface. This structure houses the travel mode selector in the form of a drop-down menu and the journey planner (containing the origin and destination points as well as the route recommendation configuration criteria). The starting point automatically evaluates as a 'LatLng()' object representing the geolocation of the building on campus that is closest to the current user's location. This is done by implementing the base operation of the 'nearest neighbour' algorithm which has a stable linear complexity within the input size in its current implementation:

```

1 //locations.js
2 function findClosestCampusLocation(source) { .. }

```



```

3 //markers_routes.js
4 function retrieveDistance(loc1, loc2) { //retrieve the distance between two LatLng() objects in meters
5     return parseInt((google.maps.geometry.spherical.computeDistanceBetween(loc1, loc2).toFixed(2)));
6 }

```

The function 'findClosestCampusLocation(source)' iterates over all of the 'places' in the database and finds the shortest distance using the linear search algorithm. As seen in the implementation of 'retrieveDistance(loc1, loc2)' the API function 'google.maps.geometry.spherical.computeDistanceBetween()' has been used to compute the distances between each pair of 'LatLng()' objects. The haversine formula was implemented originally in a separate function in order to achieve this, but then it was replaced by the much more compact function call as shown in the code snippet above.

Once the selections have been made, when the 'Apply' button is tapped/ clicked, the application retrieves the data input and calls the 'generateUserRoutes()' function (if all the necessary parameters have been supplied) which generates a request to Google Directions API:

```

1 //route_generation
2 function generateUserRoutes() {
3     mode = $("#mode").val(); //the selected travel mode is retrieved from the front-end element
4     var request = {
5         origin: starting_point, // journey origin location, closest building by default
6         destination: destination_point, // destination point
7         provideRouteAlternatives: true, // retrieve multiple routes
8         travelMode: google.maps.TravelMode[mode], // set the travel mode
9         unitSystem: google.maps.UnitSystem.METRIC // display distance using m and km
10    }; ..

```

The sorting criteria selected by the user determine how the results will be classified depending on their 'overall pollution index'. The general equation uses the weighted sum of random variables classification approach for decision making. It assigns different weights to the variables. The weight depending on how relevant they are to this application's needs: W(duration) = 0.01, W(distance) = 0.01 (least significant), W(noise) = 0.20, W(CO) = 0.20, W(NO2) = 0.58 (this air pollutant is most toxic and hence has the highest coefficient, influencing the score the most). The dangerous levels as for noise, CO, NO2 and the maximum expected route distance and duration on campus have been used to calculate the maximum index score:

```

1 //constants_global.js
2 var absoluteMinNoise = 50; // in dB, decibels
3 var absoluteMaxNoise = 130;
4 var absoluteMinCO = 0; //in ppm, parts per million
5 var absoluteMaxCO = 30; //30-40 dangerous
6 var absoluteMinNO2 = 0; //in ppm, parts per million
7 var absoluteMaxNO2 = 150;
8 ..
9 function calculateMaximumOverallPollutionIndex() {
10    maximumOverallPollutionIndex = absoluteMaxNoise * noiseMultiplier + absoluteMaxCO * coMultiplier +
11    absoluteMaxNO2 * no2Multiplier + absoluteMaxRouteDistance * routeDistanceMultiplier + absoluteMaxRouteDuration *
12    routeDurationMultiplier;
13 }
14 //route_generation.js
15 var score = noiseAVG * noiseMultiplier + coAVG * coMultiplier + no2AVG * no2Multiplier + routeDistance * routeDistanceMultiplier +
16    routeDuration * routeDurationMultiplier;

```

$$E(X) = \sum_{i=1}^n w_i x_i$$

(a) Expected value of a weighted sum of random variables (search criteria)

Figure 5.5: Route recommendation based on the 'overall pollution index'

As displayed in the code snippet above, the final score for each retrieved route is calculated by using the weighted sum of the independent variables. The expected value of the weighted sum of the random variables, specified as search criteria by the user (distance, duration, avg. CO, NO2, noise) can be expressed with the general equation shown in figure 5.5. The score is scaled via using the 'rangePercentage()' function where the maximum value is retrieved by 'calculateMaximumOverallPollutionIndex()'. The current value is calculated in a similar fashion as the dot product of the variable current values vector and the variable multipliers vector:

OverallPollutionIndex = currentNoise * noiseMultiplier + currentCO * coMultiplier + currentNO2 * no2Multiplier + currentRouteDistance * routeDistanceMultiplier + currentRouteDuration * routeDurationMultiplier

The maximum possible value as well as the current overall pollution value is calculated based on the variable selections in the 'Route Statistics' list on the 'Routes' page (as shown in figure 5.13a). Variables which are not selected are assigned '0' multipliers and therefore do not influence the scaling and final results in any way. Based on the resulting percentage value, the 'identifyBestRoute()' function sorts the array of routes in ascending order and color codes them using 4 main hues: green, yellow, orange and red. If the resulting recommendations are more than 4, every consecutive one is color-coded in red:

```

1  scores = scores.sort(compareIntegers);
2  for (var i = 0; i < scores.length; i++) {
3    if (i < colors.length) {
4      scoresColors[scores[i]] = colors[i];
5    }
6    else {
7      scoresColors[scores[i]] = colors[colors.length - 1]; //always red
8    }
9  }

```

The 'Locations' view is implemented via using the HTML5 Geolocation API[21] as well as the Google 'visualization'[52] library in order to render images for each building on campus. Building data is kept in a .json file - 'locations.json' under the resource directory for maintenance purposes. a request is used to retrieve the data, sort it and dynamically populate the view. The snippet below shows the storage structure of the file:

```

1  [{"label": "Adam Smith Building", "loc": [55.873727, -4.289915]}, ..] //store the building name and geolocation

```

After processing the JSON contents in the file, the Google Street View Image API[53] is used in order to generate 200 by 120 px images of each building:

```

1  \!tset{language=JavaScript}
2  //locations.js
3  $.getJSON("/resources/locations/locations.json", function(json) { .. } //retrieve data request

```

The API call uses the object latitude and longitude to determine the location. The 'heading' parameter indicates the compass heading of the camera (hence, some buildings might appear to the side of images). The possible values here range from 0 to 360 degrees depending on orientation. 'fov', which is the field of view represents the camera zoom level and ranges from 0 to 120. The maximum value has been used to render images from a maximum distance, without any zoom, that is, in order to ensure optimum results. 'Pitch' is responsible for the vertical orientation of the camera, ranging between -90 and 90 degrees according to the official documentation. A slightly upwards-looking angle of 10 degrees has been chosen after some experimenting with the resulting images:

```

1  ..<img class='streetViewImage' src='https://maps.googleapis.com/maps/api/streetview?size=200x120&location=" + entry.loc[0] + "," +
   entry.loc[1] + "&heading=34&pitch=10&fov=120'> ..

```

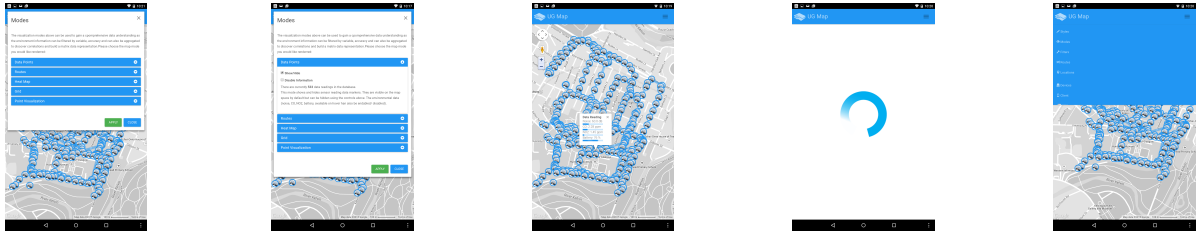
This view can also be used to stop and start tracking the user's location in order to conserve battery power and improve application performance. The user is tracked by default in order to enable the application to display a message when the destination point has been reached:

```

1  //map.js
2  userWatch = navigator.geolocation.watchPosition(function (position) { //track the device's geolocation
3    renderMarker(map, currentUserLocation, position.coords.latitude, position.coords.longitude);
4  });
5  //navigator.js
6  function renderMarker( map, marker, latitude, longitude ) {
7  ..
8    if (currentlyTrackingDestination) { //if the tracking flag is active, hence GPS is enabled
9      if (retrievedDistance(source, destination) < 50) { //retrieve the distance to the destination and if the device is within 50m of
10         the destination, proceed
11         currentlyTrackingDestination = false; //turn off the flag
12         alert("You have reached your destination!"); //alert the user
13         navigator.geolocation.clearWatch(userWatch); //stop the tracking process
14         console.log("Stopped watching for location changes."); //log the changes to the console
15       }
16     }

```

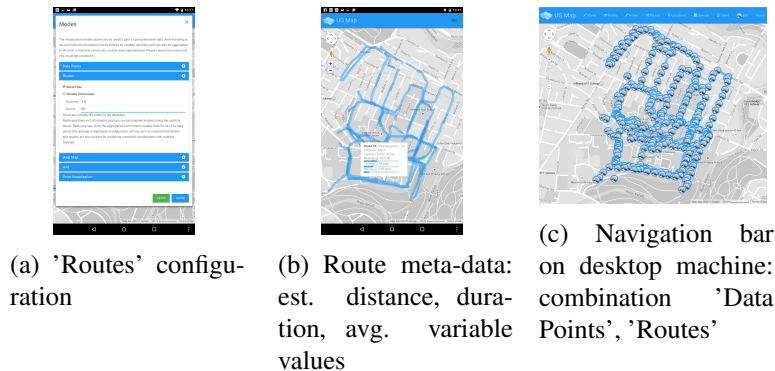
The structure of the main views of the server application can be split into two main categories - user-interactive and statistical. They have been described in the next section ('Front End', 5.3.2.5).



(a) Accordion menu: used to compact the style of the user interface
 (b) Data points configuration
 (c) Data points display on the map with pop-ups enabled: showing meta-data
 (d) Main page loading screen to improve user experience
 (e) Server mobile navigation bar optimized for smaller screens (Google Nexus 7 device)

Figure 5.7: Server 'Modes', 'Data Points'

route. The following formula is used: $final_variable_value = variable_levels_sum / route_datapoints_number$. The meta-data for each route also includes an estimated distance (in meters) as well as journey duration (assuming walking speed). Two configuration options that facilitate combining the route poly-lines with other visualization modes are the thickness and transparency field (figure 5.8b). The total number of routes in the database is also displayed.



(a) 'Routes' configuration
 (b) Route meta-data: est. distance, duration, avg. variable values
 (c) Navigation bar on desktop machine: combination 'Data Points', 'Routes'

Figure 5.8: Server 'Modes', 'Routes'

As the values of the noise and air pollutant variables are constantly changing, assessing the reliability of the data set is a complex task. Depending on the time of day when the data was collected, there can be significant differences. For instance - noise values will expectedly be low during late hours compared to daytime data collection. A similar tendency can be identified when analysing air pollution levels (CO and NO2 levels normally peak during day-time around lunch when the motor vehicle traffic is high). In order to tackle this, the 'Heat Map' visualization has been devised. It uses the 'HSL' color scheme - with colors ranging from red (high) to yellow to green (low) depending on the saturation of data points in a particular area. Red areas on the campus map indicate that there is a higher concentration of data readings in those locations. This dataset property can be used to infer that the map is more accurate at the highlighted (red hue) locations as more frequent sensor probing has been carried out there (as shown in figure 5.9b). The functionality for adjusting the radius scaling of each data point has been implemented. This can be used to emphasize the more accurate areas better depending on the scale and zoom level chosen by the user. Similar to other modes, the transparency of this visualization can be controlled (in percentage) so that it can be combined with other elements - such as data reading markers in order to inspect specific variable levels in dB and ppm (figure 5.9c). The data point radius scale can also be specified (figure 5.9a).

The most important mode is the 'Grid' visualization. It makes use of the matrix JavaScript[29] data structure comprising of indexed cells in order to generate a color-coded grid surface as an overlay atop the Glasgow University campus map. There are multiple configuration options in order to allow users to produce highly-

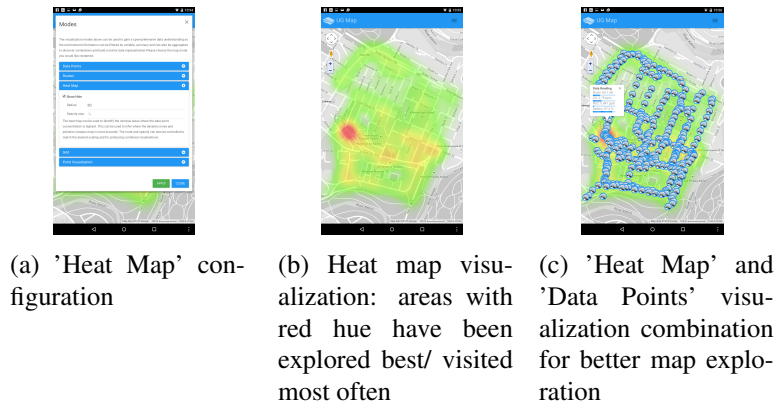


Figure 5.9: Server, 'Modes', 'Heat Map'

customized visualizations (settings shown in figure 5.10a). Unlike the first two modes, where environmental information is available for all of the variables, in this case variables are isolated so that more dataset granularity can be achieved (noise: figure 5.10b, CO: figure 5.10c, NO₂: figure 5.10d). Depending on the variable choice in the 'Variables' radio button group, the engine renders different color mappings based on the selected type.

Each matrix cell has an 'infoWindow' object association that is triggered on a tile tap/ click event. The contents of the pop-up include the index of the grid cell, the number of data readings in the full dataset that are located within the bounds of that cell's surface as well as the average environmental variable values.

Additionally, a grid outline in the form of a black border can be rendered so that areas on the map that have not been explored yet can be located more efficiently (using the 'Outline' checkbox under the 'Grid' sub-section) as shown in figure 5.10e.

Other options are the gradient choice - using HSL (red-yellow-green, figure 5.10c) it is easier to identify the middle value range while RGB (which uses red-green, figure 5.10d) allows for more comprehensive comparison between any pair of cells. Data scaling can also be controlled - relative data scaling allows users to identify the most polluted areas in the campus as the range of the data is always constrained to the maximum value in the current data set (figure 5.10h). Using absolute scaling, however, allows users to see the information in relation to the norms imposed by the government and uses the following data constraints in terms of maximum values (calculated in ppm /parts per million/ and dB correspondingly): 30-40 ppm is the danger threshold for CO, while for NO₂ this is 1.5-1.6 ppm (as it is most toxic). In terms of noise, the pain threshold is around 130-150 dB and this is used for its scaling. In order to improve the accuracy of the grid, the matrix cell size can be controlled. The minimum size is 20 m, while the default one is 50 m (meaning that each tile has a width = height = 50 m.). The opacity of the cells' filling and outline can be controlled as well to improve the output when generating combined visualizations. Map exploration progress is presented as a percentage in the form of a progress bar: $totalExploration = exploredCellNumber / totalCellNumber$. The exploration level will change depending on the cell size the map is rendered with.

The grid map and marker visualization allow only a single variable to be rendered on the map surface at any given time. In order to be able to inspect all of the variables simultaneously, the point visualization mode has been designed. It allows users to plot the three main variables on the map, choosing different color coding and scaling the data point radius (using multipliers as follows: x1, x2, x4, x10) for each one. This makes highest/lowest values very easy to inspect and enables users to be able to conclude whether there are correlations in the locations where maximum/minimum values for noise, CO and NO₂ are recorded (examples are shown in figure 5.11). This mode can be combined with the grid visualization in order to aid the data analysis as displayed in figure 5.11d.

The next view of the mobile web application is the data filtering engine. This functionality facilitates dataset filtering based on the specification of variable range limits. There are range sliders (double-sided sliders) for noise (figure 5.12a), CO and NO₂ as well as time (based on filtering by days, shown in figure 5.12b). Using these

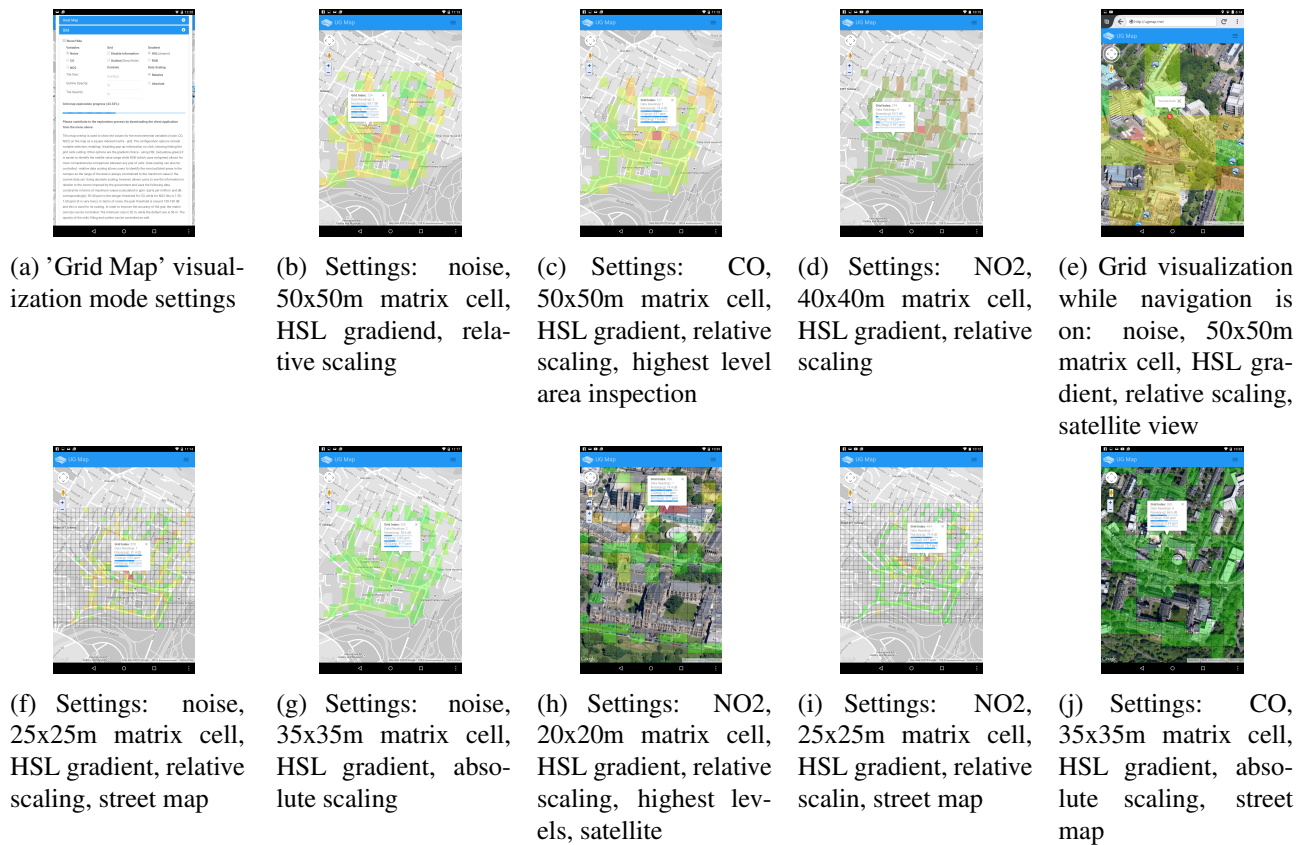


Figure 5.10: Grid map visualizations

sliders the user can combine the range constraints and inspect the resulting map overlay changes immediately. In order to streamline the filtering process further, scales showcasing the normal and dangerous levels for the environmental variables have been included in each element of the accordion menu. In the process of variable slider interaction, the number of data points in the resulting data set is immediately rendered at the top right area above the relevant jQuery slider to improve the system feedback. The available slider range is determined based on the minimum and maximum values of each variable (which have been used in other visualizations too). When parsing the data, datapoint values are compared directly, while for routes the average levels for each variable are used. The 'Data Points' and 'Grid Map' modes can be combined (figure 5.12c) to achieve a more comprehensive visualization.

The route recommendation UI component is implemented in 'routes.jsp'. The main interaction elements are a travel mode selector and a journey planner (figure 5.13a). The available options for travel mode are - walking (default, no vehicle), bicycling, driving (using a personal motor vehicle), transit (using any public transport, such as the subway, a public bus, etc.). Using the 'Journey Planner', the starting point and the destination can be specified. The implementation automatically populates the 'Starting Point' field with the lat/lon data for the database building which is currently closest to the device's location. The destination can be specified by typing the building name manually (the search engine on the web application is based around campus buildings as the main location entities) while being assisted by the autocomplete functionality implementation. The more efficient option is to tap/ click on any building on the map surface, automatically populating the destination field via using jQuery[31] event probing that retrieves the 'closest neighbour' (nearest campus building to the place where a tap occurs). Based on the criteria selections (duration, distance, noise, CO and NO2) routes are scored and sorted. The overall pollution index can be inspected to compare the color-coded routes (figure 5.13b and 5.13c).

The user can constantly keep track of their current location (figure 5.14c) so that they know how far from their destination they are. As described in the 'Evaluation' chapter (7), most users who participated in the evaluation

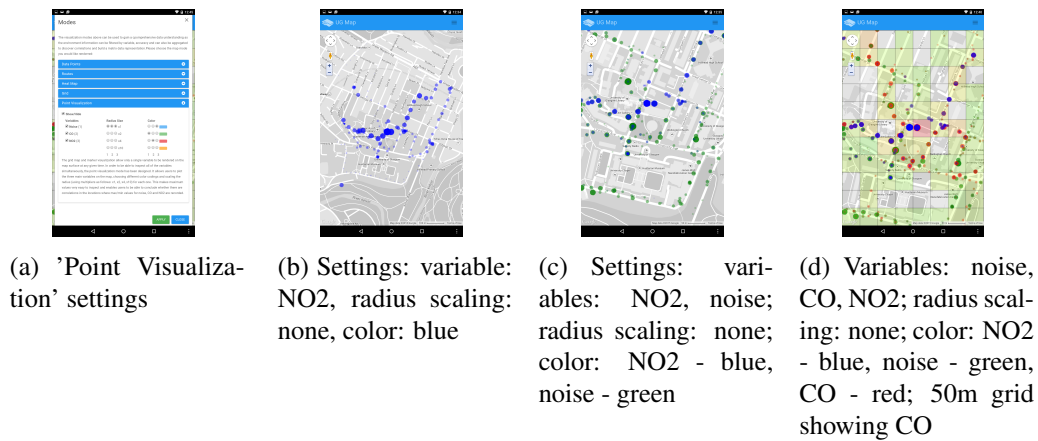


Figure 5.11: 'Point Visualization' mode settings and map rendering

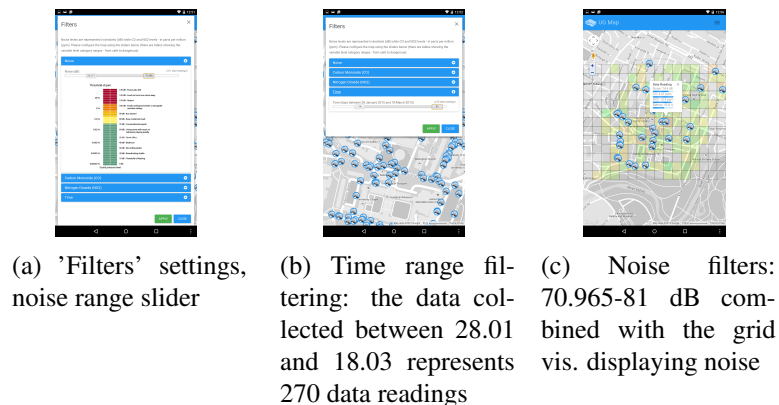


Figure 5.12: Variable filters settings and map rendering

process knew the Glasgow University Campus quite well, so although they found this tracking functionality very useful, some of them preferred not to use it for battery conservation. The back-end algorithm produces a user prompt ('You have reached your destination') once the user is within a 50 m radius of the chosen destination place (as shown in figure 5.13e).

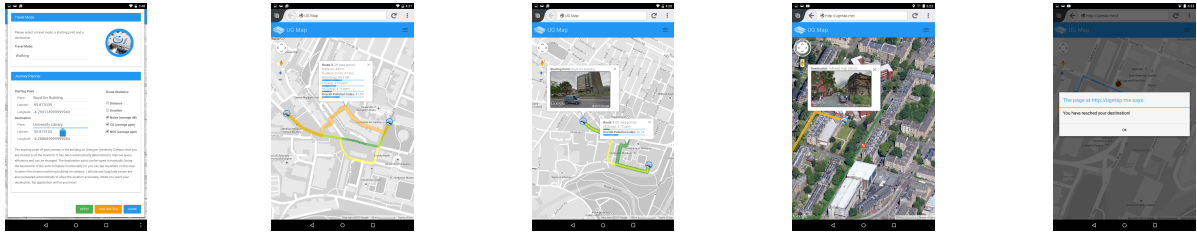
This 'Locations' modal view (figure 5.14a) displays a list of all the locations in the database. It has control buttons for the geolocation tracking functionality (it can be turned on and off (figure 5.14b) in order to facilitate battery power conservation). Google's Street View Image API[53] has been used to generate street view perspective imagery for each building. All of these operations are processed dynamically, which means that when new buildings are included to the data-store, the front-end will be updated according to the applied changes. A jQuery accordion menu stylized by using standard Bootstrap 3.3 elements has been used to facilitate the implementation of this component. Map buildings can be inspected by using the 'View' button which pans and zooms the main map directly above the selected university campus location.

Statistical (secondary views)

These pages have mostly information purpose and have been designed to aid the user when interacting with the rest of the server views. They contain general instructions and are aimed at improving the overall user understanding of the project's goal and application implementation.

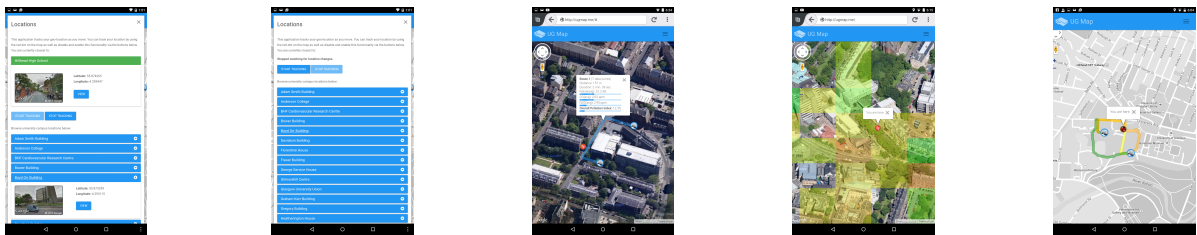
The data view has not been designed for user viewing - it has been used for the benchmarking sessions with SQL and NoSQL DBMS (as described in 7.1). Its purpose is to assess the efficiency of a data-store mechanism based on predefined criteria - the speed of a single (and thousands of) record insertion, retrieval and delete as well as the retrieval of all records currently in the database.

The 'devices.jsp' page view displays meta data about users and devices that have transmitted data to the



(a) Boyd Orr - library; travel mode - walking; sorting variables - noise, CO, NO2
 (b) Inspecting the returned routes and checking what the overall pollution index is (distance and duration criteria added)
 (c) Boyd Orr - J. Watt South; travel mode - bicycle; sorting variable - NO2
 (d) Boyd Orr - Hill-head School user tracking using the street map.
 (e) The user is prompted upon reaching their destination.

Figure 5.13: Route generation functionality, based on the grid data structure



(a) Displaying the closest building to the current user's location; showing the selection of 'Boyd Orr' in the list of buildings.
 (b) Users can stop the geolocation tracking to conserve the battery power of their device.
 (c) The red dot displays the current user's location if tracking is enabled.
 (d) Walking on campus and inspecting the CO levels using the street map.
 (e) Location is updated as the user is moving from the starting to the destination point (Western Inf. - Davidson bldg).

Figure 5.14: Campus locations and geolocation tracking

server based on the Smart Citizen Kit API[41] calls. The JSON object transmitted after each route tracking session contains this meta-data. If any attributes have changed values, the new 'User' and device object models are persisted in the NoSQL database. The code listing below shows how the JSON objects are loaded to models before being inserted in the 'Users' and 'Devices' collection accordingly:

```

1 //controllers/HomeController.java
2
3
4 public String addRoute(..){
5     User user = new Gson().fromJson(userJSON, User.class);
6     Device device = new Gson().fromJson(deviceJSON, Device.class);
7     ..
8 }

```

'sck.jsp' is a page containing general information regarding the Smart Citizen Kit. It aims to help potential users by providing an introductory paragraph about the sensor board and a link to the official website. The 'About' page ('about.jsp') has a similar purpose. However, the content is associated to the idea of the whole system and how the Smart Citizen Kit device contributes to the implementation of the application. As this is a highly-specific context, many users need a reference point in order to understand how the system functions so that they can make a better use of it. The addition of this information resulted from the user pilot evaluation session (more information is available in the 'Evaluation' chapter, section 7.4). Furthermore, it is beneficial in terms of SEO (improves on-page search engine optimisation).

The client modal page enables users to download the latest version of the client application for environmental data collection. The available mobile operating system choices at the moment are Google Android[48] (versions

1.9 and newer), Windows Phone[68]. Apple iOS[50] is currently not available as it turned out that all of the users who participated in the evaluation process own a primary Android mobile device. Furthermore, compiling an executable application file for iOS requires an iPhone developer certificate that needs to be granted by the Apple developer portal.

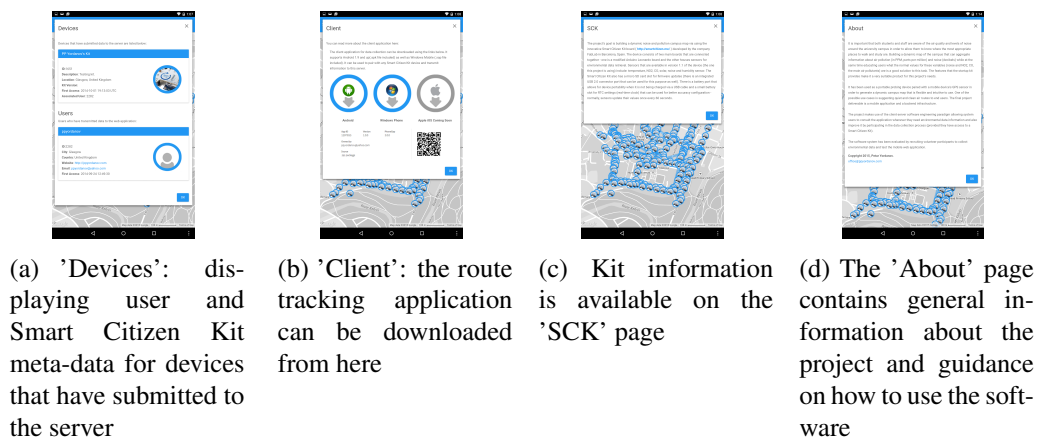


Figure 5.15: 'Point Visualization' mode settings and map rendering

All of the page views described above have been extracted into separate '.jsp' files in order to improve modularity. They have been injected in the main page ('home.jsp'). In order to optimize the overall server front-end structure and reduce the boilerplate code to a minimum (producing a template architecture), the navigation bar menu has also been extracted in a separate module - 'navigation.jsp'. Thus, it can be injected in the source code of each main page of the web application. The structure of this page has been highlighted below:

```

1 <@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5 .. <!-- metadata -->
6 <meta content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0" name='viewport' /> <!-- scaling on smaller
7 screens ensures that element proportionality is kept on smaller devices -->
8 .. <!-- style sheet attachments -->
9 .. <!-- JavaScript library and custom script attachments -->
10 </head>
11 <body>
12 .. <!-- main map initialization -->
13 .. <!-- navigation bar, data parser and views injection -->
14 </body>
</html>

```

In order to produce an unique identity for the project's look and feel, a custom logo was produced. It is available as a vector graphic (.fla' file) in the project's codebase and has been designed using Adobe Flash CS5.

Chapter 6

Testing

Performance testing, self-evaluation, usability, unit and integration testing was performed. The implementation has been refactored to improve the specified indicators.

6.1 Unit and Integration Testing

Unit testing was performed on the back-end structures in order to ensure that the models function as expected. All key interactions with the database instance were tested by using DAO (database access object) tests. The interface classes, extending 'MongoRepository<>' objects in the repository package 'com.springapp.mvc.repositories' were used to facilitate this process. The model and controller tests are available in the 'test' package in the 'ControllersTest' and 'ModelsRepositoryTests' classes. They can be run from the project parent directory by executing the 'mvn test' command.

The DAO tests use the '@Autowired' annotation to access the main repository objects. The tests are implemented using JUnit4 and run with the 'SpringJUnit4ClassRunner.class' as well as use the context configuration annotation '@ContextConfiguration' to link relevant resources. The four main objects are tested by attribute modification using the accessor and mutator methods and primarily by inserting, updating, deleting and searching for (based on id) them in the collections. The base framework of the DAO unit tests is presented below:

```
1 // package test/java/com.springapp.mvc
2 public class ModelsRepositoryTests {
3     ..
4     @Before
5     public void setup() { .. } //set up tests, create test variables and save them in the data store using the autowired repository
6         objects
7     @Test
8     public void dataReadingFindByRouteIdTest() throws Exception { .. } //ensure that the retrieved entity belongs to the set of objects
9         with the same route id
10    @Test
11    public void dataReadingFindByDeviceIdTest() throws Exception { .. } //ensure that the retrieved entity belongs to the set of objects
12        with the same device id
13    @Test
14    public void dataReadingFindByIdTest() throws Exception { .. } //find by id
15    @Test
16    public void dataReadingInsertTest() throws Exception { .. }
17    @Test
18    public void dataReadingUpdateTest() throws Exception { .. }
19    @Test
20    public void dataReadingDeleteTest() throws Exception { .. } //similarly structured DAO tests have been created for the rest of the
        models
    @After
    public void tearDown() { .. } //reset the database to its previous state, removing test-data
}
```

The tests have been configured to delete all of the test data inserted in the database after each run so that it remains in the same state. Mock data is saved during the 'setup()' method execution prior to running each test and removed in the 'tearDown()' function. This ensures that each unit test case is using the same test data set representation in case there are any variable modifications.

The integration tests have been designed in order to test the separate system components that operate with multiple models (the main reason why unit tests are implemented first is that they test the basic data structures

and allow composite components to be tested afterwards). They follow the 'happy-sad path' testing - there are multiple tests for each endpoint, a successful and a failing (invalid input/ bad request) test. The 'MockMvc' object has been used. It facilitates the generation of customizable requests to the server's endpoints and allows for comprehensive testing by inspecting the response. Content-type and headers can be specified as well as request body and all of this is done programmatically with high precision, again, relying on the context configuration ('mvc-dispatcher-servlet.xml'). The controller tests have been designed to print out the request and response data so that it can be carefully inspected on execution completion.

The 'ControllersTests' class is similarly structured to the 'ModelsRepositoryTests' class. Mock data in the form of JSON strings is loaded into objects and it has been used to add a new route to the database - along with the corresponding data points as well as a device and a user (if they do not exist, otherwise are only updated). Below is a code listing showcasing the success route insertion test:

```

1 // package test/java/com.springapp.mvc
2 public void routeSuccessTest() throws Exception {
3     mockMvc.perform(post("/addRoute").contentType(MediaType.APPLICATION_JSON) // specify POST request content-type
4         .param("context", context) // add the context parameter as specified by the method signature
5         .param("user", userJSON) //user parameter
6         .param("device", deviceJSON) // and device
7         .andDo(print()))
8         .andExpect(status().isOk()); // assert that the response is 200 (OK, created)
9 }

```

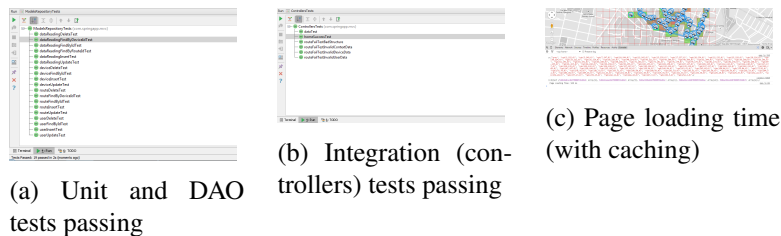


Figure 6.1: Server unit, integration testing and loading performance

6.2 Performance

The performance of the server and the client has been tested. This involved tracking loading times for page assets - images/ graphics as well as textual content and data parsing on page refresh. The rendering time for map configuration changes was also inspected. The performance assessment function for page loading time has the following implementation:

```

1 //utility/utility.js
2 function logPageLoadingTime() {
3     var loadingComplete = Date.now();
4     var userLoadTime = loadingComplete - performance.timing.navigationStart; //calculate the page loading time
5     console.log("Page Loading Time: " + userLoadTime + " ms"); //log the page loading time to the console
6 }

```

What this function calculates is the time passed between the point a user navigates to a specific page and the point when all or most of the page assets have finished loading. The expected page loading time without caching that would be considered within the norm is 0-2 seconds. Currently the page is loading for about 1-1.5 seconds on average after code refactoring has been performed. A jQuery page loader was implemented on the front-end in order to improve the user experience when the map takes 2-3 seconds to render.

Usability testing and user interface intuitiveness assessment was also performed. As the hosting server provides detailed graphs regarding the bandwidth, disk and CPU usage, those statistics have been analyzed. They are split in two main categories - system and user resource utilization. A clear correspondence can be observed between the data collection periods and the spikes in disk and bandwidth usage when viewing the data from the last year (365 days), while the CPU statistics are only useful for shorter time spans. Logging has also been implemented to facilitate testing and debugging (section 5.3, chapter 5).

Chapter 7

Evaluation

7.1 DB Benchmarks

Considering the fact that data store would be an integral part of the implementation process, a number of database systems were considered and compared in terms of performance. They can be split into two main categories:

MySQL and Mongo DB Comparison

Having measured the approximate maximum load that the system should sustain (section 3.1, chapter 3), the next task was to simulate this load on the server code by setting up an instance of each database engine and performing standardized testing, followed by comparison of the results to determine best performance. Below 7.1 are presented more details regarding the benchmarks carried out as well as the results obtained at the end. 5000 sample data readings collected from the device while indoors were used as input data. The database management systems were evaluated based on a number of different criteria. The following tests were carried out:

Data Write	Data Read	Additional Tests
✓insert thousands of records (5 000, 60 000, 200 000) records in the key DB table (DataReading)	✓single instance of type data reading insertion	✓update a single instance of type data reading
✓retrieve all records in DataReading table	✓retrieve a single instance of type data reading from the table	
✓delete a single record from the table		

Table 7.1: MySQL versus NoSQL benchmarking criteria.

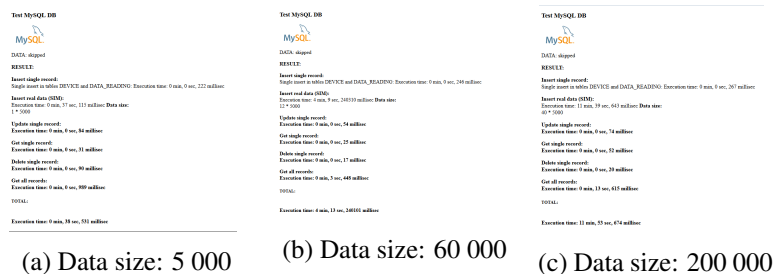
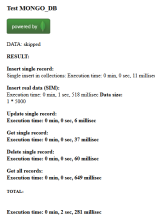


Figure 7.1: Figures 7.1a, 7.1b, 7.1c showing MySQL tests

The test results can be seen in figures 7.1(MySQL) and 7.2(MongoDB). MySQL, having stable referential integrity enforcing mechanisms and as expected, fit very well in the designed entity-relationship model. It



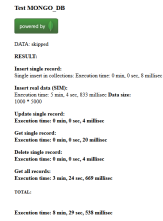
(a) Data size: 5 000



(b) Data size: 60 000



(c) Data size: 200 000



(d) Data size: 5 000 000

Figure 7.2: Figures 7.2a, 7.2b, 7.2c, , 7.2d showing MongoDB tests

also demonstrated great data caching capabilities. The data access operations did internally rely on JDBC and prepared statements to improve data retrieval efficiency. Hibernate was used as a form of a more advanced architecture to improve the potential need of scaling and provide robustness. Hibernate is an *ORM* (object-relational mapping) library that significantly improves the interaction between the classes of an object oriented application and a relational database (which MySQL is). Combined with bean parameters in a framework such as Spring MVC it is a very efficient way to retrieve data from as well as insert data in the database. It facilitates the creation of efficient *DAO* (data-access-object) structures and uses a dedicated session management tool - a *SessionFactory*, to persist objects. *HQL*[27] is a powerful query language that has a syntax similar to SQL. It was used in combination with MySQL to manipulate *DAOs* and is fully object oriented, which allows it to parse software development hierarchy such as polymorphism, association and inheritance. It also uses prepared statements internally, meaning that query strings are pre-compiled and stored at run-time for maximum efficiency and increased reusability (statement caching).

Mongo DBs configuration utilized the `MongoRepository` object along with a Spring MVC XML configuration bean parameter to set up the database and add the relevant repositories for data access. A code listing of both can be seen below:

```

1 //DataReading.java
2
3 package com.springapp.mvc.repositories;
4 import com.springapp.mvc.models.DataReading;
5 import org.springframework.data.mongodb.repository.MongoRepository;
6 /**
7  * Created by Peter Yordanov on 29.10.2014
8  */
9 public interface DataReadingRepository extends MongoRepository<DataReading, String> {
10
11     public DataReading findByRouteId(String routeId);
12     public DataReading findByDeviceId(String deviceId);
13     public DataReading findByBattery(Double battery);
14     public DataReading findById(String id);
15
16 }

```

```

1 <!-- mvc-dispatcher-servlet.xml -->
2
3 <beans>
4     ..
5     <mongo:repositories base-package="com.springapp.mvc.repositories"/>
6     <bean id="mongo" class="org.springframework.data.mongodb.core.MongoFactoryBean">
7         <property name="host" value="localhost"/>
8     </bean>
9     <!-- MongoTemplate for connecting and querying the documents in the database -->
10    <bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
11        <constructor-arg name="mongo" ref="mongo"/>
12        <constructor-arg name="databaseName" value="ugmap"/>
13    </bean>
14 </beans>

```

`MongoTemplate` is another option for database collection access which provides more fine-grained control over data store querying but for the needs of this project the `MongoRepository` object type proved to be more efficient.

Couch DB is also a good NoSQL alternative and it was researched extensively. However, Mongo DB was chosen as the NoSQL representative in the benchmarking process as it is more extensively documented online and provides a 'query'-style language support.

Both database systems performed well with reasonably small-sized input. However, as the input set size

increased, Mongo DB performed times faster than MySQL (as shown in figures 7.1 and 7.2) proving to be much better suited for the needs of the system and thus became the DBMS of choice. The source code for the database benchmarks is located in the 'DataPopulation and 'Benchmark' classes in the package 'utilities'.

7.2 Prototype Evaluation

Two prototype (section 5.2 in chapter 5) evaluation sessions were conducted to test the functionality and evaluate the usability of the product prior to commencing the final implementation process(section 5.3 in chapter 5). Both sessions lasted for about two hours and were conducted on the premises of the university campus. The SCK case needed to be modified prior to this as the container that was previously used to store and carry it around obstructs the noise, CO and NO2 sensors on the board due to the micro-climate environment it creates. In order improve the sensor accuracy and reliability a new version was designed to protect the board and uncover the vital sensors that the system relies on (the noise detector as well as the CO and NO2 chips) while still maintaining the portability of the device. A small food storage container was used and two 2 ventilation holes were cut - one in the lid and on the side so that air could flow freely as well as avoid sound wave obstructions. The SCK device was carefully placed in the container using foam to introduce additional protection.

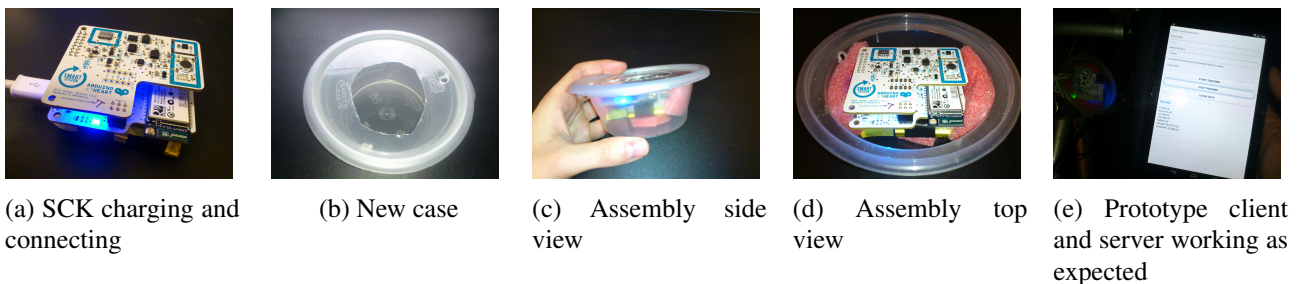


Figure 7.3: Prototype evaluation, case modification

There were some minor issues during the first evaluation session. Data received from the device was constantly monitored to verify that there is stable network access and updates are sent on regular time intervals. Server route aggregation was also checked after each tracking session completion. However, Wi-Fi network latency and delays as well as movement speed influenced transmission time and environmental data was not updating properly. The accuracy of the GPS sensor embedded in the mobile device used (Nexus 7) started varying which caused some fluctuations in the output. It was diagnosed that this variance results from the fact that the GPS sensor needs some time (a few minutes) to precisely locate the device, improving the precision of each consecutive geolocation update. The first prototype version of the client turned the GPS sensor on and off (every 30 sec. by default) in order to reduce the resource consumption levels, which turned out to be the issue. The final implementation relies on the GPS sensor being turned on throughout the whole route tracking session (although updates are retrieved every 30 sec.) in order to improve location precision.

7.3 Planning

For the purposes of user evaluation, a task sheet was designed in order to record objective opinion of each volunteer participant. The task sheet was accompanied by a questionnaire. Some questions were asked before the task sheet was handed to the participants in order to assess their understanding of the system functionality and evaluate how usable the interface is based on initial impressions. On the other hand, the closing questions were designed in order to discuss ideas and try to identify opportunities for future improvement. Those questions did not only ask for comments and thoughts, they also allowed participants to rate different aspects of the application. A Nasa TLX type form was used (appendix section A.4). This provides a very stable usability testing platform. A questionnaire with three sections was also designed to get more feedback. The first section consists of opening

questions that are used to collect general statistical data. This is followed by a task section requiring participants to use the route tracking and visualization software, while carrying the sensor boards. The third section consists of the questions in the TLX form as well as questions aiming to document user experience. Participants are also asked to give improvement suggestions. The complete task sheet is available in the appendix, section A.4. The standard documentation - a consent form (section A.4), an introduction (section A.4) and a debrief script (section A.4) are also available in the appendix.

7.4 Pilot Evaluation

Figure 7.4 shows the results from the pilot evaluation session. Participants considered the application useful and 75% said that they would use it if it becomes available on the web application catalogue for their device. Valuable suggestions for improvement have been collected and addressed as documented in section 7.5.

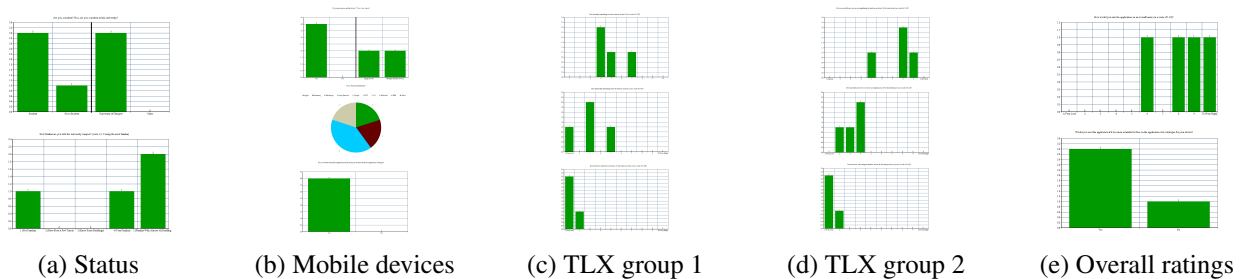


Figure 7.4: Pilot evaluation questions and feedback

7.5 Final Evaluation

After the pilot evaluation session plenty of ideas for improvement were collected. All of them were implemented, although they are entirely beyond the scope of the original functional requirements for the application. Automatic population of the starting point with the users current location (or building on campus that is closest to the current location) is currently available after being requested. Users can click anywhere on the map to select the closest building in the database as a destination point, without having to type it in. There is also a navigator to show the current location of the user and display a pop-up with a message ('You have reached your destination!') when they reach their desired location. Start/stop tracking functionality has been added to enable battery power conservation. Instructions and guidance information specifying how to interact with the UI has been included on the server (charts displaying the safe levels for noise, CO and NO₂ have also been included) in order to improve the usability of the software.

The results of the final evaluation session are shown in figure 7.5. Most participants stated that they would use the application if it was to be released. Participants also helped to identify future work ideas by giving suggestions for further improvement (section 8.2).

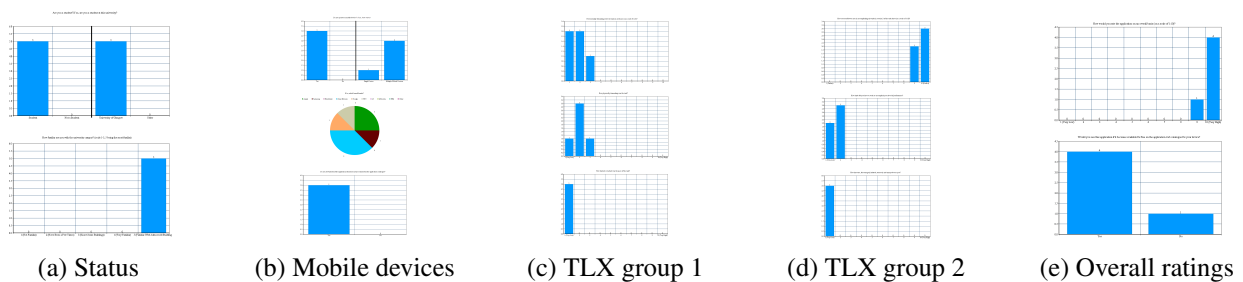


Figure 7.5: Pilot evaluation questions and feedback

Chapter 8

Conclusion

8.1 Summary

The main aim of this project was to develop an application that could be accessed from mobile devices and would display a noise and pollution map of the Glasgow university campus. The implementation was facilitated by the SCK sensor board which has the required sensors to enable comprehensive environmental tracking. The two evaluation sessions that have been carried out ('Evaluation', chapter 7) have indicated that users consider the application useful and would use it if it becomes available for their mobile device.

8.2 Future Work

After the pilot evaluation and the implementation of suggested features, the volunteer participants in the final evaluation continued to give suggestions for improvement. They helped to identify the following additional features that could be implemented in future system releases:

1. Export functionality so that visualizations can be saved for future reference. This can not only be limited to image exports, '.csv' table files with environmental values can also be saved for example.
2. Implementing an automatic trigger on the Android application which starts tracking once the user is moving to reduce the complexity of the process (this could be a useful feature, as long as it can be enabled and disabled).
3. Additional search places when generating routes, such as specific streets and addresses.
4. Additional device support for Windows Phone and especially Apple iOS to increase the number of potential users.
5. The design of a specialized user guide that demonstrates all of the features one by one and leads the user using hints available on the UI.

8.3 Lessons Learnt

The project has been of significant importance for the developer and author as it has provided a stable experience in the Android (Phonegap) and mobile web application development (Spring MVC) using the model-view-controller programming paradigm. It has also led to the obtainment of significant knowledge for data set visualization and composite sensor technology (referring to the Smart Citizen Kit and Arduino development). NoSQL database experience has been gained as well, specifically during the DBMS benchmarking research process and test validation. Additionally, the author's knowledge about REST API[57] endpoints manipulation as well as client-server model system development and maintenance (unit, integration testing) has greatly improved.

