Plant view – an augmented reality android application

Matthew Frost – L1426439

Contents

[Introduction 2](#_Toc480203504)

[Abstract 2](#_Toc480203505)

[Rationale 2](#_Toc480203506)

[Ethical, Legal and Social Issues 2](#_Toc480203507)

[Research & Analysis 3](#_Toc480203508)

[Justification of choices 3](#_Toc480203509)

[Android Development Language 3](#_Toc480203510)

[Web Service Development Language 3](#_Toc480203511)

[Integrated Development Environments (IDEs) 4](#_Toc480203512)

[Data Storage 4](#_Toc480203513)

[Requirements 4](#_Toc480203514)

[Must 5](#_Toc480203515)

[Should 5](#_Toc480203516)

[Could 5](#_Toc480203517)

[Will not have this time 5](#_Toc480203518)

[Estimates 5](#_Toc480203519)

[Design 6](#_Toc480203520)

[Mock-up designs 6](#_Toc480203521)

[Architecture 7](#_Toc480203522)

[Database Diagrams 8](#_Toc480203523)

[Class Diagrams 9](#_Toc480203524)

[Location Web