Bibliography

- [1] Android Debug Bridge , a command-line tool for emulator instance connection and communication. <http://developer.android.com/tools/help/adb.html>. Accessed: 2015-03-25.
- [2] Android SDK , android software development kit. <http://developer.android.com/sdk/index.html>. Accessed: 2015-03-25.
- [3] Apache Ant , a java library and a command-line tool used for process management. <http://ant.apache.org/>. Accessed: 2015-03-25.
- [4] Apache Cordova . <http://cordova.apache.org/>. Accessed: 2015-03-25.
- [5] Apache Maven , a software project management and comprehension tool. <http://maven.apache.org/>. Accessed: 2015-03-25.
- [6] Apache Slf4 . <http://www.slf4j.org/>, a Java logging facade. Accessed: 2015-03-25.
- [7] Boilerplate , a robust web application development framework. <https://html5boilerplate.com/>. Accessed: 2015-03-25.
- [8] Django Framework , a high-level python web framework that encourages rapid development and clean, pragmatic design. <https://www.djangoproject.com/>. Accessed: 2015-03-25.
- [9] DNPCM Client , client source code. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/tree/client>. Accessed: 2015-03-25.
- [10] DNPCM Client and Server Deployment , deployment instructions. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/blob/client/README.md>. Accessed: 2015-03-25.
- [11] DNPCM Client compiled , client compiled files. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/tree/client/GEN>. Accessed: 2015-03-25.
- [12] DNPCM development documentation , github issues. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/issues>. Accessed: 2015-03-25.
- [13] DNPCM milestones , github milestones and relevant tasks. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/milestones>. Accessed: 2015-03-25.
- [14] DNPCM Server , server source code. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/tree/master>. Accessed: 2015-03-25.
- [15] DNPCM Video , source file. <https://drive.google.com/file/d/0B4VtbfdyYAnbV1pwc1BpZW5YUkU/view?usp=sharing>. Accessed: 2015-03-25.
- [16] DNPCM Wiki , project wiki page. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/wiki>. Accessed: 2015-03-25.

- [17] Dynamic Noise and Pollution Campus Map , demonstration video. <https://www.youtube.com/watch?v=8V8YSHsBpTA>. Accessed: 2015-03-25.
- [18] Dynamic Noise and Pollution Campus Map , github repository page. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map>. Accessed: 2015-03-25.
- [19] Dynamic Noise and Pollution Campus Map , official web page. <http://ugmap.me/>. Accessed: 2015-03-25.
- [20] Flask , a microframework for python based on werkzeug, jinja 2. <http://flask.pocoo.org/>. Accessed: 2015-03-25.
- [21] Geolocation , html5 geolocation. http://www.w3schools.com/html/html5_geolocation.asp. Accessed: 2015-03-25.
- [22] Google Android Studio , android development ide. <http://developer.android.com/sdk/installing/index.html?pkg=studio>. Accessed: 2015-03-25.
- [23] Google Directions API , google maps api web services. <https://developers.google.com/maps/documentation/directions/>. Accessed: 2015-03-25.
- [24] Google Gson , official documentation. <https://code.google.com/p/google-gson/>. Accessed: 2015-03-25.
- [25] Google Maps , official page. <https://maps.google.com/>. Accessed: 2015-03-25.
- [26] Google Maps API V3 , the google map javascript api. <https://developers.google.com/maps/documentation/javascript/>. Accessed: 2015-03-25.
- [27] HQL , the hibernate query language. <https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html>. Accessed: 2015-03-25.
- [28] HTML5 , html5 documentation. http://www.w3schools.com/html/html5_intro.asp. Accessed: 2015-03-25.
- [29] JavaScript, official mozilla documentation. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Accessed: 2015-03-25.
- [30] Jetty , a web server and servlet container provider. <http://eclipse.org/jetty/>. Accessed: 2015-03-25.
- [31] jQuery , a fast, small, and feature-rich javascript library. <http://jquery.com/>. Accessed: 2015-03-25.
- [32] jQuery Mobile , official documentation. <http://jquerymobile.com/>. Accessed: 2015-03-25.
- [33] MapBox , one of the biggest providers of custom online maps for major websites such as foursquare, pinterest, evernote, the financial times and uber. <https://www.mapbox.com/>. Accessed: 2015-03-25.
- [34] Microsoft Visual Studio , official website. <https://www.visualstudio.com/>. Accessed: 2015-03-25.
- [35] Mongo DB Benchmarks , source code. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/tree/db/mongo>. Accessed: 2015-03-25.
- [36] MySQL Benchmarks , source code. <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/tree/db/mysql>. Accessed: 2015-03-25.

- [37] Node JS , a platform built on chrome's javascript runtime for easily building fast, scalable network applications. <https://nodejs.org/>. Accessed: 2015-03-25.
- [38] OpenStreetMap , built by a community of mappers that contribute and maintain data about roads, trails, cafs, railway stations, and much more, all over the world. <http://www.openstreetmap.org/copyright>. Accessed: 2015-03-25.
- [39] PhoneGap , a free and open source framework that allows you to create mobile apps using standardized web apis. <http://phonegap.com/>. Accessed: 2015-03-25.
- [40] SCK 0.9.1 , release beta 0.9.1. <https://github.com/fablabbcn/Smart-Citizen-Kit/releases/tag/v0.9.0>. Accessed: 2015-03-25.
- [41] SCK REST API , smart citizen kit api reference. <http://api.smartcitizen.me/>. Accessed: 2015-03-25.
- [42] Spring MVC , official page. <https://spring.io/>. Accessed: 2015-03-25.
- [43] Twitter Bootstrap , a popular html, css, and js framework for developing responsive, mobile first projects on the web. <http://getbootstrap.com/>. Accessed: 2015-03-25.
- [44] Zepto JS , a minimalist javascript library for modern browsers with a largely jquery-compatible api. <http://zeptajs.com/>. Accessed: 2015-03-25.
- [45] Agile Manifesto, manifesto for agile software development. <http://agilemanifesto.org/>. Accessed: 2015-03-25.
- [46] Agile, the essence of agile. <https://gordonmcmahon.wordpress.com/2013/01/21/the-essence-of-agile/>. Accessed: 2015-03-25.
- [47] Air Quality Index, air pollution in united kingdom: Real-time air quality index visual map. <http://aqicn.org/map/unitedkingdom/>. Accessed: 2015-03-25.
- [48] Android, official web page. <https://www.android.com/phones/>. Accessed: 2015-03-25.
- [49] Apache Couch DB, official website. <http://couchdb.apache.org/>. Accessed: 2015-03-25.
- [50] Apple iOS, official web page. <https://www.apple.com/ios/>. Accessed: 2015-03-25.
- [51] Client-Server, programming model description. http://en.wikipedia.org/wiki/Client%E2%80%93server_model. Accessed: 2015-03-25.
- [52] Google Libraries, official documentation. <https://developers.google.com/maps/documentation/javascript/libraries>. Accessed: 2015-03-25.
- [53] Google Street View Image API, official documentation. <https://developers.google.com/maps/documentation/streetview/>. Accessed: 2015-03-25.
- [54] Mongo DB, official website. <https://www.mongodb.org/>. Accessed: 2015-03-25.
- [55] MySQL, official website. <http://www.mysql.com/>. Accessed: 2015-03-25.
- [56] Native vs. Web Development, trade-offs between the two approaches. <http://thenextweb.com/dd/2014/02/08/decide-responsive-website-native-mobile-app/>. Accessed: 2015-03-25.
- [57] REST, what are restful web services. <http://docs.oracle.com/javasee/6/tutorial/doc/gijqy.html>. Accessed: 2015-03-25.

- [58] SCK Firmware, firmware releases. <https://github.com/fablabbcn/Smart-Citizen-Kit/releases>. Accessed: 2015-03-25.
- [59] SCK Kickstarter, start-up campaign page. <https://www.kickstarter.com/projects/acrobotic/the-smart-citizen-kit-crowdsourced-environmental-m>. Accessed: 2015-03-25.
- [60] SCK Updates, new sensor board models news. <https://smartcitizen.me/posts/view/9>. Accessed: 2015-03-25.
- [61] Smart Citizen Kit, official page. <https://smartcitizen.me>. Accessed: 2015-03-25.
- [62] SOAP, soap services description. http://www.w3schools.com/webservices/ws_soap_intro.asp. Accessed: 2015-03-25.
- [63] SolarCity, official website. <http://www.solarcity.com/>. Accessed: 2015-03-25.
- [64] Tesla Motors, official page. <http://www.teslamotors.com/>. Accessed: 2015-03-25.
- [65] Web API, description of web apis. http://en.wikipedia.org/wiki/Web_service#Web_API. Accessed: 2015-03-25.
- [66] Web Service Types, types description. <http://www.w3schools.com/webservices/>. Accessed: 2015-03-25.
- [67] Web Service, what are web services. <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice/>. Accessed: 2015-03-25.
- [68] Windows Phone, official web page. <http://www.windowsphone.com/en-us>. Accessed: 2015-03-25.
- [69] Julien Gedeon Roman Brtl Max Mhlhuser Immanuel Schweizer, Christian Meurisch. Noisemap: multi-tier incentive mechanisms for participative urban sensing. *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*, (9), 2012.
- [70] Mauricia Benedito Sergi Trilles Arturo Beltrn Laura Daz Joaquin Huerta Irene Garcia Mart, Luis E. Rodriguez. *Mobile Application for Noise Pollution Monitoring through Gamification Techniques*. Springer Berlin Heidelberg, Berlin, Germany, 2012.
- [71] Richard Moss. Crowdsourced Pollution Maps , smartphone sensor generates crowdsourced pollution maps. <http://www.gizmag.com/smartphone-sensor-crowdsourced-pollution-map-karlsruhe/32932/>. Accessed: 2015-03-25.
- [72] J. ; Stojanovic D. ; Aberer K. Predic, B. Zhixian Yan ; Eberle. *ExposureSense: Integrating daily activities with air quality using mobile participatory sensing*. PhD thesis, Department of Computing Science and Mathematics; University of Ni, Serbia, 2013. Accessed: 2015-03-25.
- [73] Nicholas Tufnell. Noise Pollution, noise pollution mapped by data collected on smartphones. <http://www.wired.co.uk/news/archive/2013-10/23/crowd-sourced-noise-pollution-monitoring>. Accessed: 2015-03-25.

Appendices

Appendix A

Appendices

A.1 Appendices

A.2 Resources

- Dynamic Noise and Pollution Campus Map Github repository page [18]
- DNPCM Server server source code[14]
- DNPCM Client client source code[9]
- DNPCM Client compiled client compiled files[11]
- Mongo DB Benchmarks source code[35]
- MySQL Benchmarks source code[36]
- DNPCM Wiki project wiki page[16]
- DNPCM development documentation Github issues[12]
- DNPCM milestones Github milestones and relevant tasks[13]
- DNPCM Client and Server Deployment deployment instructions[10]
- DNPCM Demonstration Video[17]
- DNPCM Video source file[15]
- Dynamic Noise and Pollution Campus Map official web page [19]

A.3 Progress Reports, Sprint Retrospectives

The reports have been ordered in chronological order:

A.3.1 Progress Report 1

Dynamic Noise and Pollution Campus Map

Tehnologies to be Used

- Spring MVC RESTful web service for back-end processing
- Twitter Bootstrap
- Google Developer Tools/PhoneGap(Apache Cordova) for the mobile client implementation
- Google Maps API
- Smart Citizen Kit device and RESTful API
- jQuery for data vizualization
- D3 js (option)
- CouchDB/ SQL DBMS for remote data storage

Work Done

- UI prototyping
- user scenarios
- SCK 1.1 configuration
- general structural planning
- requirements capture

Work Under Way

- environment configuration
- server and database server setup
- data structure

Known Issues + Resolutions

- SCK firmware configuration via Arduino IDE (Windows) -> resolved by changing the OS (UNIX) and using the serial monitor for manual upload
- Spring MVC model-view pair configuration -> resolved by updating the dispatcher servlet to discover all classes

/Some of these choices are just options, there might be some changes in tools used as development progresses./

Petar Yordanov, 14.10.2014

A.3.2 Progress Report 2

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- prepared structural diagrams
 - ER diagram (including Chen's Notation) - [link](#)
 - sequence diagram - [link](#)
 - high level architectural diagram - [link](#)
- Spring initial models setup - [link](#)
- uploaded the updated 0.9.0 SCK firmware on the device (now transmits data every 30 sec.) - [link](#)
- emailed SCK team to ask if they have open source authentication API to be used on the server side /no response yet, got a detailed response for previous enquiries though/

Work Under Way

- database server setup
- MVC server configuration
- Bootstrap setup
- updating the [wiki](#) (interview Q&A, report 1 already published)
- data set simulation on server side
- looking at DNSimple and cloud based hosting services to serve DB and accommodate server framework

Known Issues + Resolutions

- some minor issues when updating the software:
 - the WiFly module would not detect any Wi-Fi networks
 - the device could not connect
- *resolution – used the IDE serial monitor to set network ssid, password and type manually and reset the device*

Petar Yordanov, 20.10.2014

A.3.3 Progress Report 3

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- MySQL database server setup
- MVC server configuration
- Bootstrap setup
- data set simulation on server side
- DigitalOcean droplet (5\$/ month Ubuntu 14.10 x64 20 GB ssd 512 RAM) setup

Work Under Way

- populate DB
- visualise data (SIM)

Known Issues + Resolutions

- SCK developers are now implementing an oAuth system with open source API on the official website which is going to be available in mid 2015, will not be possible to use their authentication mechanism – this is good to know, not an issue, can process data anonymously

Petar Yordanov, 28.10.2014

A.3.4 Progress Report 4

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- Parsing + loading models from JSON (custom deserialisation)
- MySQL tests (benchmarks: 5000, 60000, 200000 el. set)
 - Results: fast update, retrieve and delete, slower insertion; larger data sets lead to a significant slow-down; however, when caching is used, very fast execution
- MongoDB tests (benchmarks: 5000, 60000, 200000, 5000000 el. set)
 - Results: extremely fast insertion -> 200 000 records for around 20 sec.; however, caching is significantly less efficient while MySQL guarantees better performance in the long run (in my opinion, looking at the test results)

Work Under Way

- visualise data (simulation)

Known Issues + Resolutions

- Auto-wiring the MongoDB template to the controller -> resolved by using the @Autowired Spring annotation for bean parameters.
- Configuring the models to work with MongoDB, especially the Timestamp object (tried to use mongo's BSONTimestamp) -> resolved by reverting back to java.sql.Timestamp

Test Conclusions:

Mongo DB does an excellent job regarding data insertions - its capabilities/ insertion algorithm/ have no match considering this. Data retrieval, update and delete of single records is also very fast, however, it is difficult to argue if much faster than *MySQL* .

All of the benchmarks have been completed without caching mechanisms being used. Not surprisingly, *Mongo DB* has shown around x1.5 up to x2 faster data manipulation than *MySQL* and this tendency is present for all of the data sets (I did not even think of running a 5 million data set insertion script for the *SQL DBMS*).

Efficient data caching is one of the main advantages of *MySQL* in this comparison. Although I have not uploaded results with memory caching in use, I noticed that query execution (especially

database reading) for more lengthy operations dramatically decreases up to a multiplier of 2 after each consecutive call to the database, which will inevitably lead to a very improved efficiency in a long run. I would assume that it would be even much better than mongo. On the other hand, *Mongo DB* delivers a great first impression for the users who will visit the page more rarely (in other words, not make use of in-memory cache).

There is no need to consider the relational db structural benefits against the document-based, as the plans for MySQL did not involve querying more than a table at a time (or 2) and this would not affect the final decision much.

I did a quick search for similar tests/evaluations, and found some information confirming my observations so far:

- <http://www.moredevs.ro/mysql-vs-mongodb-performance-benchmark/>

Repository Benchmarks:

- DB System Choice: <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/issues/27>
- NoSQL DBMS Evaluation: <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/issues/30>
- MySQL Evaluation: <https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/issues/29>

A.3.5 Progress Report 5

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- visualise data (simulation):
<https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/issues/10>
 - set up a VC pair (view-controller) to use the parsed JSON models and display them on different locations around the university campus map
 - display additional information to the system user (data readings and battery status):
 - CO
 - NO2
 - Noise
 - Battery status
 - scaled the data, defining maximum values for the relevant measures and displayed it visually in bars; color-coding, etc. can be applied to improve interpretation and usability
- updated wiki (added new sections to improve overall system documentation and project structure)
- researched CO, NO2 and noise data scaling:
<https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/issues/47>
- researched data formatting (CO and NO2 are raw sensor values at the moment, displayed in kilo Ohms) -> the standard requires them to be in *ppm :
<https://github.com/ppyardanov/Dynamic-Noise-and-Pollution-Map/issues/48>
- code refactoring
 - optimised controller
 - improved JavaScript data processing

Work Under Way

- test multiple visualisation styles
- build up on the user interface of the mobile web app

Known Issues + Resolutions

- contacted the SCK team regarding NO2 and CO data formatting and they responded promptly with details; the idea behind using raw data (kilo Ohms resistance) is to ensure that when conversion to *ppm is used the sensors are going to be calibrated according to the surrounding environment and Fab Lab are working on a solution to this at the time being
- VC pairing for main controller and home view took a lot of trial and error as data needs to be passed (after being parsed into models) to the client-side code .jsp file and then accessed by JavaScript for populating a data structure in the script and visualizing this information on the map canvas
- map infoWindow() instances content getting overwritten leading to same data being displayed for each data reading -> resolved by using function closure, transferring marker generation in a separate function after thorough research on the issue

**PPM – parts per million*

A.3.6 Progress Report 6

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- client prototyping: [link](#)
- Android development environment setup (Windows OS):
 - Microsoft Visual Studio (2012 or later recommended)
 - nodeJS v0.10.33
 - PhoneGap v3.6.3-0.22.1 (requires the previous 2)
 - standalone Android SDK: [link](#)
 - AVD (Android Virtual Device)
 - ADB (Android Debug Bridge)
 - Apache Ant v1.9.4
- Android permissions research: [link](#)
- project building with Apache Ant: [link](#)
- initialized base PhoneGap Android project: [link](#)
- registered the test app on build.phonegap.com for maintainability: [link](#)

Work Under Way

- Android prototype implementation: [link](#)
- prototype testing/ evaluation

Known Issues + Resolutions

- PhoneGap configuration issues: could not avoid installing the required Visual Studio to enable the software to initialize an empty project
- Android SDK configuration issues: failed access to SDK manager-> resolved by manually editing one of the .bat scripts
- Android ADB would not recognize plugged in Android devices-> resolved by reinstalling through SDK manager (tried a universal driver which did not work) after thorough research

Petar Yordanov, 18.11.2014

A.3.7 Progress Report 7

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- client further prototyping:
 - GUI update: [link](#)
- functionality added: [link](#)
 - contextual information
 - configuration options
- code debugging
- test data gathering (brief evaluation of the prototype): [link](#)

Work Under Way

- upgrading the client
- visualizing data on the map (debugging)
- more test data generation

Known Issues + Resolutions

- Google Map API v3 InfoWindow() instances are not visible on the map: [link](#)
- Internet connectivity checking with user prompt prior to tracking needs to be handled

Petar Yordanov, 25.11.2014

A.3.8 Progress Report 8

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- map visualizations techniques research (grid index)
- further Android client testing

Work Under Way

- implementing server-client interaction
- facilitating server-client communication by checking current network settings and Wi-Fi connectivity

Known Issues + Resolutions

- grid indexing involves a trade-off between efficiency/ performance and precision; significant precision would involve rendering very small tiles that would most definitely impact rendering speed when basic map browsing is used (zoom –in and –out, dragging around)

Petar Yordanov, 03.12.2014

A.3.9 Progress Report 9

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- Digital Ocean server configuration: [link](#)
- server code deployment: [link](#)
- Spring MVC RESTful infrastructure implementation for client communication: [link](#)
- event logging facilitated by SLF4J improved: [link](#)
- debugging
 - client-server data transmission: [link](#)
 - client position check window: [link](#)

Work Under Way

- testing around campus: [link](#)
- data filtering

Known Issues + Resolutions

- Digital Ocean confusing documentation and lacking support for Java MVC web application frameworks (WAF) as documented in the source links above: <https://www.digitalocean.com/community/questions/spring-mvc-nosql-server-setup> ; resolved by researching for alternative toolkits, example: used “sysv-rc-conf” which borrows its syntax directly from the currently deprecated UNIX package “chkonfig” that was listed on one of the tutorials for automatic service initialization at startup (used for a Mongo DB instance)
- CORS vs JSONP data transmission (advantages and disadvantages, trade-offs), Google’s Gson vs Jackson JSON Processor -> highlighted in RESTful infrastructure issue
- HTML5’s geo-location service function changed to improve window interval specification (further details in “debugging” links)

Petar Yordanov, 09.12.2014

A.3.10 Progress Report 10

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- SCK Eduroam WEP64 Connection: [link](#)
- domain registration: [link](#)
- Android application tracking accuracy improvement
- evaluation:
 - 12.12.2014 Experiment: [link](#)
 - 14.12.2014 Evaluation: [link](#)

Work Under Way

- development + continuous testing

Known Issues + Resolutions

- GPS time interval and geo-location lock inaccuracies: resolved by changing the device data retrieval approach; previous implementation: get location once every 30 seconds (or depending on user input for window size) – very inaccurate and unstable, timeouts/delays; current implementation – use constant geo-tracking, no timeouts, much more accuracy using helpers – accelerometer and previous location estimation techniques. Results and evaluation: (links to evaluation sessions contain information on this)
- SCK Eduroam connection issue – resolved by setting up my device as a portable access point on a private Wi-Fi network (I used GiffGaff)

Petar Yordanov, 14.12.2014

Dynamic Noise and Pollution Campus Map

Personal Project 4 – Semester 1 Retrospective

Petar Yordanov (1103620)

School of Computing Science,

University of Glasgow

15.12.2014

Background

The project's goal is building a dynamic noise and pollution campus map via using the innovative Smart Citizen Kit board (<https://smartcitizen.me/>) developed by the company FabLab in Barcelona, Spain. The device consists of two main boards that are connected together - one is a modified Arduino Leonardo board and the other houses sensors for environmental data retrieval. Sensors that are available in version 1.1 of the device (the one this project is using) include – temperature, NO2, CO, solar, noise and humidity sensor.

The Smart Citizen Kit also has a micro SD card slot for firmware updates (there is an integrated USB 2.0 connector port that can be used for this purpose as well). There is a battery port that allows for device portability when it is not being charged via a USB cable and a small battery slot for RTC settings (real-time clock) that can be used for better accuracy configuration - normally, sensors update their values once every 60 seconds.

Description

It is important that both students and staff are aware of the air quality and levels of noise around the university campus in order to allow them to know where the most appropriate places to walk and study are. Building a dynamic map of the campus that can aggregate information about air pollution (in PPM – parts per million) and noise (decibels) while at the same time educating users what the normal values for these variables (noise and NO2, CO, the main air pollutants) is a good solution to this task.

The features that the startup kit provides make it a very suitable product for this project's needs. It can be used as a portable probing device paired with a mobile device's GPS sensor in order to generate a dynamic campus map that is flexible and intuitive to use. One of the possible use cases is suggesting quiet and clean air routes to end users. The final project deliverable is a mobile application and a backend infrastructure. The project makes use of the client-server software engineering paradigm allowing system users to consult the application whenever they need environmental data information and also improve it by participating in the data collection process (provided they have access to a Smart Citizen Kit).

The software system will be evaluated by recruiting volunteer participants to collect environmental data and test the mobile application.

Progress

During the first semester, the following system components and features have been implemented (and tested) to develop a final prototype:

- **SCK configuration:** firmware update and modification to improve accuracy
- **Server:**
 - cloud hosting configuration and domain registration

- Database Evaluation: SQL versus NoSQL comprehensive evaluation and testing with sample data (5 000 up to 5 000 000 records insertion and retrieval) was carried out to compare performance; database systems tested: MySQL and Mongo DB; Mongo DB was chosen as operation execution is significantly faster
- RESTful API implementation to facilitate client-server communication
- environmental data visualization
- **Client** (Android application):
 - route-tracking prototype
 - configuration options
 - SCK update retrieval functionality
 - user tracking history storage
 - client-server data transmission
- **Evaluation:**
 - Smart Citizen Kit case modifications to get better sensor exposure
 - generating environmental data readings (walking with different speed, trying different kit containers)
 - aggregating route information on the server's database

Plan

The plan for the second semester is to increase the configuration options for the client application and the server. The Android application needs to check mobile device internet connectivity and GPS accuracy and let the user know if there is no satellite/ network coverage (more user feedback in general). Data filters will be implemented on server side to enable users to identify the loudest/ quietest and most/ least air-polluted places around campus much easier (currently they need to look at the sensor values to do that).

Configuration settings will be based on an algorithm that will also let users extract routes they are interested in (in close proximity to their current location for example) and have certain PPM and decibel ranges for NO₂, CO and noise accordingly.

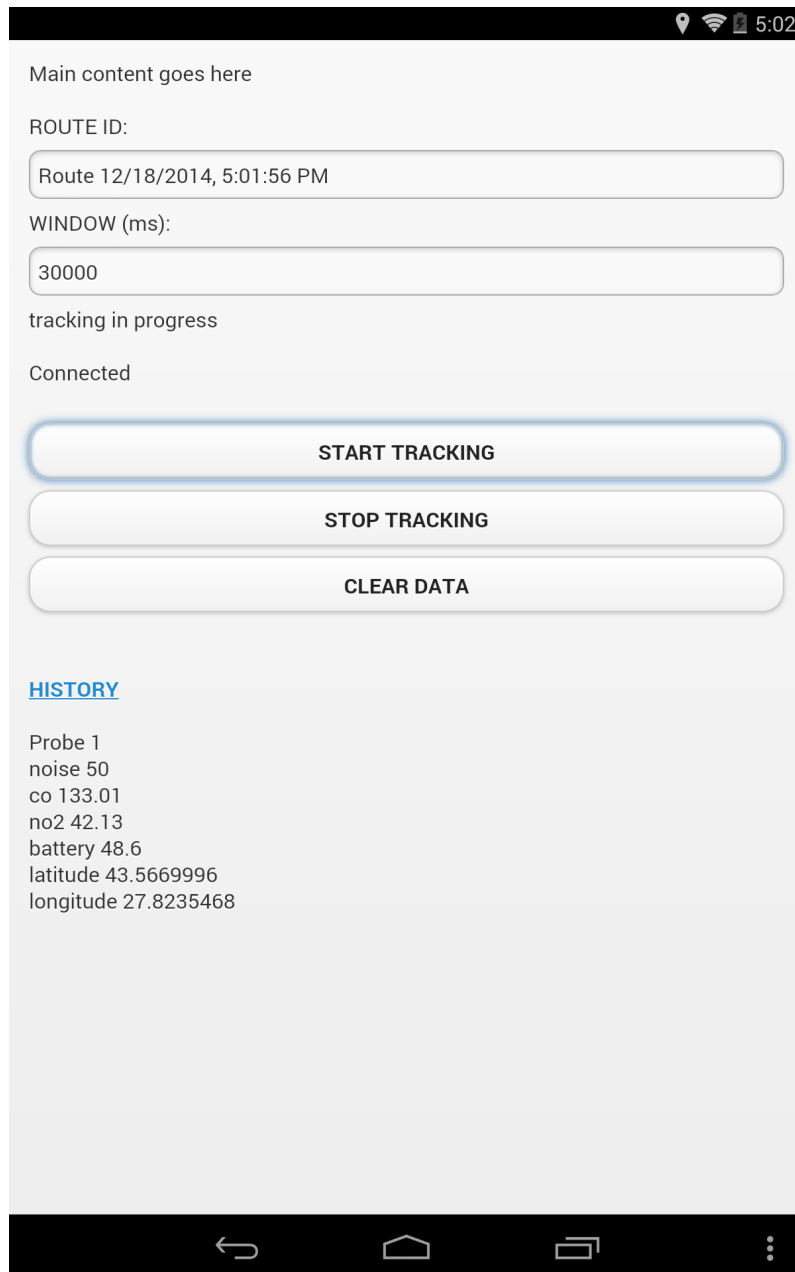
Problems

Some of the more significant issues encountered include:

- Arduino IDE does not work very well with the prototype version of the kit and this required significant research (contacted the FabLab team continuously) as well as trial-and-error; successfully resolved
- GPS location accuracy variance and irregular time updates; resolved by changing the location tracking approach
- “Eduroam” network SCK connection issues; resolved by using a mobile device as a portable access point and sharing its Wi-Fi connection resources with the kit

Appendix A

- Android Client:



- System Server:



A.3.12 Progress Report 11

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Semester 2 Plan

The second semester will be split into a few main periods facilitating final project presentation (expected completion dates have been listed against each component):

- **implementation finalisation:** *completion: around 15 February-March*
 - **functionality**
 - **testing**
- **further heuristic evaluation:** *completion: mid-February-March*
- **user testing + system improvements:** *completion: mid-February-around the start of March (till mid-March)*
- **final report preparation + refinement:***completion: mid-February-March to allow plenty of time for execution*

Implementation Layout (Incl. Heuristic Evaluation)

Below is a more detailed execution plan for the implementation component. It will include several agile sprints to allow for performance improvements through retrospective sessions. The incremental development process will build on the existing client-server framework which provides a base (prototype) enabling more complex functionality integration. The main layout consists of the following roughly described steps:

- client UI improvements and configuration
- map configuration options + routes
- unit testing
- refactoring
- deployment
- heuristic evaluation

Petar Yordanov, 15.01.2015

A.3.13 Progress Report 12

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- Mongo DB server data exported for backup and local development: [link](#)
- JSP structure refactoring: [link](#)
 - split main views into sub-components to create a template structure
 - clearly labelled utility files ('data.jsp' and 'parse.jsp') used for raw data processing
- jQuery & Bootstrap issue – discovered a bug on the UI and some jQuery UI library files missing, now restored: [link](#)

Work Under Way

- Map Configuration : [link](#)
- Performance Assessment: [link](#)

Known Issues + Resolutions

- Mobile device screens UI misbehaviour – the server menu was not collapsing properly; resolved by re-arranging included “js” script files (a conflict between Bootstrap and Google Maps): [link](#)

*I will be presenting you this development iteration next week/week after

Petar Yordanov, 20.01.2015

A.3.14 Progress Report 13

Dynamic Noise and Pollution Campus Map

/progress report/

Work Done Previous Week

- server UI: [link](#)
- sensor data formatting: [link](#)
- sensor data scaling: [link](#)
- map configuration: [link](#)

Work Under Way

- map configuration: [link](#)

Known Issues + Resolutions

- server performance assessment: [link](#)

Petar Yordanov, 27.01.2015

A.3.15 Progress Report 14

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- contacted Stephen Brewster and discussed project evaluation; he granted me permission to conduct the sessions
- emailed the SCK team to ask for permission to use one of their BETA versioned RESTful endpoint: key/device.js; using it to generate user/device data on the server: [link](#)
- updated the client's UI and finalized Android implementation: [link](#)
- finalized map configuration and server implementation:
 - [link1](#)
 - [link2](#)
 - [link3](#)
 - [link4](#)
 - [link5](#)
 - [link6](#)
- logo design (files on the repository)
- project DEMO

Work Under Way

- unit testing of the key server operations
- evaluation preparation
- dissertation

Known Issues + Resolutions

- used java.sql.timestamp to load the json into models before returning to the client javaScript code; conflicts arised as Mongo DB's BSON timestamp does not have a converter/ mapper from the Java data type; resolved by using the Date() object from java.util.date; now additional data such as timestamps can be used for filtering

Petar Yordanov, 04.02.2015

A.3.16 Progress Report 15

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- updated the user interface to show route recommendations more comprehensively
- added user options for route ranking based on different statistics (duration, distance, noise, CO, NO2)
- data collection sessions
- final interface interactions: [link](#) (showcasing all report points)
- unit testing completed (run 'mvn test' to execute tests): [link](#)

Work Under Way

- evaluation setup & planning

Known Issues + Resolutions

- had deprecated Spring MVC functions issues, was a very tricky issue that took a while to resolve; the solution involved updating the framework version to make it compatible with all of the installed dependencies (dependency data included to repository)

Petar Yordanov, 10.02.2015

A.3.17 Progress Report 16

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- planned evaluation
- modified client to enable using multiple SCK devices: [link](#)
- prepared evaluation documentation: [link](#)
- full database backup: [link](#)

Work Under Way

- dissertation + pilot evaluation execution

Known Issues + Resolutions

- fixed a bug on the server when updating user and device information

Project Files

- <http://ugmap.me/>
- [Android Route Tracking Client](#) (.apk updated)

Petar Yordanov, 17.02.2015

A.3.18 Progress Report 17

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- submitted evaluation application documentation
- final report LaTeX template setup
- report structural skeleton
- report draft 'Introduction' chapter completed:
 - [document link](#)

Work Under Way

- revise 'Introduction'
- final report 'Context' and 'Implementation' chapter

Note: The document link provided above is a draft version. It will be revisited and further changes are pending. The Google Doc is just a tool for easy sharing of the textual representation of the report. This will enable efficient change introduction upon receipt of feedback. All of this information will be transferred to the final LaTeX template with all the corresponding images/appendices and code listings to generate the final .pdf version.

Petar Yordanov, 25.02.2015

A.3.19 Progress Report 18

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- report structural skeleton updated
- Chapter 'Related Work'
- Chapter 'Planning'
- Chapter 'Design'
- report draft 'Introduction' chapter completed:
 - [document link](#)

Work Under Way

- revise chapters
- final report 'Context' and 'Implementation' chapter

Note: The LaTeX document is rendered in parallel. Once it has more strict formatting, I will share it as well.

Petar Yordanov, 02.03.2015

A.3.20 Progress Report 19

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- pilot evaluation has been completed (used [Nasa TLX](#) forms and a questionnaire as well as user monitoring to track relevant data); all participants collected environmental data ; obtained useful feedback, new features have been implemented on the web application server : [link](#) (important)
 - a bug with marker rendering on the map has been resolved
 - automatically populate the starting point with the user's current location (or building on campus that is closest to the current location)
 - automatically populate lat/lon fields on location selection
 - click anywhere on the map to select the closest building in the database as a destination point, without having to type it in
 - user navigator to show the current location of the user and display a pop-up (for instance "You have reached your destination") when they reach their desired location
 - timestamp filters have been improved
 - added start/stop tracking functionality under 'Locations'
 - added relevant instructions to the server specifying how to interact with the UI and included image tables to show safe levels for noise, CO and NO2
 - added download links for the client application (Android and Windows Phone) as well as completed 'SCK' and 'About' pages

- report draft revision:
 - [document link](#)

Work Under Way

- revise chapters
- final report 'Implementation' and 'Evaluation' chapter

Note: The LaTeX document is rendered in parallel. Once it has more strict formatting, I will share it as well. The final evaluation should be commencing shortly.

Petar Yordanov, 10.03.2015

A.3.21 Progress Report 20

Dynamic Noise and Pollution Campus Map

/progress report & planning/

Work Done Previous Week

- final evaluation has been completed; all participants collected environmental data ; it was very rewarding to see that people’s satisfaction has certainly increased after the implementation of suggested features during the pilot evaluation process (since most participants were the same in both sessions); this can be seen on the forms that all participants have filled out and will become particularly visible when the data is plotted on graphs in the final report; the answers of the open questions involved praising comments (which made me feel even more satisfied with the final result)
- ‘Implementation’ and ‘Evaluation’ draft has almost been completed
- the most up-to-date report version has been transferred to the LaTeX document and will be made available to you for review by next week

Work Under Way

- the demonstration video is currently being prepared; it will be made available to you and the project reader Dr. Alessandro Vinciarelli by the end of this week (I would like to refine it)
- complete ‘Implementation’, ‘Testing’ and ‘Evaluation’ chapters in the dissertation
- revise the final report according to your feedback
- project presentation (at 9:50 on 27.03.15 in SAWB 422) preparation:

LEVEL 4 PROJECT PRESENTATION					
Friday 27/3/2015					
SAWB 422					
Time	Name	Title	Sup	Reader	Chair
9: 50	Yordanov, Petar	Building a dynamic pollution and noise street map using the Smart Citizen Kit	Ounis,I	Vinciarelli,A	Ounis,I

Petar Yordanov, 18.03.2015

A.3.22 Progress Report 21

Dynamic Noise and Pollution Campus Map

/final progress report & planning/

Work Done Previous Week

- Final report completed

Work Under Way

- Final report spell and grammar checking, editing and formatting
- Presentation preparation

Petar Yordanov, 25.03.2015

A.4 Evaluation Documentation

Participant evaluation pack:

A.4.1 Consent Form

Participant Consent Form: Dynamic Noise and Pollution Campus Map

The aim of this experiment is to evaluate the system that has been produced to collect and visualize environmental data using the Smart Citizen Kit sensor board.

The experiment will take about an hour to complete.

While the experiment is being carried out, some data will be collected. This will include:

- Participant's status: student/ not a student
- Mobile device possession: positive/ negative
- Time taken completion of each task
- Total evaluation duration
- Destination choices
- Route Choices
- Questionnaire answers and suggestions for improvement

All results will be held in strict confidence, ensuring the privacy of all participants. No personal information will be stored with the data.

Please note that it is the system, not you, that are being evaluated. You may withdraw from the experiment at any time. Any data that has been collected until the point of withdrawal will be discarded and not analyzed as a part of the summative evaluation.

If you have any further questions regarding this experiment, please contact:

Petar Yordanov
School of Computing Science,
University of Glasgow
1103620y@student.gla.ac.uk

I have read this information sheet, and agree to voluntarily take part in this experiment:

Name: _____ Email: _____
Signature: _____ Date: _____

This study adheres to the BPS ethical guidelines, and has been approved by the DCS ethics committee of The University of Glasgow.. Whilst you are free to discuss your participation in this study with the experimenter, if you would like to speak to someone not involved in the study, you may contact the chair of the DCS Ethics Committee: Prof Stephen Brewster, stephen.brewster@glasgow.ac.uk

Introduction Script

Dynamic Noise and Pollution Campus Map Evaluation

Petar Yordanov

School of Computing Science,

University of Glasgow

16.02.2015

Introduction

The general aim of this experiment is to evaluate the usability and applicability of the route generation software that was produced as a part of the project. The involvement of other people is a necessity due to the fact that potential users of the application are an invaluable source of feedback when determining the product's performance.

The experiment will involve walking around campus and making route choices based on the feedback received from the mobile application. There will be 3 destinations that you would need to choose and go to. You will also be asked to produce custom visualizations via the system's GUI. Tasks in this category will involve using the visualizations the UI enables you to generate in order to determine the lowest and highest values for noise and air pollution on campus and also understand how the locations change for each variable respectively (noise, CO, NO2), are there correlations, as well as try to find out how accurate the map is in different areas. You will also be testing the route recommendation functionality, choosing a destination on campus and walking there. While you are walking you will be carrying a sensor and collecting environmental information using the software product.

While the experiment is being carried out, some data will be collected. This will include:

- Participant's status: student/ not a student
- Mobile device possession: positive/ negative
- Time taken completion of each task
- Total evaluation duration
- Destination choices
- Route Choices
- Suggestions for improvement

During the course of the experiment, I will be walking next to you in case you have any questions. It is important to note that this experiment does not test your ability and/ or performance, it has been designed with the sole purpose of software evaluation.

You are welcome to withdraw from this evaluation session at any time.

If you have any questions prior to the start of the experiment please ask. If you agree to taking part in this evaluation session, please sign the consent sheet.

Task Sheet

Dynamic Noise and Pollution Campus Map Evaluation

Petar Yordanov

School of Computing Science,

University of Glasgow

16.02.2015

Questions (part 1)

1. Are you a student?
2. If so, are you studying at the University of Glasgow?
3. How familiar are you with the university campus (on a scale of 1-5)?
4. Do you possess a mobile device (or multiple devices)?
5. If so, what brand/ brands?
6. Do you download mobile applications from the application catalogue that track your location?

Tasks

1. Explore the data points and routes rendered on the map.
2. Using the user interface of the application, try to identify the where the highest values for noise, CO and NO2 are on the map.
3. Using the custom visualizations, plot multiple variables on the map at the same time.
4. Using one of the visualization modes, identify the areas where the map is most accurate – popular evaluation areas in other words.
5. Familiarize yourself with user interface of the Android application (once you think you are ready, I will ask you some questions and give you a briefing).
6. Start the environmental collection process (using the Android client).
7. Find route recommendations to three buildings of your choice and walk to them, while carrying the sensor and collecting data.
8. Please fill in the Nasa TLX form I will provide you with (rating mental, physical and temporal demand as well as performance, effort and frustration).

Questions (part 2)

7. How would you rate the application on an overall basis (on a scale of 1-10)?
8. Would you use this application if it becomes available for free on the application web catalogue for your device?
9. Is there anything in particular that you like/ dislike about this application? Please comment.
10. How can this application be improved in your opinion?

Debrief Script

Dynamic Noise and Pollution Campus Map Evaluation

Petar Yordanov

School of Computing Science,

University of Glasgow

16.02.2015

Participant Debrief

Thank you for agreeing to take part in this evaluation session.

The general aim of this experiment is to evaluate the usability and applicability of the route generation software that was produced as a part of the project. The involvement of other people is a necessity due to the fact that potential users of the application are an invaluable source of feedback when determining the product's performance.

My contact details are as follows. You are encouraged to note them down in case you have further questions and/ or suggestions:

Email: 1103620y@student.gla.ac.uk

Project Supervisors: **Dr. Iadh Ounis, Mr. Richard McCreddie**

If you have any questions, please feel free to contact me or the project's supervisors using the contact information provided above.

Thank you again for your participation!

A.5 Requirements Gathering Session

A requirement gathering session was conducted to discuss important aspects of the system. This was an informal requirements gathering interview to propose system implementation features and discuss system design direction opinions.

1. Should this be a client-server based application or can it be a single, standalone mobile web application using local data storage? How flexible is the framework choice?

- Any architectural hierarchy would suffice, as long as it's choice has been justified. The framework choice is flexible.

2. Is data plotting flexible and if not, what are some guiding rules?

- plotting noise and pollution on the same map /overlapping/
- representation style colour-coded blocks, gradient map(heat map style), etc.
 - flexible visualization techniques
 - will be good to use multiple visualization styles/models

3. Is colour-coding enough to represent data or can there be added numerical values for more detailed interface?

- good idea to experiment with different visualisation arrangements in order to iteratively refine the final interface

4. Is route generation and manipulation a prime task or can be defined as consequential after the initial campus map generation?

- a COULD HAVE requirement which needs to be implemented after the basic campus map functionality is present

5. Is ranking the input entities feature that users can have voting impact on or it could be entirely dependent on device readings?

- can be incorporated as a WOULD (would be nice to have) feature
6. Would you consider a good idea presenting some wider statistical data to the user as a potential feature?

- yes, a COULD/ WOULD HAVE functionality that will improve the overall performance of the application

7. Currently the device transmits data once every 60 sec. (1 minute). Do you think decreasing this to get more accurate readings (considering the fact that the device is going to be moving) should be researched and is potentially viable for this project?

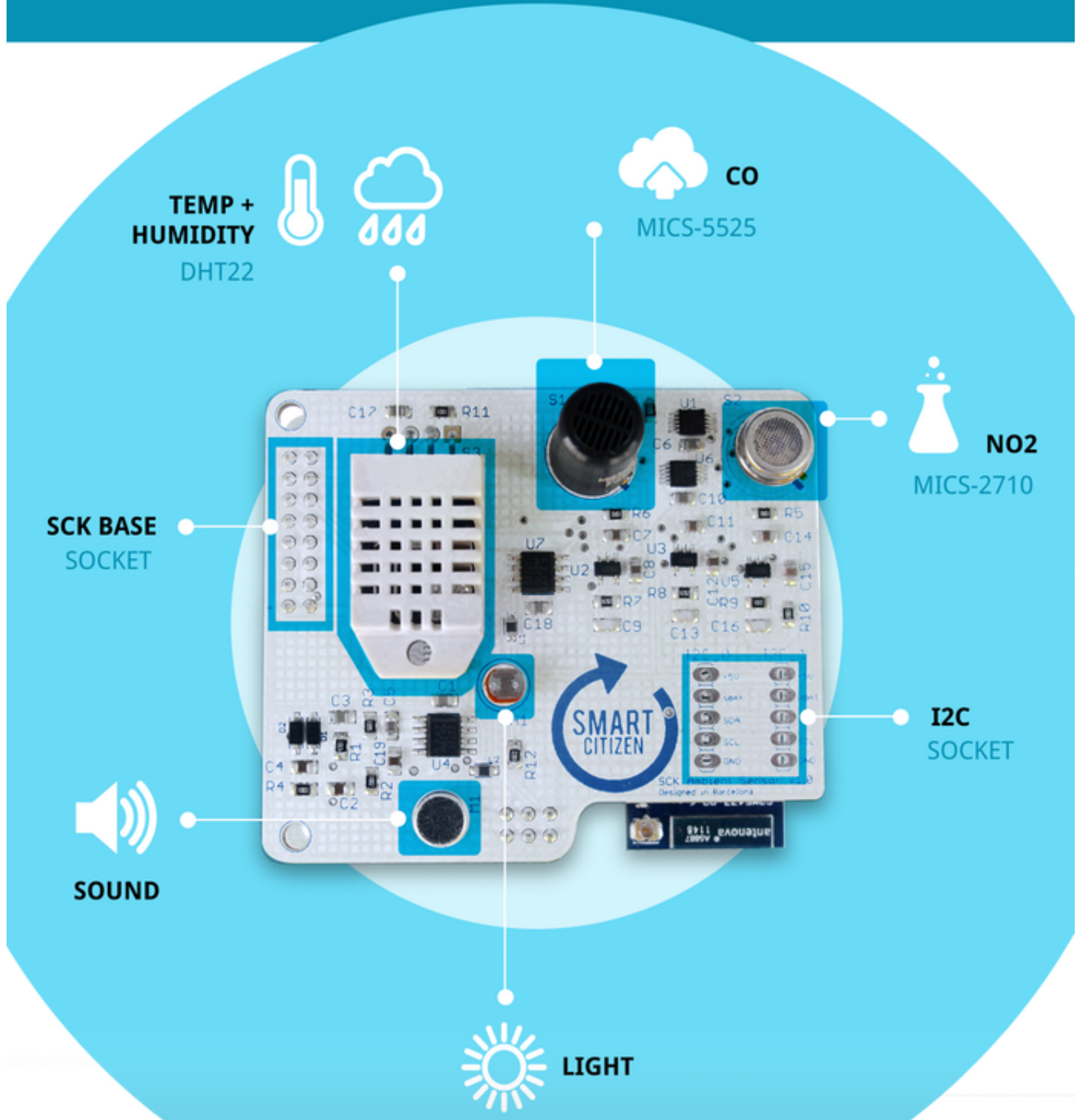
- if the firmware can be configured to do this, yes

8. The device has sensors for Carbon Monoxide and Nitrogen Dioxide. Is there any particular requirement about how the readings for those two toxic chemicals for humans should be plotted on the map (for example separately or as a single entity by adding up the values for instance; could incorporate different weights)?

- should be researched further
- add the values and if needed refine results by applying different weights depending on toxic effects

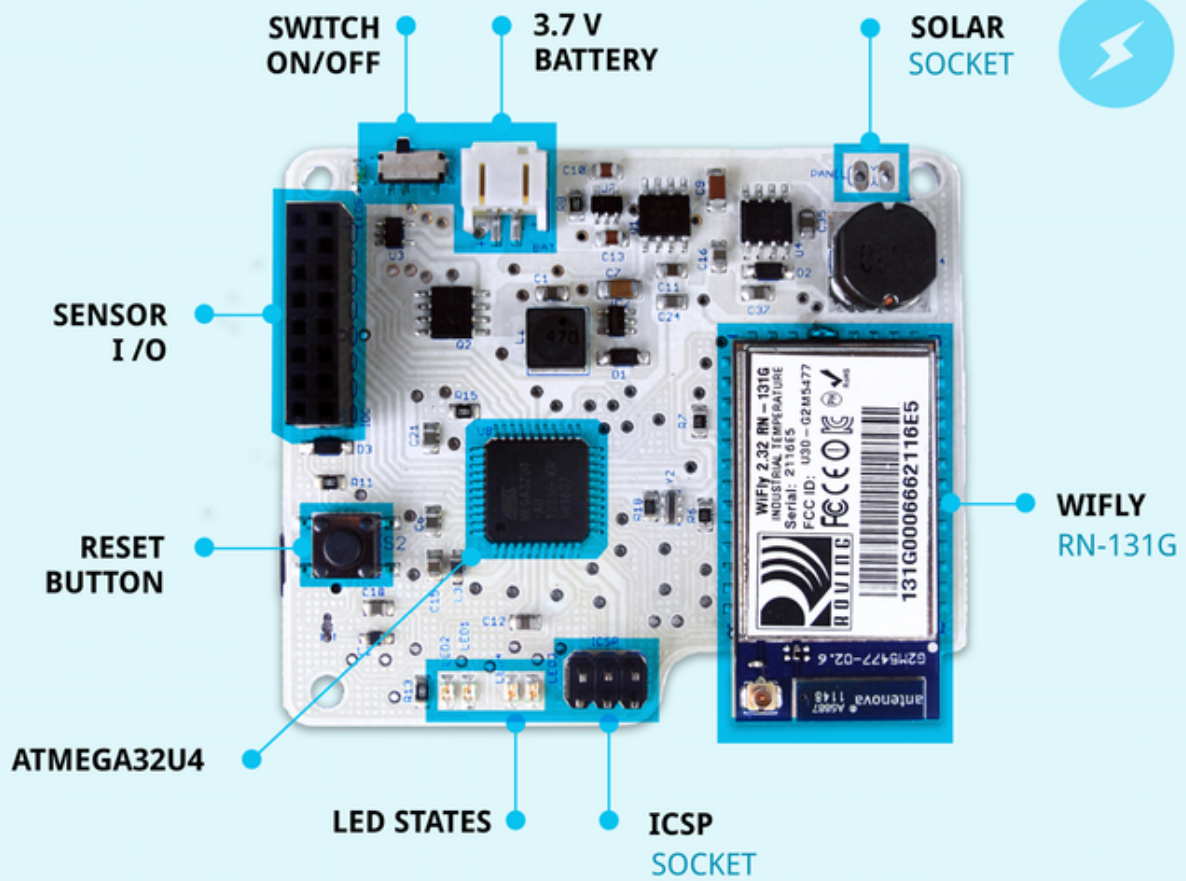
A.6 Images

AMBIENT BOARD (SHIELD)

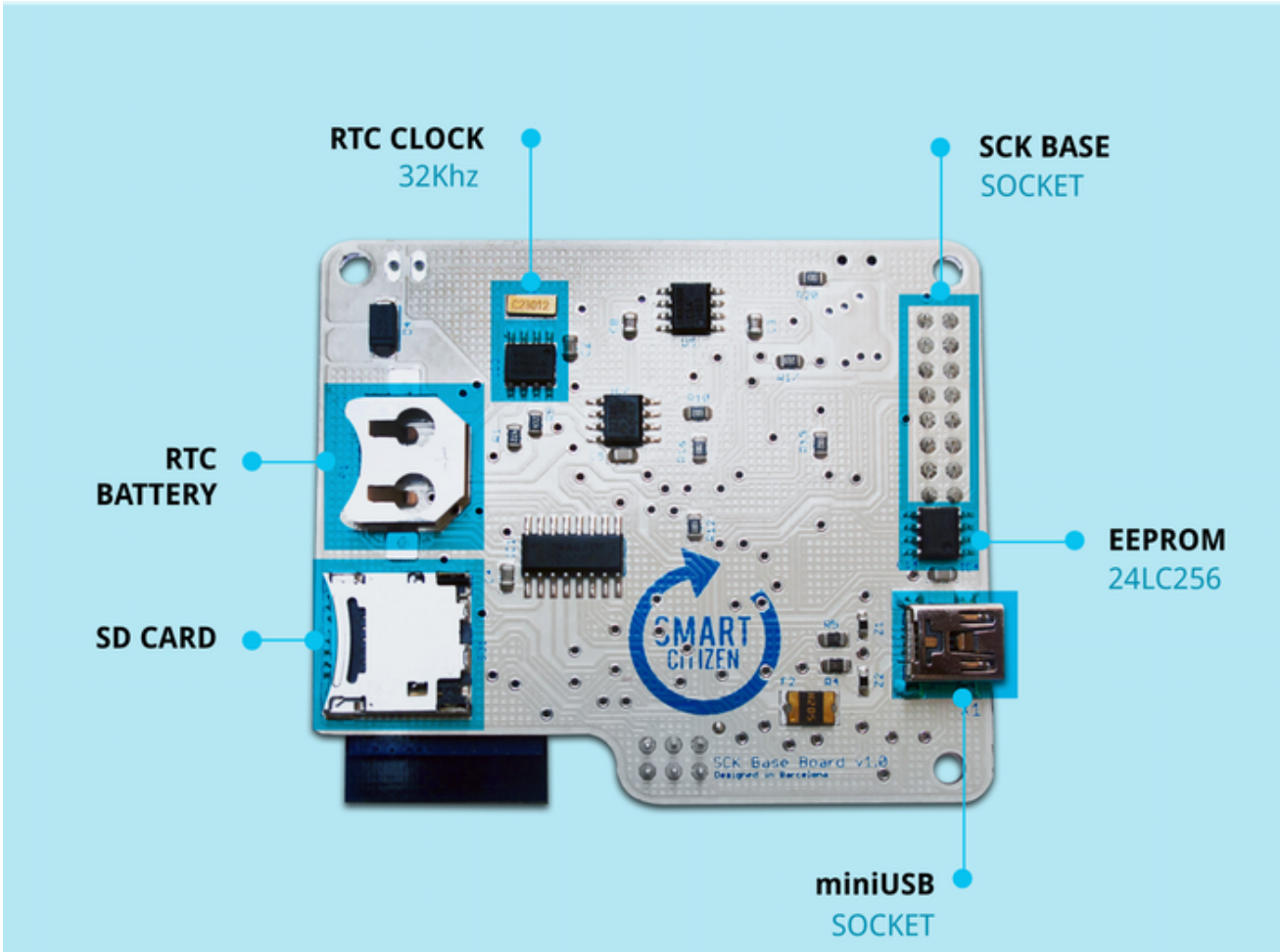


Thumbnail 1

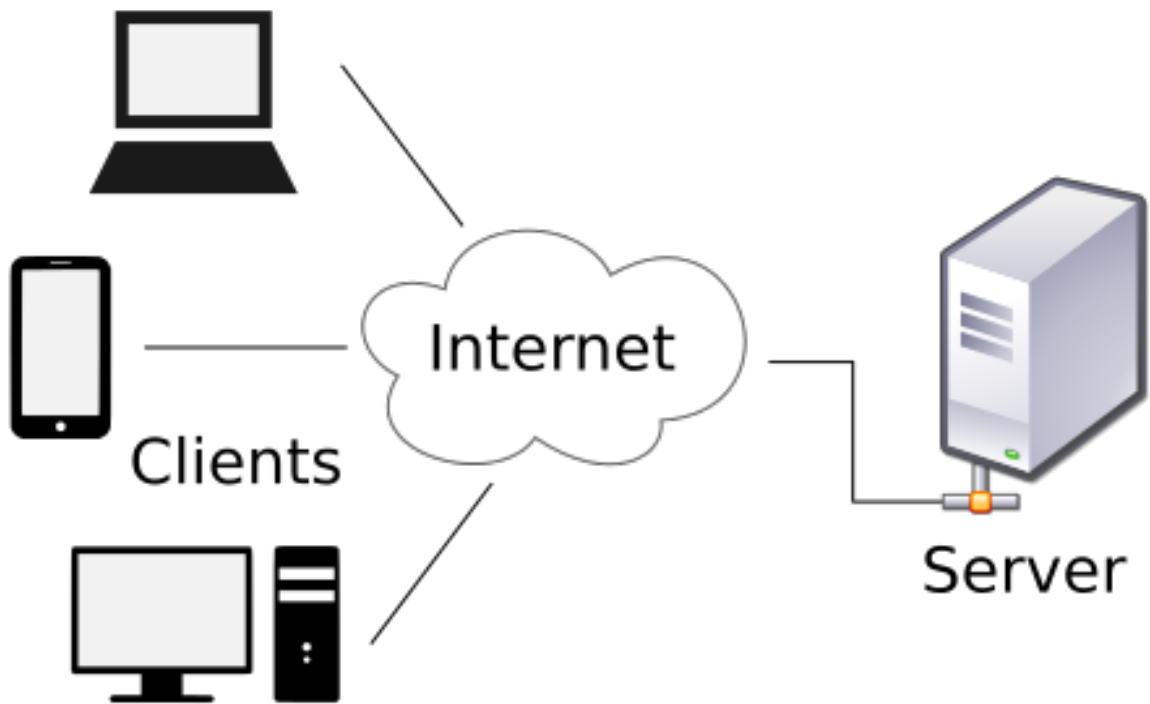
DATA-PROCESSING BOARD



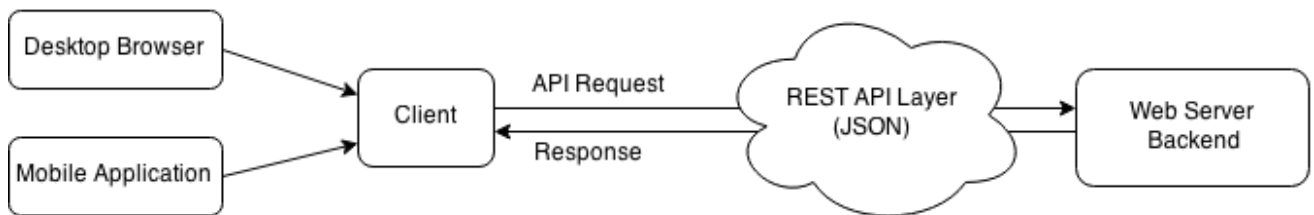
Thumbnail 2



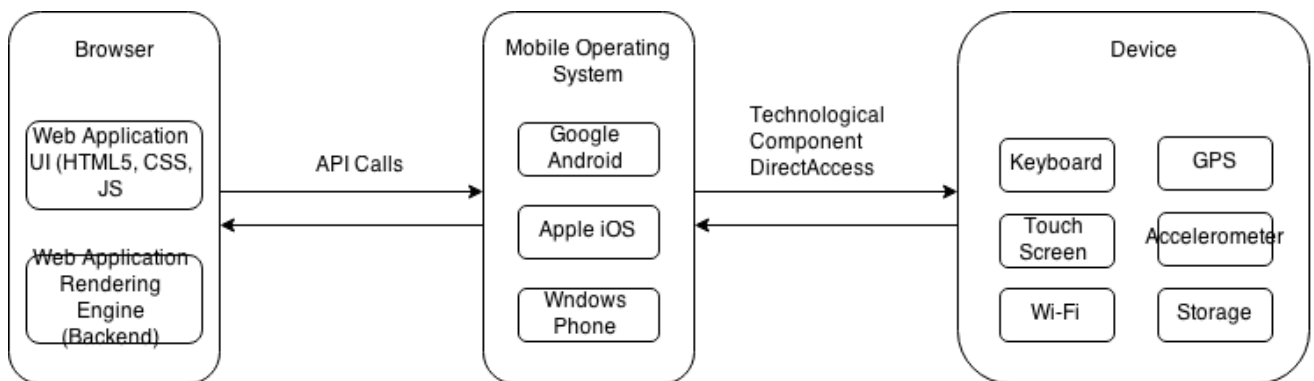
Thumbnail 3



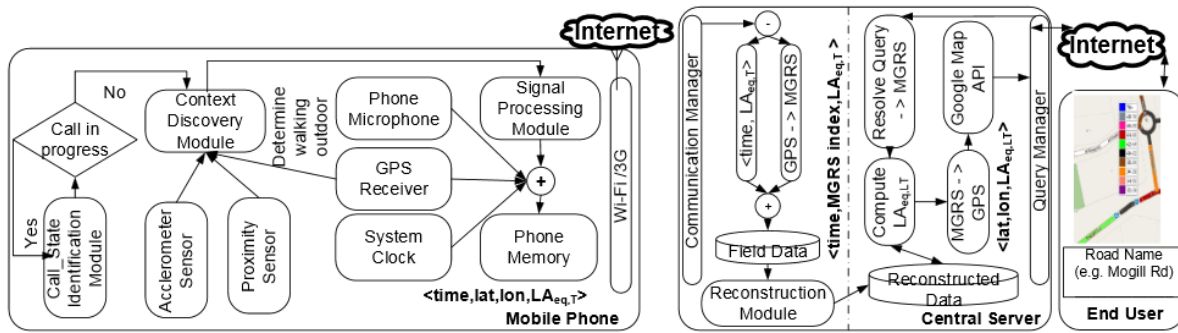
Thumbnail 4



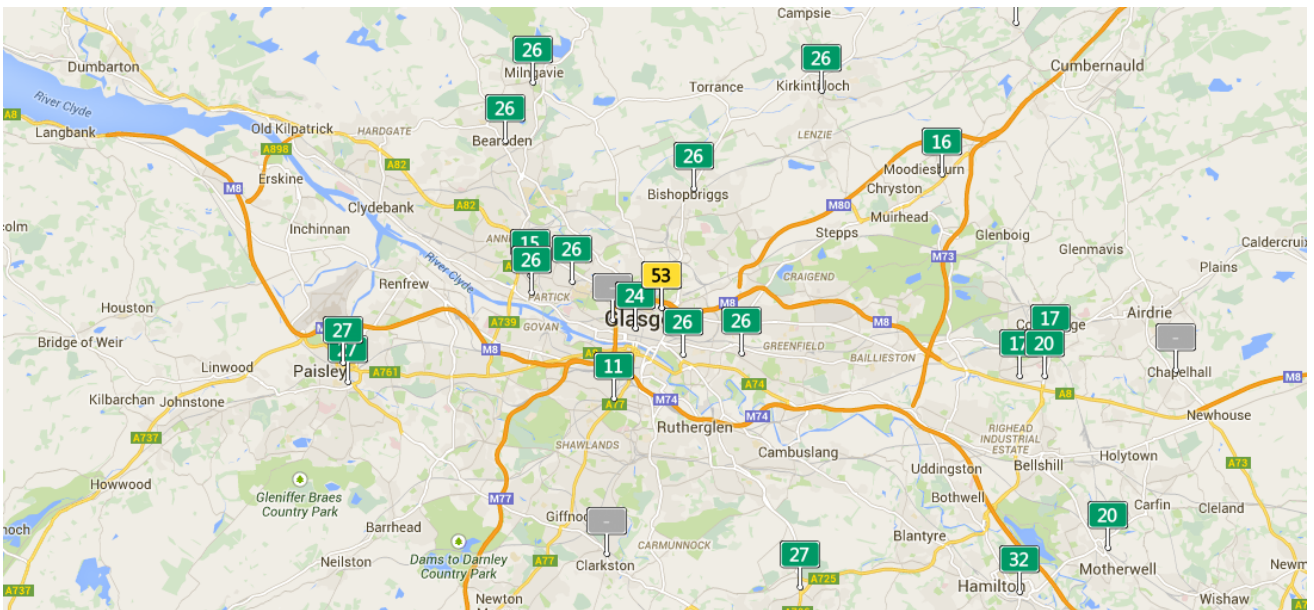
Thumbnail 5



Thumbnail 6

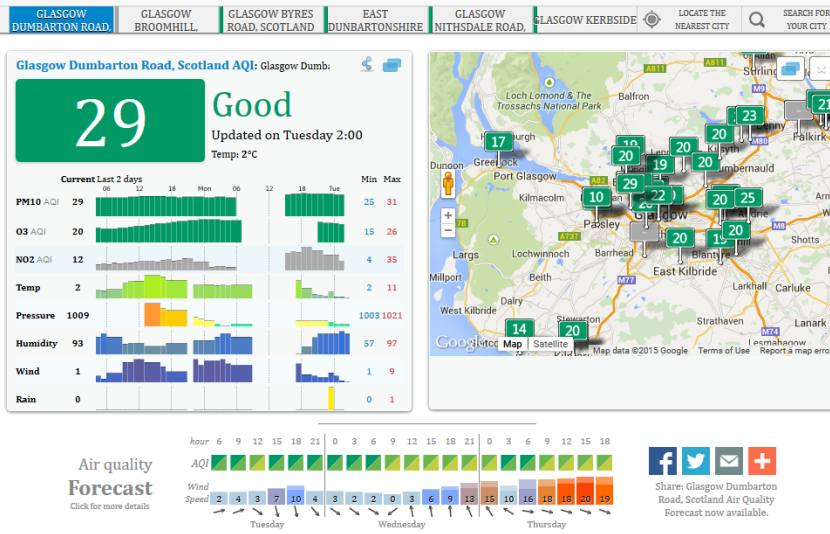


Thumbnail 7



Thumbnail 8

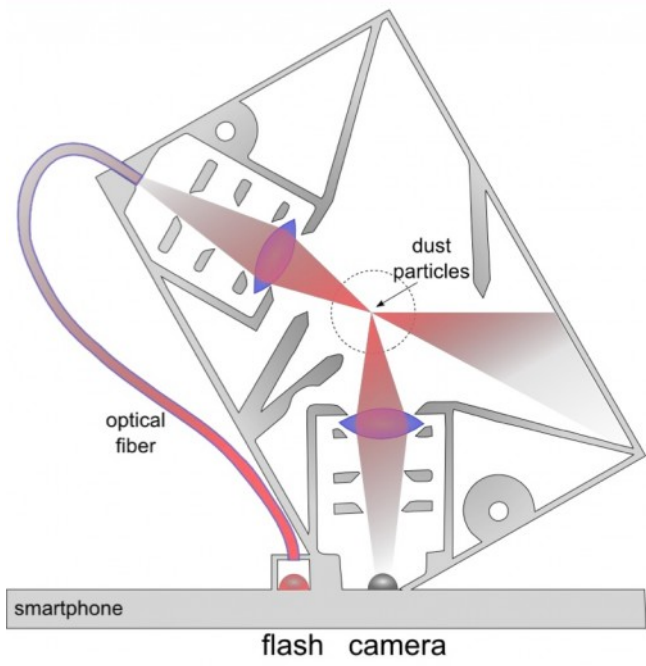
Glasgow Dumbarton Road, Scotland Air Pollution: Real-time Air Quality Index (AQI)



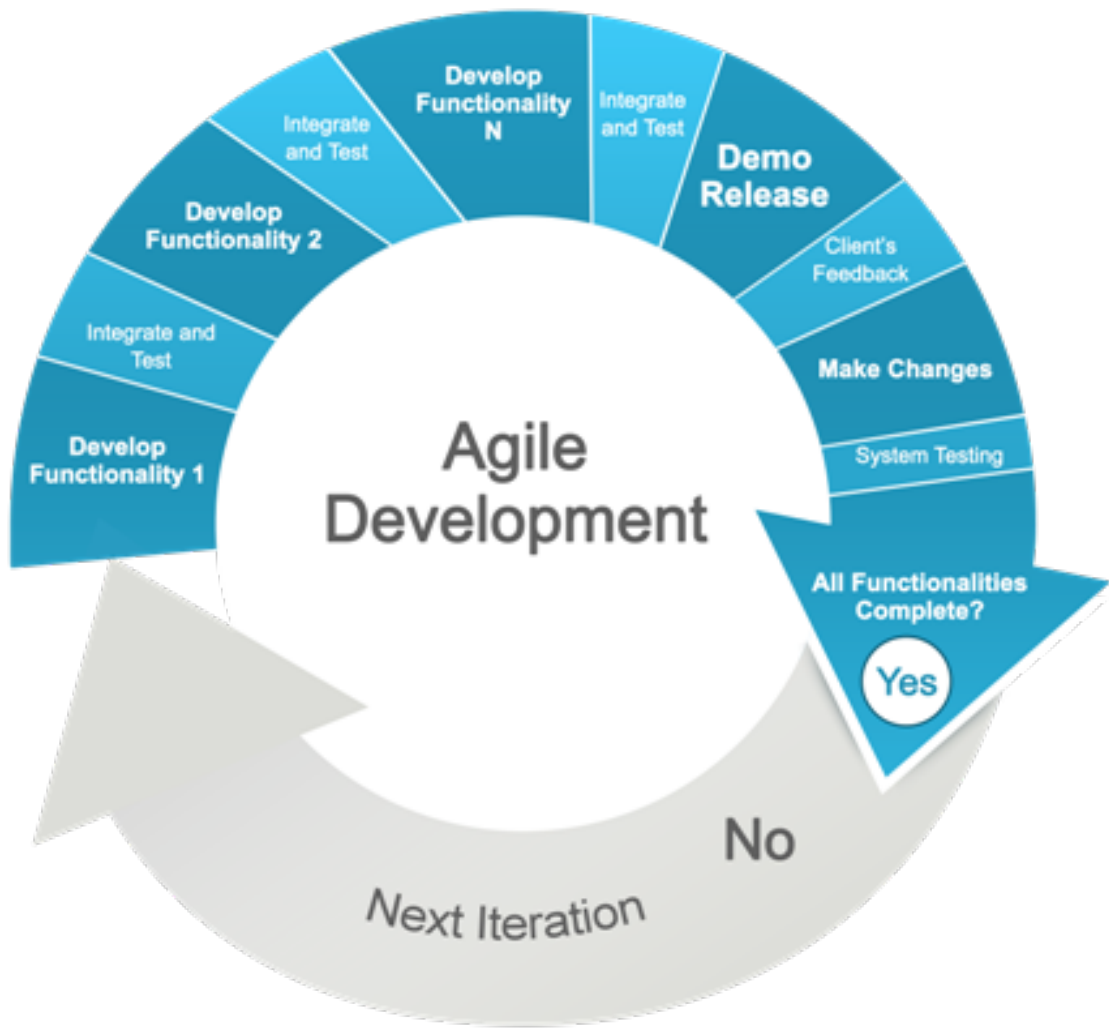
Thumbnail 9



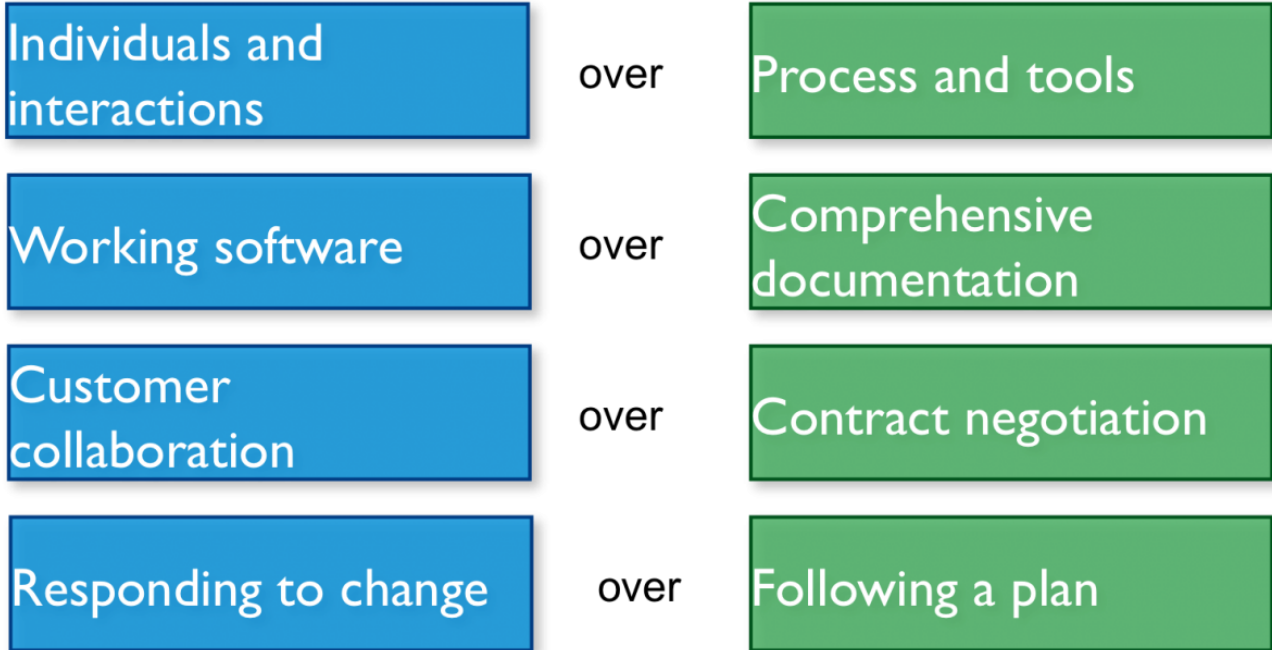
Thumbnail 10



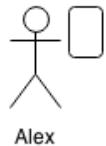
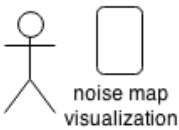
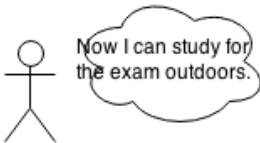
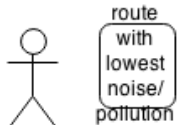
Thumbnail
11



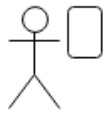
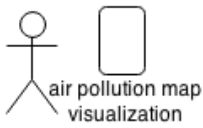
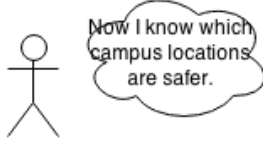
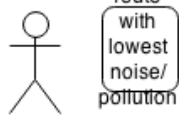
Thumbnail 12



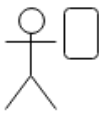

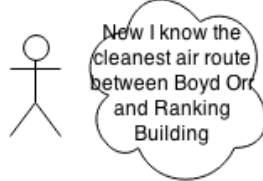

Thumbnail 13

User Case 1			
1. Alex has an exam and would like to revise	2. He installs the application on his mobile device and explores the map.	3. After identifying the quietest areas on campus, he can study outdoors without distractions	4. Alex can use the route recommendation functionality to get to the places he has chosen
 <p>Alex</p>	 <p>noise map visualization</p>	 <p>Now I can study for the exam outdoors.</p>	 <p>route with lowest noise/pollution</p>

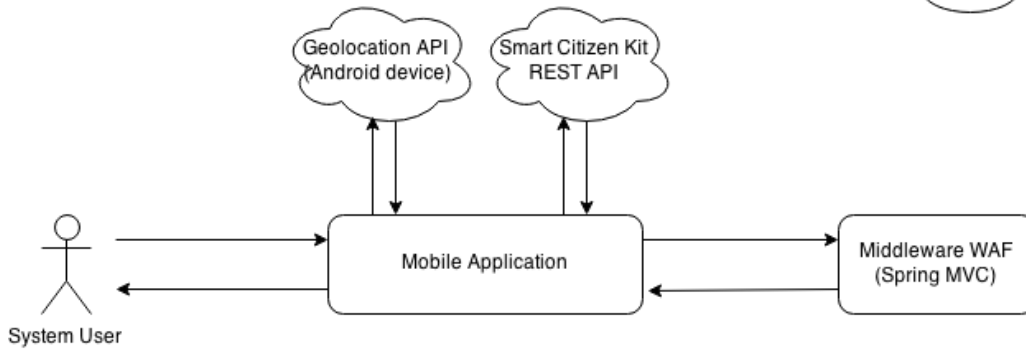
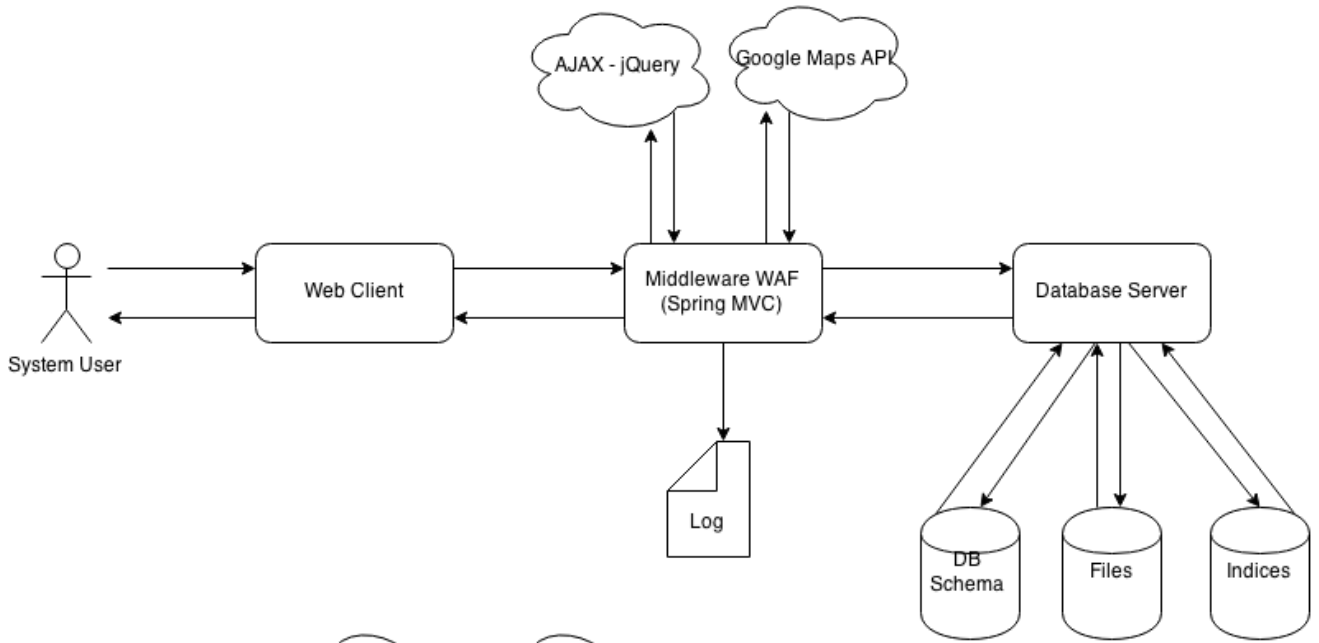
Thumbnail 14

User Case 2			
1. Maria has asthma and takes her mobile device to university every day.	2. She installs the application and explores the cleanest air map areas (lowest CO, NO2 lvl)	3. After identifying where air is cleanest, she knows what the safe locations are.	4. Alex can use the route recommendation functionality to get to the places he has chosen
 <p>Maria</p>	 <p>air pollution map visualization</p>	 <p>Now I know which campus locations are safer.</p>	 <p>route with lowest noise/pollution</p>

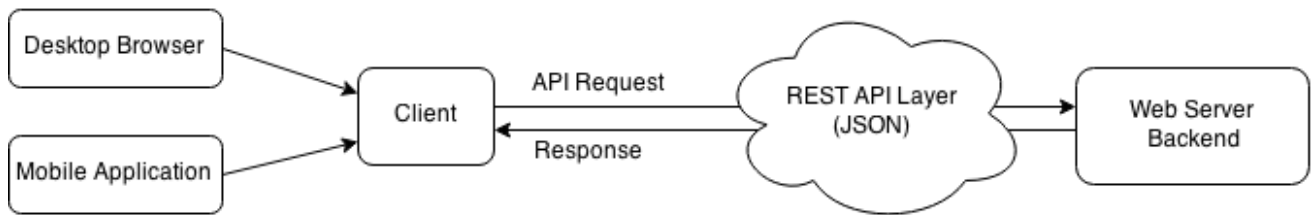
Thumbnail 15

User Case 3			
1. Ben would like to take a Saturday stroll on the premises of Glasgow University campus	2. Ben installs the DNPCM application and explores the map.	3. After searching for Rankine Bldg from Boyd Orr, Ben identifies the cleanest route.	4. Alex can use the route recommendation to walk to the building, relaxing and staying healthy at the same time.
 <p>Ben</p>	 <p>air pollution map visualization</p>	 <p>Now I know the cleanest air route between Boyd Orr and Rankine Building</p>	 <p>Feel relaxed and healthy now!</p>

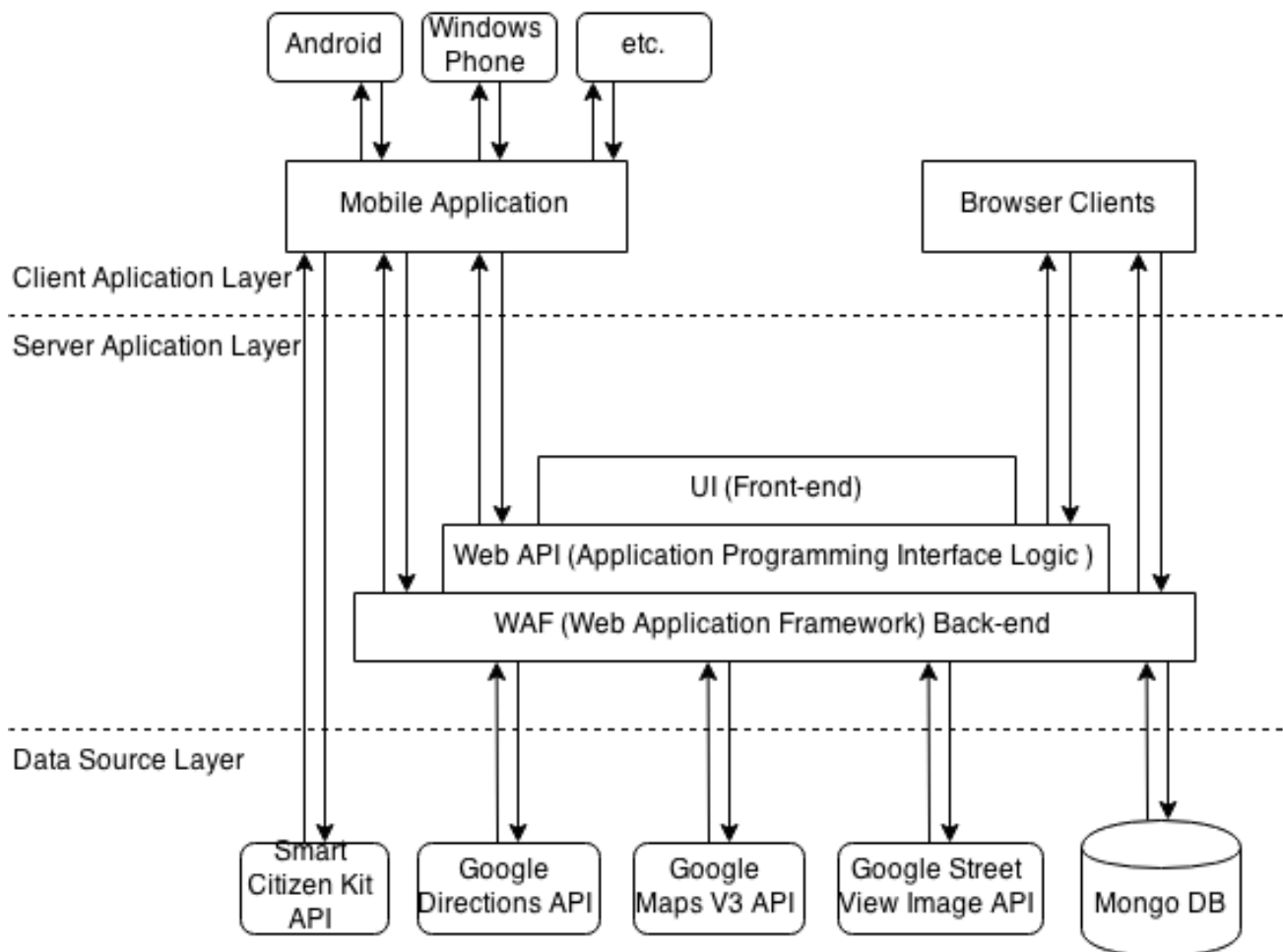
Thumbnail 16



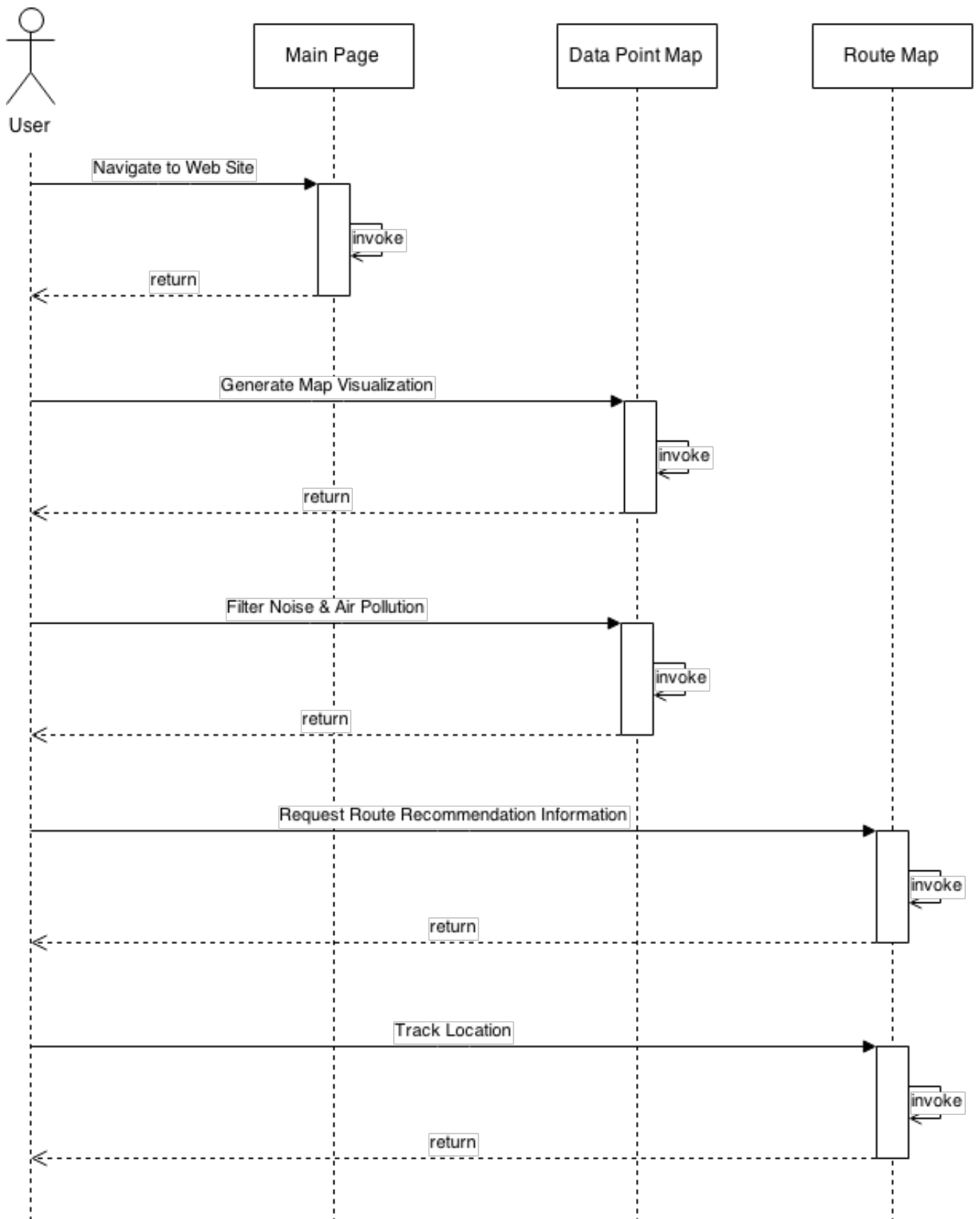
Thumbnail 17



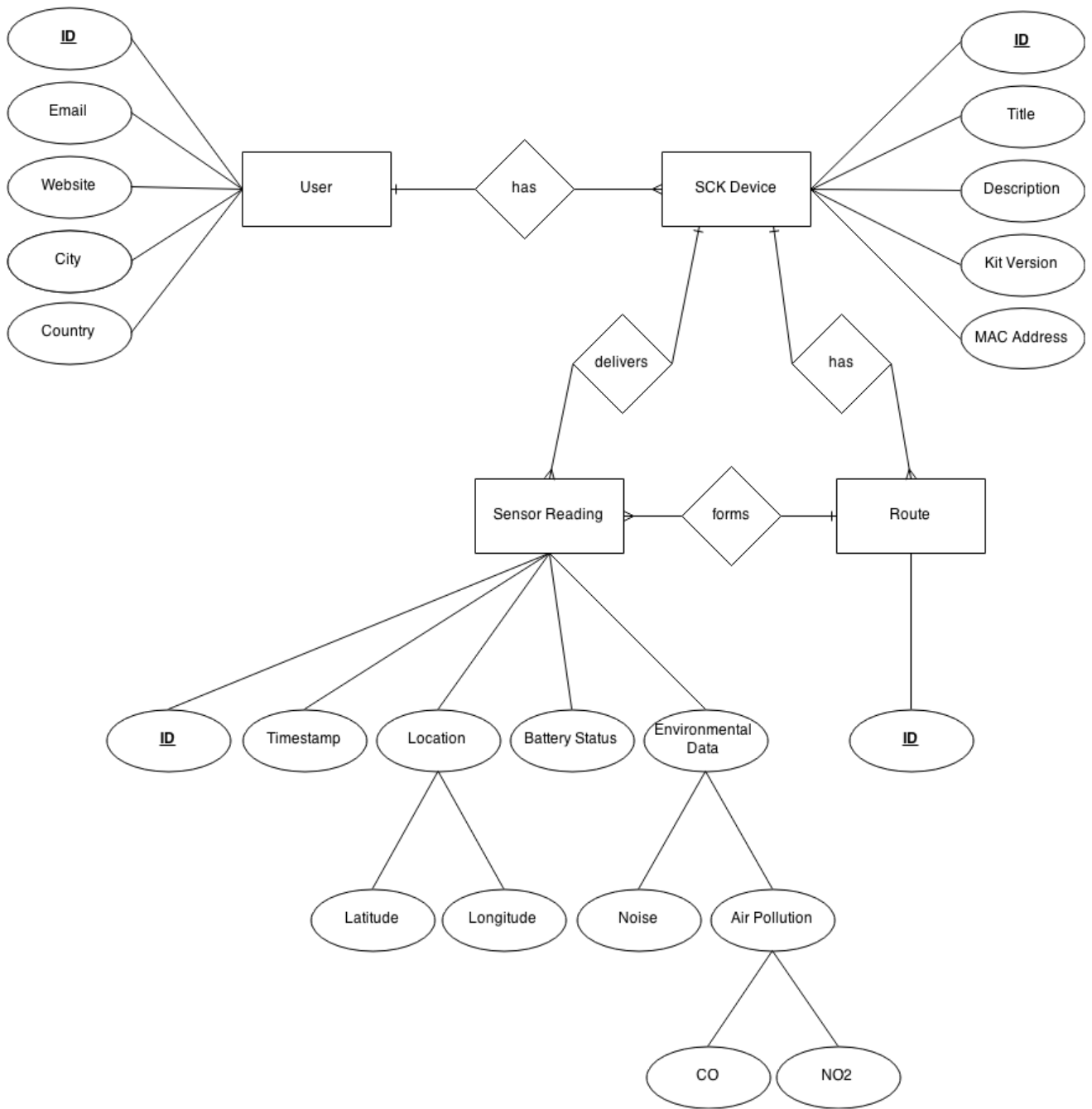
Thumbnail 18



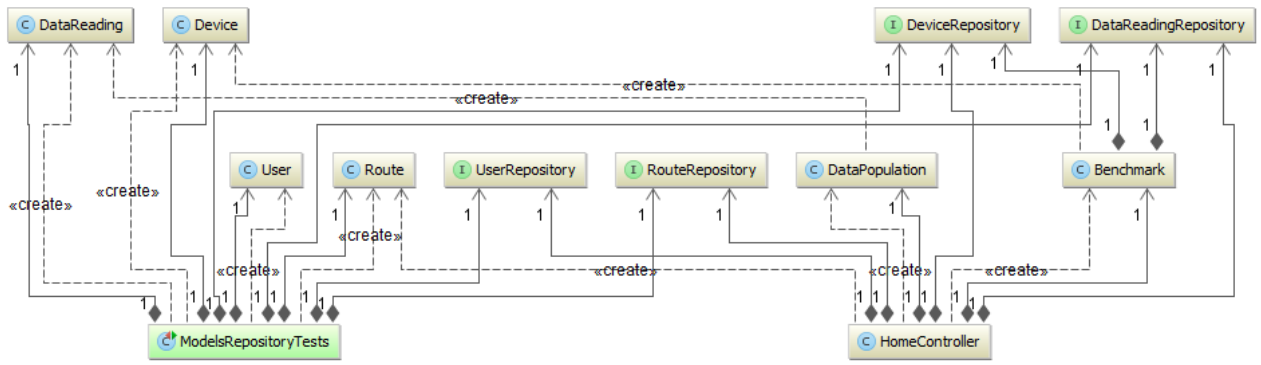
Thumbnail 19

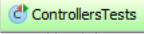
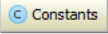


Thumbnail 20

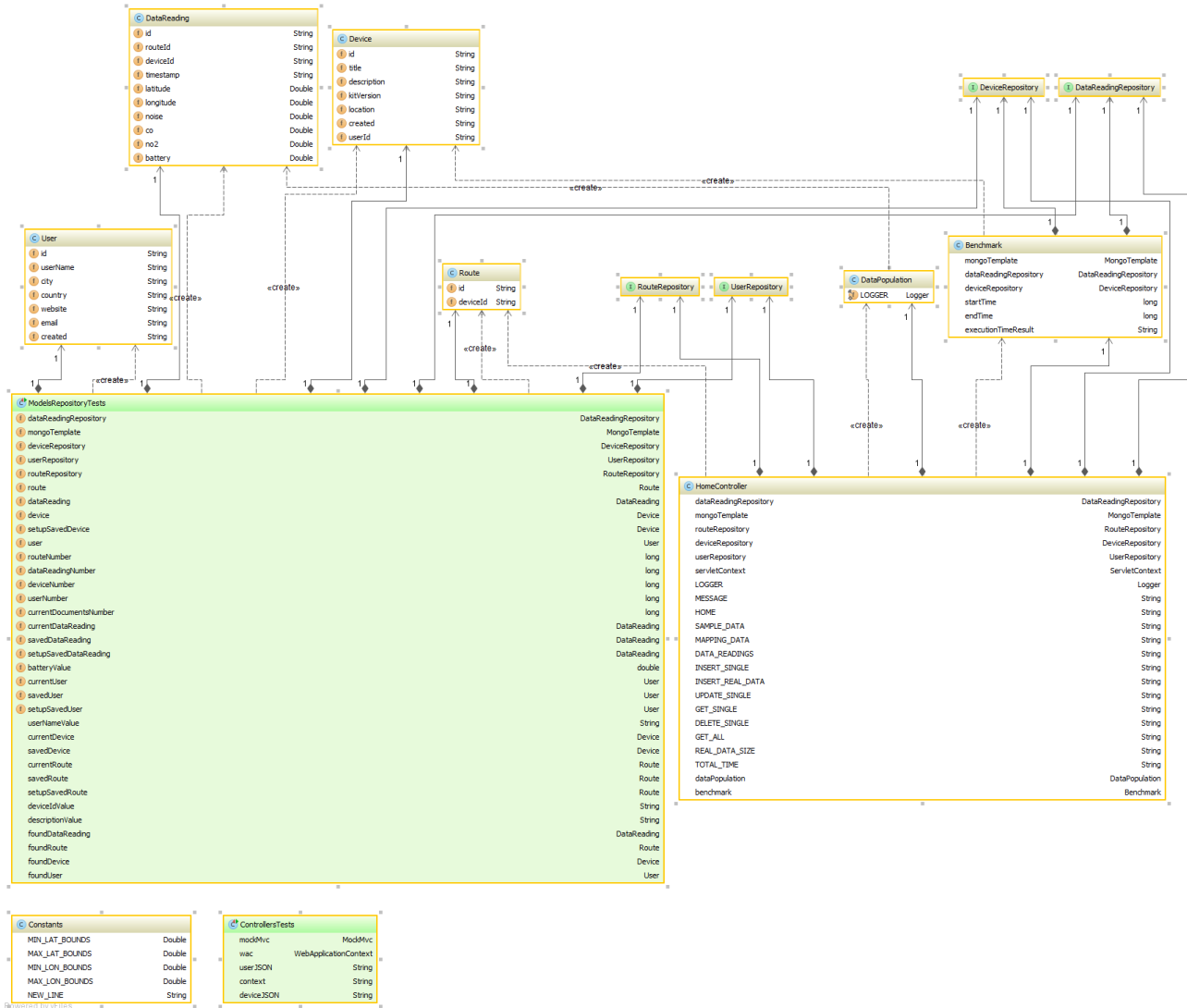


Thumbnail 21



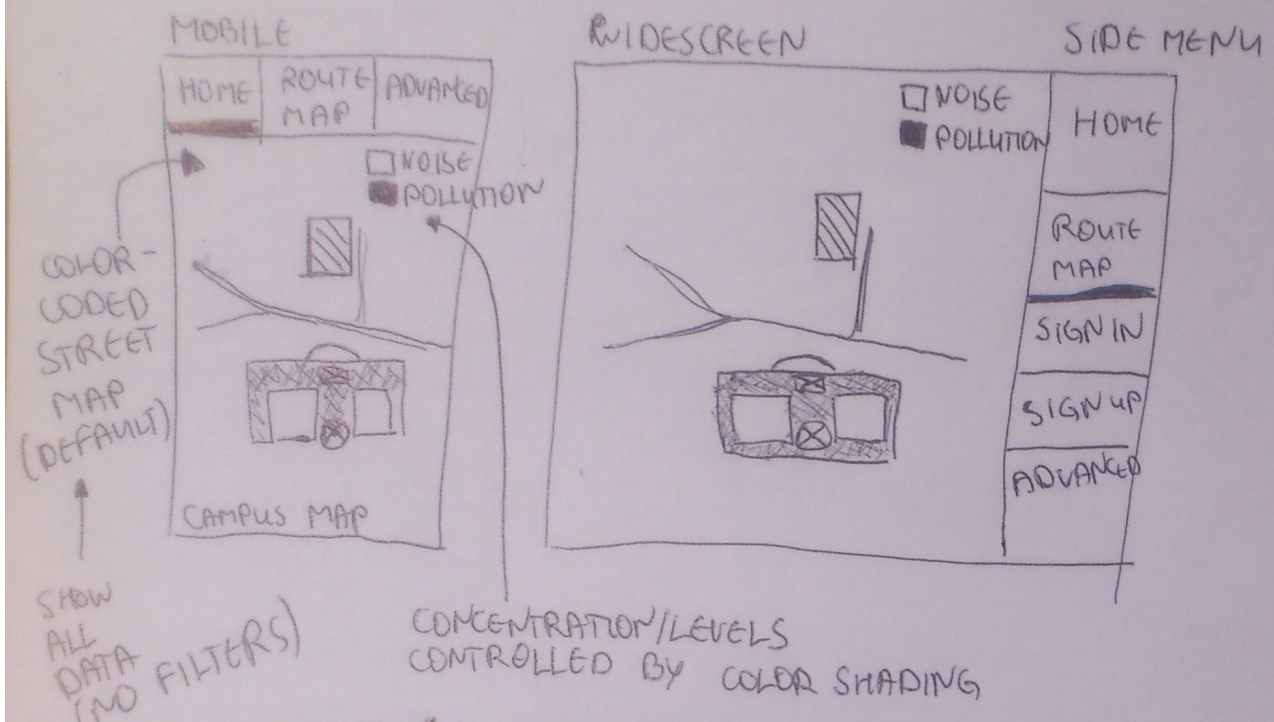


 Powered by yFiles

Thumbnail 22



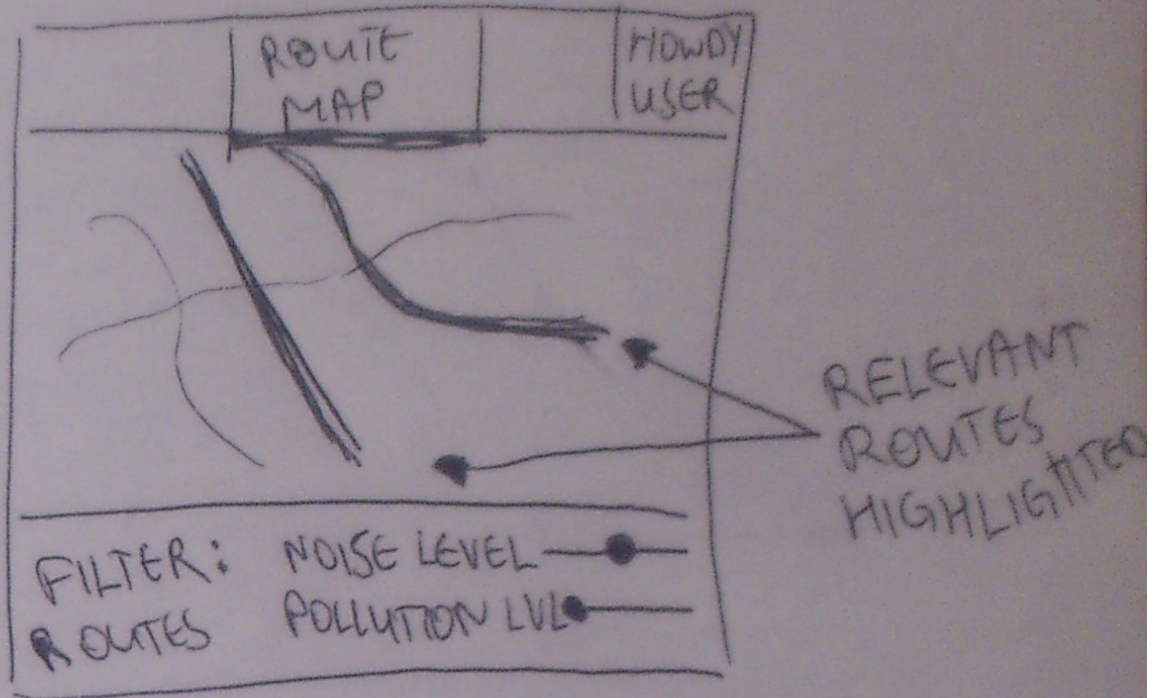
Thumbnail 23

Dynamic noise and pollution map application (prototypes)



Thumbnail 24

DIFFERENT MENU VIEW:



Thumbnail 25

map settings:

- grid ~~width~~ size (on/off) → settings
- style (map look)
- zoom
- pan
- routes
- sliders (• NO₂
• CO
• noise)

Styles

- classic
- dark blue
- grayscale
- clean
- labels
- } other options
- }

→ custom?

Configuration

- tile size
- routes display
- markers display
- grid frame

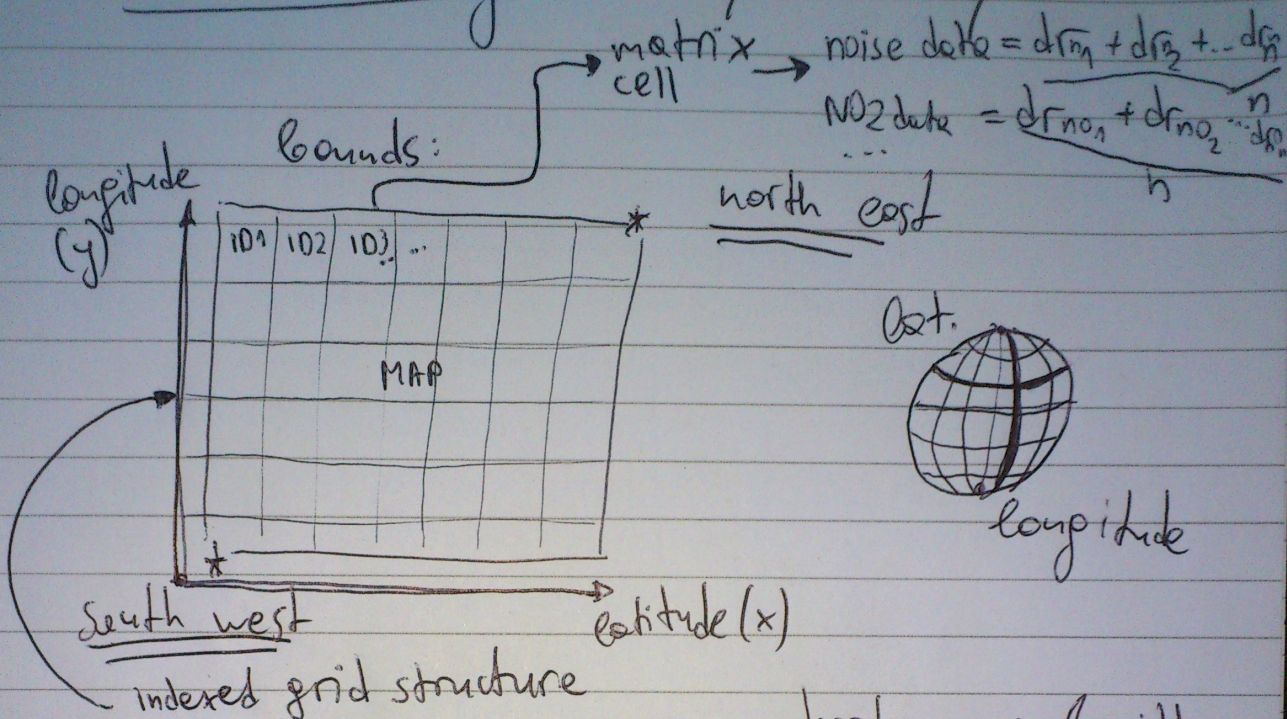
opacity + route thickness + color

↓ opacity

Filters

- Noise
- CO
- NO₂
-

visualization styles: /simulation



- heat-map algorithm
- hue/saturation (color-mapping)
 - opacity (2 distinct colors for each reading type)

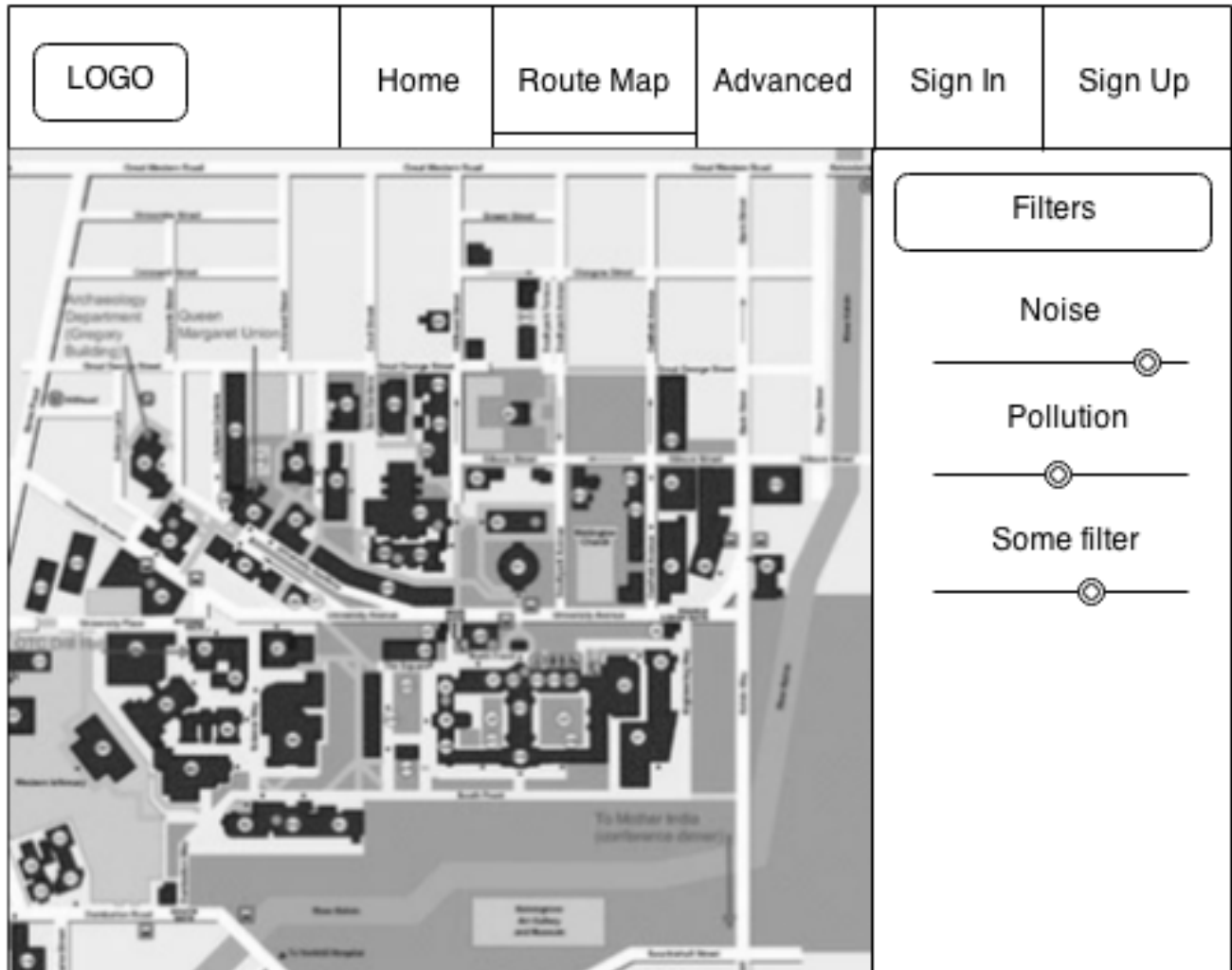
min lat: 55.870056

max lat: 55.875209 → round in this interval

min lon: 4.278787

max lon: -4.287637 → round

Route Map View Prototype v2



Thumbnail 28

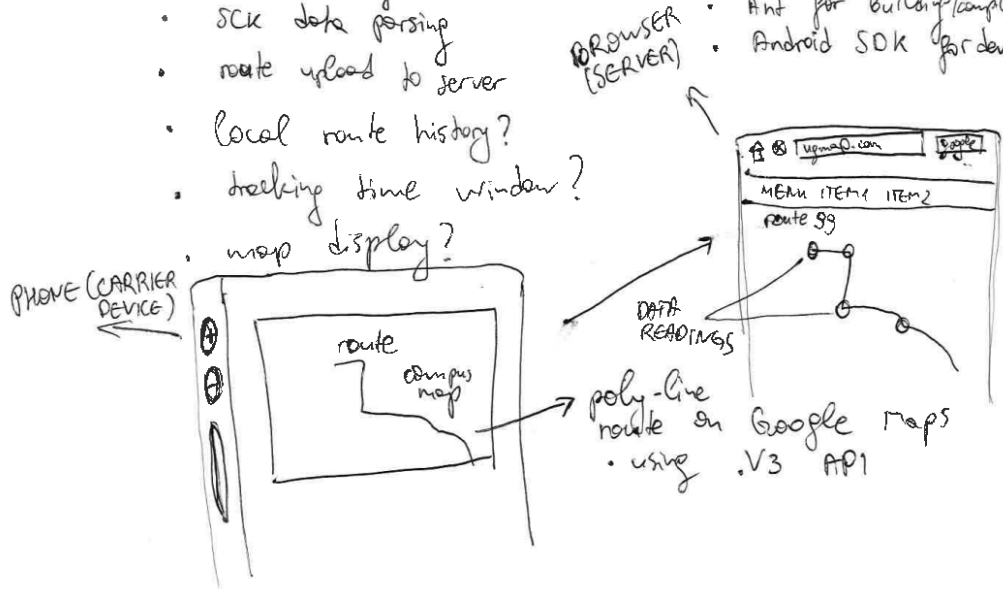
CLIENT PROTOTYPE

functionality (potential features)

- route tracking
- sock data parsing
- route upload to server
- local route history?
- tracking time window?
- map display?

technologies + tools:

- visual studio
- nodeJS to run Cordova
- Phonegap (Cordova) - source wrapper
- Google Maps API
- Ant for building/compile
- Android SDK for dev



Thumbnail 29

SCREEN 2:



UPLOAD ROUTE

version 1:



version 2:



click to upload
route to server
(parse together with SCK data?)

prototype for client (UGMAP application)

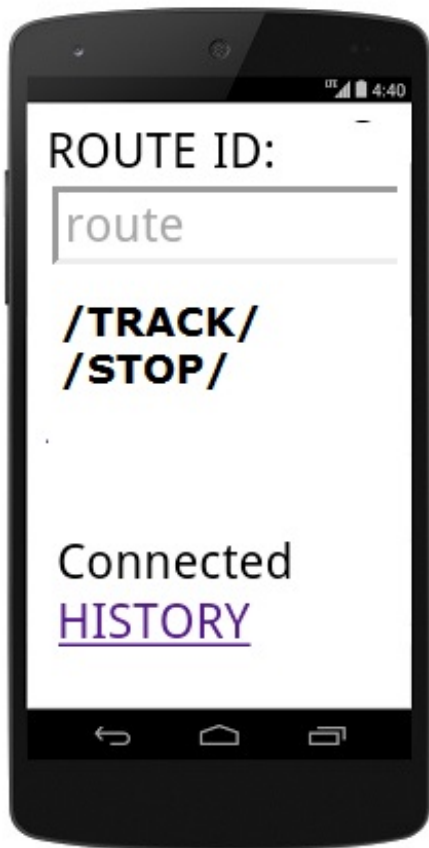
SCREEN 1:



Android required permissions:

- location (GPS resources)
- internet
- network access
- ...

- specify update time interval?
(initially 30 seconds)
/SCK updates are received every
30 seconds /



Thumbnail 32

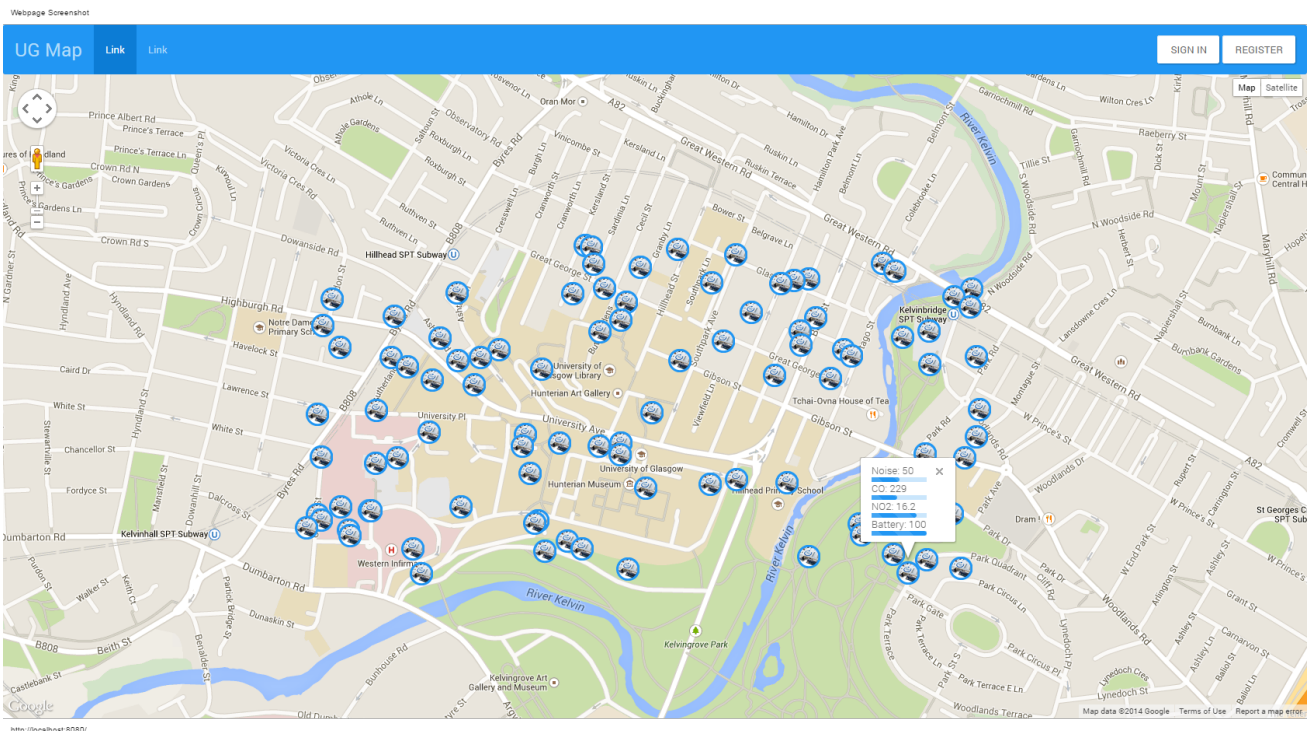
Android Device Home Page Prototype



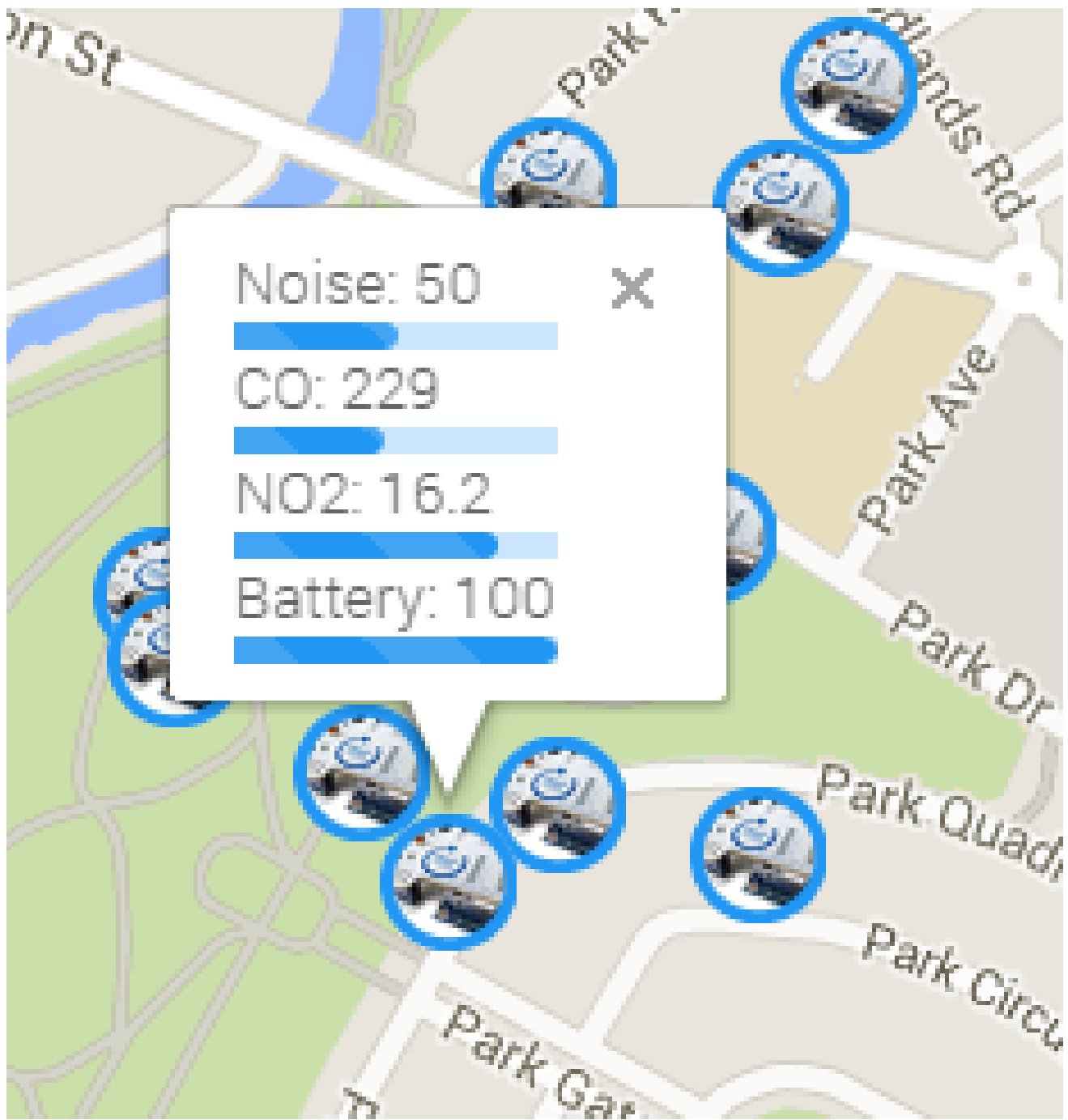
Thumbnail 33



Thumbnail 36



Thumbnail 37



Thumbnail 38

Main content goes here

ROUTE ID:

Route 11/19/2014, 12:04:05 AM

tracking in progress

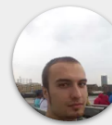
Connected

START TRACKING

STOP TRACKING

CLEAR DATA

[HISTORY](#)



Thumbnail 39

Main content goes here

ROUTE ID:

Route

WINDOW (ms):

10000

tracking in progress

Connected

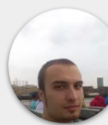
START TRACKING

STOP TRACKING

CLEAR DATA

HISTORY

Probe 2
noise 55
co 181.66
no2 51.48



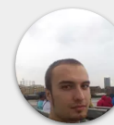
Thumbnail 40



Show history
Routes:11

- [Route](#)
- [Route 11/22/2014, 4:14:26 PM](#)
- [Route 11/22/2014, 4:15:31 PM](#)
- [Route 11/22/2014, 6:08:04 PM](#)
- [Route 11/22/2014, 6:34:40 AM](#)
- [Route 11/22/2014, 6:35:26 AM](#)
- [Route 11/23/2014, 2:54:26 AM](#)
- [Test3](#)
- [null](#)
- [test1](#)
- [test2](#)

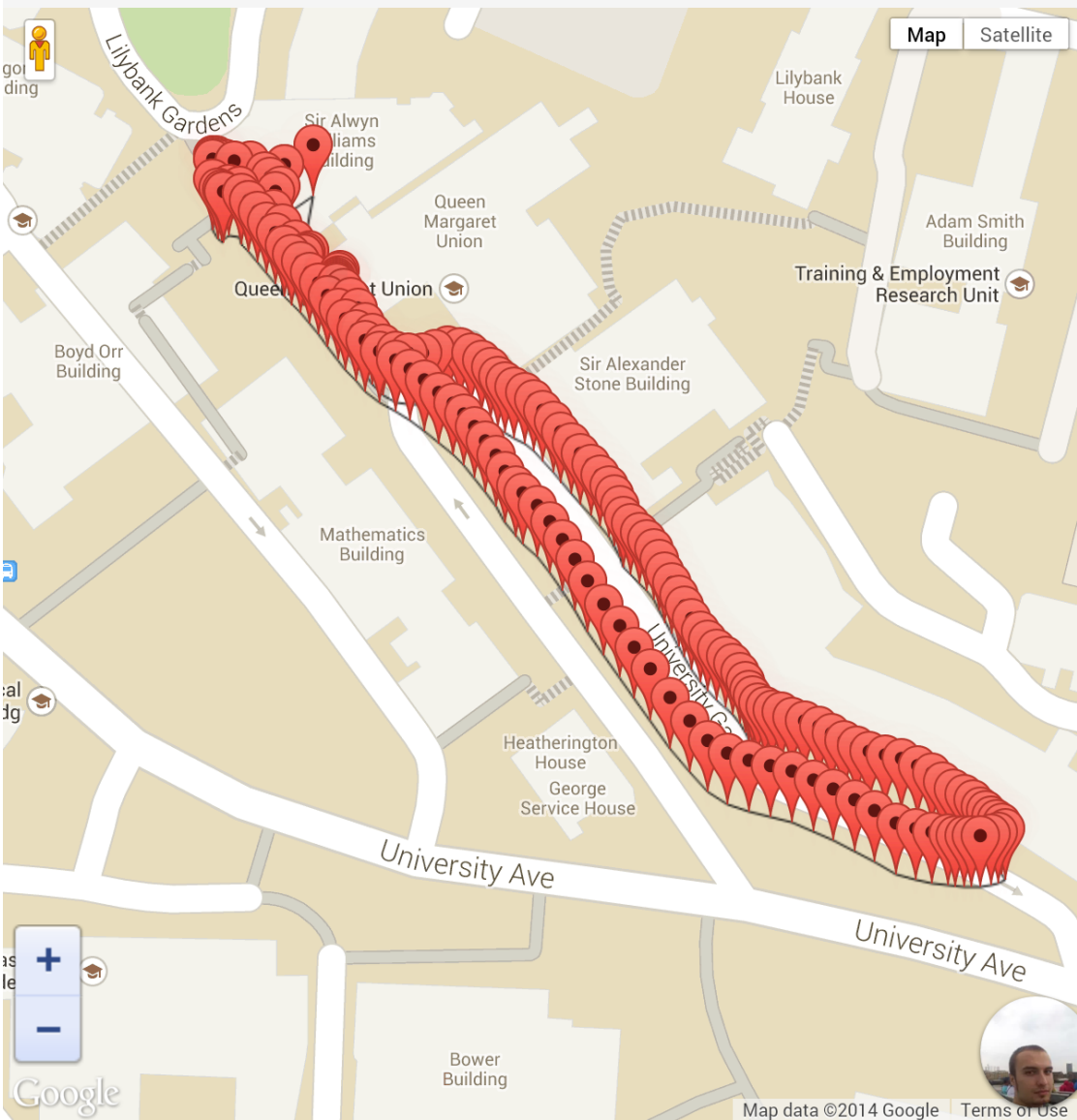
[HOME](#)



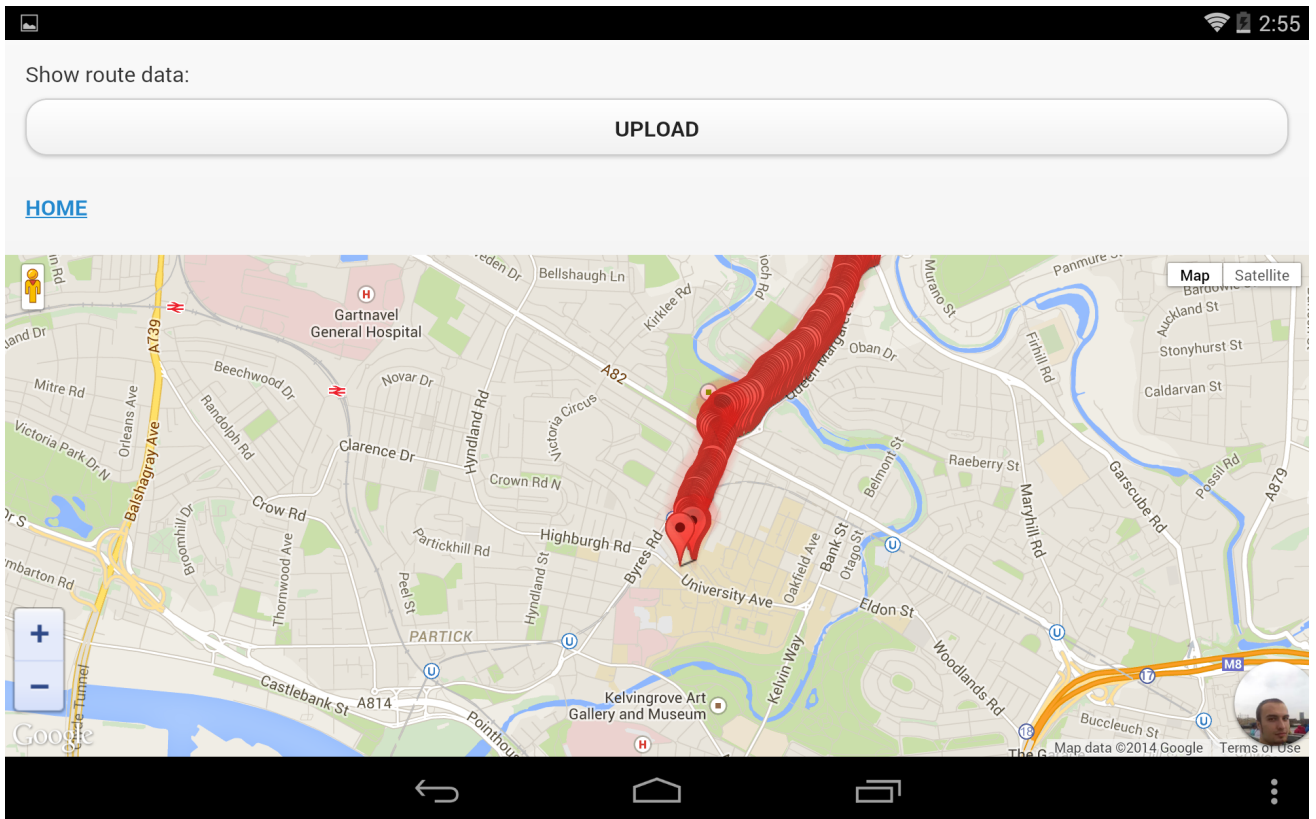
Show route data:

UPLOAD

HOME



Thumbnail 42



Thumbnail 43



Dynamic Noise and Pollution Campus Map

Route Tracking Client

Route ID:

Please Enter an Identifier

Time Window:

Window (ms)

SCK Device API Code:

084122509ae13c389bf752915861249cff652249

Connected

Environmental data tracking can be performed via the buttons below.

Track



Home



History

UG Map Copyright 2015





Dynamic Noise and Pollution Campus Map

Route Tracking Client

Route ID:

Route 3/13/2015, 6:46:26 PM

Time Window:

30000

SCK Device API Code:

084122509ae13c389bf752915861249cff652249

Tracking in progress!

Connected

Environmental data tracking can be performed via the buttons below.

Stop

Probe: 42
Noise: 63.34 dB
CO: 179.81 kOhm
NO2: 98.52 kOhm
Battery: 68.4 %
Latitude: 55.877014
Longitude: -4.2871408
Time: Fri Mar 13 2015 18:46:28 GMT+0000 (GMT)

Thumbnail 45



Dynamic Noise and Pollution Campus Map

Route Tracking Client

Route ID:

Time Window:

SCK Device API Code:

Completed tracking and transmitted data to the server!

Connected

Environmental data tracking can be performed via the buttons below.

Track

Probe: 41
Noise: 66.88 dB
CO: 169.06 kOhm
NO2: 105.91 kOhm
Battery: 68.9 %
Latitude: 55.8770314
Longitude: -4.2871726
Time: Fri Mar 13 2015 18:46:16 GMT+0000 (GMT)

Thumbnail 46



Dynamic Noise and Pollution Campus Map

Routes:66

- [Route 2/4/2015, 7:33:03 PM](#)
- [Route 2/4/2015, 7:42:37 PM](#)
- [Route 2/4/2015, 7:42:54 PM](#)
- [Route 2/6/2015, 10:10:45 AM](#)
- [Route 2/6/2015, 7:08:47 AM](#)
- [Route 2/6/2015, 7:22:47 AM](#)
- [Route 2/6/2015, 7:38:50 AM](#)
- [Route 2/6/2015, 9:47:01 AM](#)
- [Route 2/6/2015, 9:57:21 AM](#)
- [Route 2/8/2015, 10:11:44 PM](#)
- [Route 2/8/2015, 10:13:20 PM](#)
- [Route 2/8/2015, 10:16:03 PM](#)
- [Route 2/8/2015, 10:20:18 PM](#)
- [Route 2/8/2015, 10:26:04 PM](#)
- [Route 2/8/2015, 10:31:33 PM](#)
- [Route 2/8/2015, 10:38:20 PM](#)
- [Route 2/8/2015, 10:43:06 PM](#)
- [Route 2/8/2015, 10:44:37 PM](#)
- [Route 2/8/2015, 10:46:38 PM](#)
- [Route 2/8/2015, 10:49:56 PM](#)
- [Route 2/8/2015, 11:00:26 PM](#)
- [Route 2/8/2015, 11:02:09 PM](#)
- [Route 2/8/2015, 11:05:35 PM](#)
- [Route 2/8/2015, 9:47:34 PM](#)
- [Route 2/8/2015, 9:49:53 PM](#)
- [Route 3/11/2015, 11:32:28 AM](#)
- [Route 3/11/2015, 11:39:10 AM](#)
- [Route 3/11/2015, 11:47:52 AM](#)
- [Route 3/11/2015, 3:10:35 PM](#)
- [Route 3/12/2015, 5:45:37 PM](#)
- [Route 3/12/2015, 5:55:50 PM](#)
- [Route 3/12/2015, 6:02:28 PM](#)
- [Route 3/13/2015, 6:12:13 PM](#)
- [Route 3/13/2015, 6:25:38 PM](#)
- [Route 3/13/2015, 6:25:46 PM](#)



Home



History

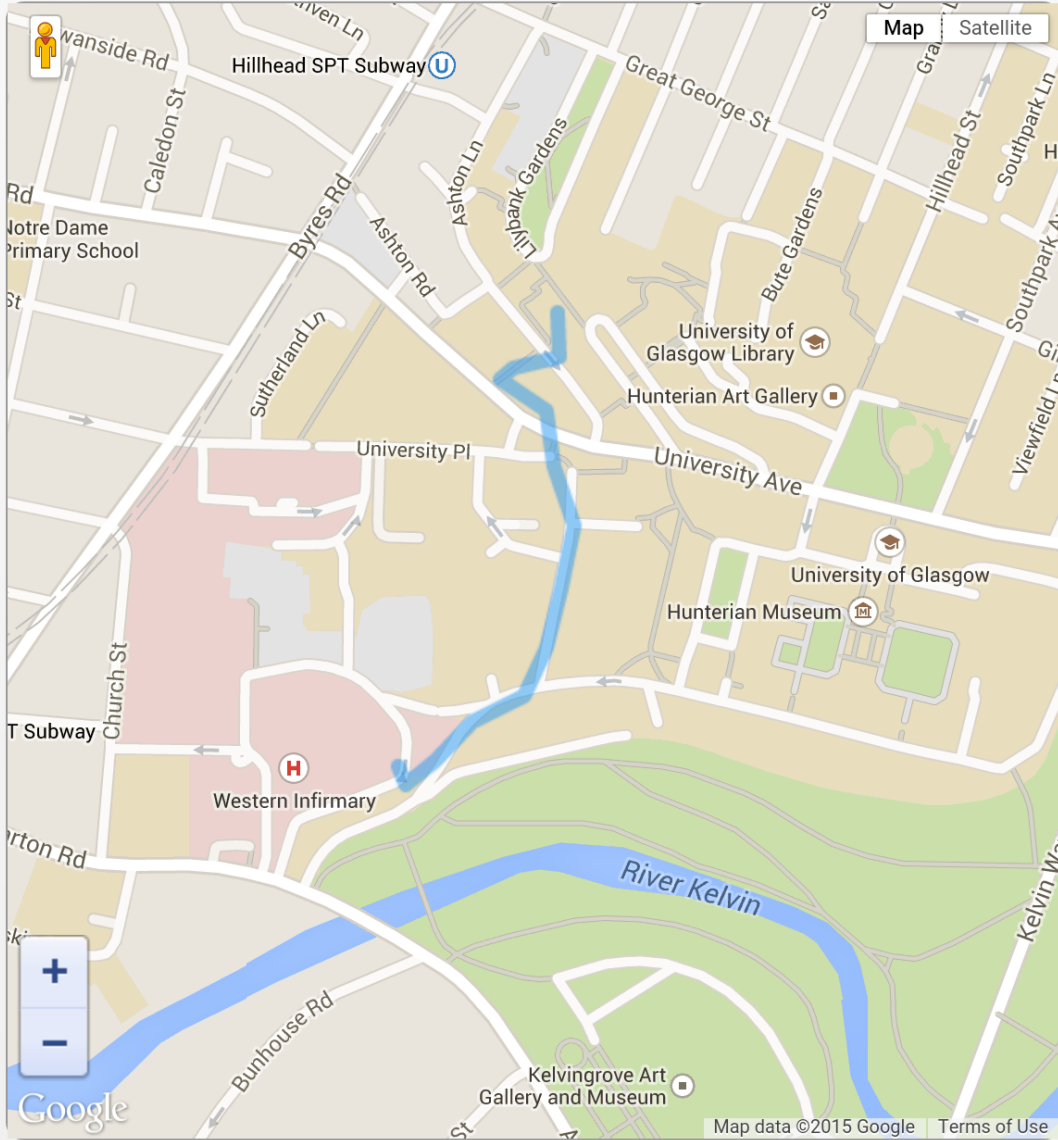
UG Map Copyright 2015



Thumbnail 47



Dynamic Noise and Pollution Campus Map



Home



History

UG Map Copyright 2015

```

2015-03-22 00:10:46 INFO DispatcherServlet:498 - FrameworkServlet 'mvc-dispatcher': initialization completed in 3765 ns
2015-03-22 00:10:46.050:INFO::Started SelectChannelConnector@0.0.0.0:80
[INFO] Started Jetty Server
2015-03-22 00:10:53 INFO HomeController:102 - Data Readings: 530
2015-03-22 00:10:53 INFO HomeController:103 - Routes: 96
2015-03-22 00:10:53 INFO HomeController:104 - Devices: 5
2015-03-22 00:10:53 INFO HomeController:105 - Users: 7

```

Thumbnail 49

```

2015-03-25 03:01:38 INFO HomeController:124 - User saved: com.springapp.mvc.models.User Object (
id: 2282
Name: ppyordanov
City: Glasgow
Country: United Kingdom
Website: http://ppyordanov.com
Email: ppyordanov@yahoo.com
Created: 2014-09-24 12:49:30
)

2015-03-25 03:01:38 INFO HomeController:127 - Device saved: com.springapp.mvc.models.Device Object (
id: 1651
title: PP Yordanov's Kit
description: Testing kit.
kitVersion: null
location: Glasgow, United Kingdom
created: 2014-10-01 19:13:03 UTC
userId: 2282
)

2015-03-25 03:01:38 INFO HomeController:133 - Route saved: com.springapp.mvc.models.Route Object (
id: 55125d52e4b0ceff77a313a5
deviceId: 1651
)

2015-03-25 03:01:38 INFO HomeController:136 - Data Size:1
2015-03-25 03:01:38 INFO HomeController:141 - Data reading saved: com.springapp.mvc.models.DataReading Object (
id: 55125d52e4b0ceff77a313a6
routeId: 55125d52e4b0ceff77a313a5
deviceId: 1651
timestamp: 2015-03-25T07:01:36.211Z
latitude: 55.8843286
longitude: -4.2803049
noise: 59.9
co: 186.22
no2: 173.21
battery: 67.9
)

```

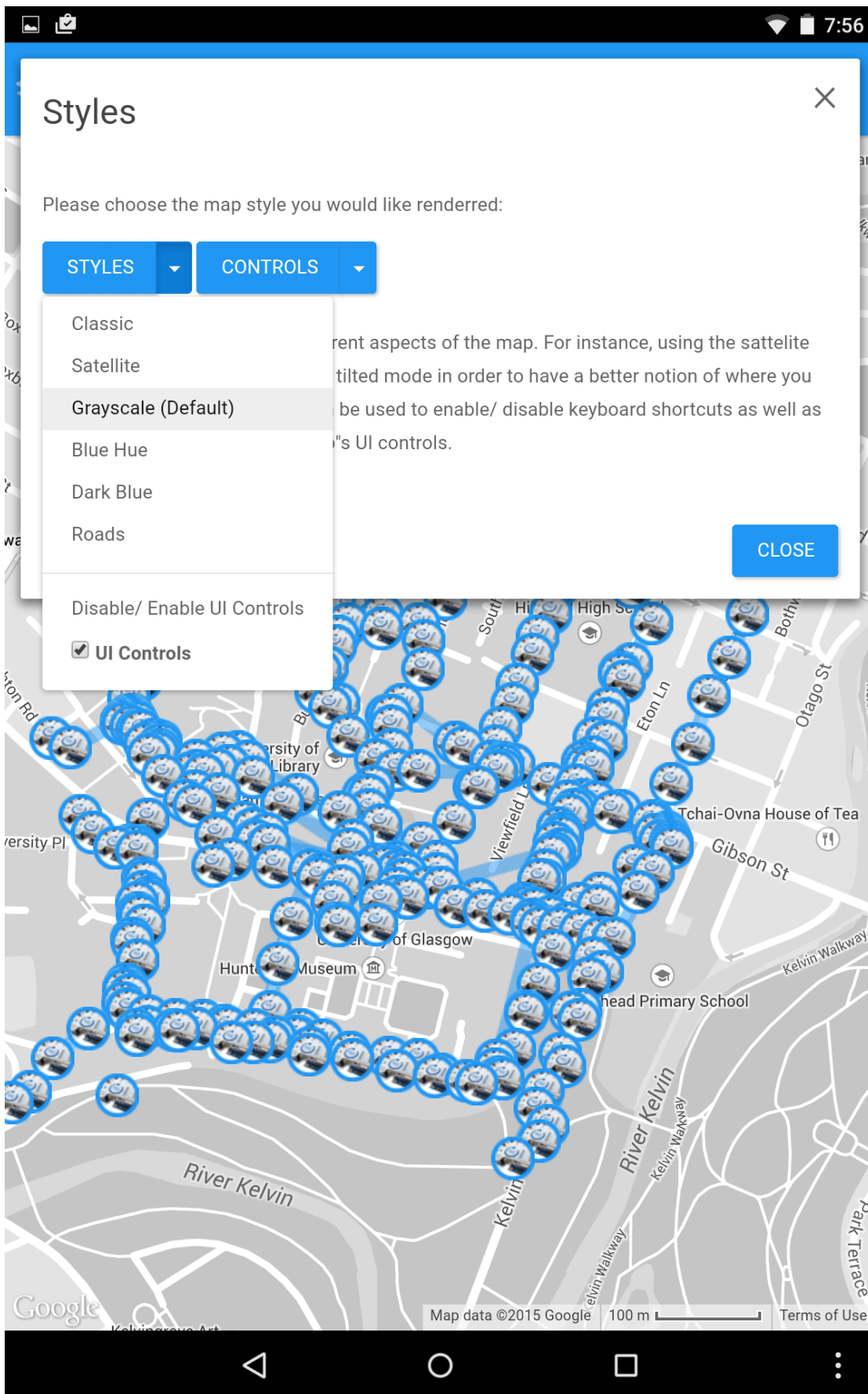
Thumbnail 50

Expected Value of a Weighted Sum of Random Variables

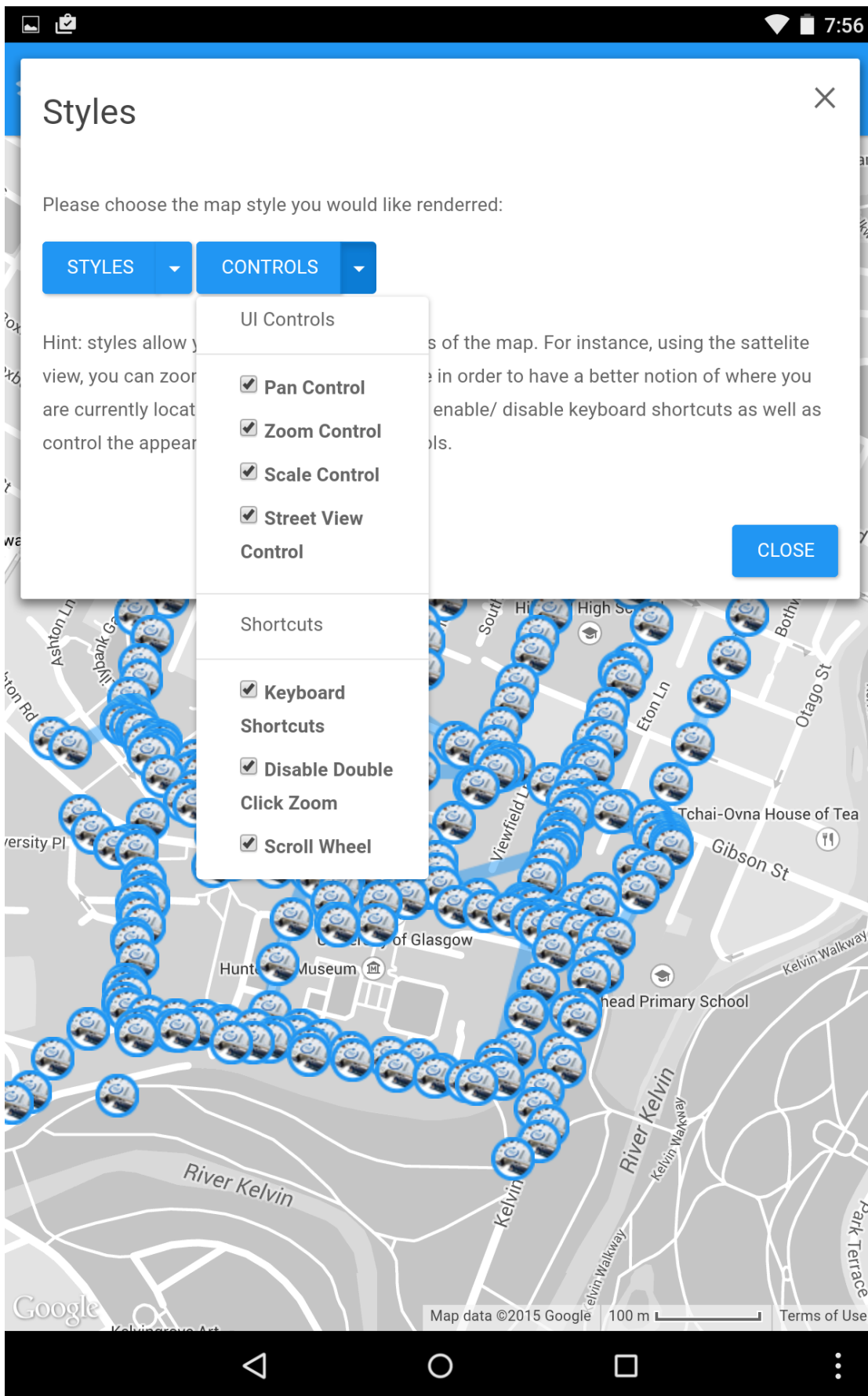
$$E(Y) = a_1E(X_1) + a_2E(X_2) + \dots + a_nE(X_n)$$

$$E(Y) = \sum_{i=1}^n a_i E(X_i)$$

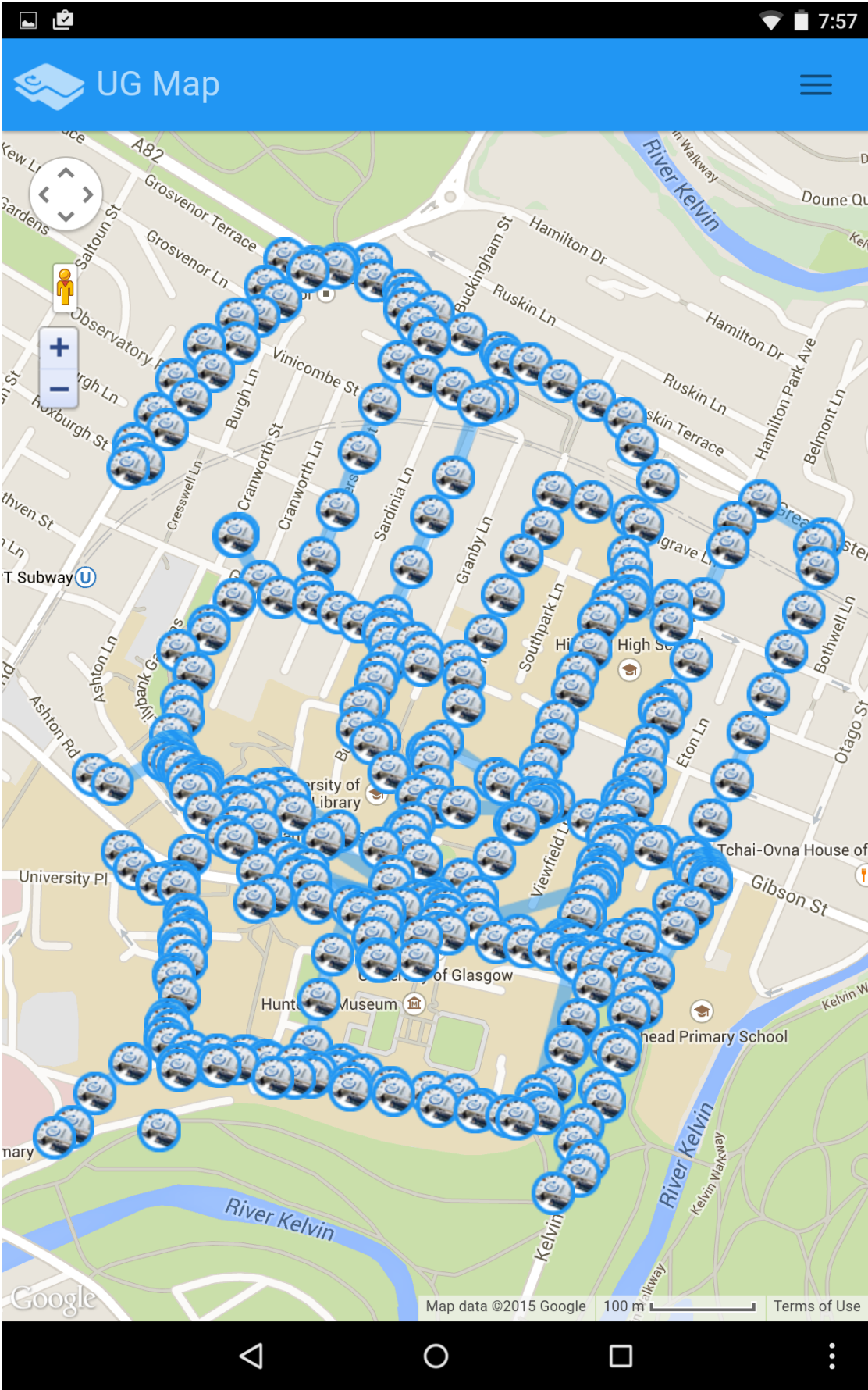
Thumbnail 51



Thumbnail 52



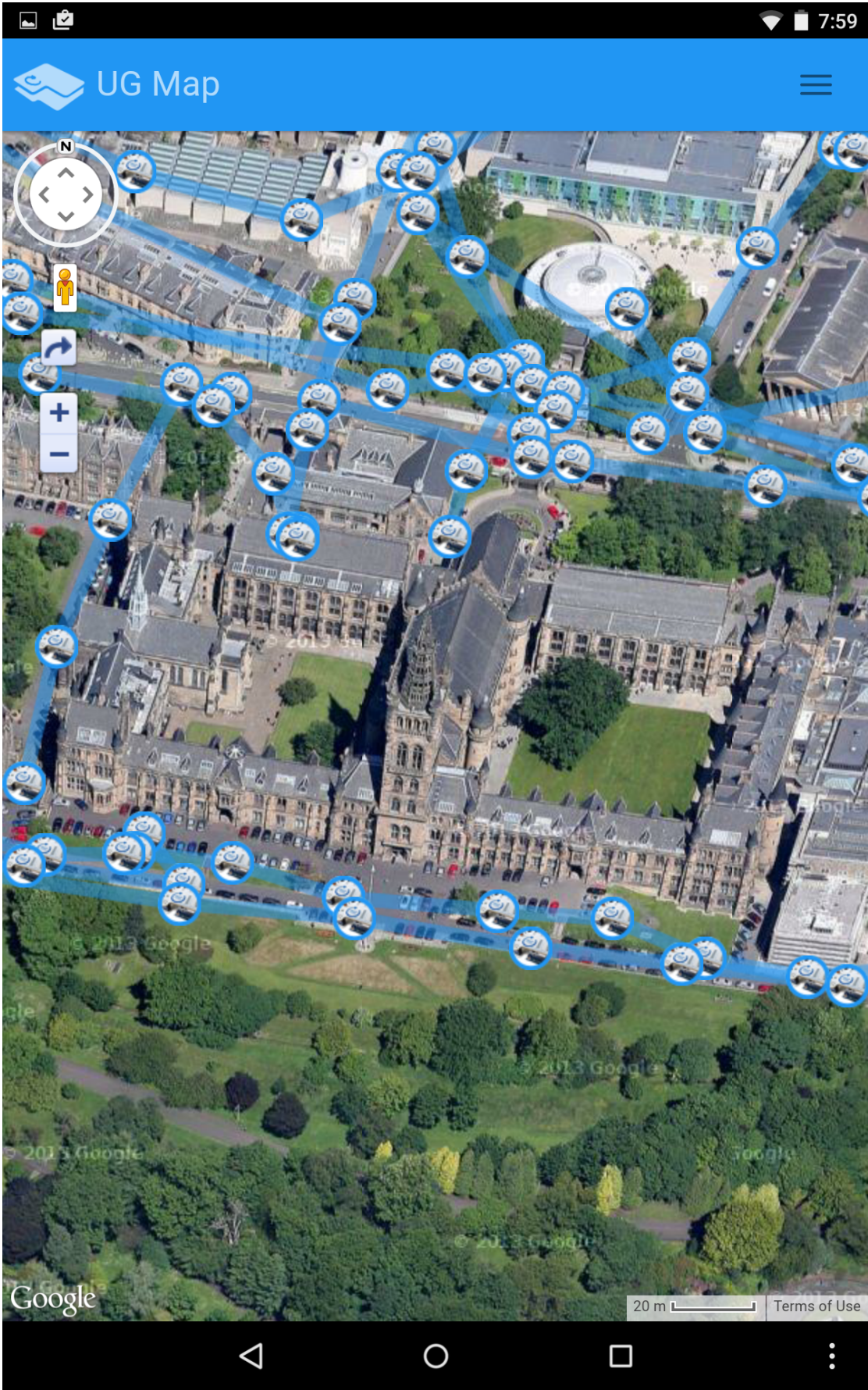
Thumbnail 53



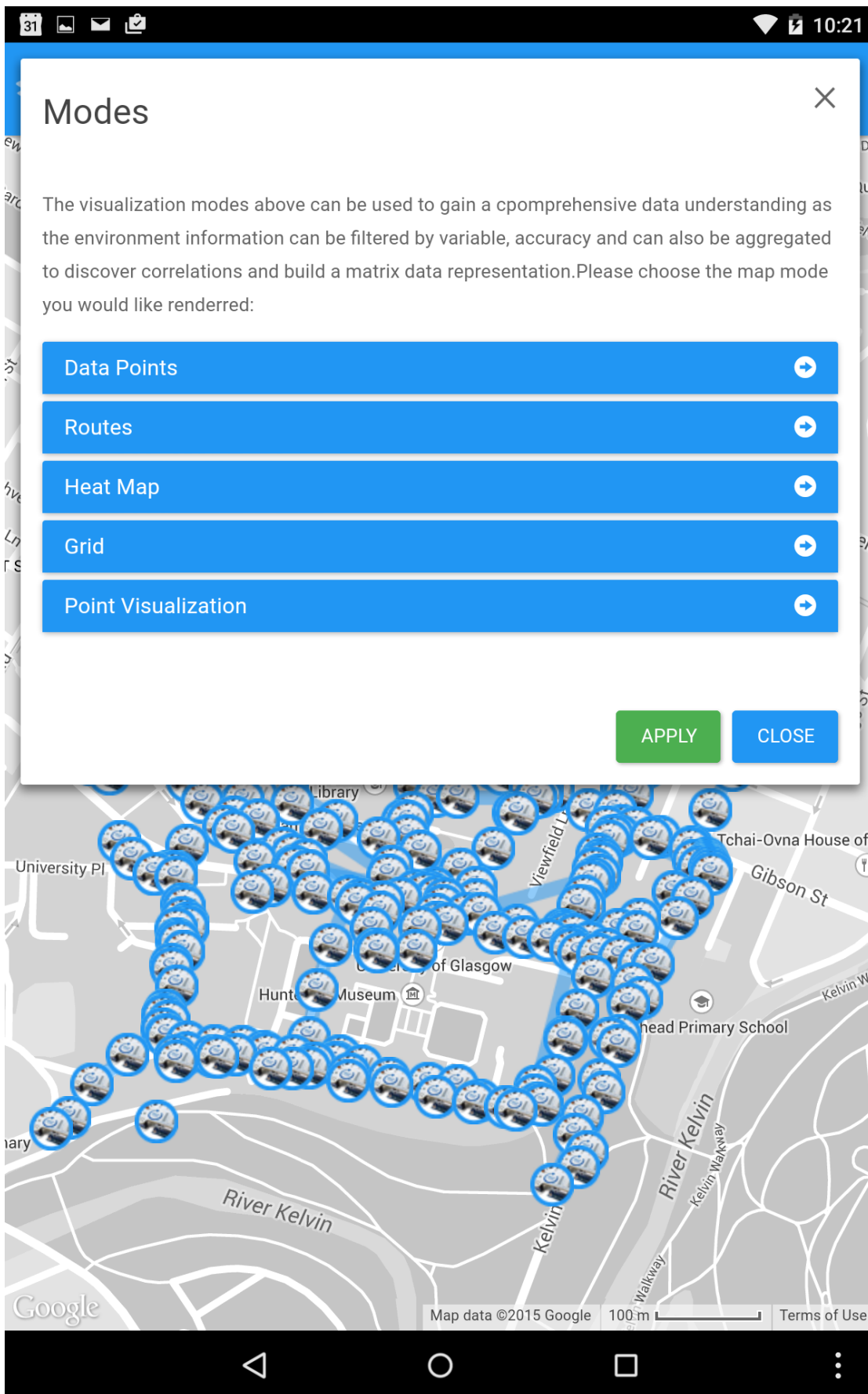
Thumbnail 54



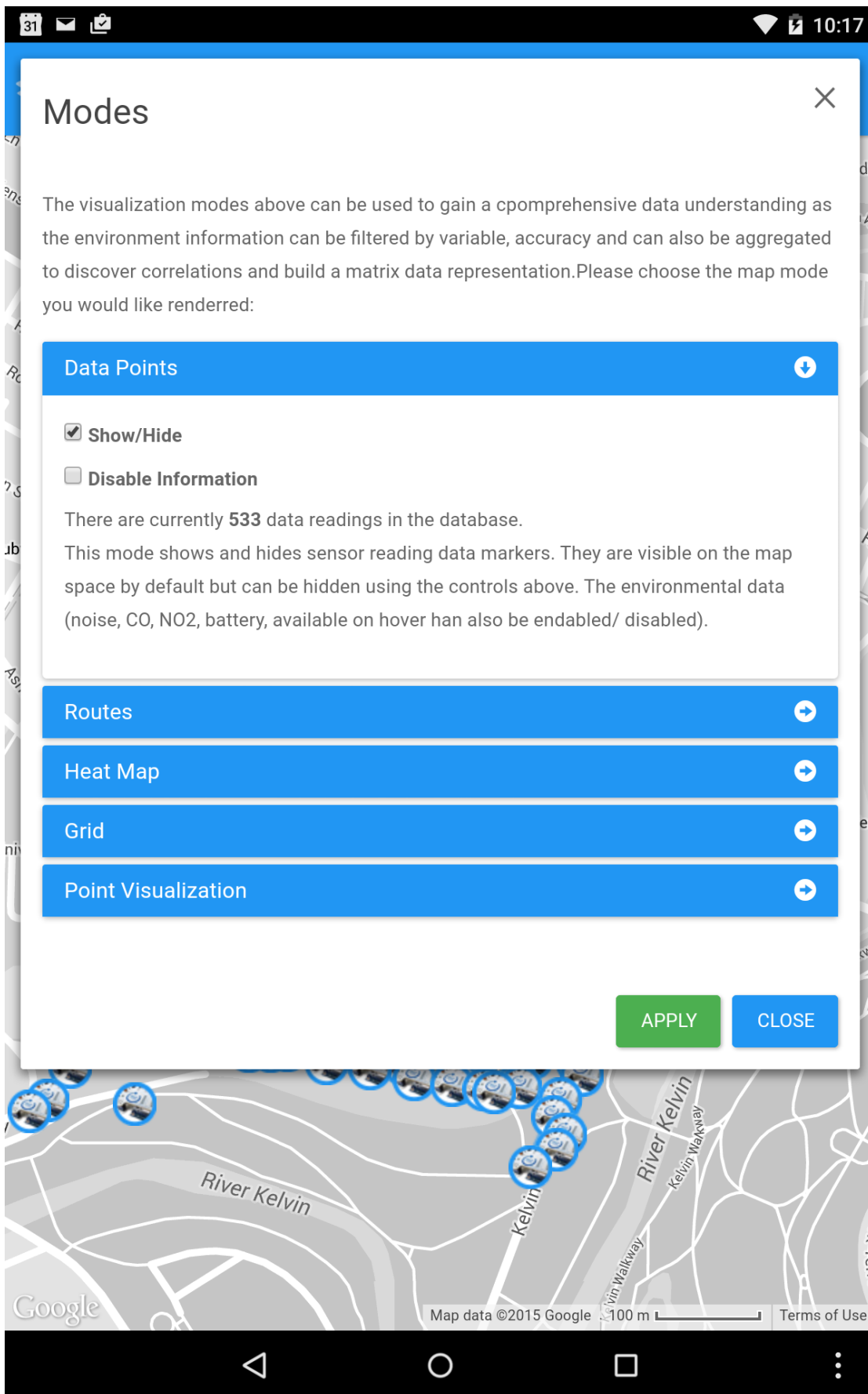
Thumbnail 55



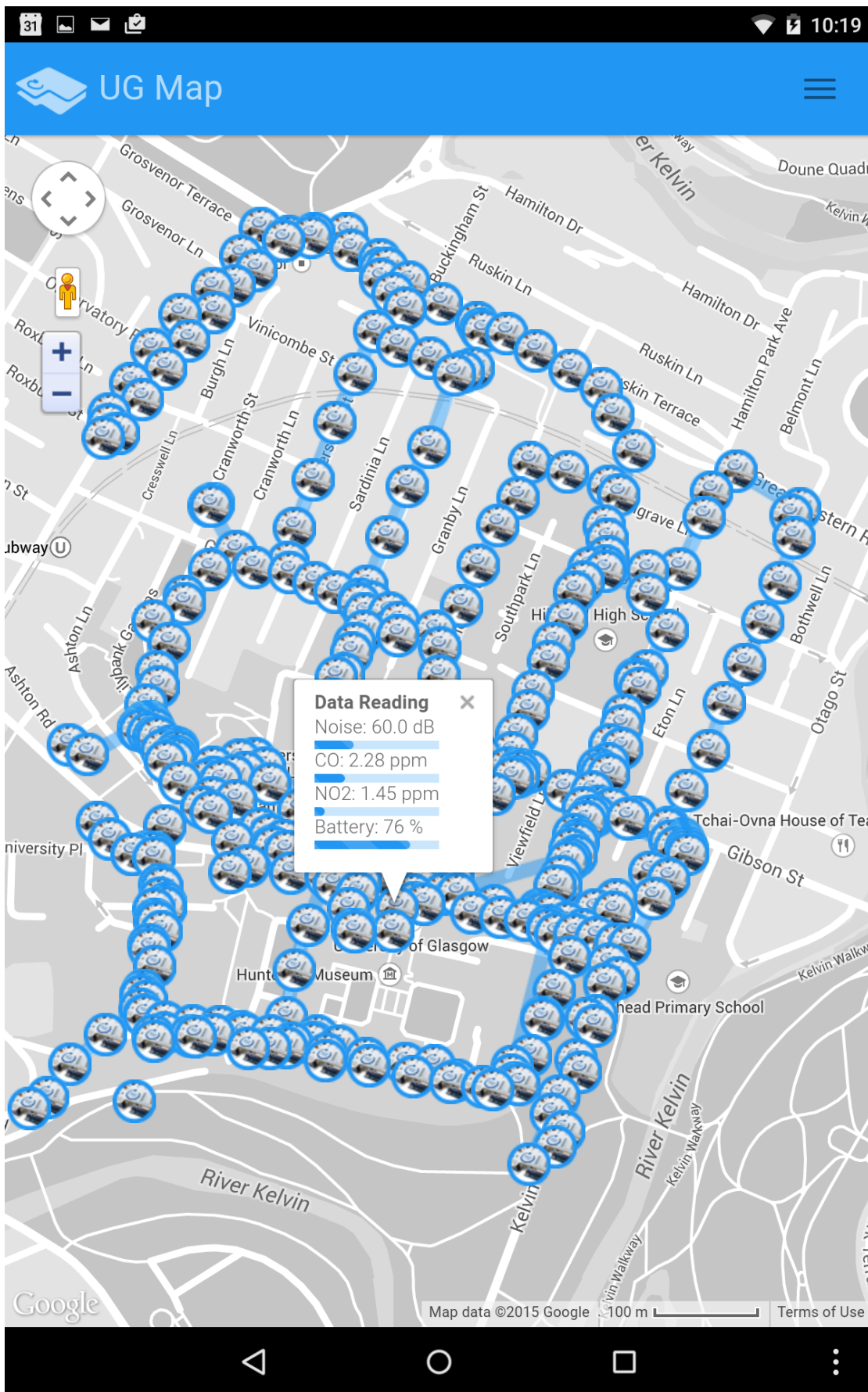
Thumbnail 56



Thumbnail 57

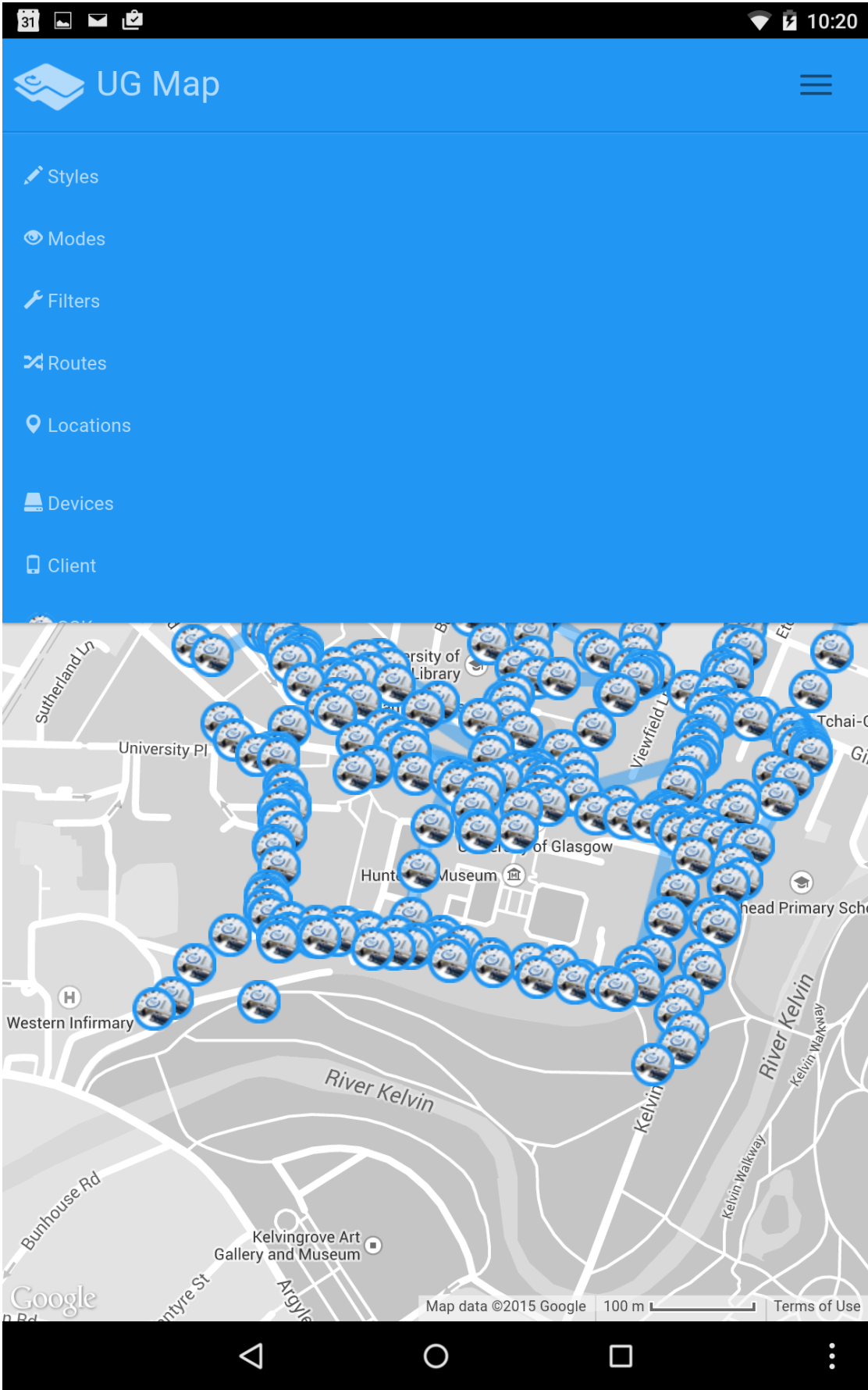


Thumbnail 58

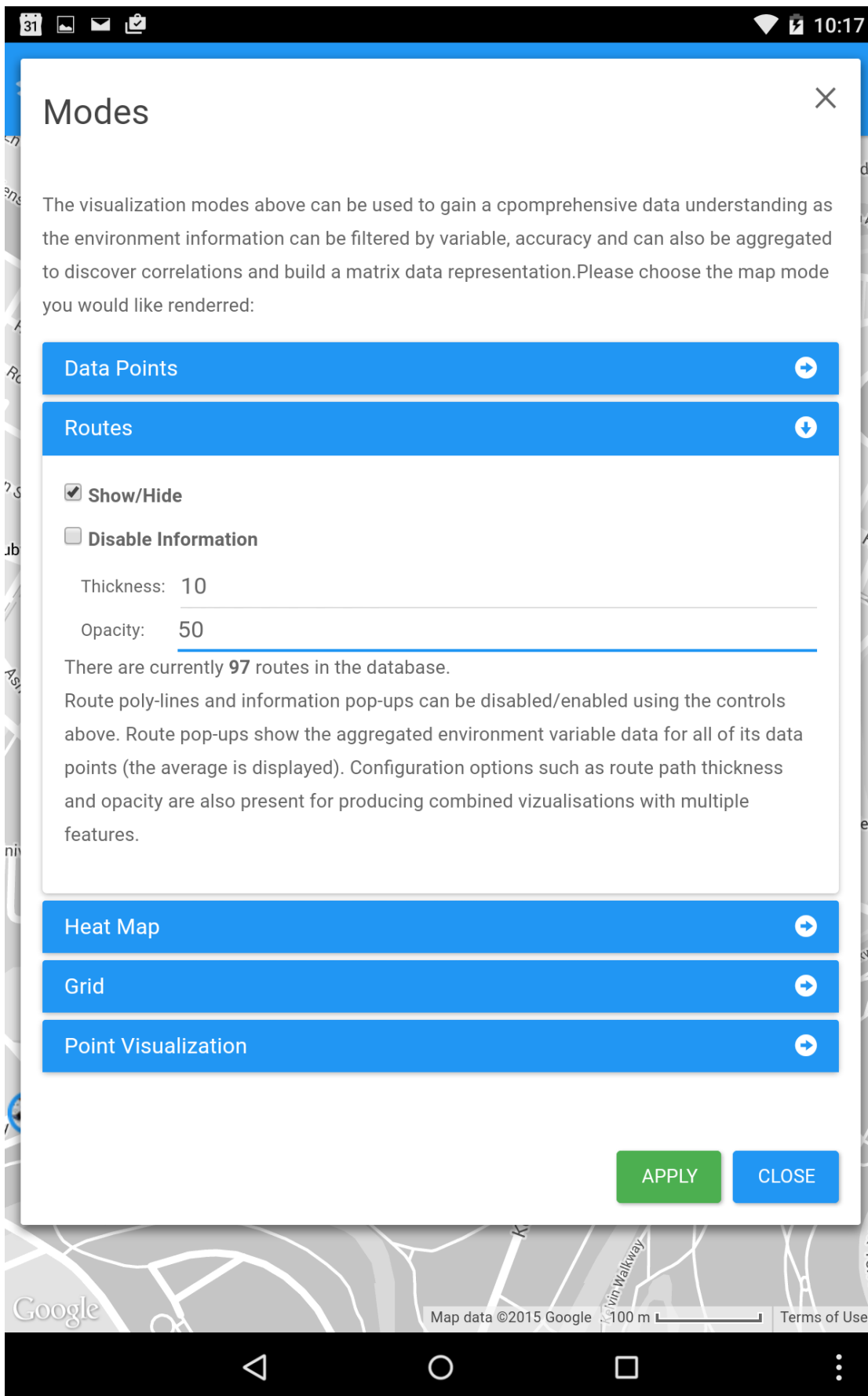


Thumbnail 59

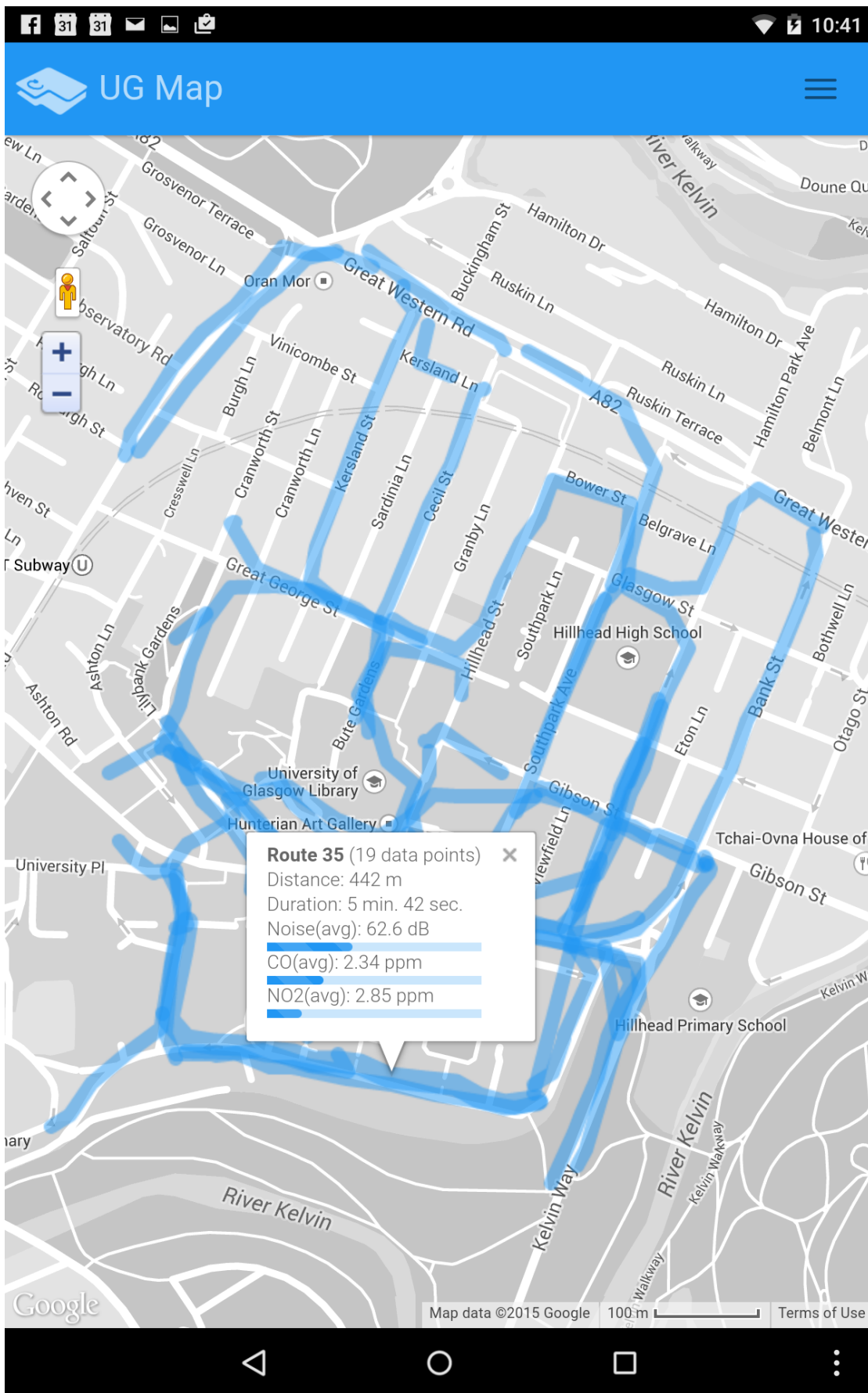




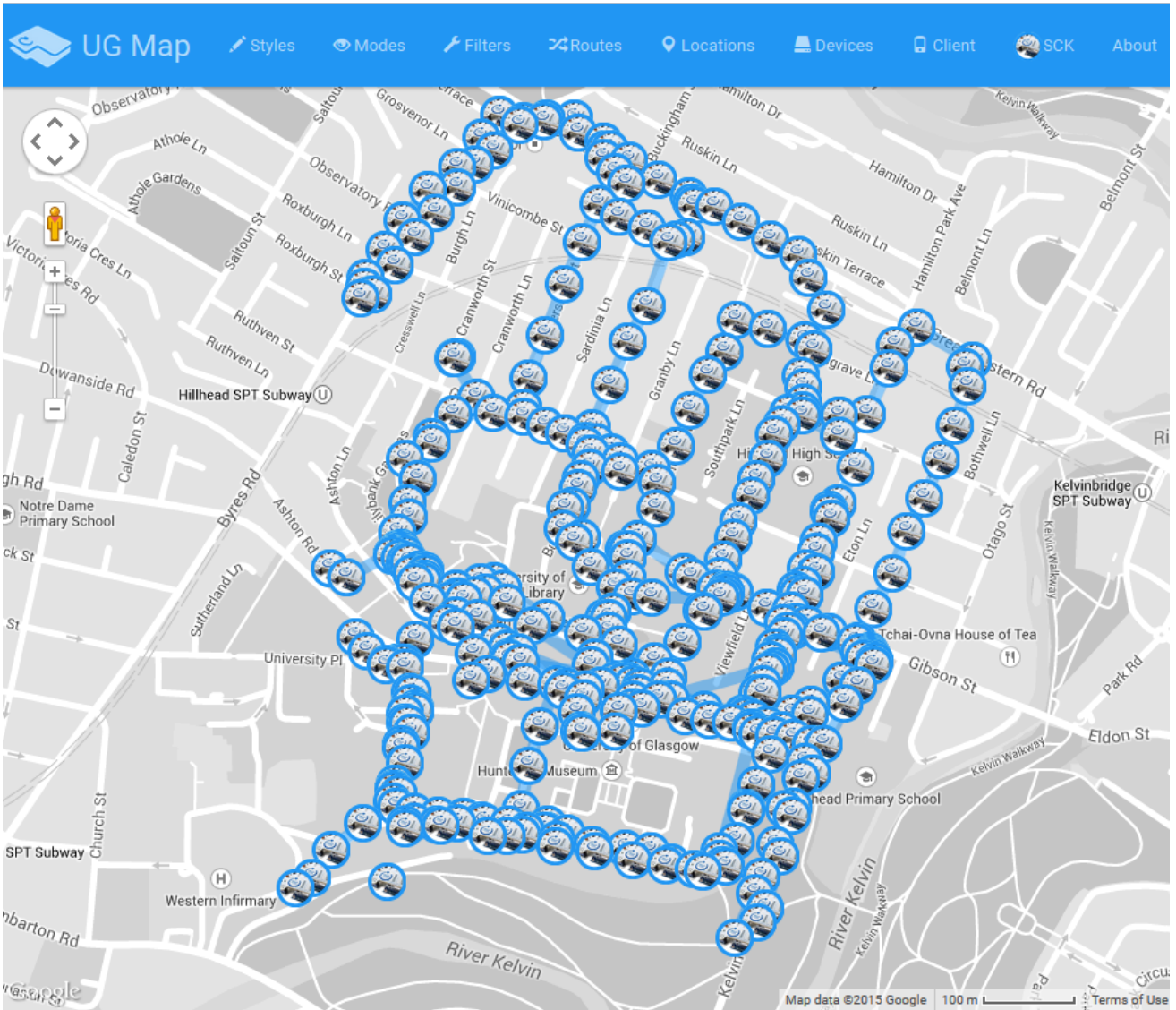
Thumbnail 61



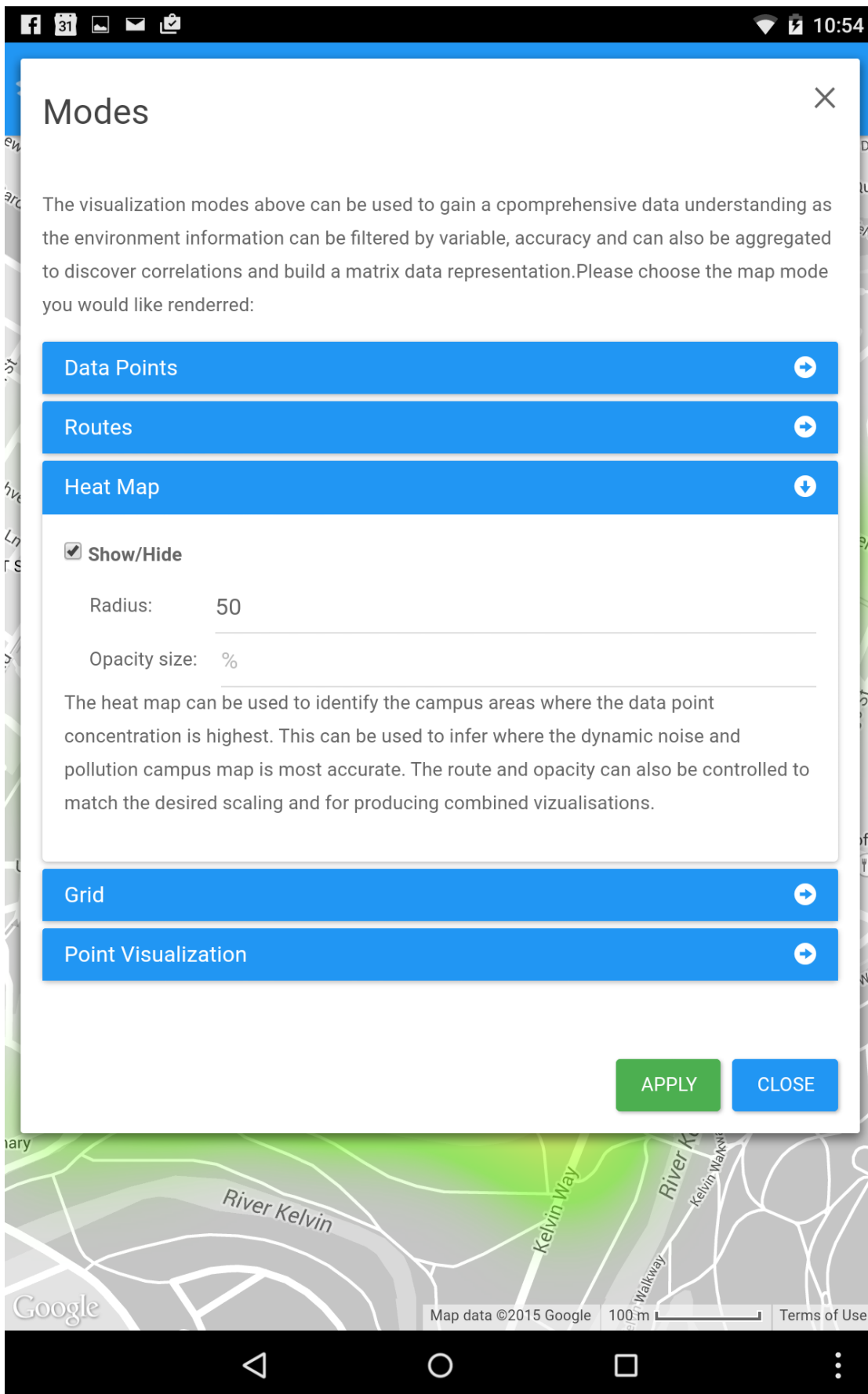
Thumbnail 62



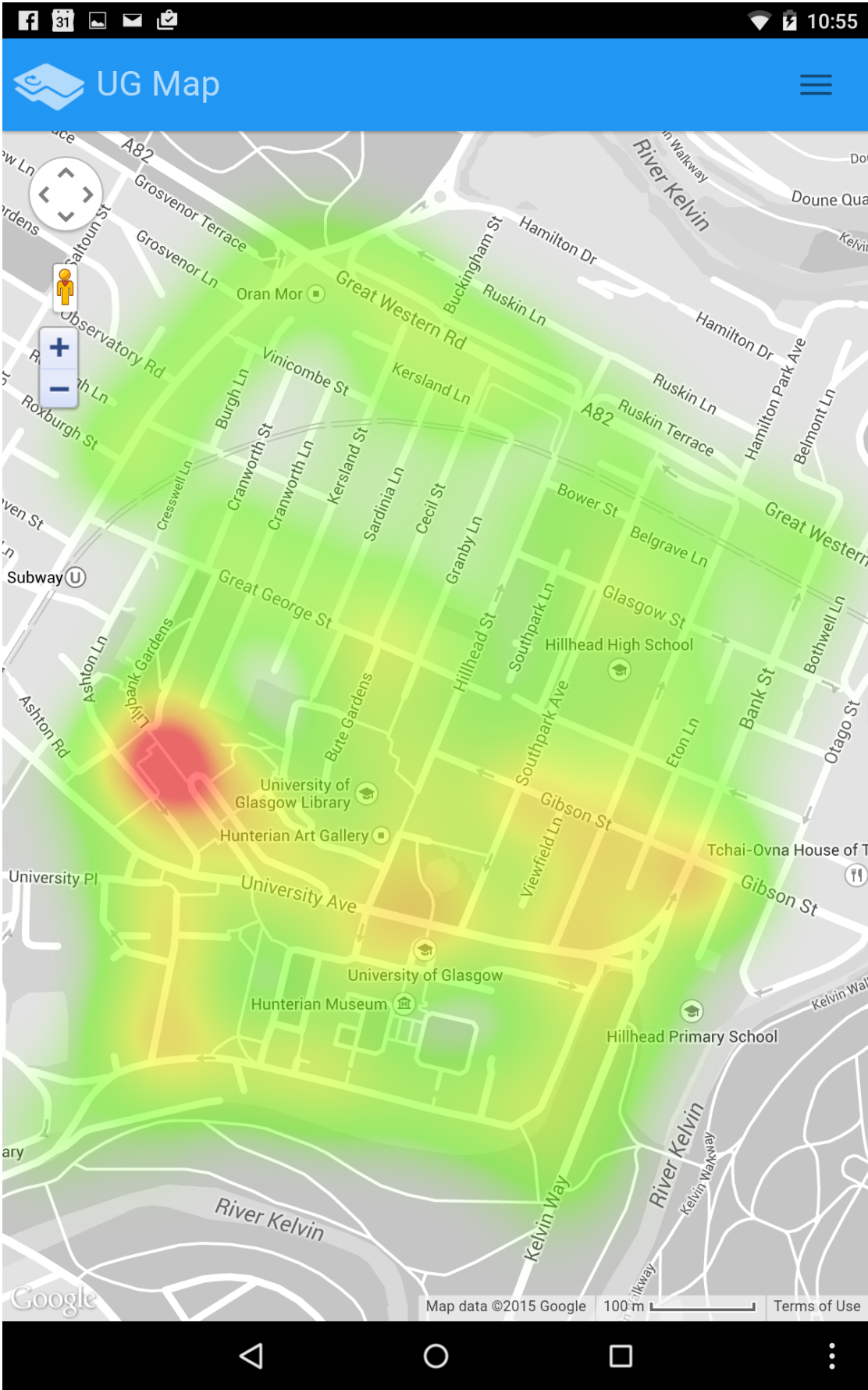
Thumbnail 63



Thumbnail 64



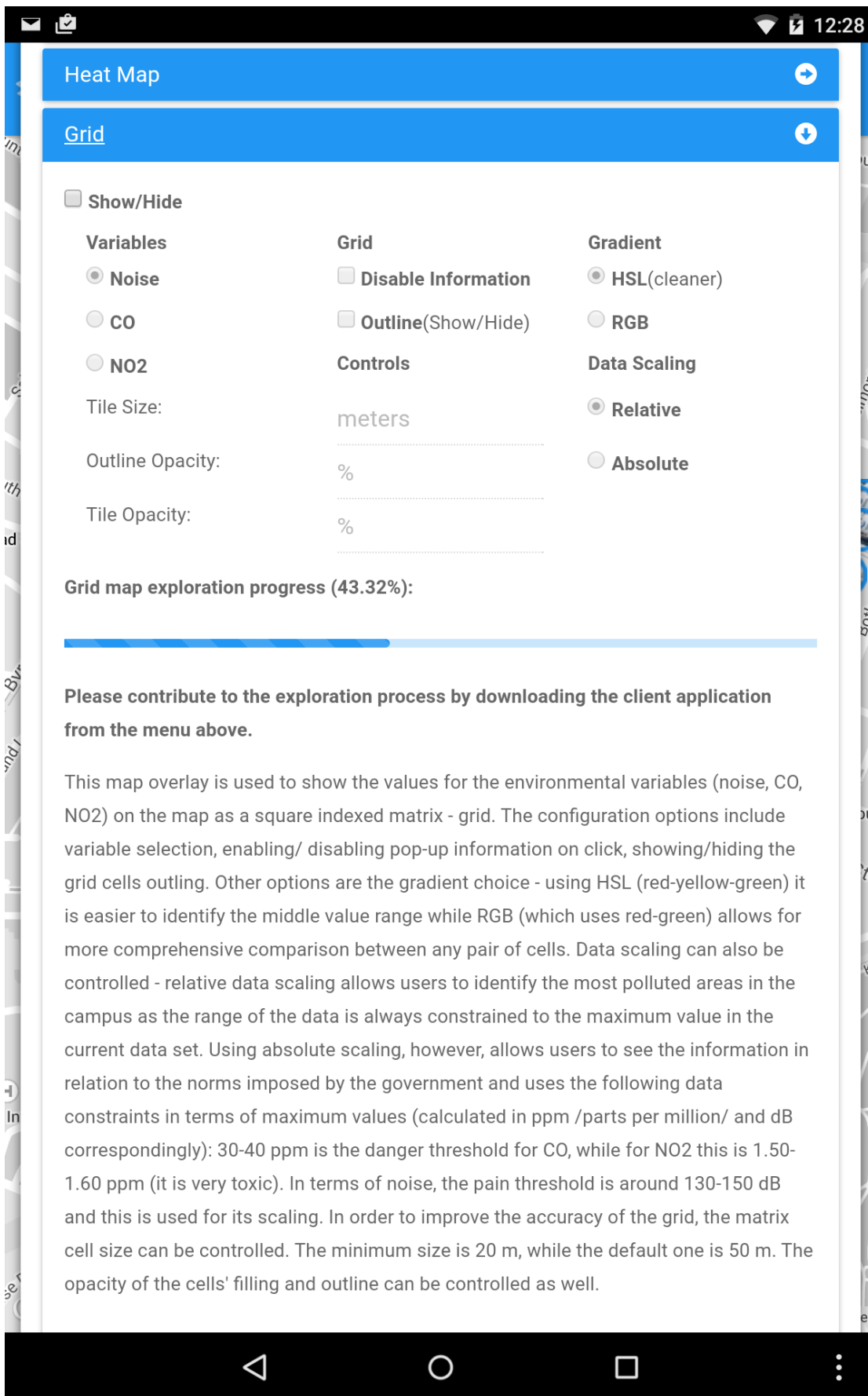
Thumbnail 65



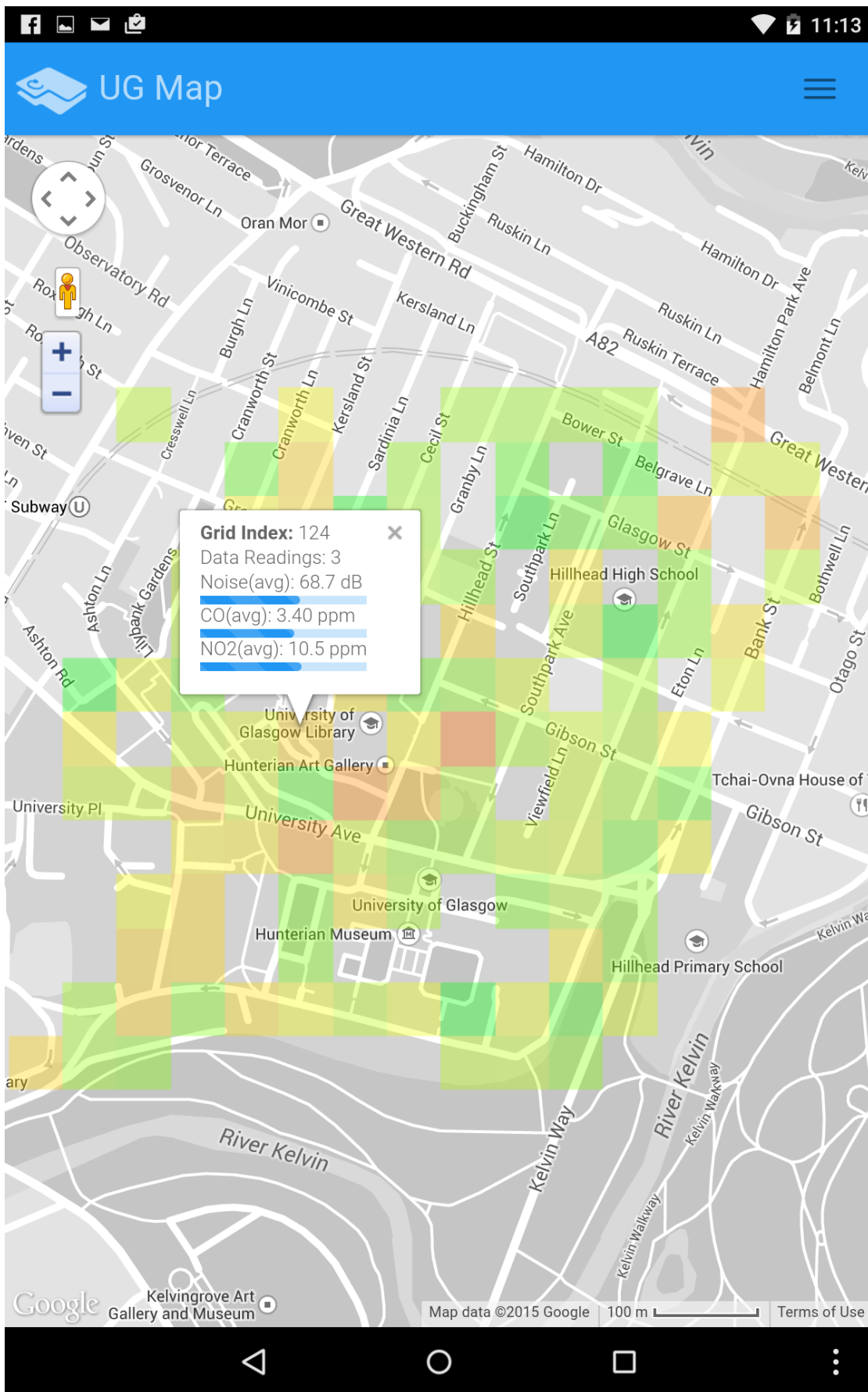
Thumbnail 66



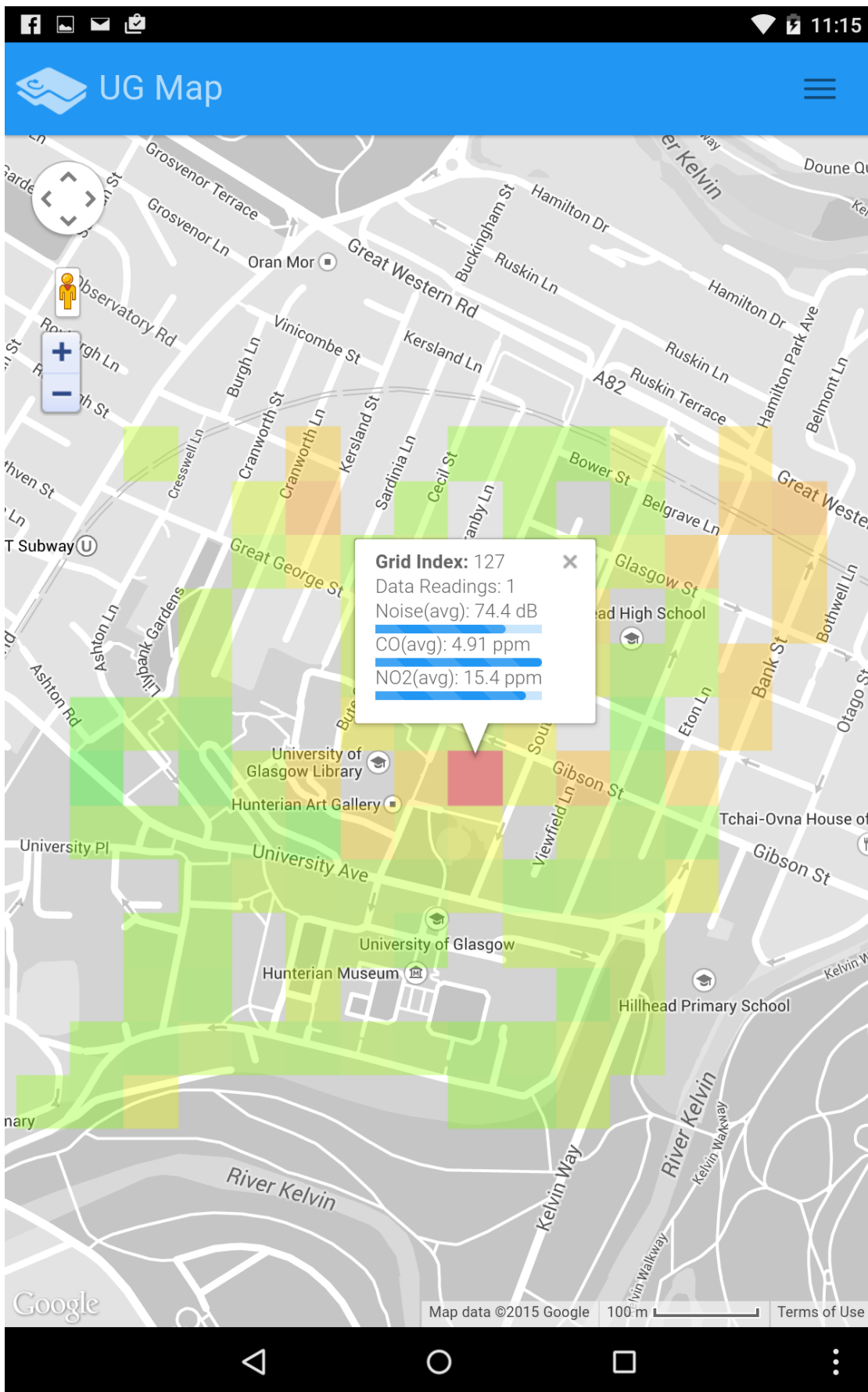
Thumbnail 67



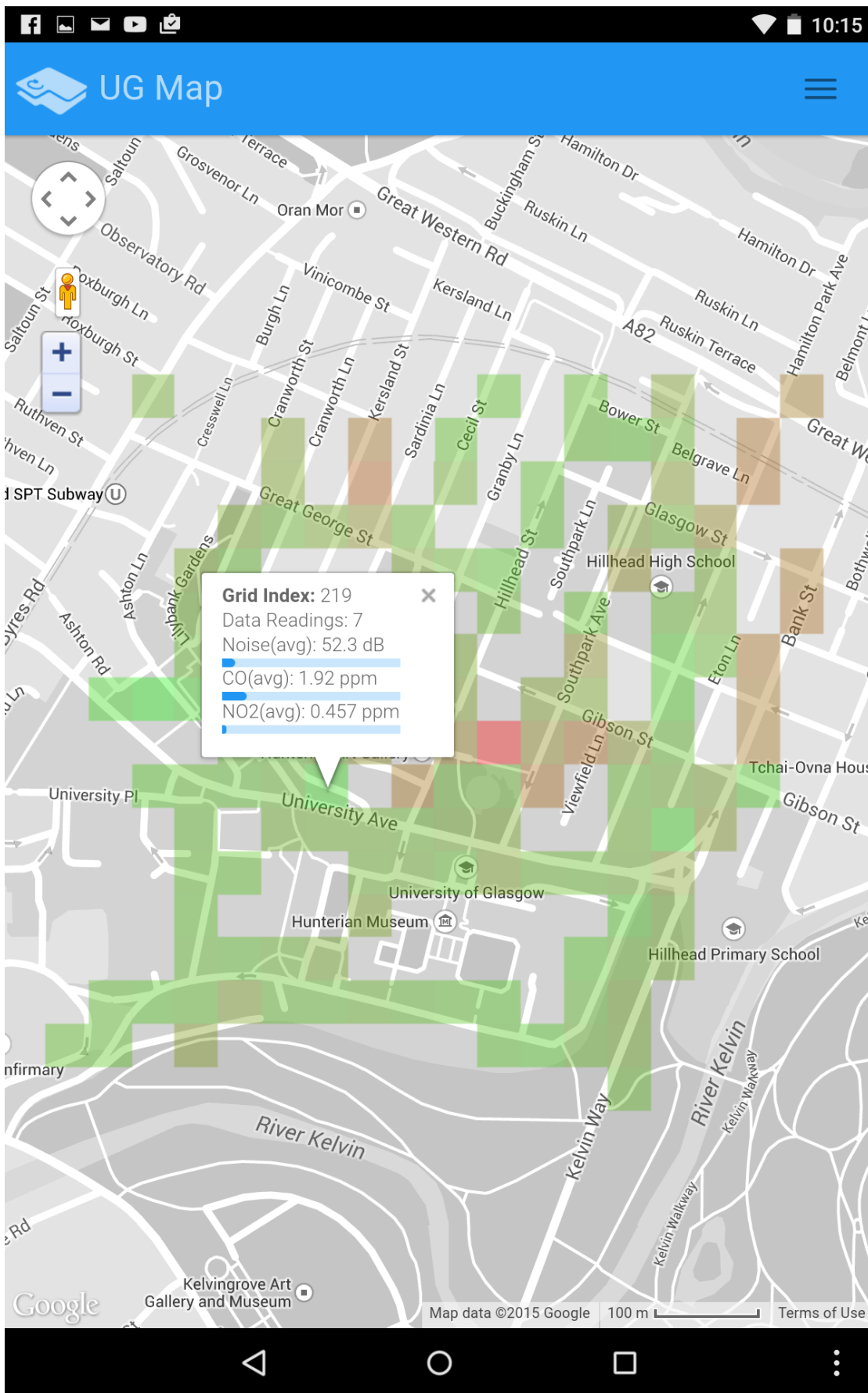
Thumbnail 68



Thumbnail 69



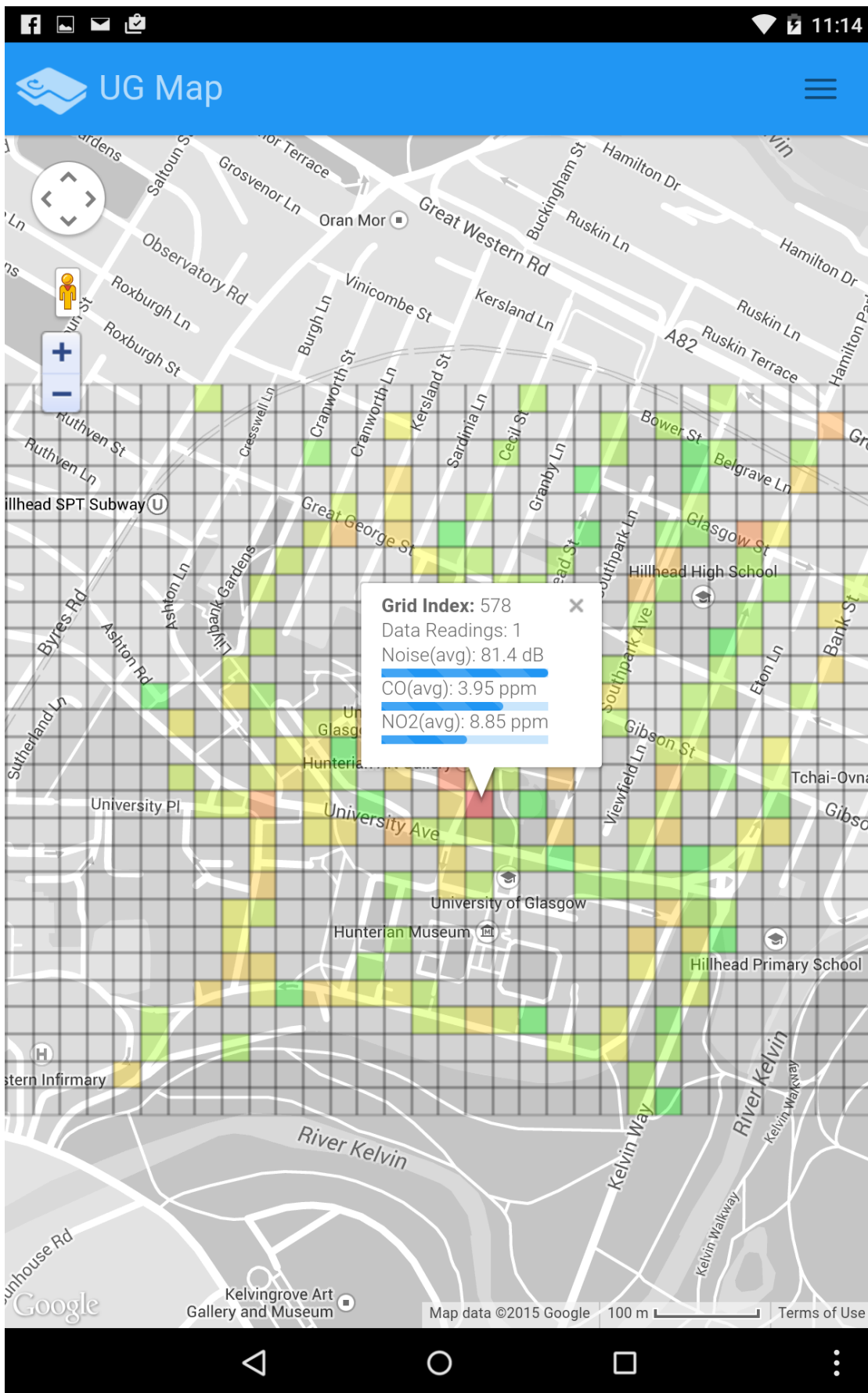
Thumbnail 70



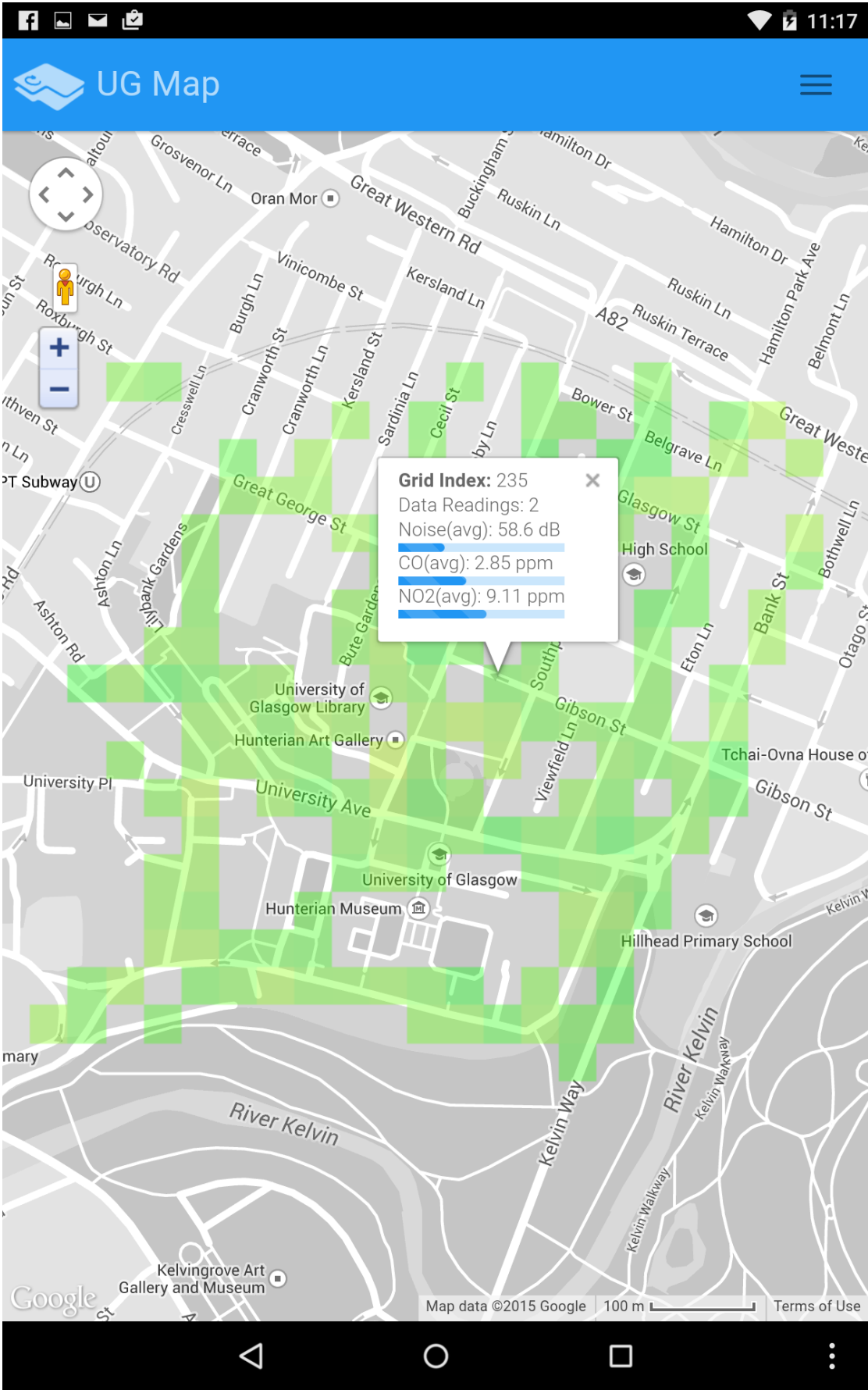
Thumbnail 71



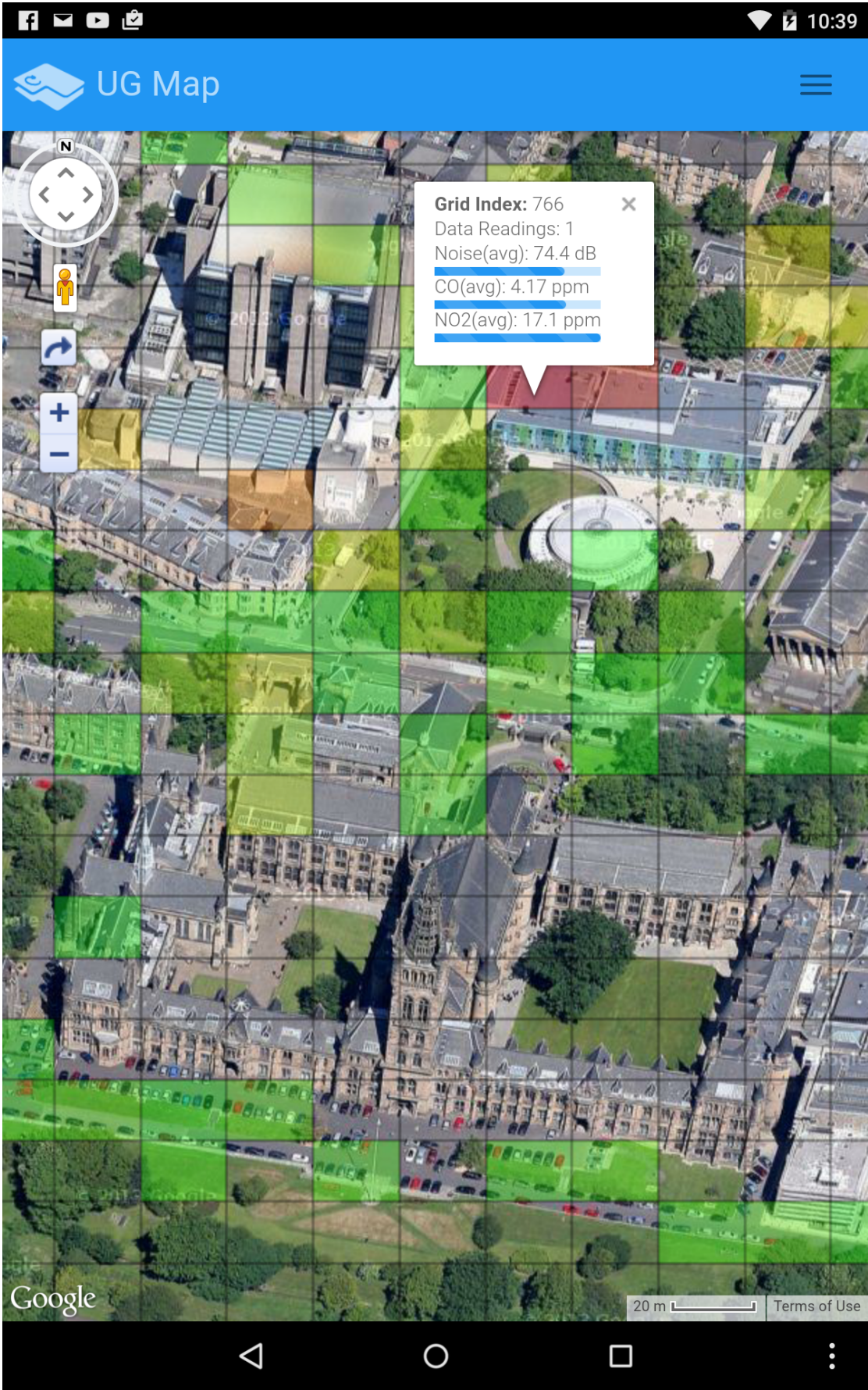
Thumbnail 72



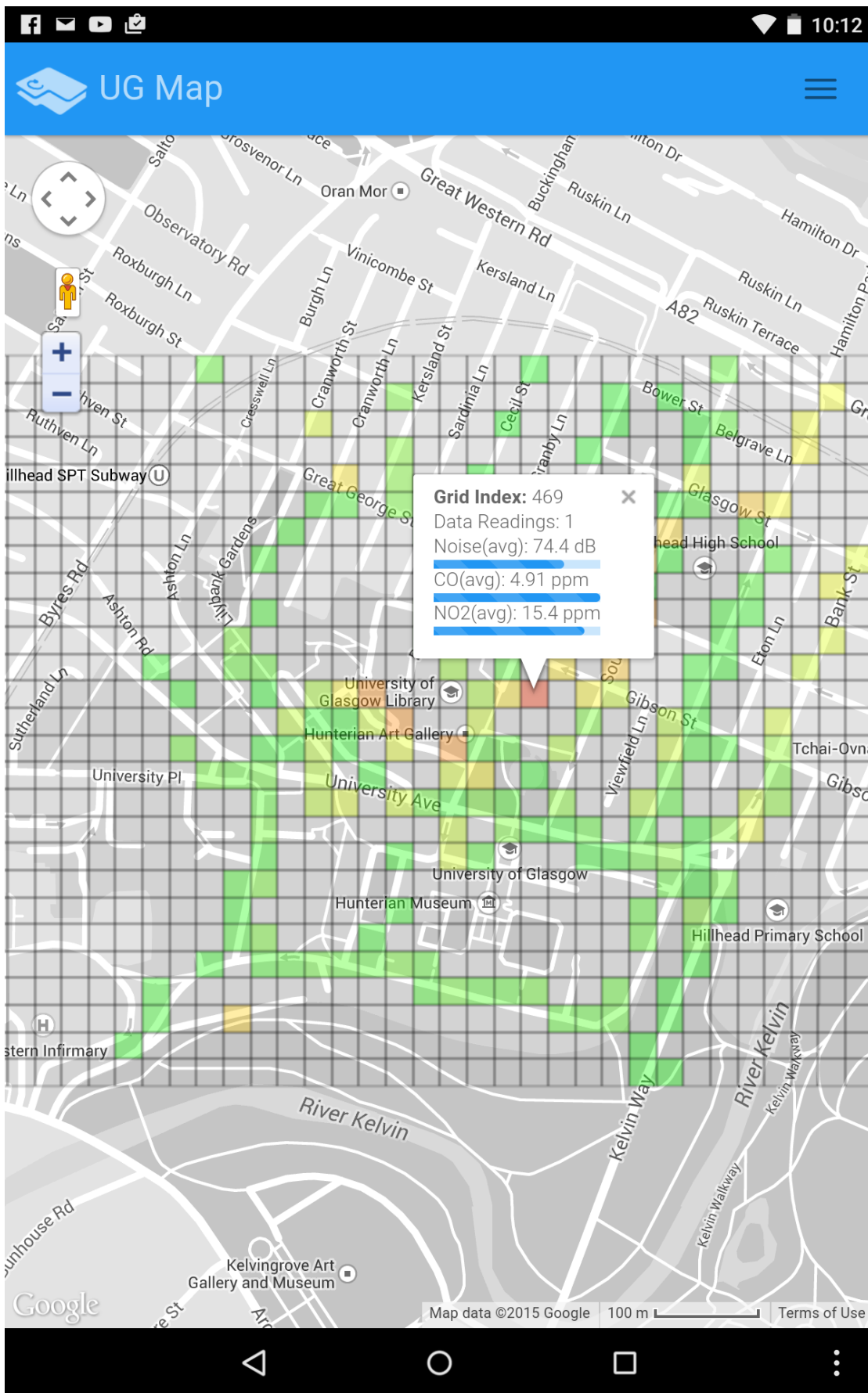
Thumbnail 73



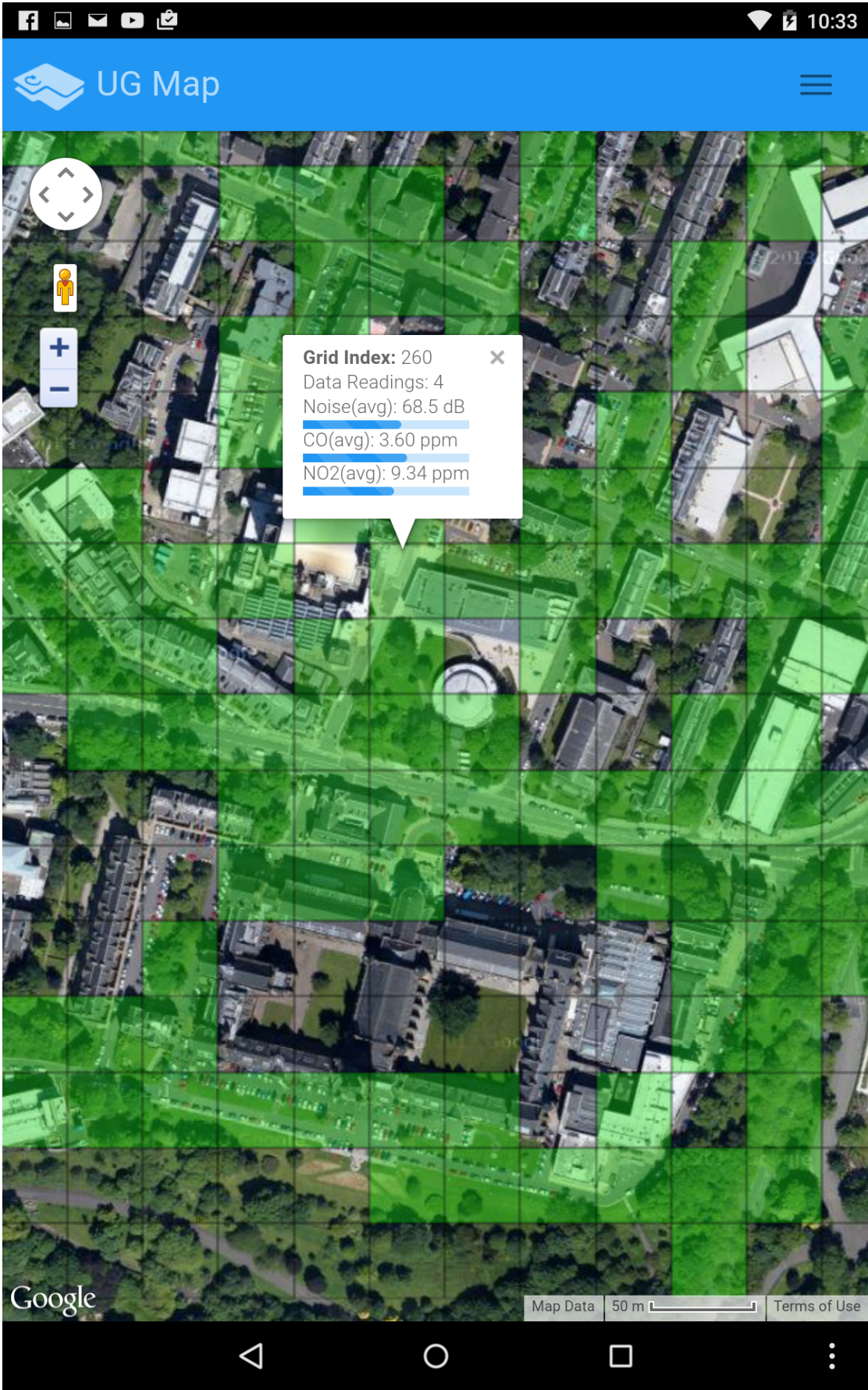
Thumbnail 74



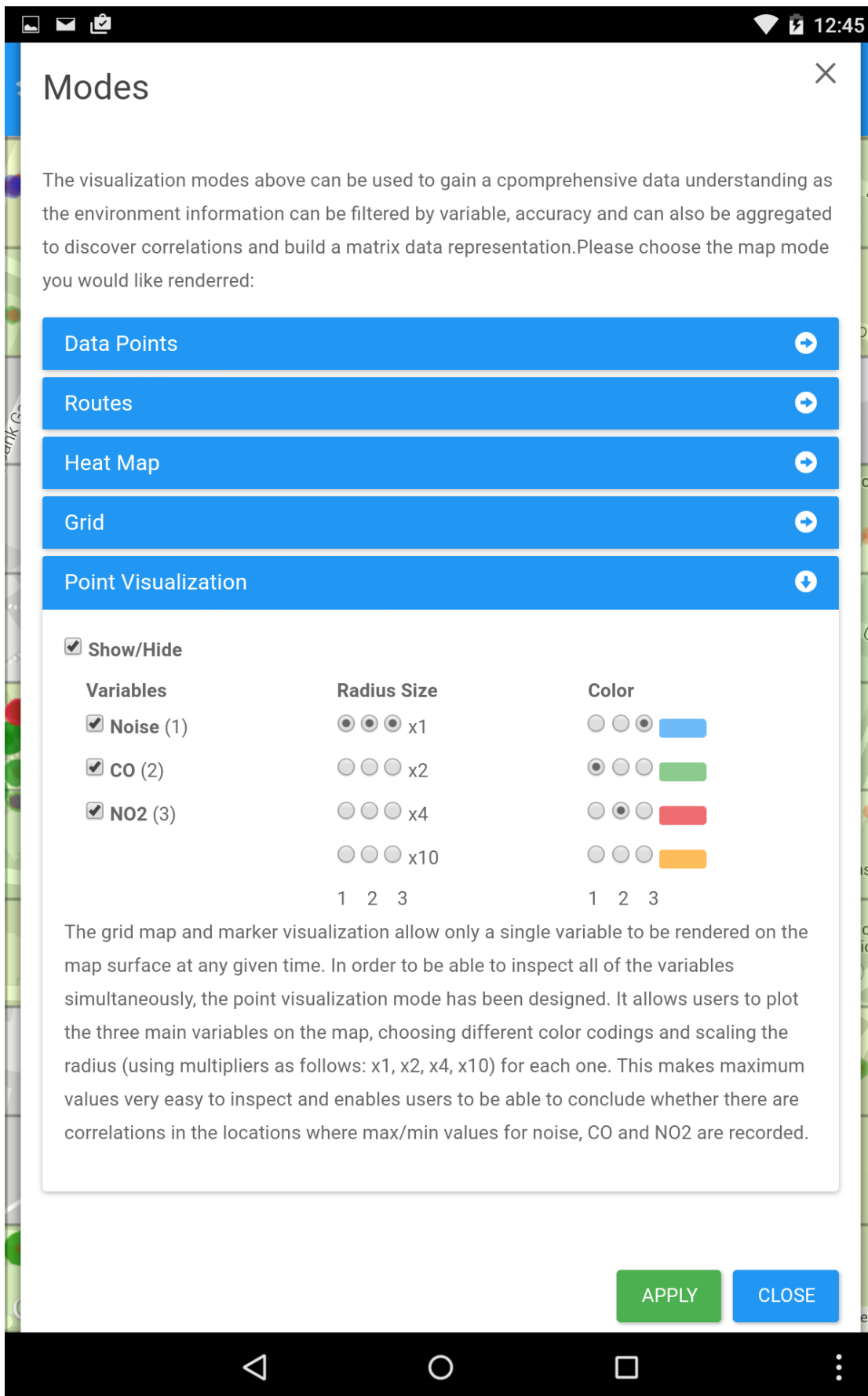
Thumbnail 75



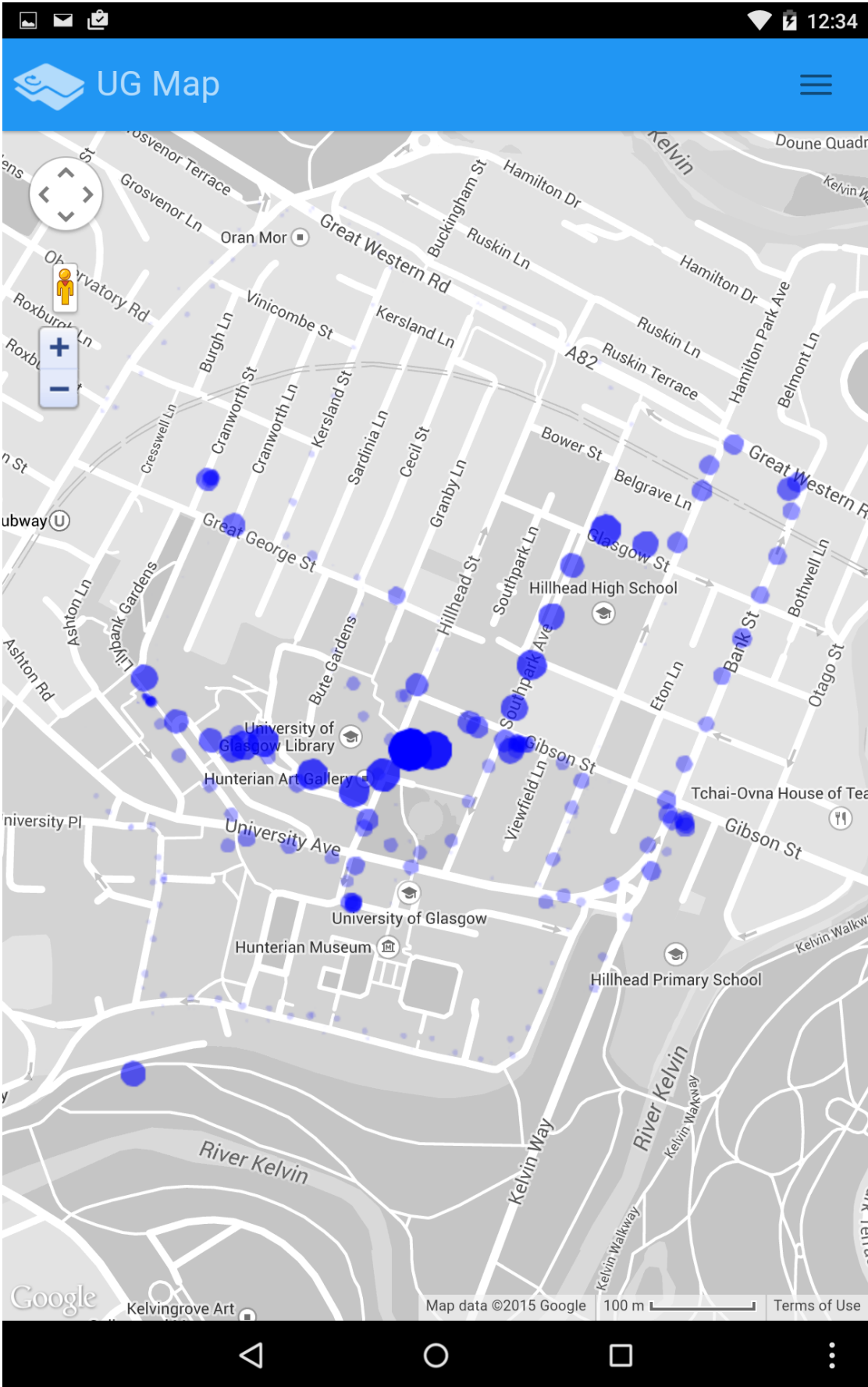
Thumbnail 76



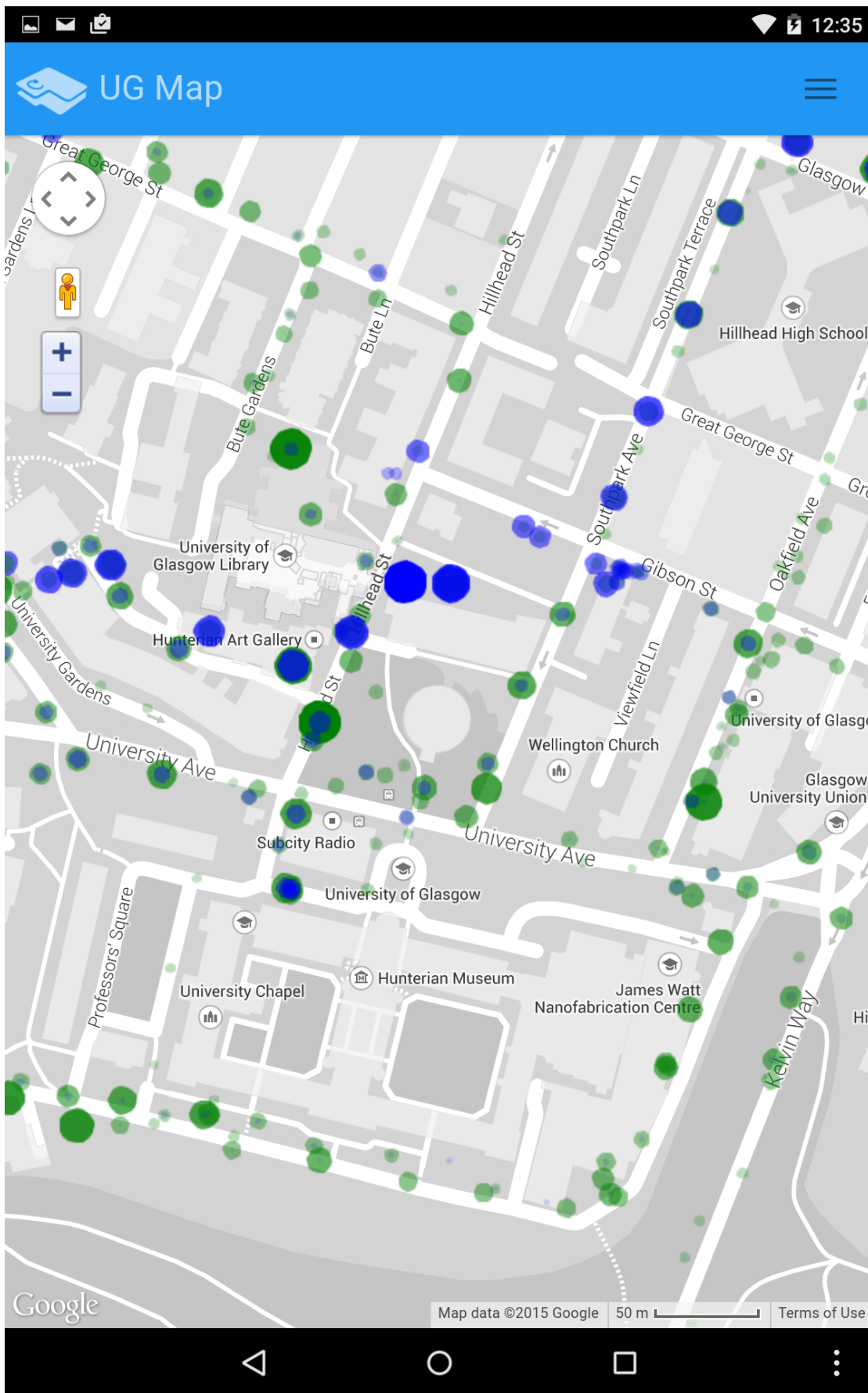
Thumbnail 77



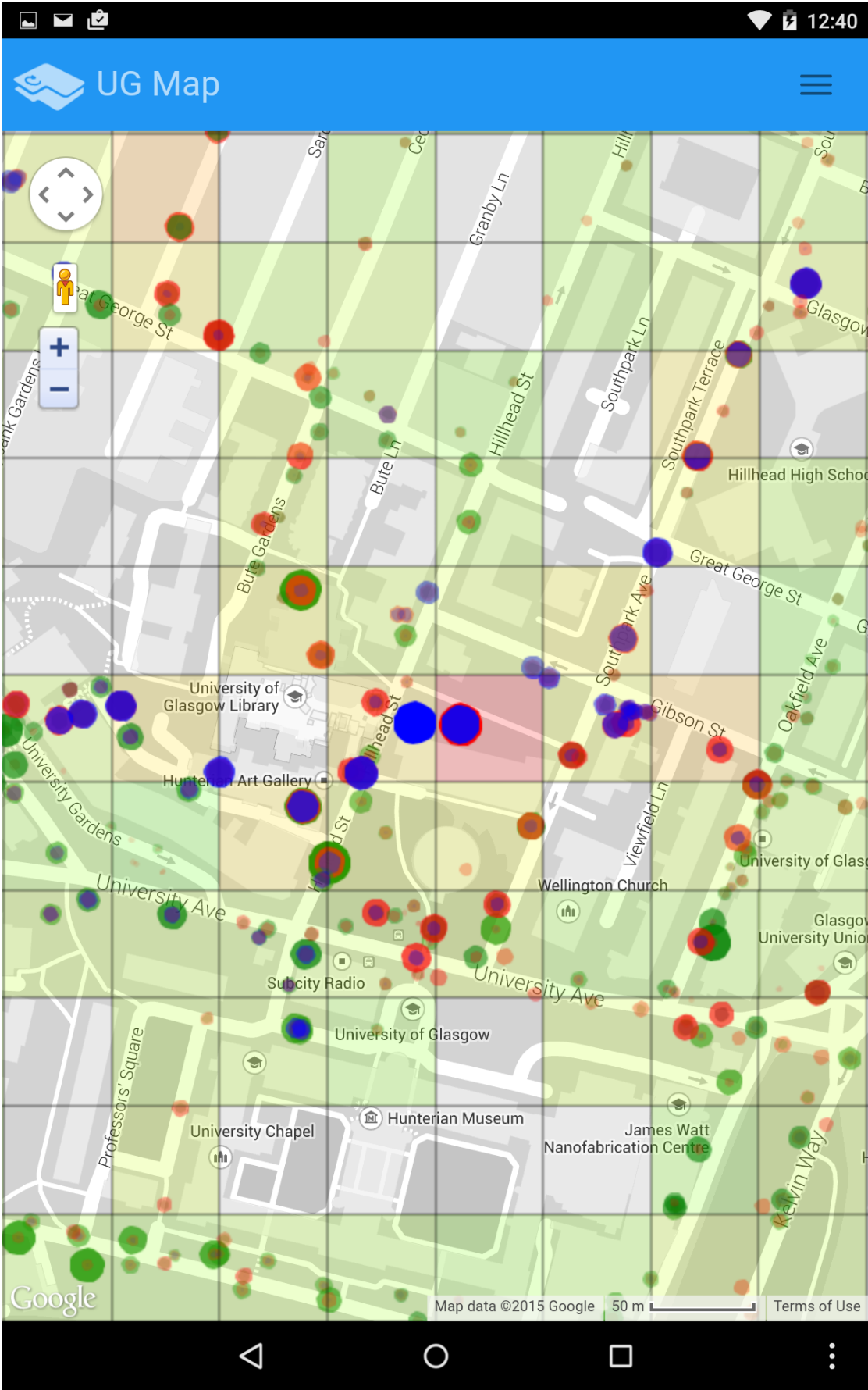
Thumbnail 78



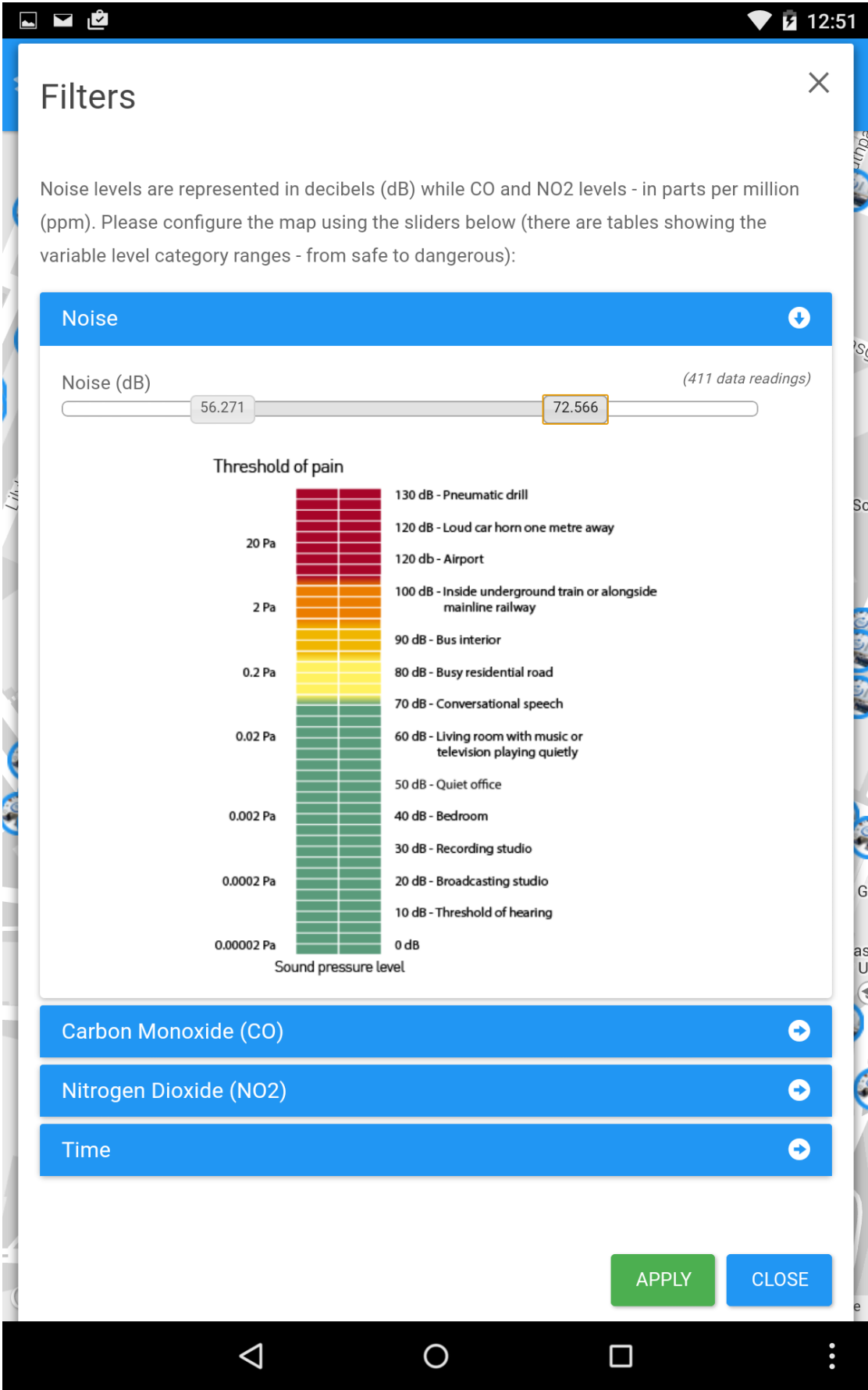
Thumbnail 79



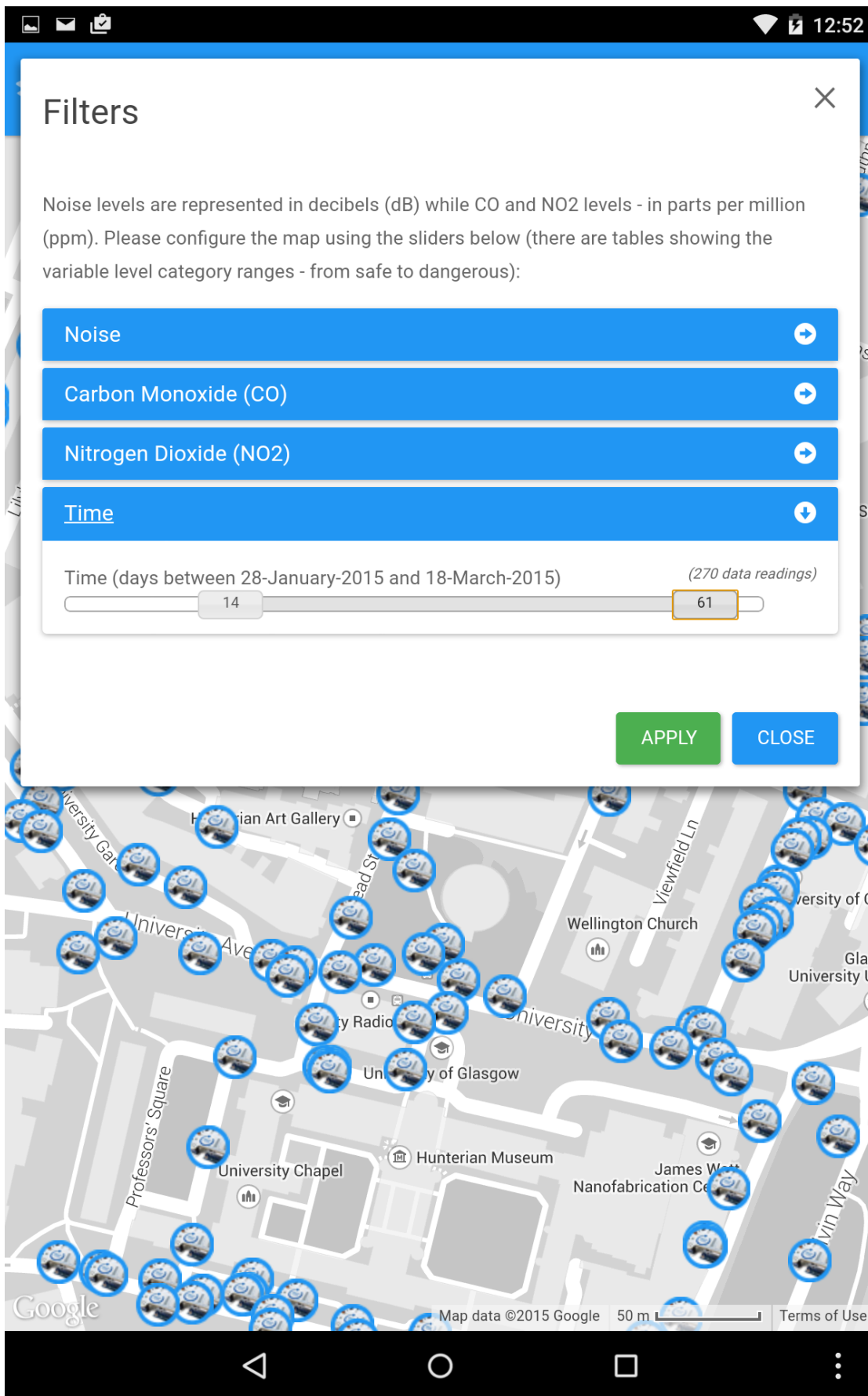
Thumbnail 80



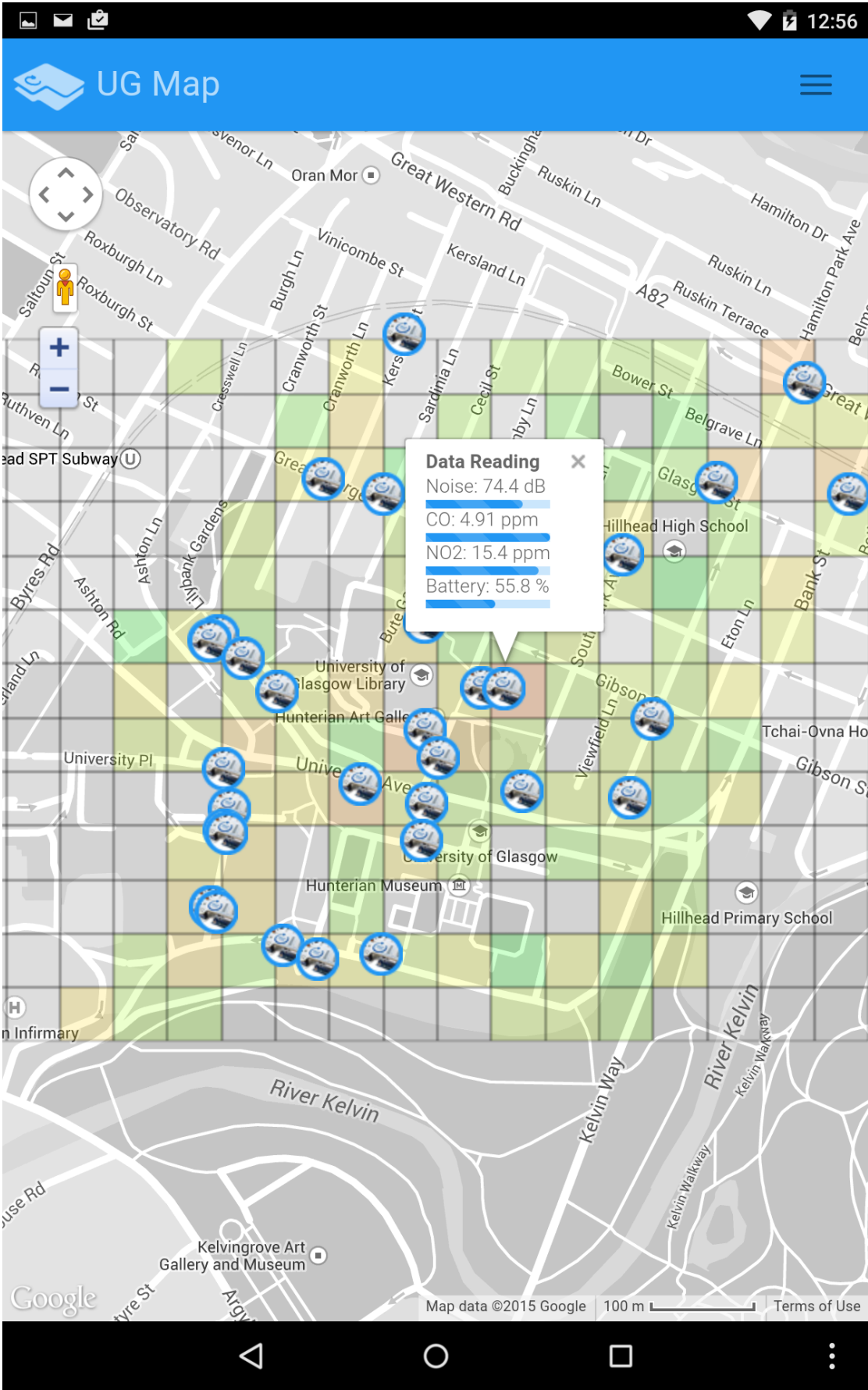
Thumbnail 81



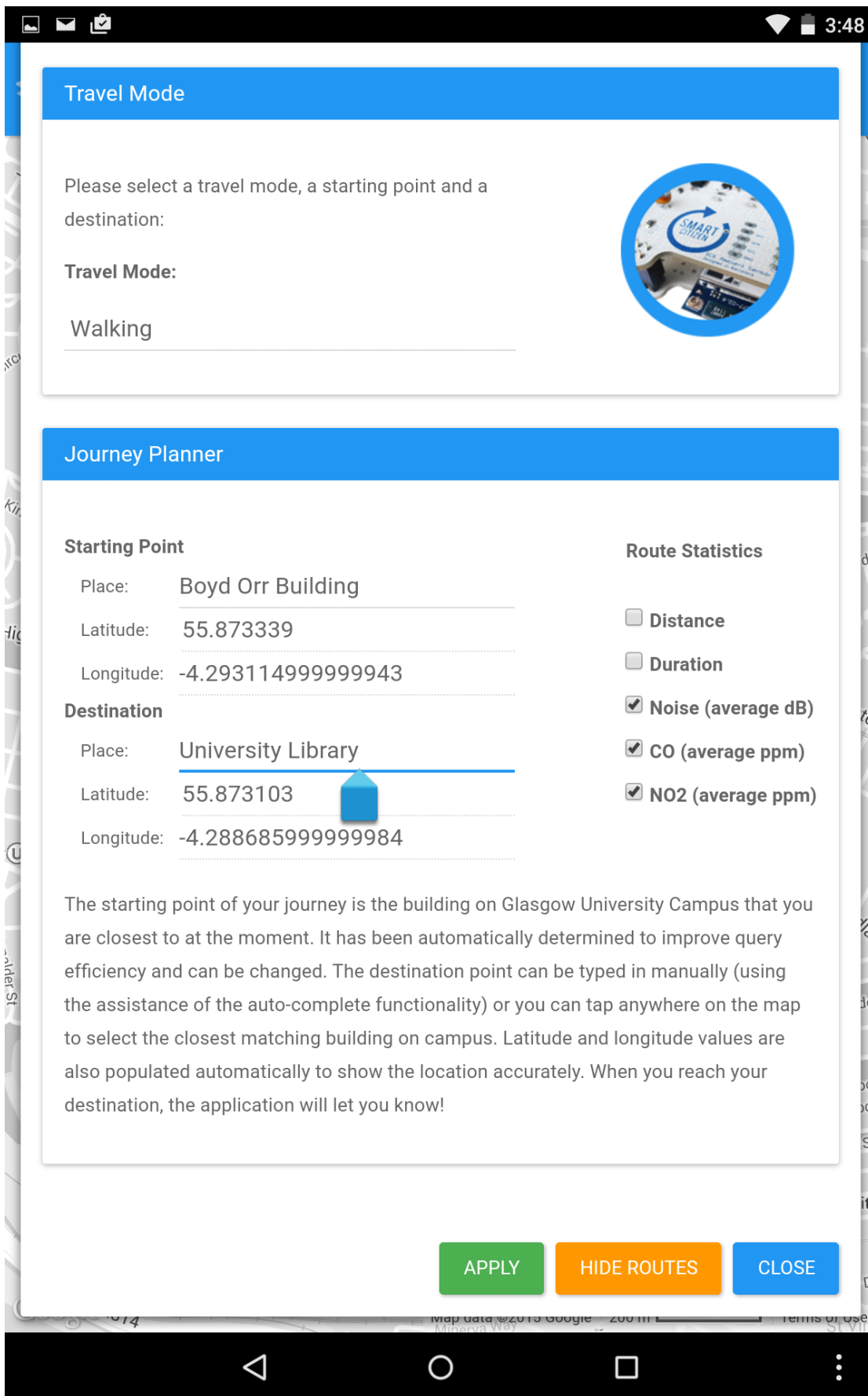
Thumbnail 82



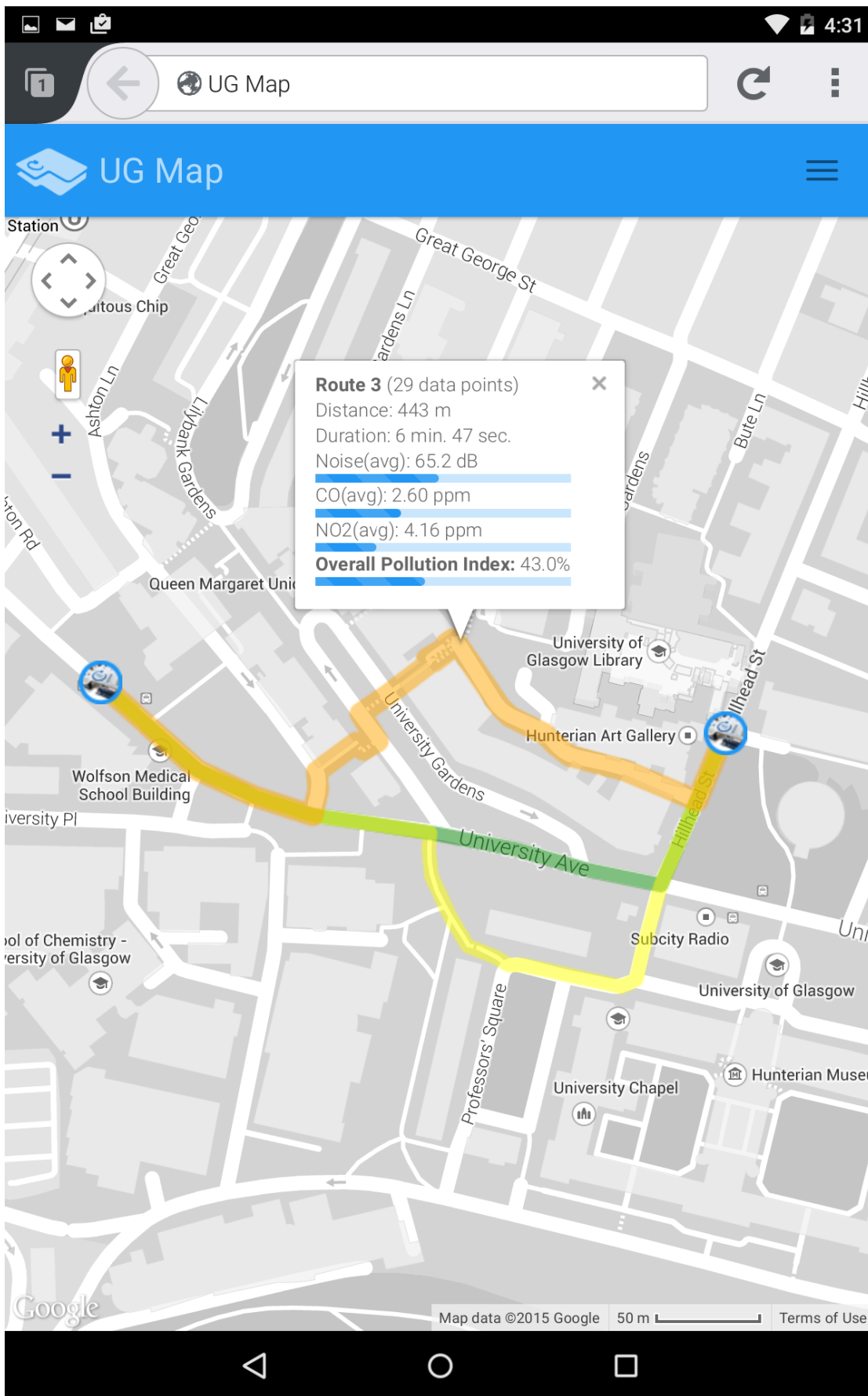
Thumbnail 83



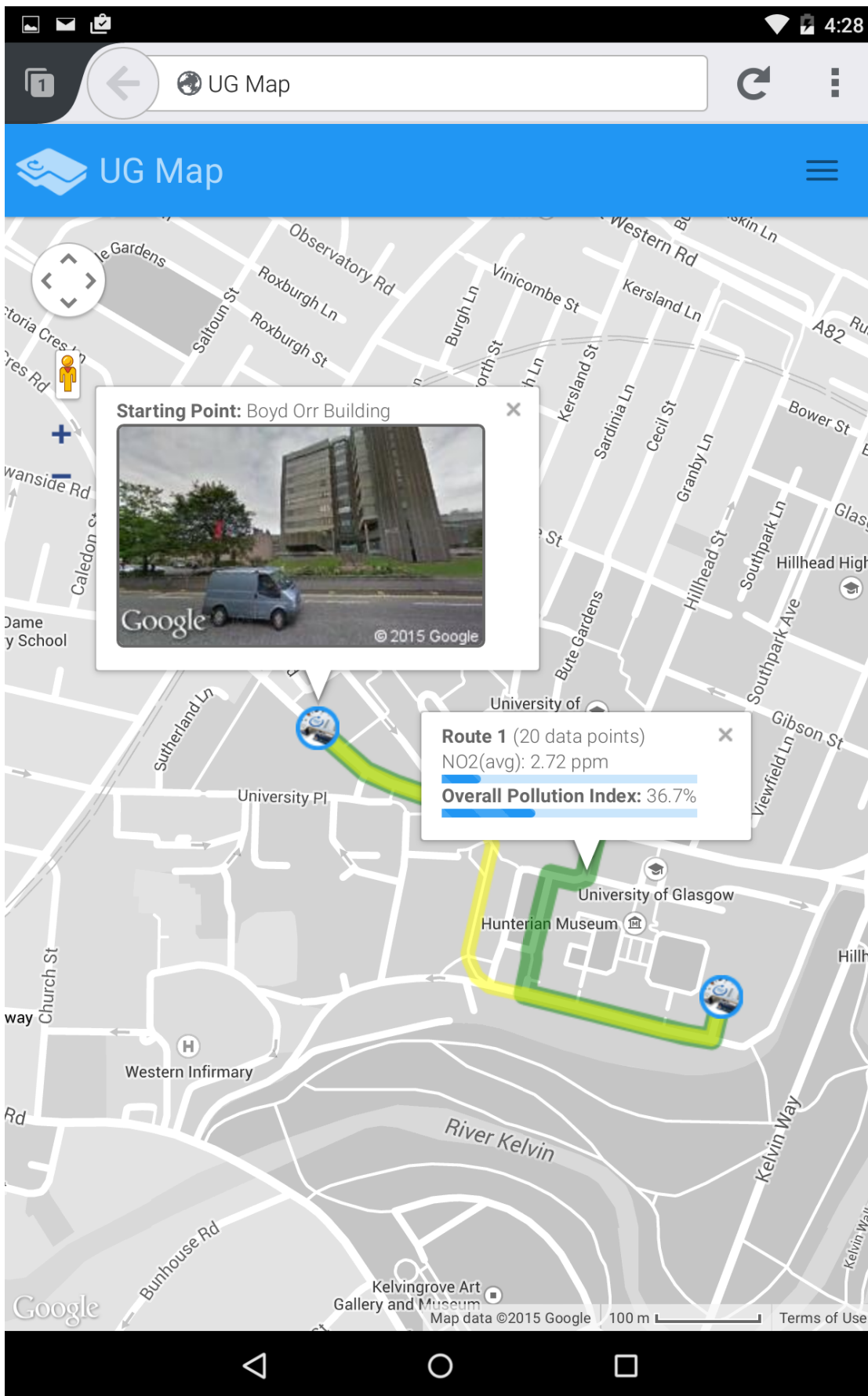
Thumbnail 84



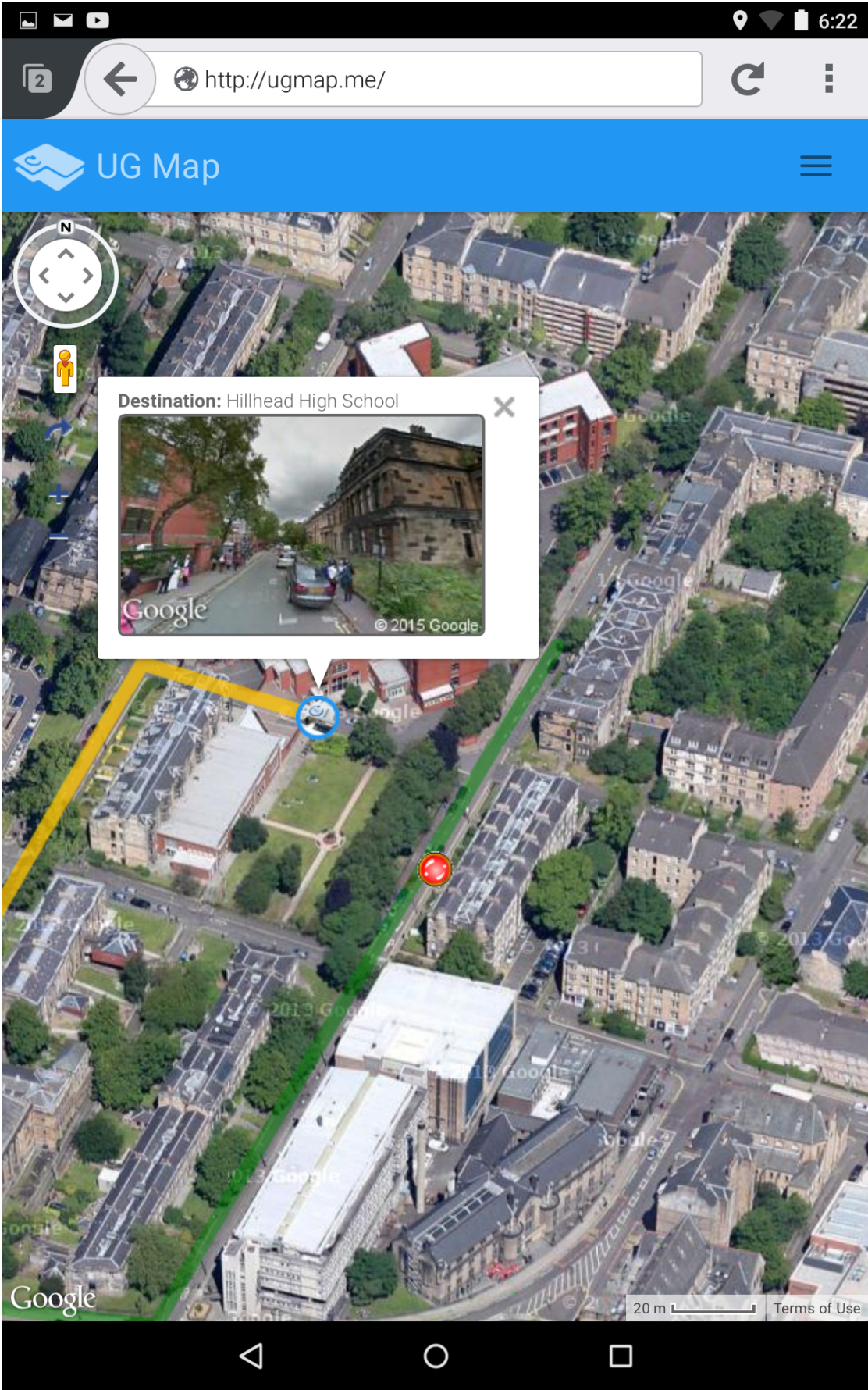
Thumbnail 85



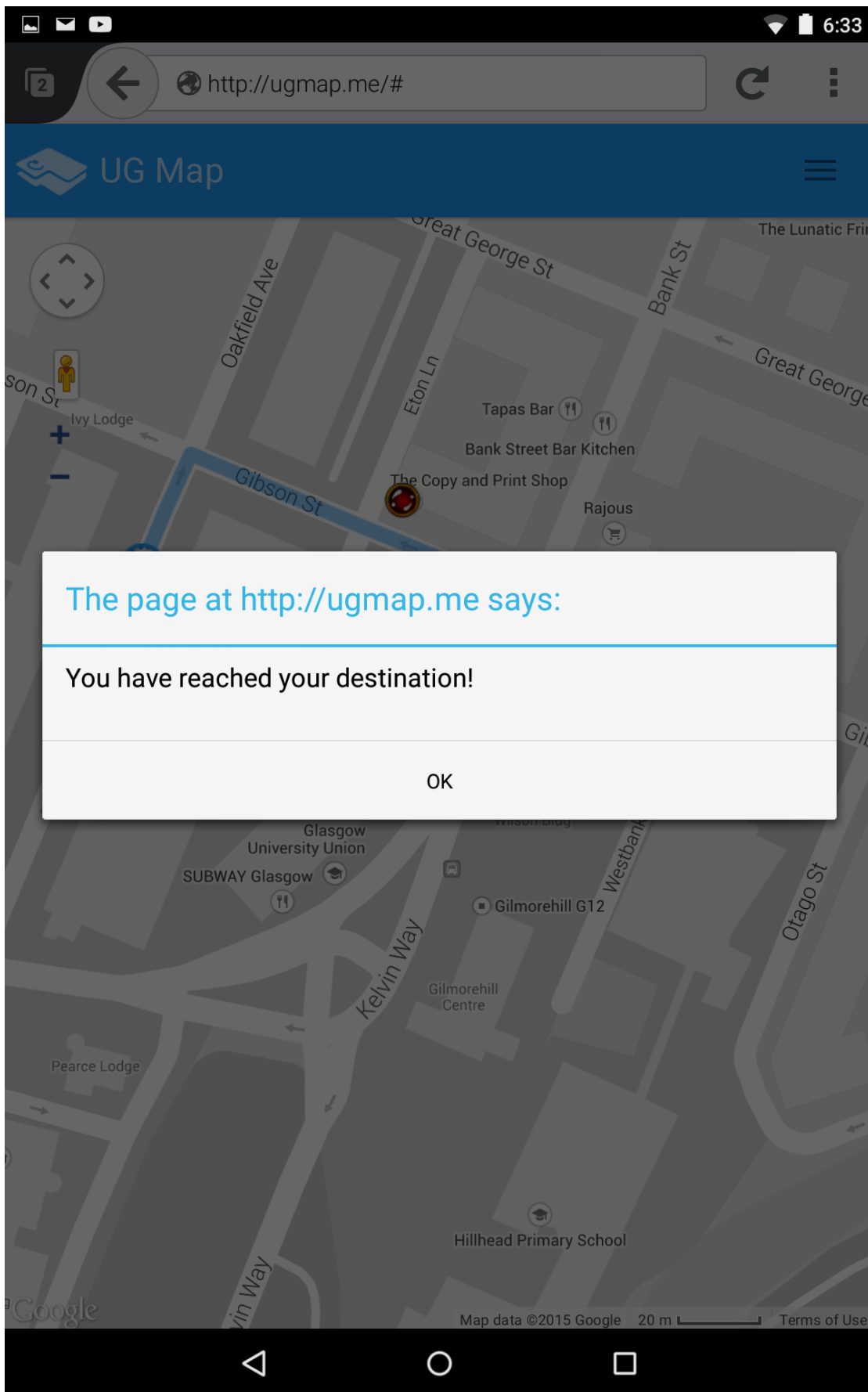
Thumbnail 86



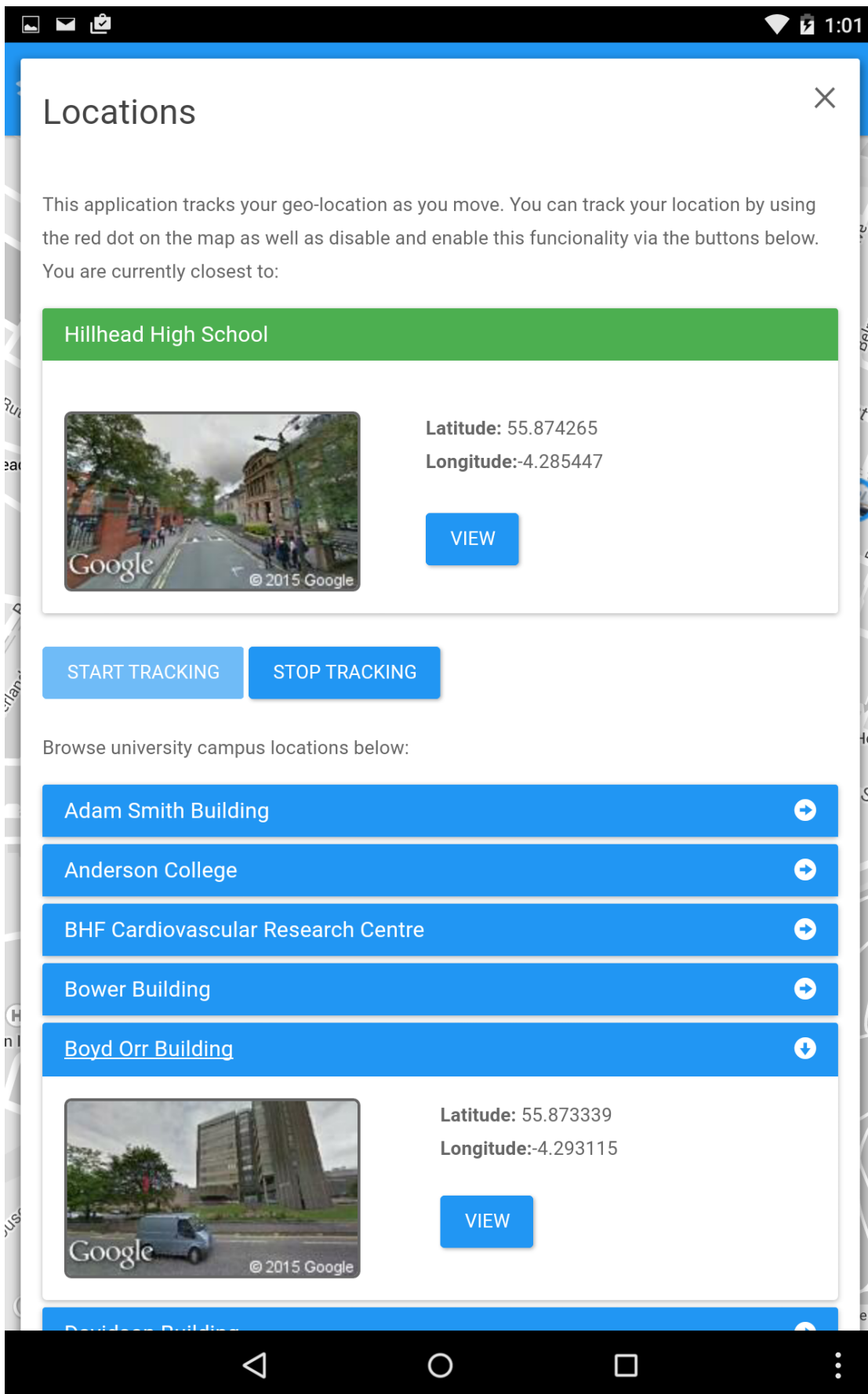
Thumbnail 87



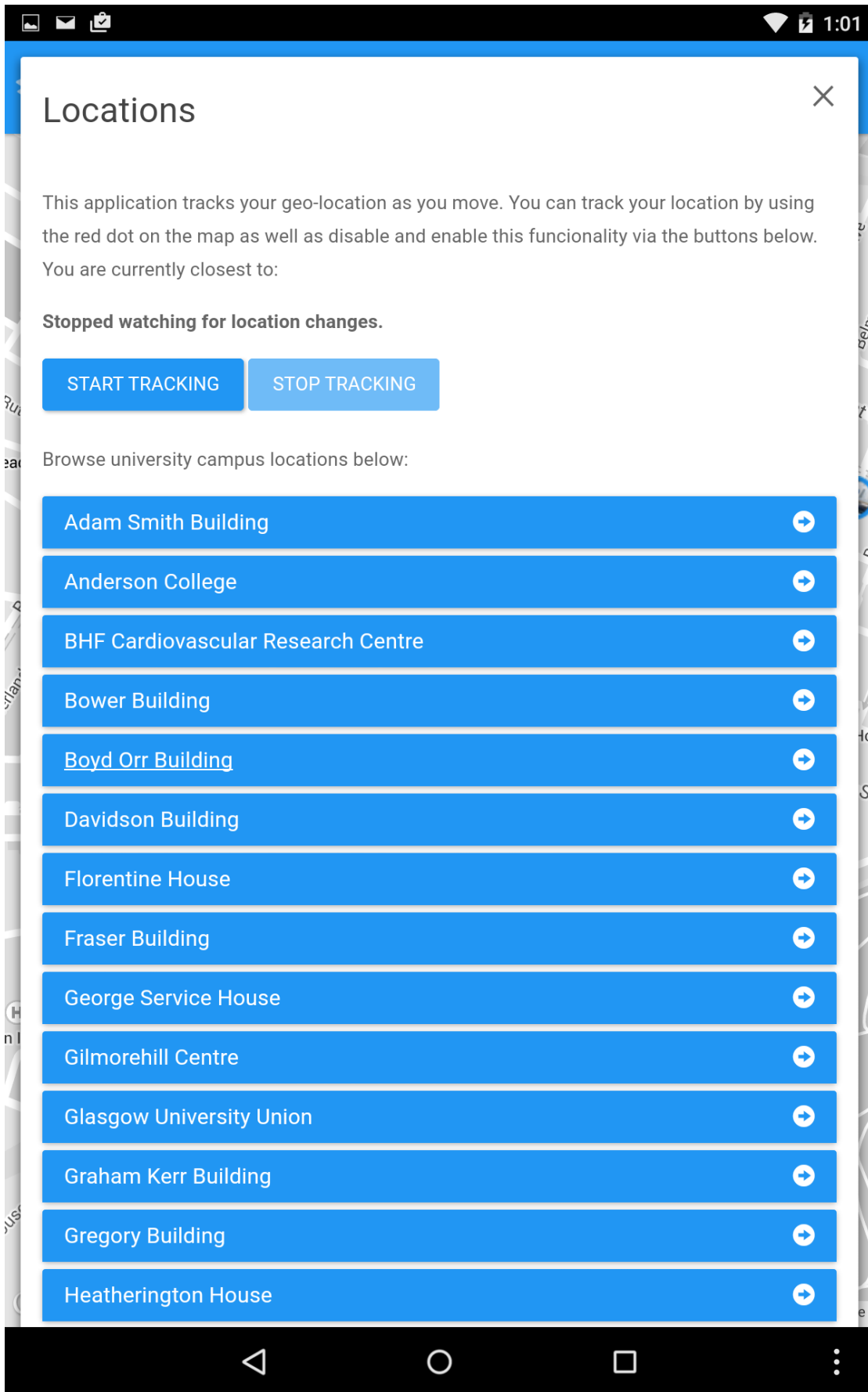
Thumbnail 88



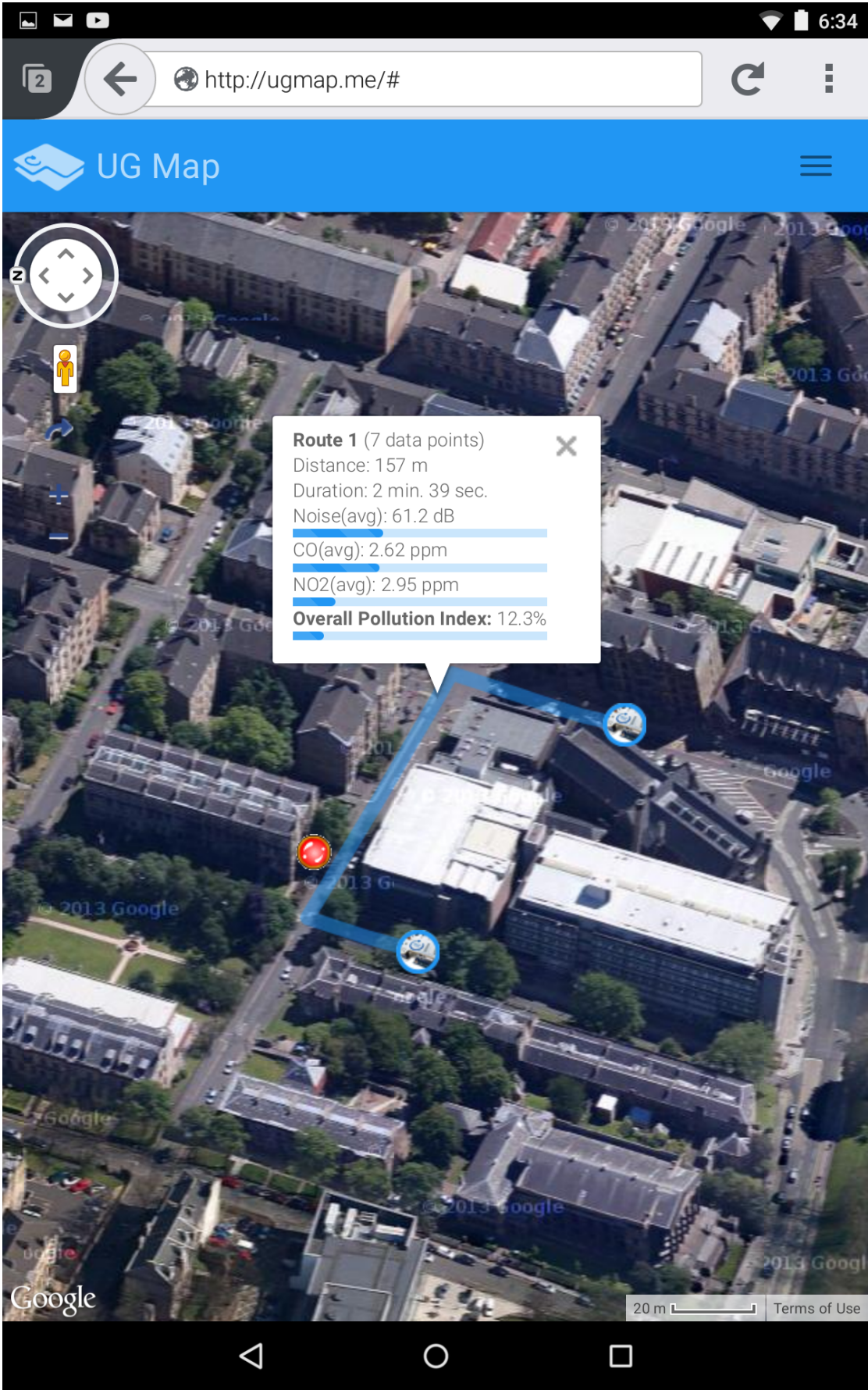
Thumbnail 89



Thumbnail 90



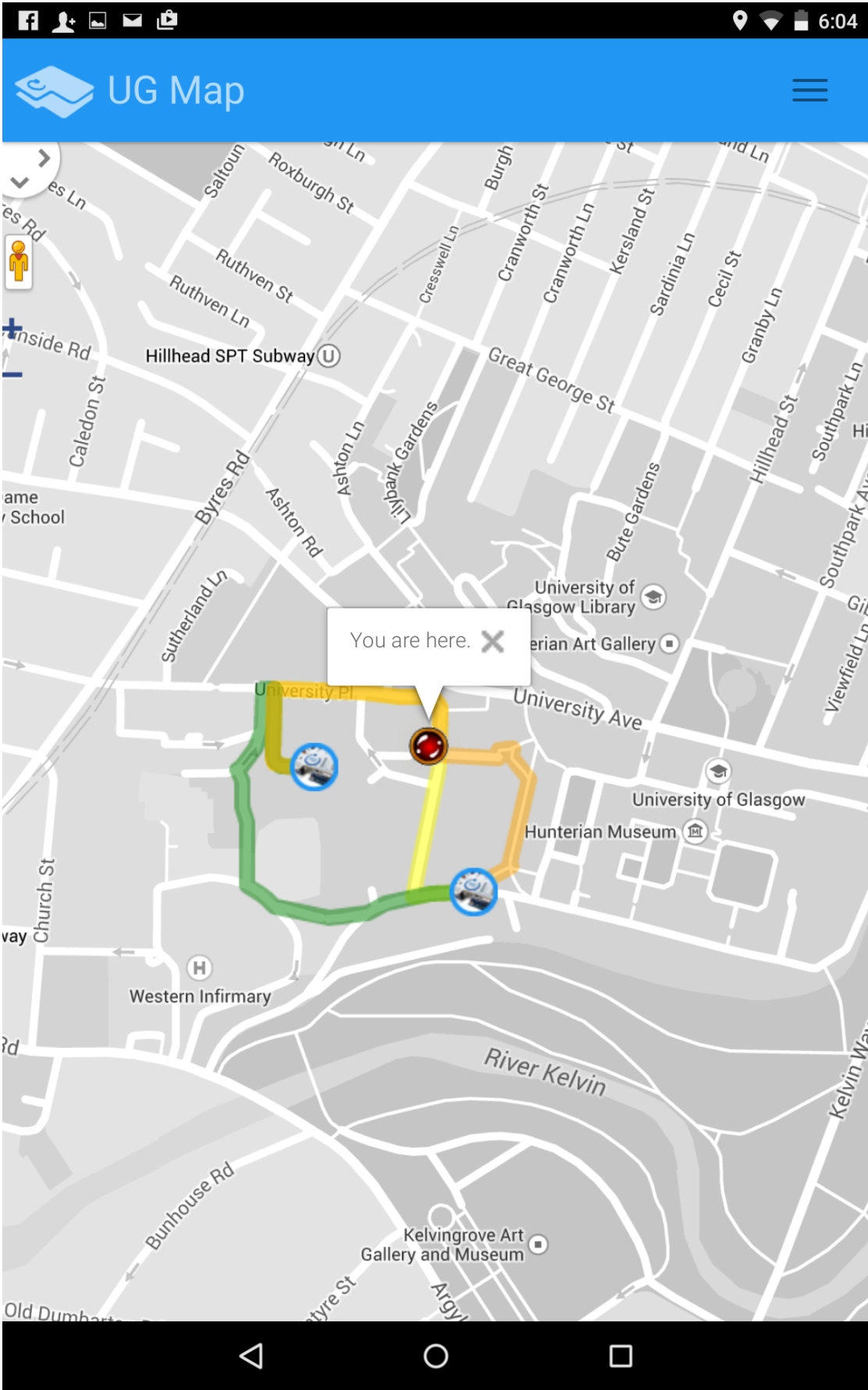
Thumbnail 91



Thumbnail 92



Thumbnail 93



Thumbnail 94

1:07

Devices

Devices that have submitted data to the server are listed below:

PP Yordanov's Kit

ID:1651


Description: Testing kit.

Location: Glasgow, United Kingdom

Kit Version:

First Access: 2014-10-01 19:13:03 UTC

Associated User: 2282



Users

Users who have transmitted data to the web application:

ppyordanov

ID:2282


City: Glasgow

Country: United Kingdom

Website: <http://ppyordanov.com>

Email: ppyordanov@yahoo.com

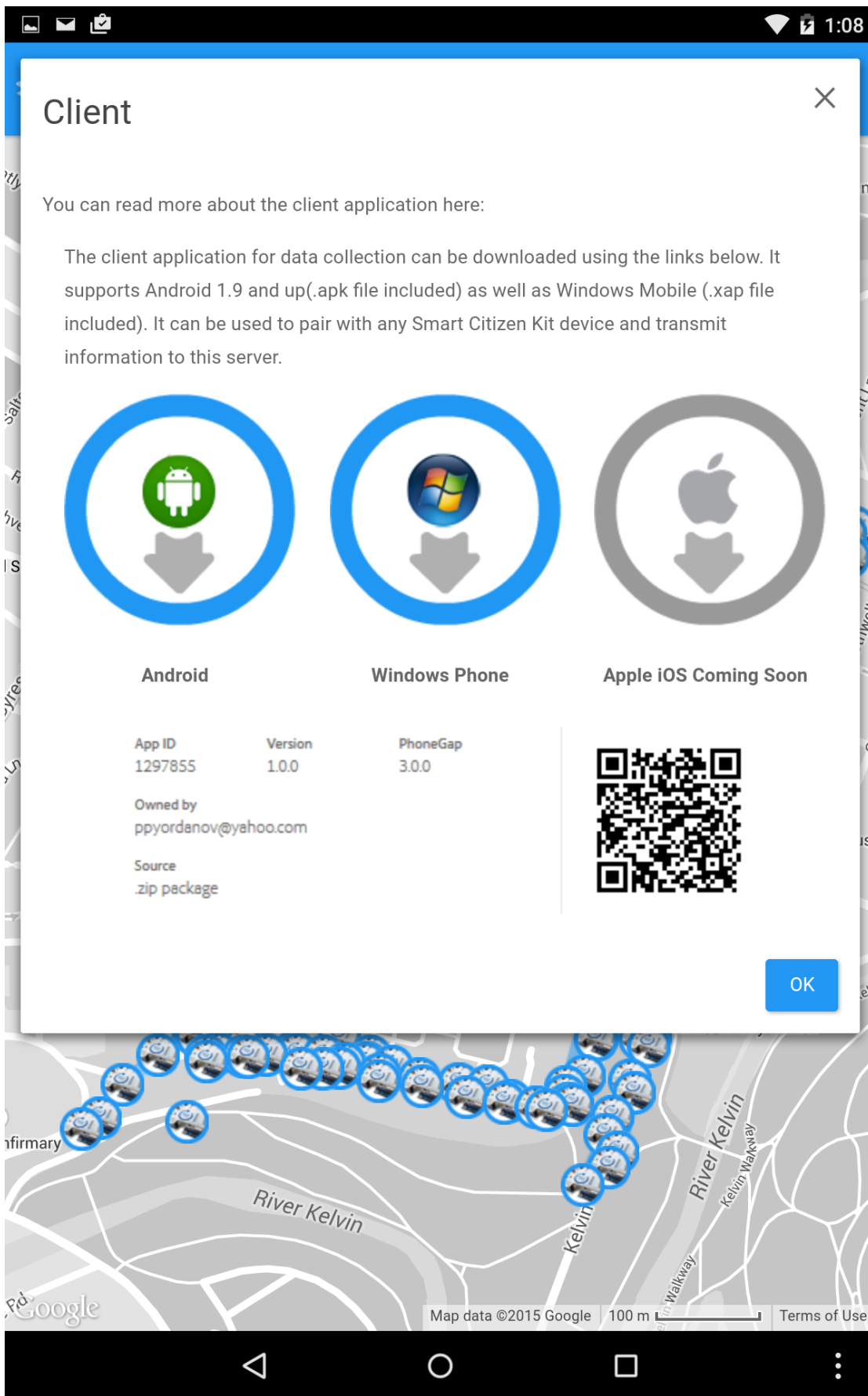
First Access: 2014-09-24 12:49:30



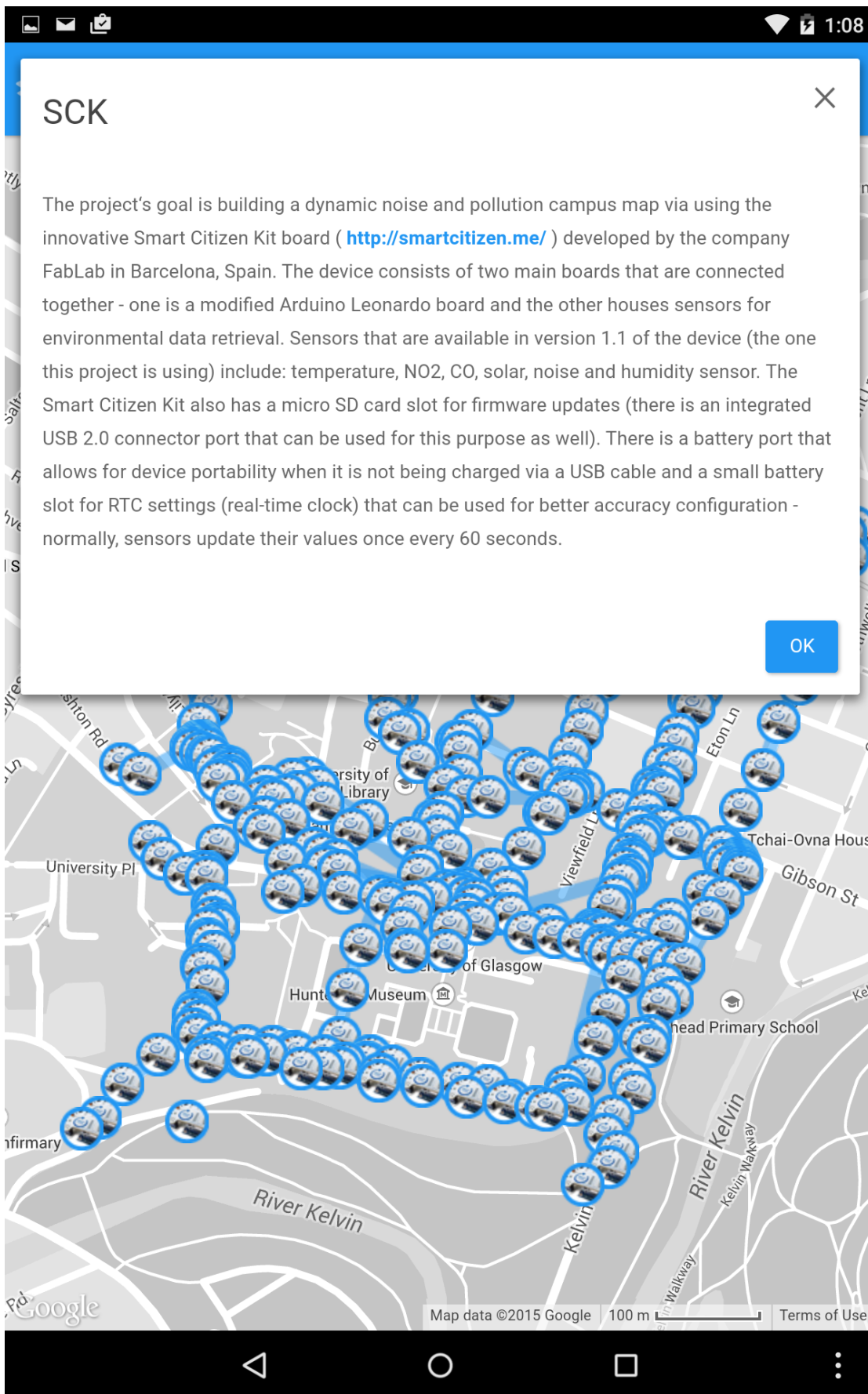
OK

Map data ©2015 Google 100 m Terms of Use

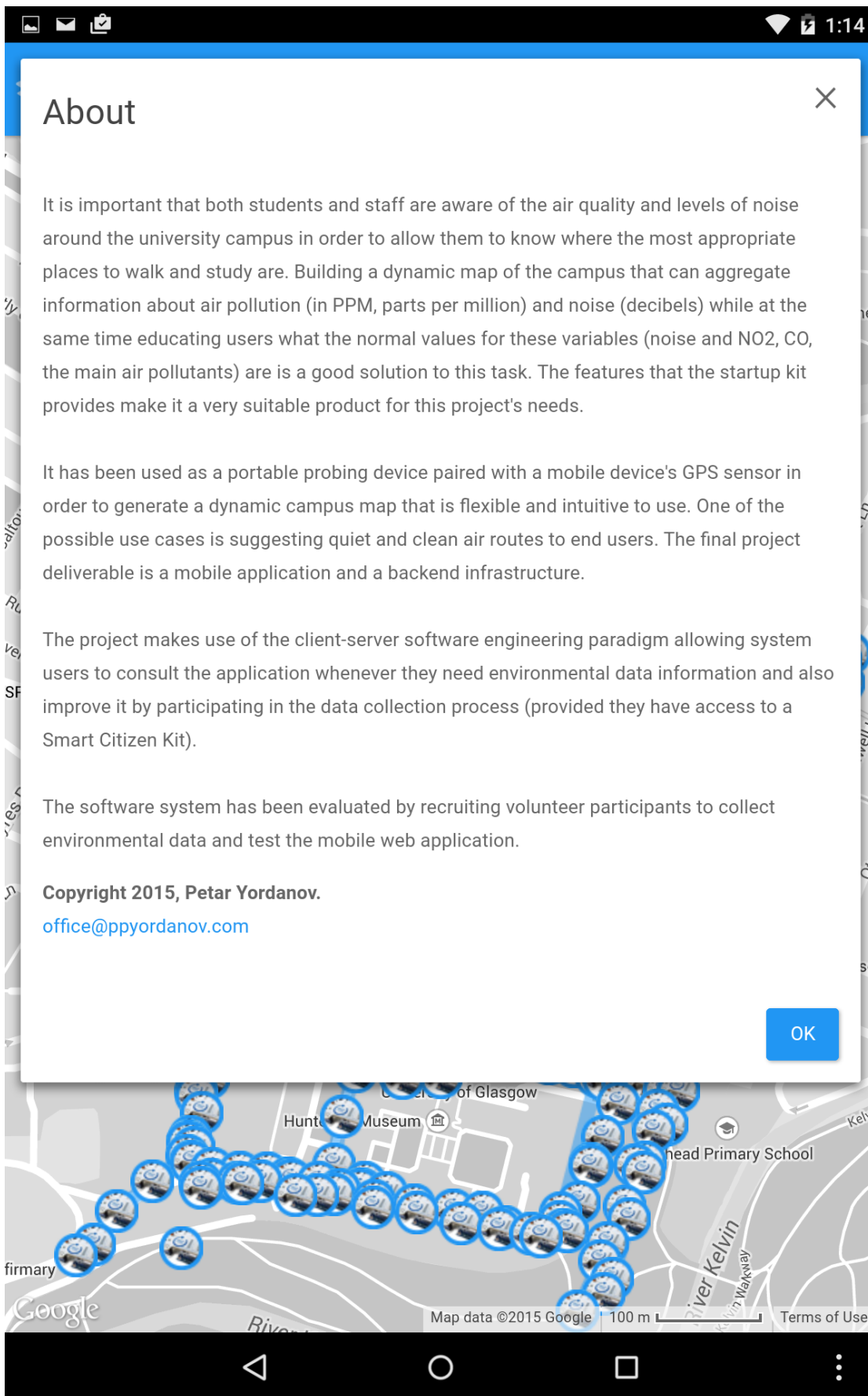
Thumbnail 95



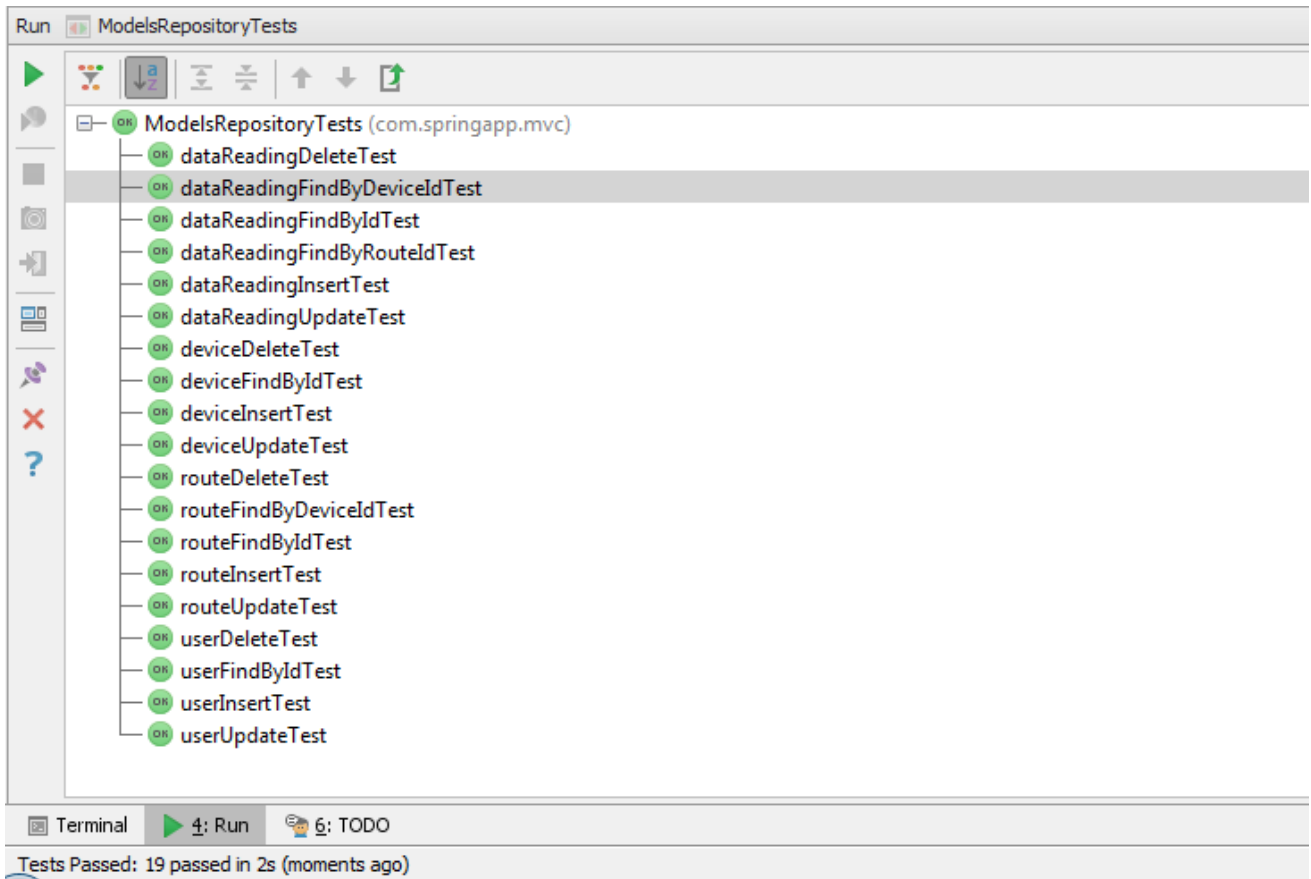
Thumbnail 96



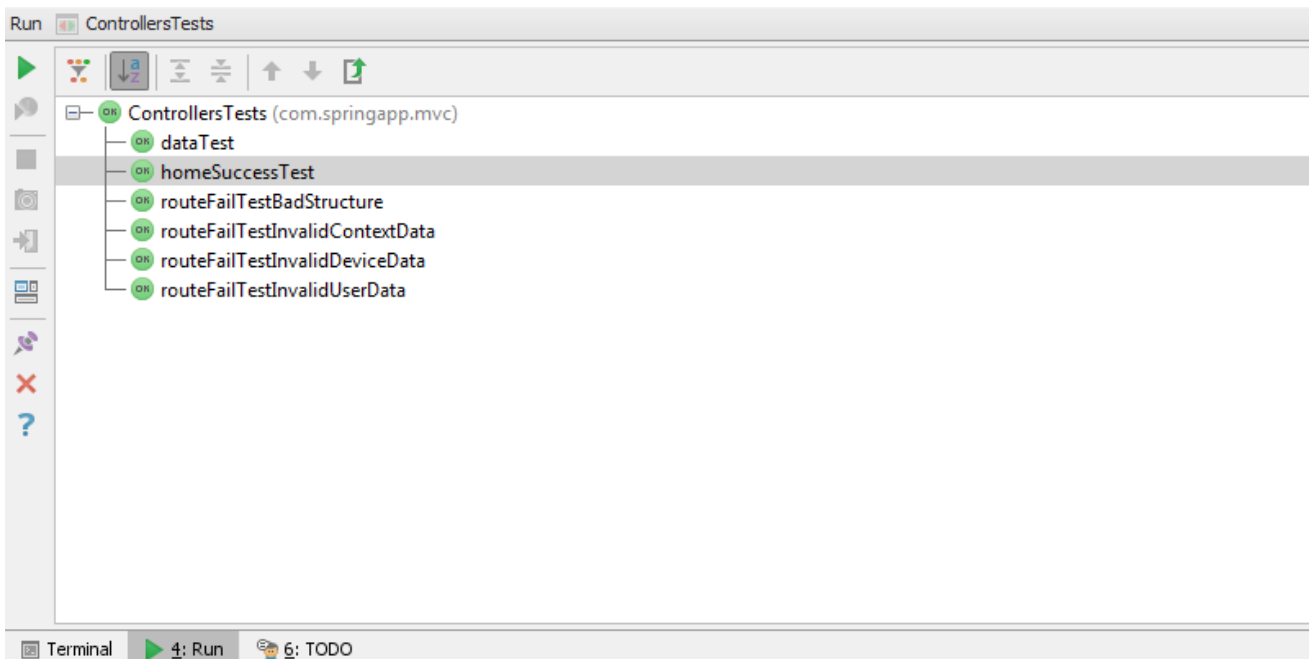
Thumbnail 97



Thumbnail 98



Thumbnail 99



Thumbnail 100

The screenshot shows a Google Maps interface with a grid of thumbnail images overlaid on a map of Glasgow. The map includes labels for 'The West End Festival (Glasgow)', 'Notre Dame Primary School', 'University of Glasgow', 'Huterian Museum', 'St Mary's Cathedral', and 'Stravaigin 11'. The thumbnail grid is composed of small circular icons, many of which are blue and white, arranged in a pattern that follows the street layout. The interface includes standard Google Maps controls like a compass, street view pegman, and zoom in/out buttons. Below the map, the Chrome DevTools console is open, displaying a large array of 'rgb' color values and some object references.

```

map.is:150
▶ [{"rgb": "rgb(2,252,0)", "rgb(5,249,0)", "rgb(7,247,0)", "rgb(10,244,0)", "rgb(12,242,0)", "rgb(15,239,0)", "rgb(17,237,0)", "rgb(20,234,0)", "rgb(22,232,0)", "rgb(25,229,0)", "rgb(28,226,0)", "rgb(30,224,0)", "rgb(33,221,0)", "rgb(35,219,0)", "rgb(38,216,0)", "rgb(40,214,0)", "rgb(43,211,0)", "rgb(45,209,0)", "rgb(48,206,0)", "rgb(51,204,0)", "rgb(53,201,0)", "rgb(56,198,0)", "rgb(58,196,0)", "rgb(61,193,0)", "rgb(63,191,0)", "rgb(66,188,0)", "rgb(68,186,0)", "rgb(71,183,0)", "rgb(73,181,0)", "rgb(76,178,0)", "rgb(79,175,0)", "rgb(81,173,0)", "rgb(84,170,0)", "rgb(86,168,0)", "rgb(89,165,0)", "rgb(91,163,0)", "rgb(94,160,0)", "rgb(96,158,0)", "rgb(99,155,0)", "rgb(102,153,0)", "rgb(104,150,0)", "rgb(107,147,0)", "rgb(109,145,0)", "rgb(112,142,0)", "rgb(114,140,0)", "rgb(117,137,0)", "rgb(119,135,0)", "rgb(122,132,0)", "rgb(124,130,0)", "rgb(127,127,0)", "rgb(130,124,0)", "rgb(132,122,0)", "rgb(135,119,0)", "rgb(137,117,0)", "rgb(140,114,0)", "rgb(142,112,0)", "rgb(145,109,0)", "rgb(147,107,0)", "rgb(150,104,0)", "rgb(153,102,0)", "rgb(155,99,0)", "rgb(158,96,0)", "rgb(160,94,0)", "rgb(163,91,0)", "rgb(165,89,0)", "rgb(168,86,0)", "rgb(170,84,0)", "rgb(173,81,0)", "rgb(175,79,0)", "rgb(178,76,0)", "rgb(181,73,0)", "rgb(183,71,0)", "rgb(186,68,0)", "rgb(188,66,0)", "rgb(191,63,0)", "rgb(193,61,0)", "rgb(196,58,0)", "rgb(198,56,0)", "rgb(201,53,0)", "rgb(204,51,0)", "rgb(206,48,0)", "rgb(209,45,0)", "rgb(211,43,0)", "rgb(214,40,0)", "rgb(216,38,0)", "rgb(219,35,0)", "rgb(221,33,0)", "rgb(224,30,0)", "rgb(226,28,0)", "rgb(229,25,0)", "rgb(232,22,0)", "rgb(234,20,0)", "rgb(237,17,0)", "rgb(239,15,0)", "rgb(242,12,0)", "rgb(244,10,0)", "rgb(247,7,0)", "rgb(249,5,0)", "rgb(252,2,0)"}]
(index):5868
▶ Object {548a6a8ce4b07808853bd0a4: Array[1], 548a6a8ce4b07808853bd0a6: Array[5], 548a6560e4b07808853bd0ac: Array[1], 548a6a8ce4b07808853bd0ae: Array[22], 548a696e4b07808853bd0c5: Array[2]}
Page Loading Time: 549 ms
map.is:86

```

Thumbnail 101

Test MySQL DB



DATA: skipped

RESULT:

Insert single record:

Single insert in tables DEVICE and DATA_READING: Execution time: 0 min, 0 sec, 222 millisec

Insert real data (SIM):

Execution time: 0 min, 37 sec, 115 millisec **Data size:**
1 * 5000

Update single record:

Execution time: 0 min, 0 sec, 84 millisec

Get single record:

Execution time: 0 min, 0 sec, 31 millisec

Delete single record:

Execution time: 0 min, 0 sec, 90 millisec

Get all records:

Execution time: 0 min, 0 sec, 989 millisec

TOTAL:

Execution time: 0 min, 38 sec, 531 millisec

Thumbnail 102

Test MySQL DB



DATA: skipped

RESULT:

Insert single record:

Single insert in tables DEVICE and DATA_READING: Execution time: 0 min, 0 sec, 246 millisec

Insert real data (SIM):

Execution time: 4 min, 9 sec, 240310 millisec **Data size:**
12 * 5000

Update single record:

Execution time: 0 min, 0 sec, 54 millisec

Get single record:

Execution time: 0 min, 0 sec, 25 millisec

Delete single record:

Execution time: 0 min, 0 sec, 17 millisec

Get all records:

Execution time: 0 min, 3 sec, 448 millisec

TOTAL:

Execution time: 4 min, 13 sec, 240101 millisec

Thumbnail 103

Test MySQL DB



DATA: skipped

RESULT:

Insert single record:

Single insert in tables DEVICE and DATA_READING: Execution time: 0 min, 0 sec, 267 millisec

Insert real data (SIM):

Execution time: 11 min, 39 sec, 643 millisec **Data size:**
40 * 5000

Update single record:

Execution time: 0 min, 0 sec, 74 millisec

Get single record:

Execution time: 0 min, 0 sec, 52 millisec

Delete single record:

Execution time: 0 min, 0 sec, 20 millisec

Get all records:

Execution time: 0 min, 13 sec, 615 millisec

TOTAL:

Execution time: 11 min, 53 sec, 674 millisec

Thumbnail 104

Test MONGO_DB



DATA: skipped

RESULT:

Insert single record:

Single insert in collections: Execution time: 0 min, 0 sec, 11 millisec

Insert real data (SIM):

Execution time: 0 min, 1 sec, 518 millisec **Data size:**

1 * 5000

Update single record:

Execution time: 0 min, 0 sec, 6 millisec

Get single record:

Execution time: 0 min, 0 sec, 37 millisec

Delete single record:

Execution time: 0 min, 0 sec, 60 millisec

Get all records:

Execution time: 0 min, 0 sec, 649 millisec

TOTAL:

Execution time: 0 min, 2 sec, 281 millisec

Thumbnail 105

Test MONGO_DB

powered by



DATA: skipped

RESULT:

Insert single record:

Single insert in collections: Execution time: 0 min, 0 sec, 9 millisec

Insert real data (SIM):

Execution time: 0 min, 10 sec, 420 millisec **Data size:**

12 * 5000

Update single record:

Execution time: 0 min, 0 sec, 5 millisec

Get single record:

Execution time: 0 min, 0 sec, 25 millisec

Delete single record:

Execution time: 0 min, 0 sec, 7 millisec

Get all records:

Execution time: 0 min, 0 sec, 504 millisec

TOTAL:

Execution time: 0 min, 10 sec, 971 millisec

Test MONGO_DB



DATA: skipped

RESULT:

Insert single record:

Single insert in collections: Execution time: 0 min, 0 sec, 11 millisec

Insert real data (SIM):

Execution time: 0 min, 14 sec, 549 millisec **Data size:**
40 * 5000

Update single record:

Execution time: 0 min, 0 sec, 4 millisec

Get single record:

Execution time: 0 min, 0 sec, 21 millisec

Delete single record:

Execution time: 0 min, 0 sec, 5 millisec

Get all records:

Execution time: 0 min, 4 sec, 107 millisec

TOTAL:

Execution time: 0 min, 18 sec, 697 millisec

Test MONGO_DB



DATA: skipped

RESULT:

Insert single record:

Single insert in collections: Execution time: 0 min, 0 sec, 8 millisec

Insert real data (SIM):

Execution time: 5 min, 4 sec, 833 millisec **Data size:**
1000 * 5000

Update single record:

Execution time: 0 min, 0 sec, 4 millisec

Get single record:

Execution time: 0 min, 0 sec, 20 millisec

Delete single record:

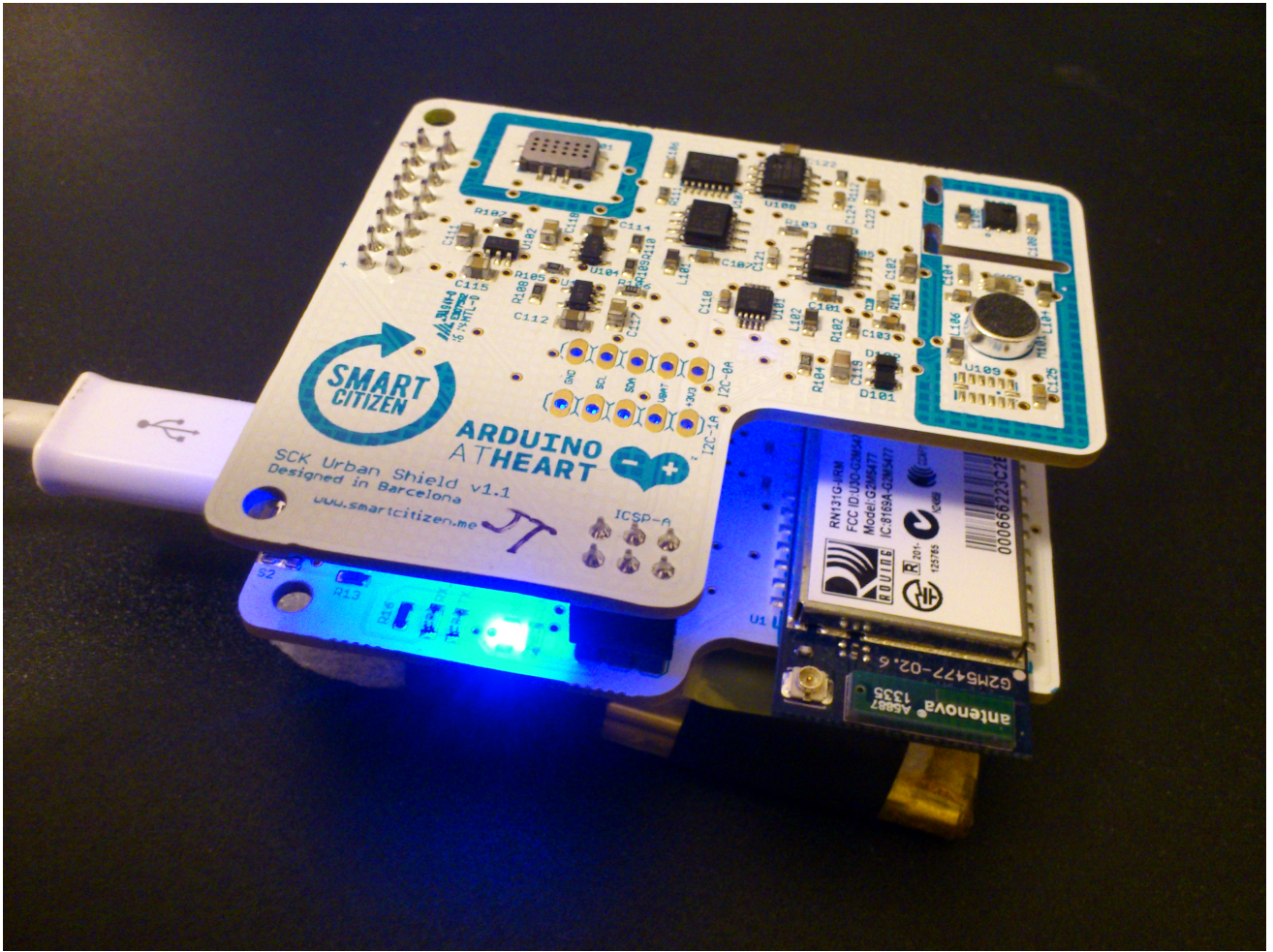
Execution time: 0 min, 0 sec, 4 millisec

Get all records:

Execution time: 3 min, 24 sec, 669 millisec

TOTAL:

Execution time: 8 min, 29 sec, 538 millisec



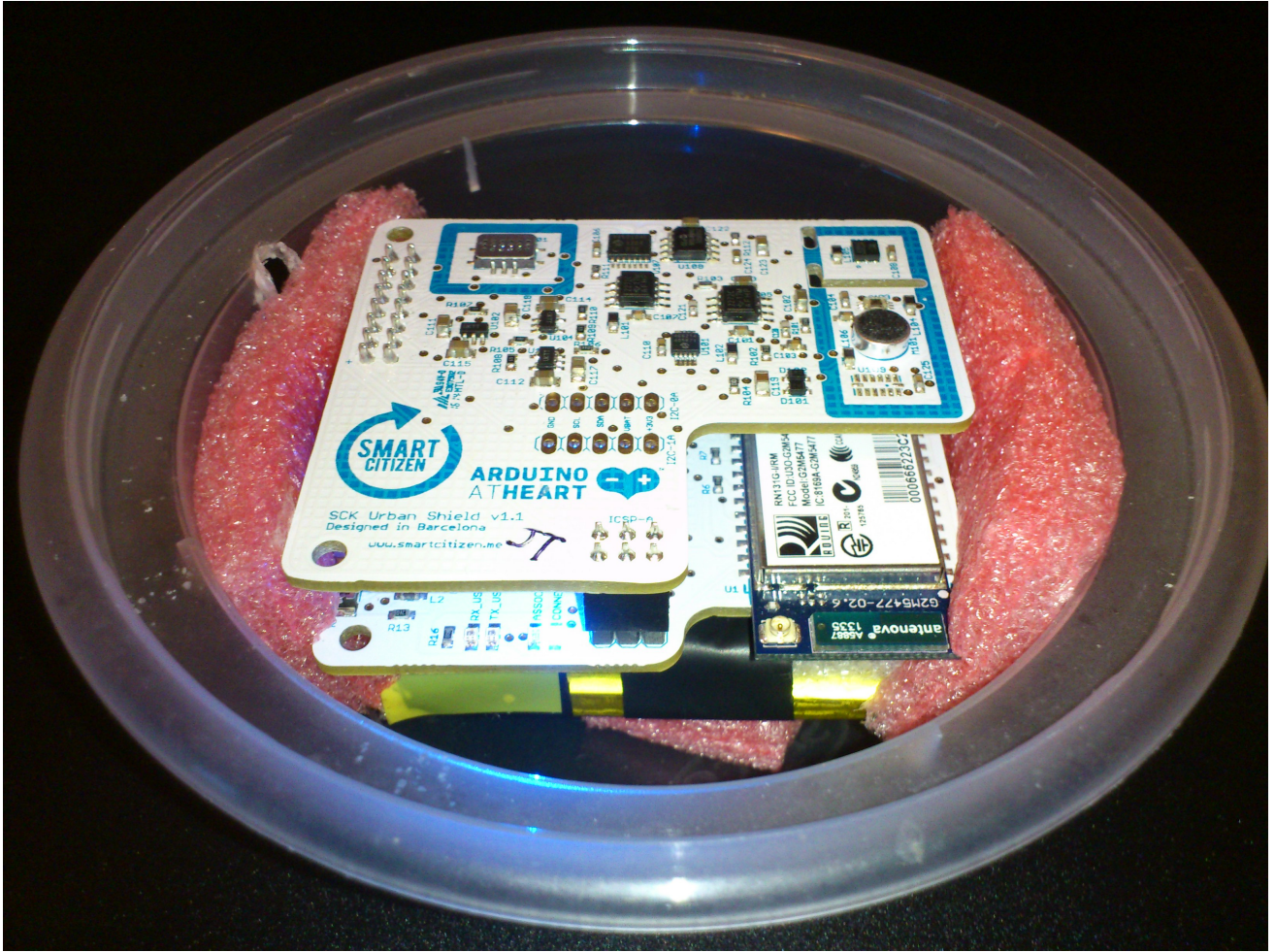
Thumbnail 109



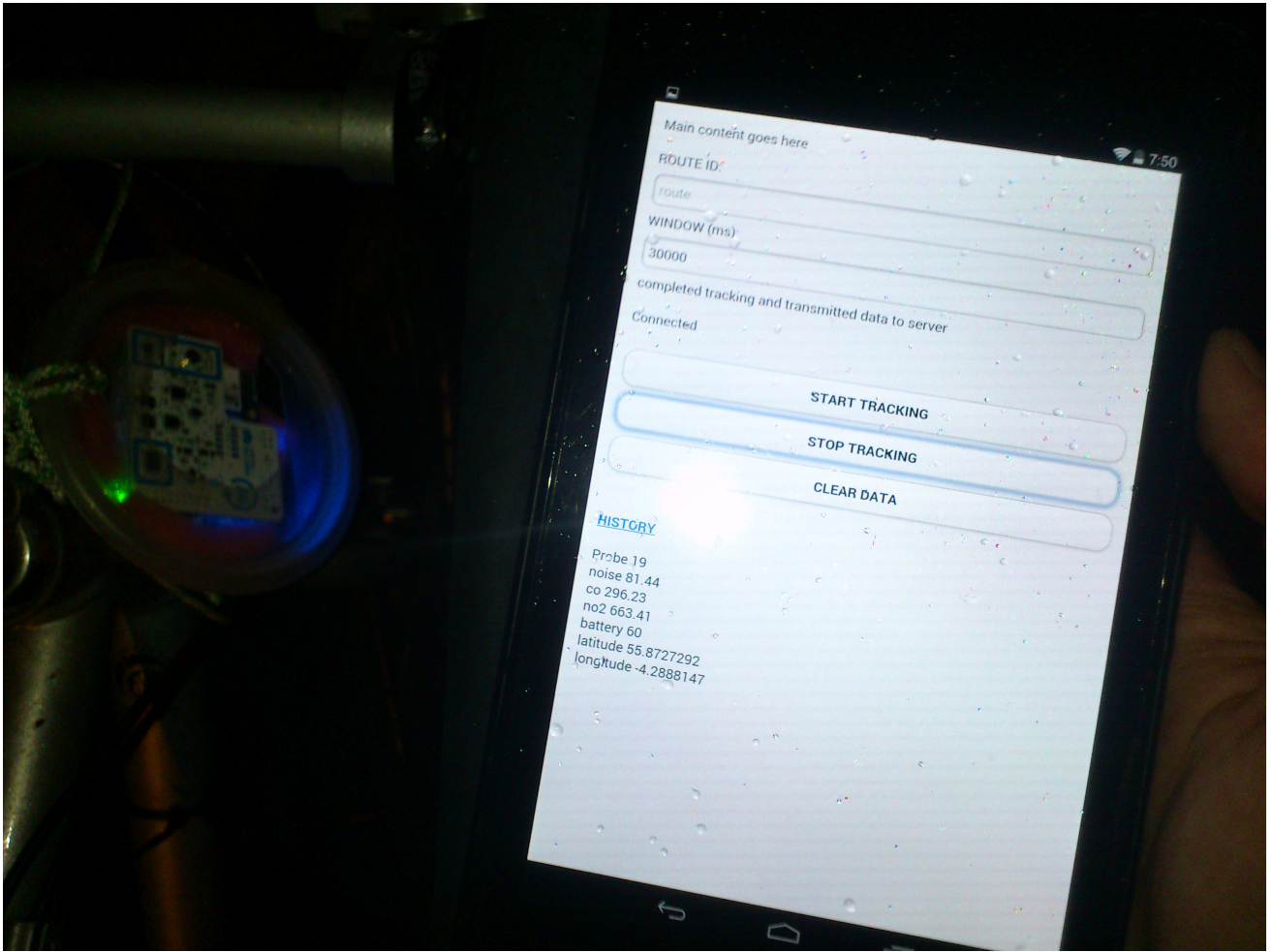
Thumbnail 110



Thumbnail 111

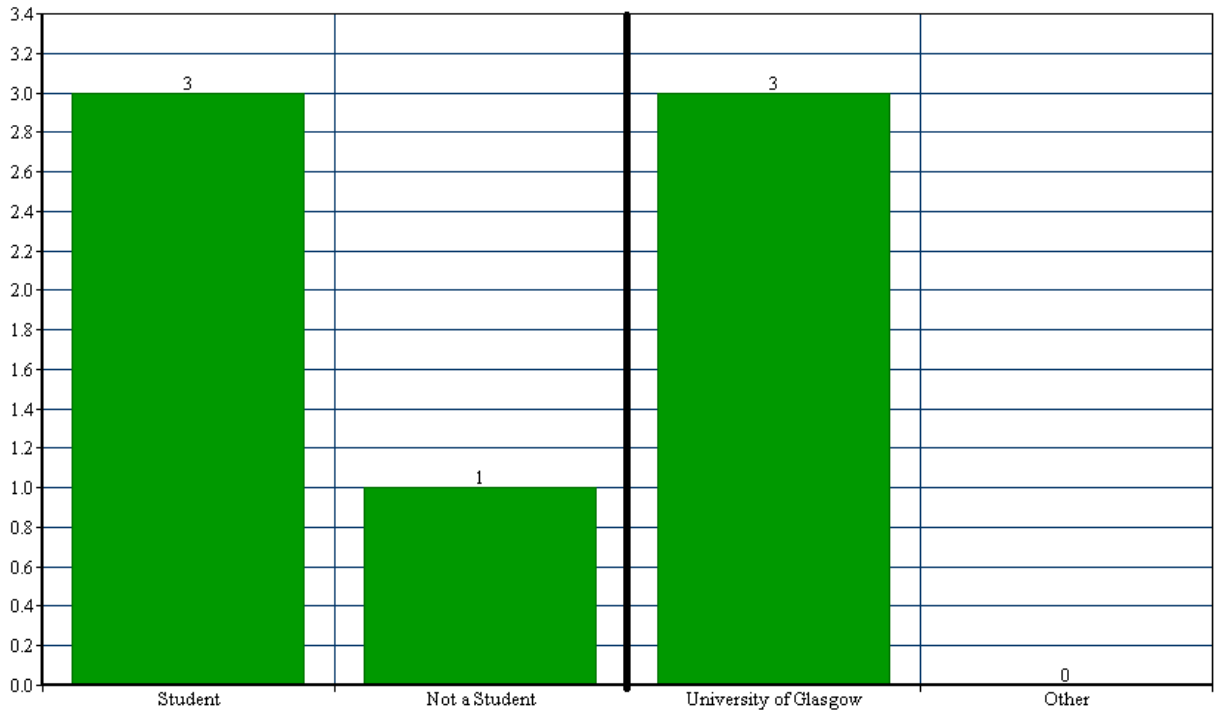


Thumbnail 112

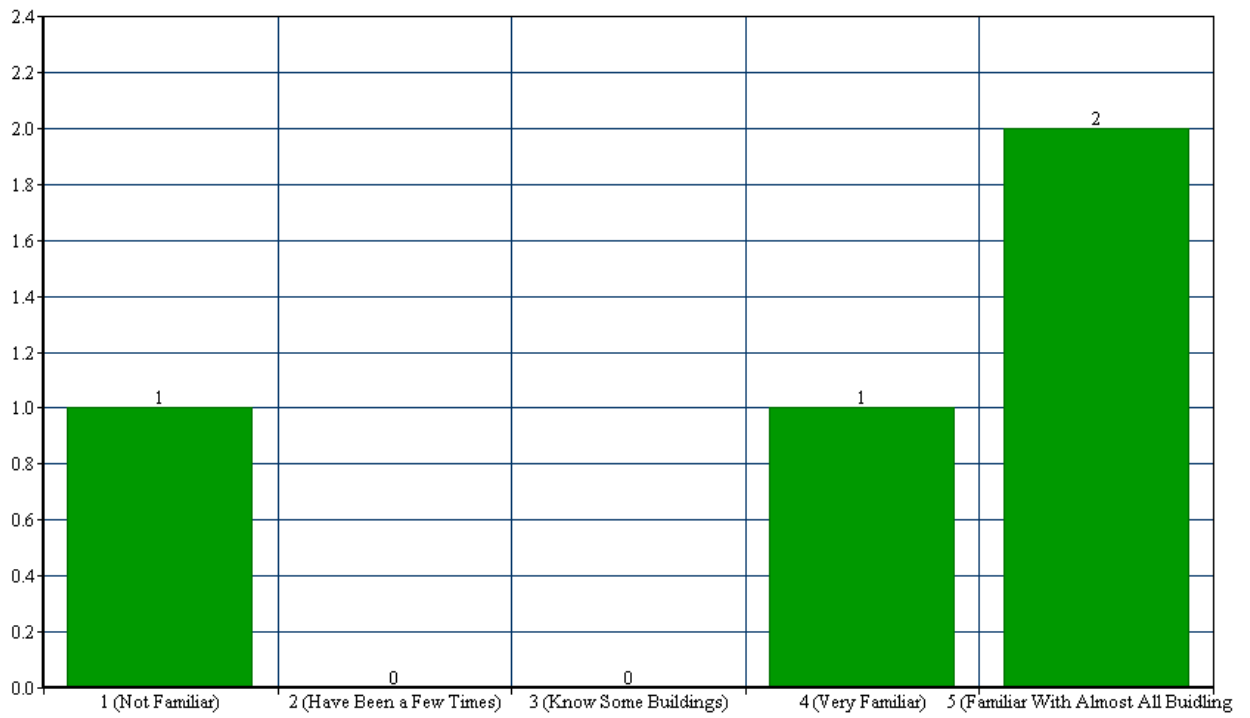


Thumbnail 113

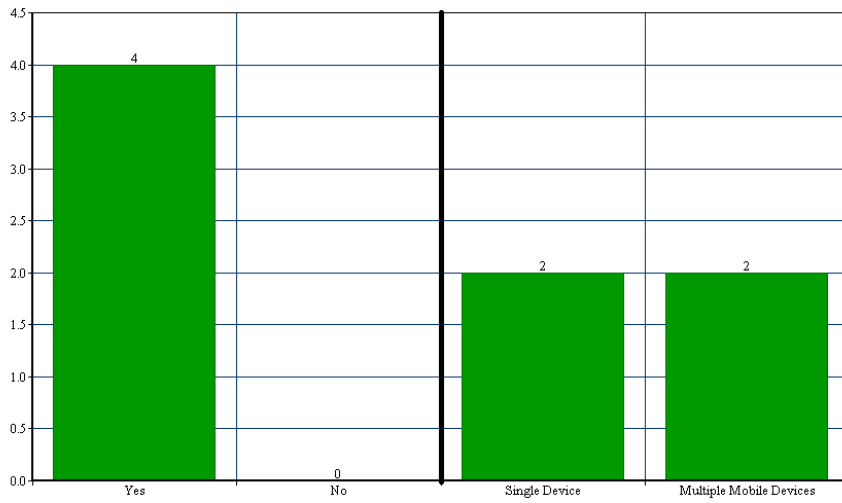
Are you a student? If so, are you a student at this university?



How familiar are you with the university campus? (scale 1-5, 5 being the most familiar)

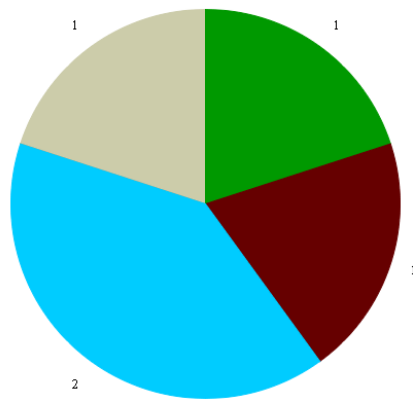


Do you possess a mobile device? If so, how many?

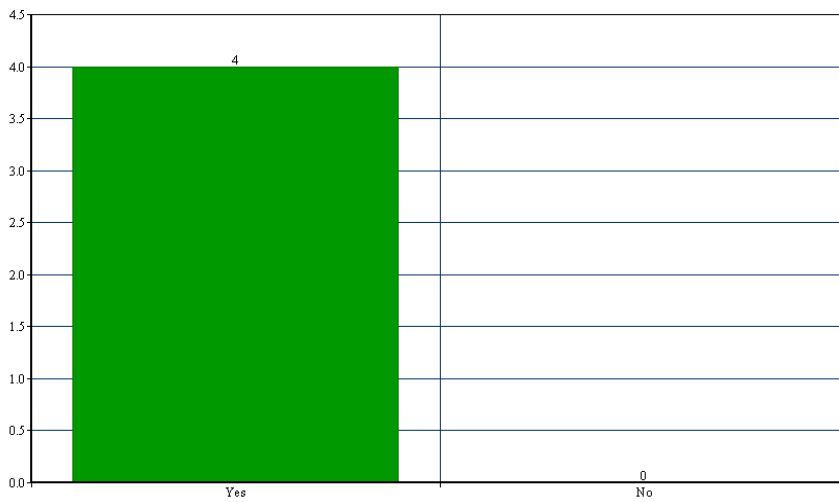


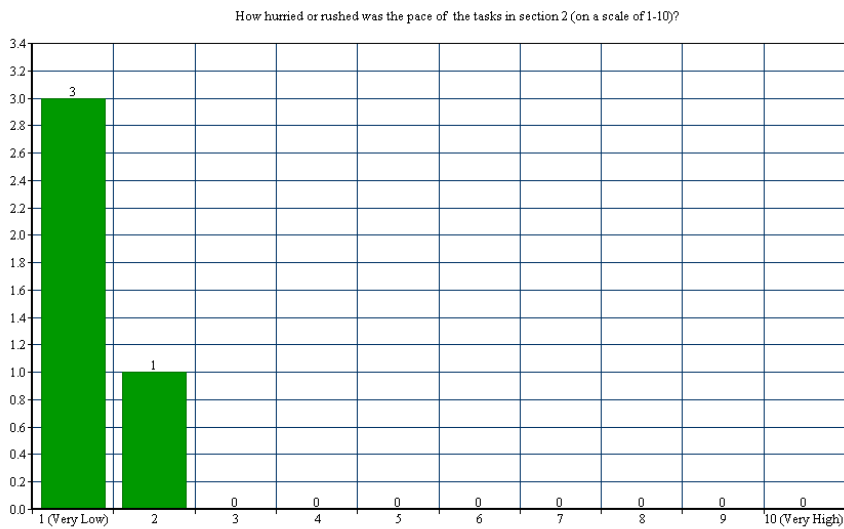
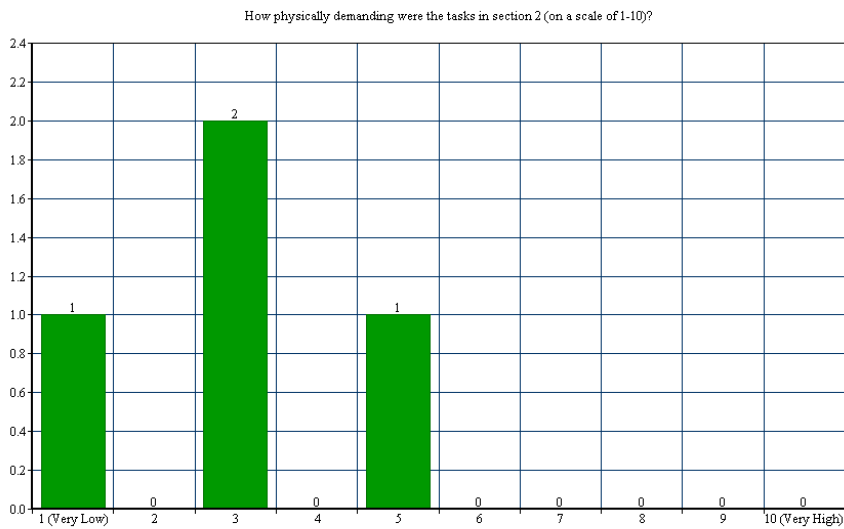
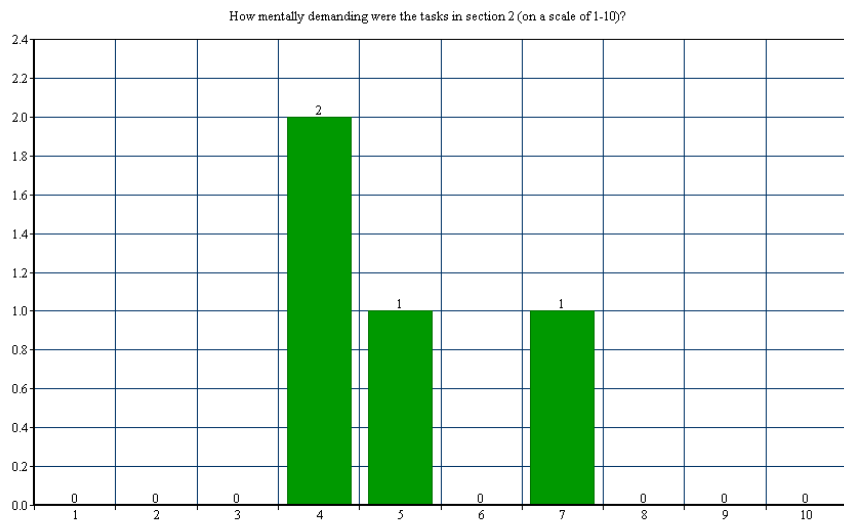
If so, which brand/brands?

■ Apple
 ■ Samsung
 ■ Blackberry
 ■ Sony Ericson
 ■ Google
 ■ HTC
 ■ LG
 ■ Motorola
 ■ RIM
 ■ Other

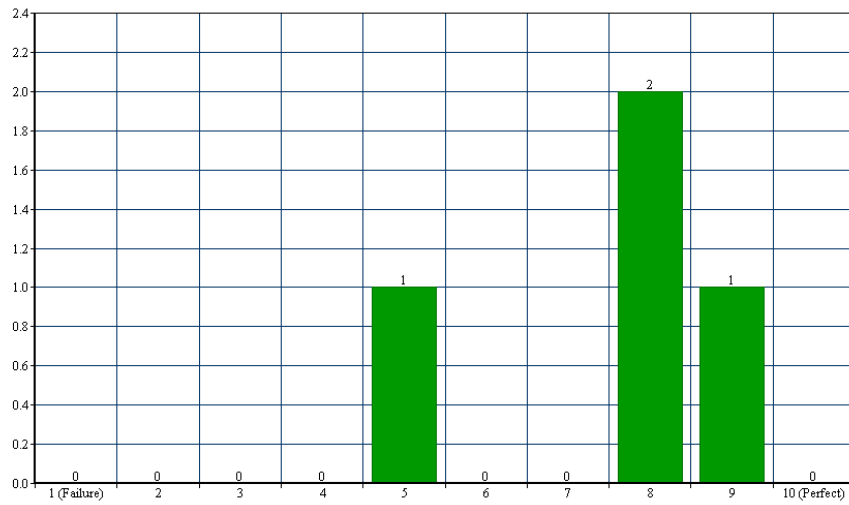


Do you download mobile applications that track your location from the application catalogue?

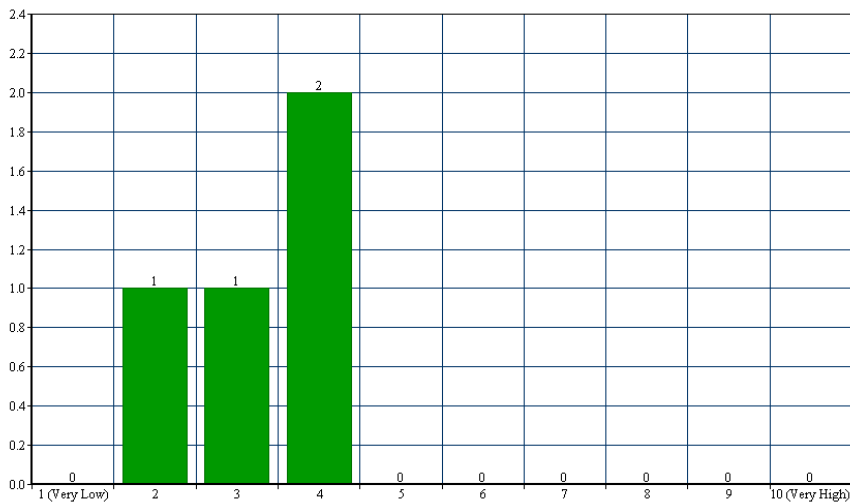




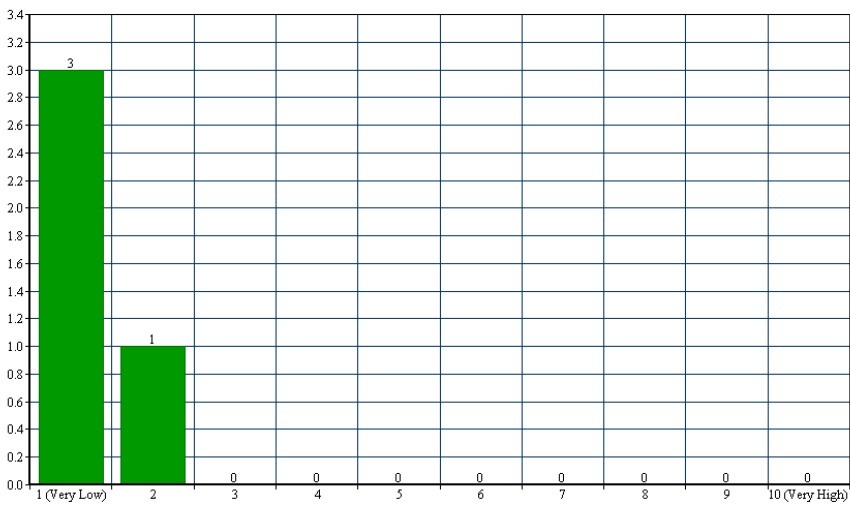
How successful were you in accomplishing the tasks in section 2 of the task sheet (on a scale of 1-10)?



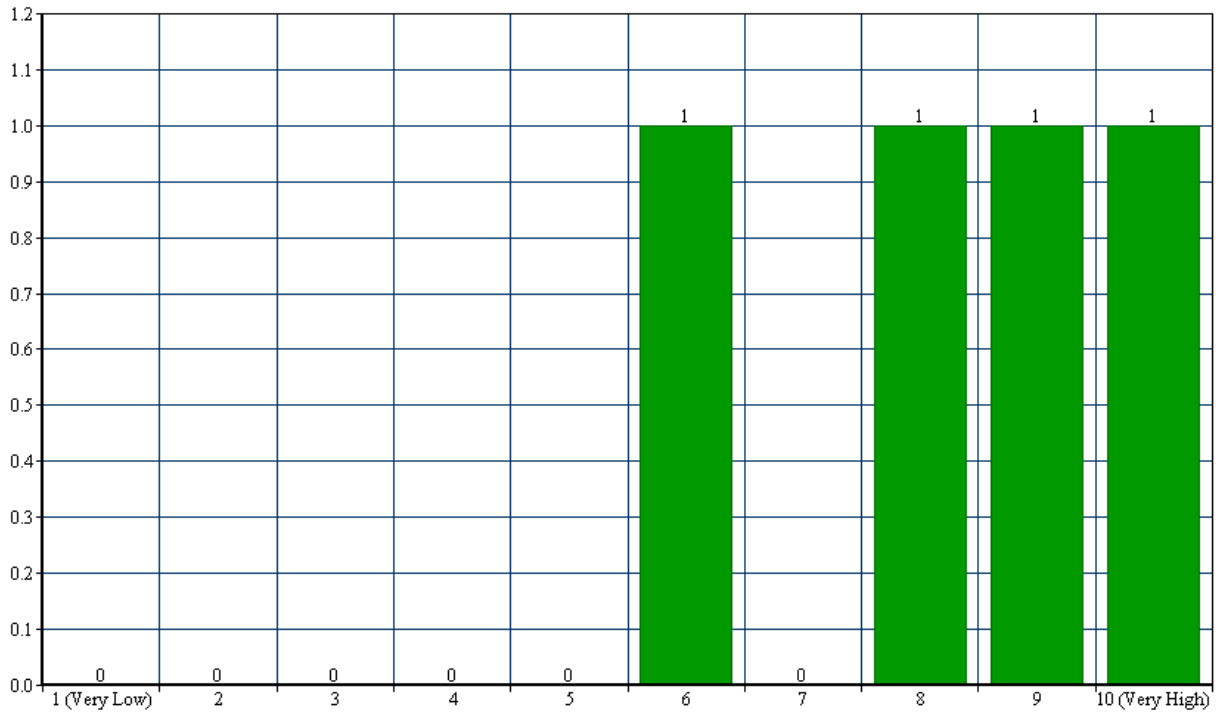
How hard did you have to work to accomplish your level of performance (on a scale of 1-10)?



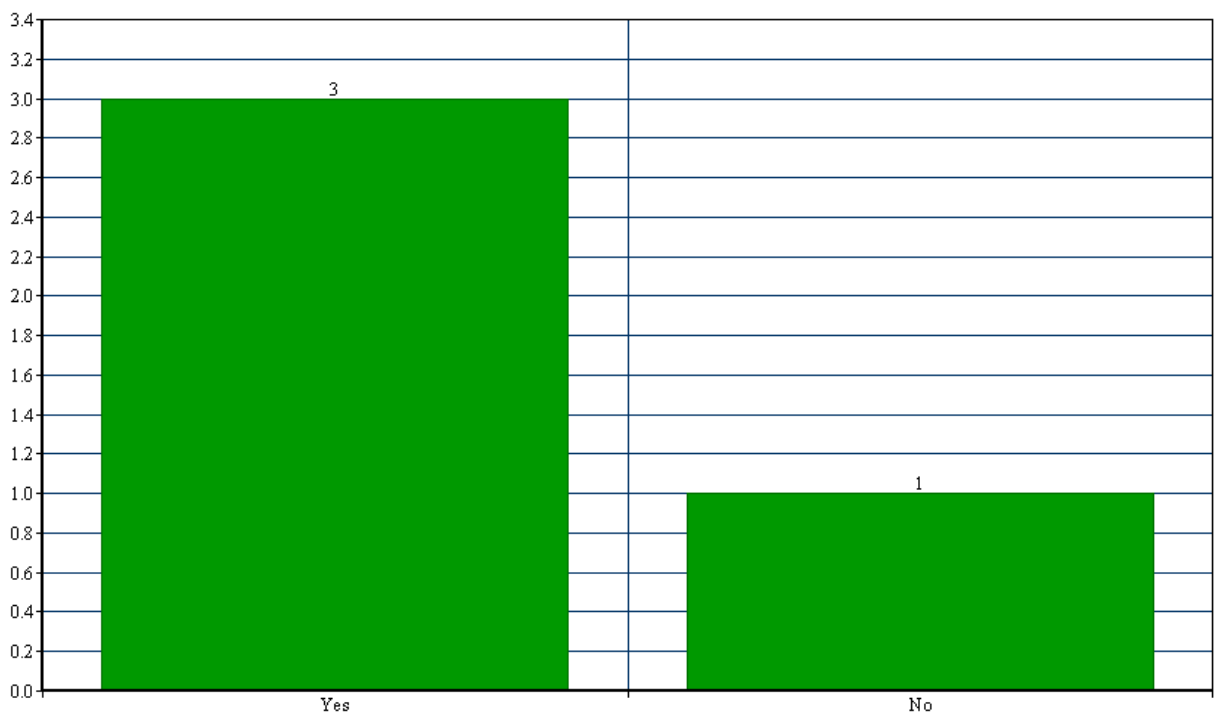
How insecure, discouraged, irritated, stressed, and annoyed were you (on a scale of 1-10)?



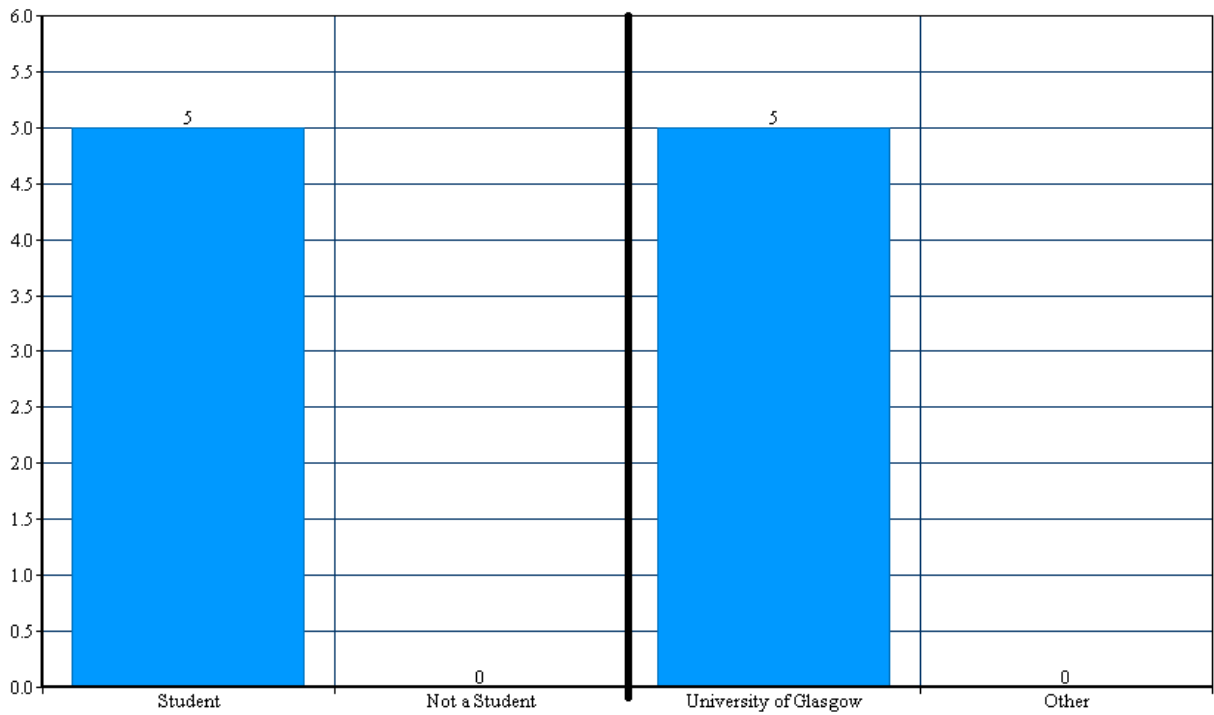
How would you rate the application on an overall basis (on a scale of 1-10)?



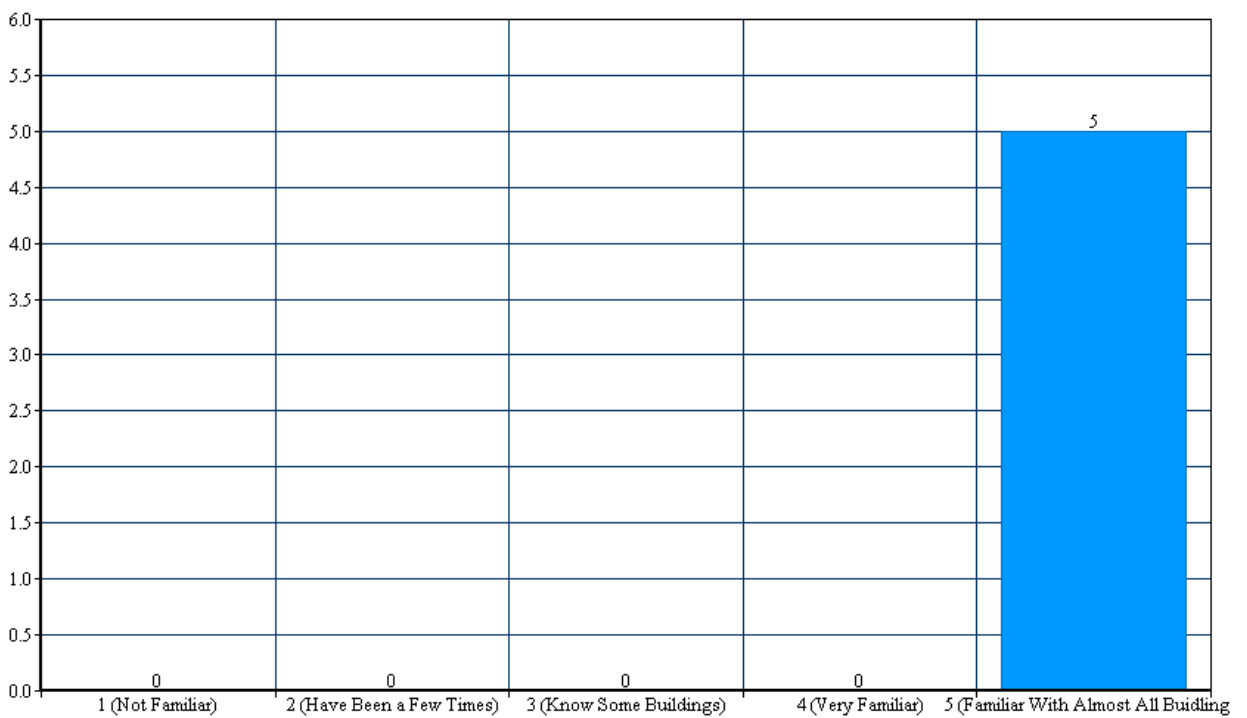
Would you use this application if it becomes available for free on the application web catalogue for your device?

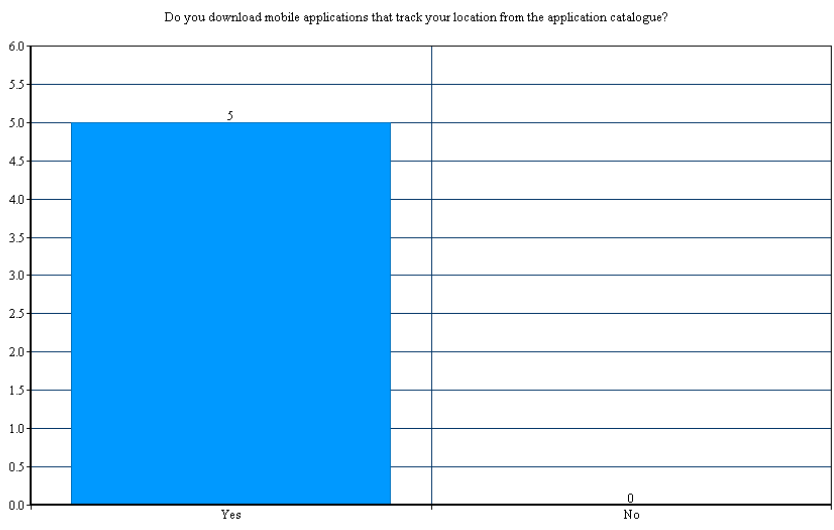
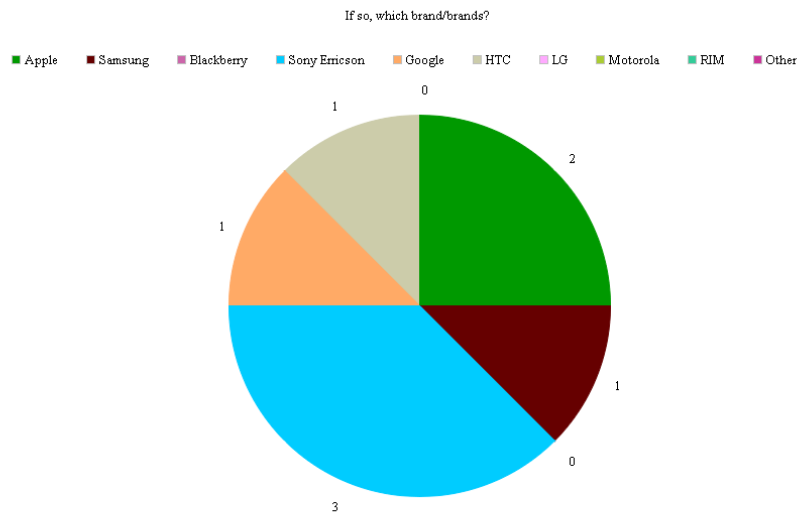
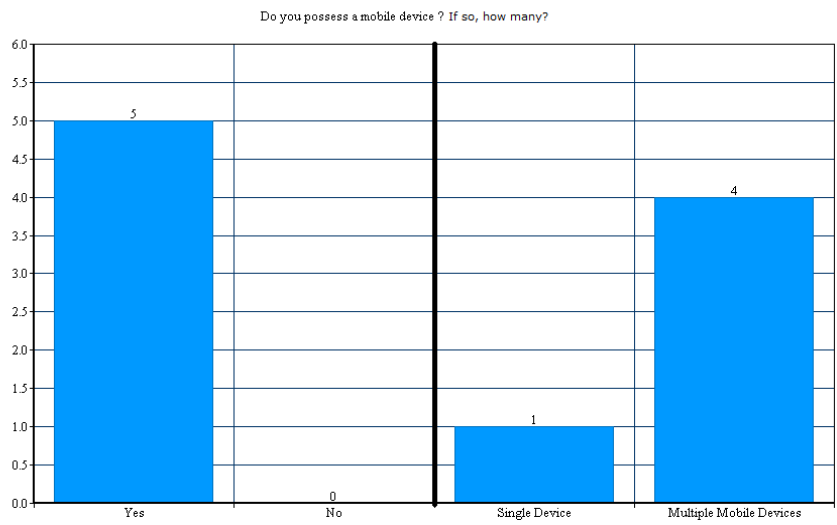


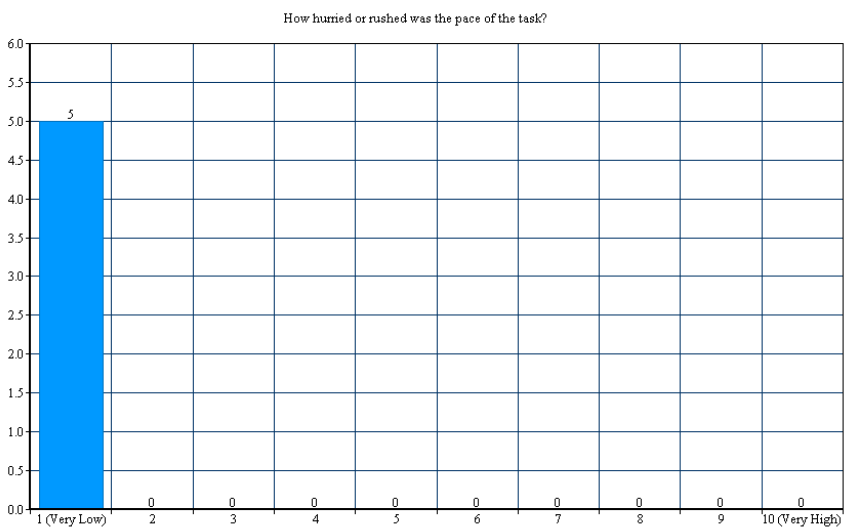
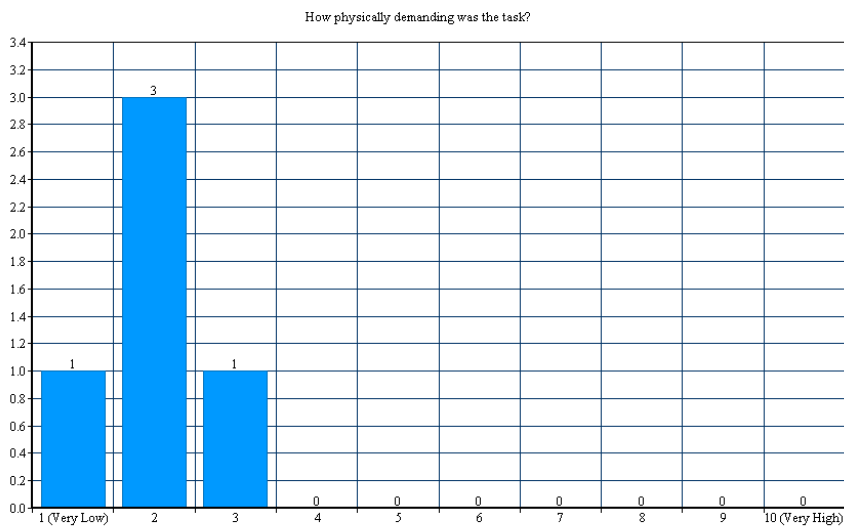
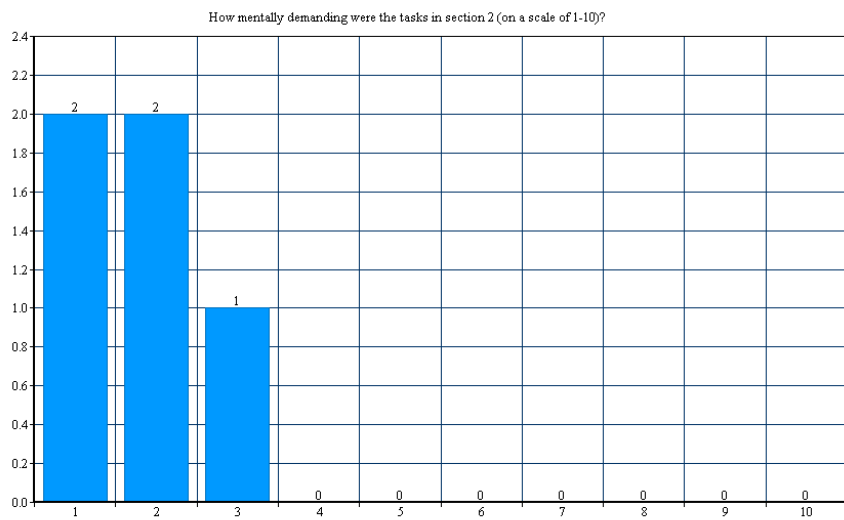
Are you a student? If so, are you a student at this university?

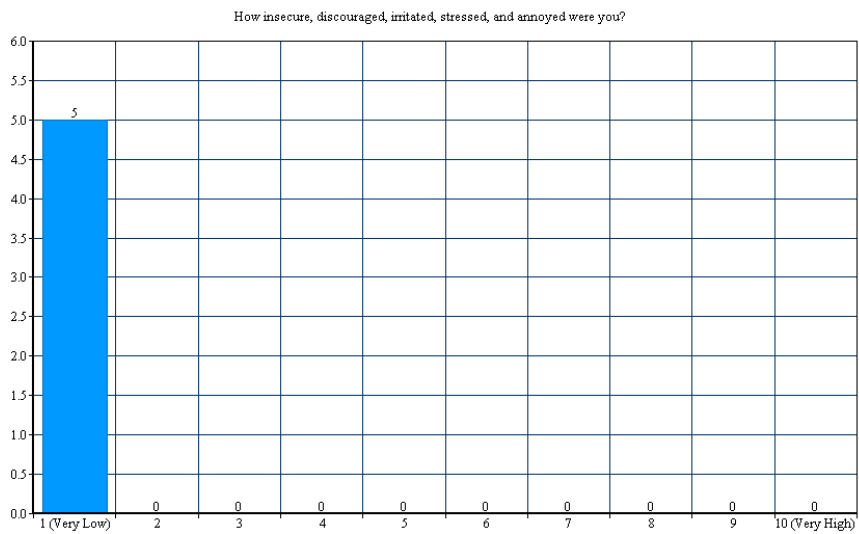
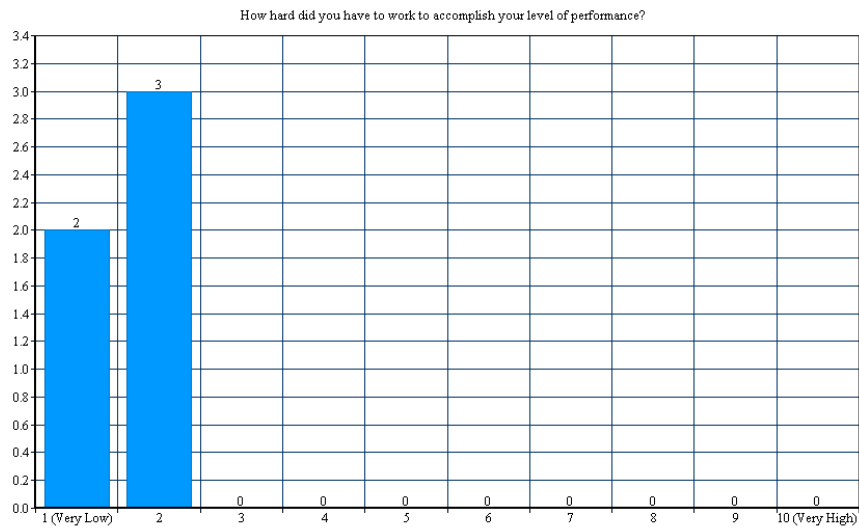
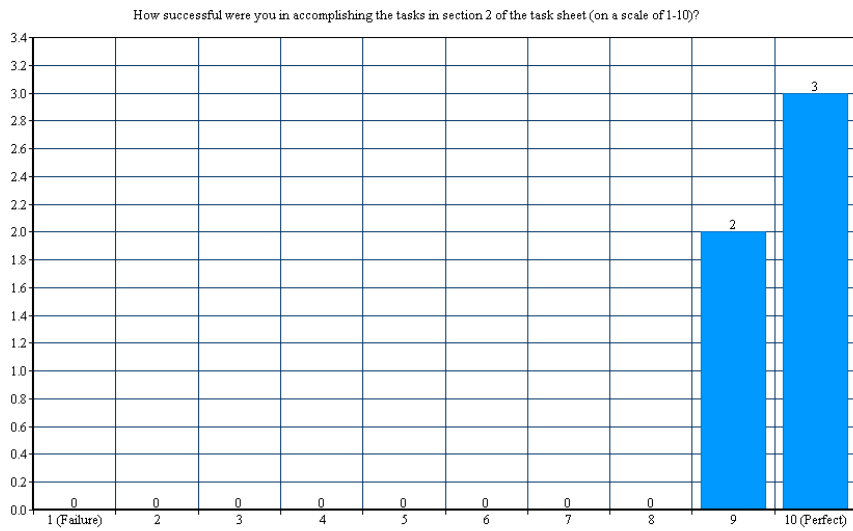


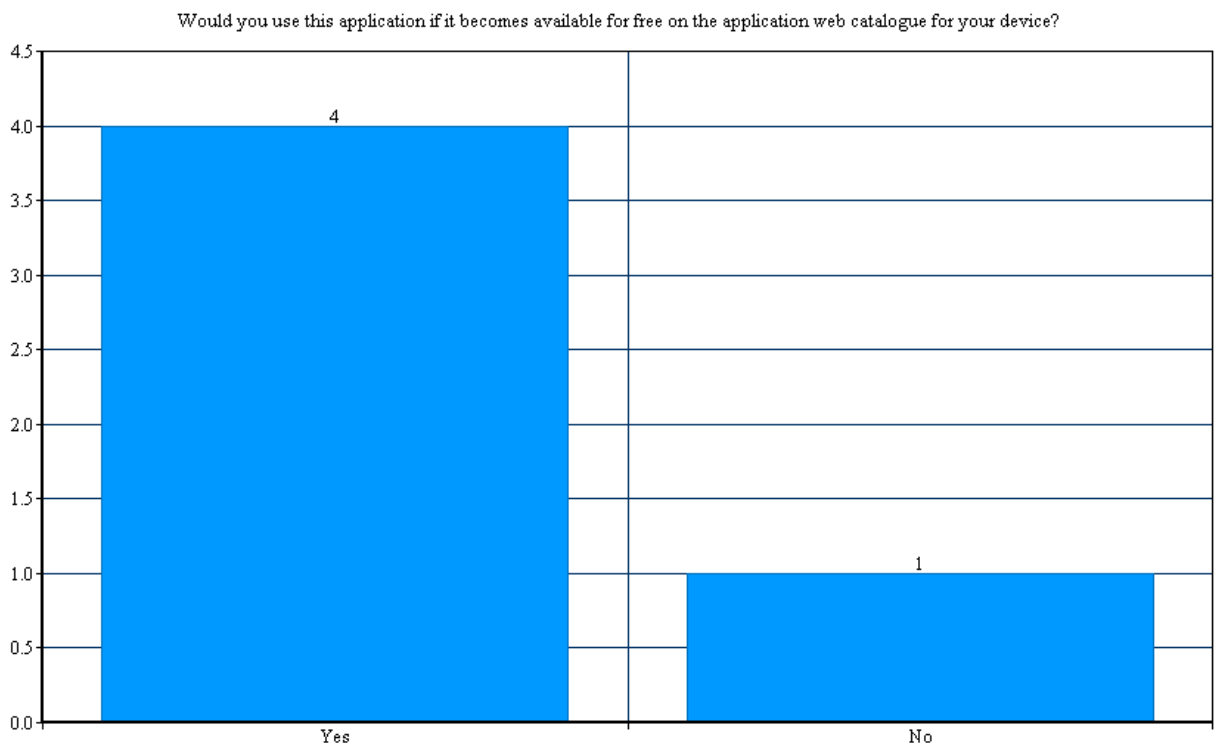
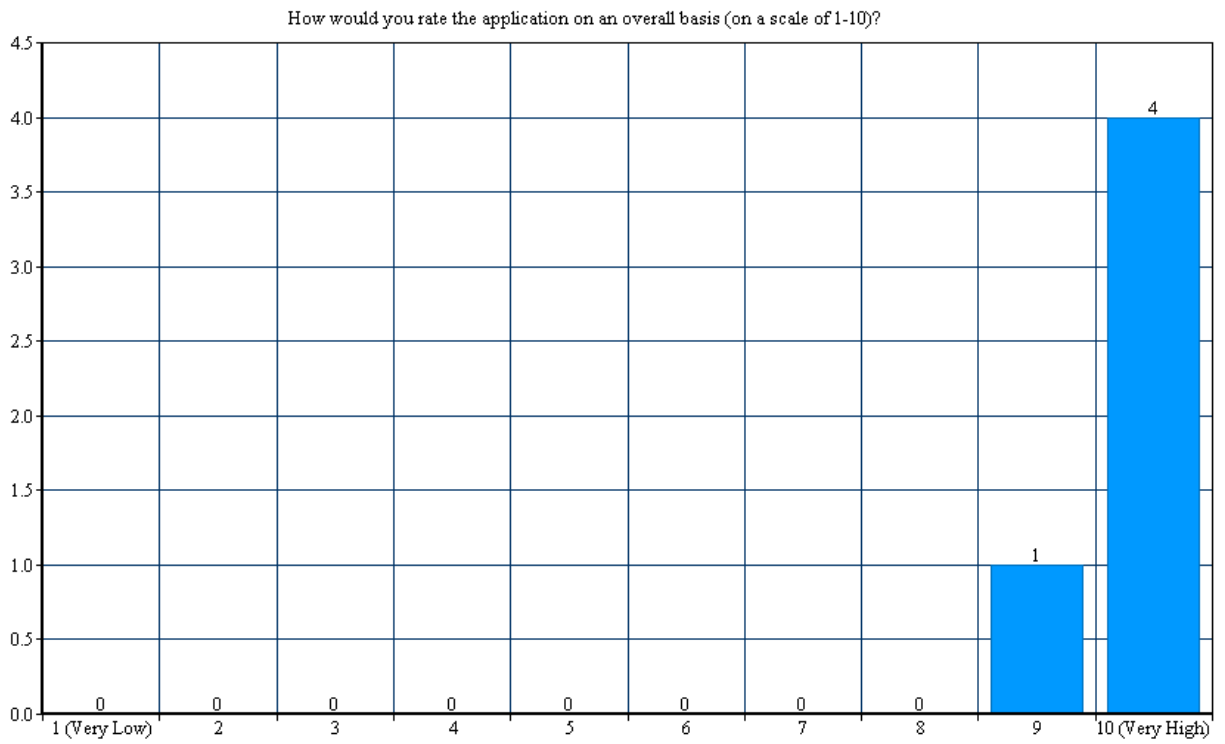
How familiar are you with the university campus? (scale 1-5, 5 being the most familiar)











Thumbnail 123

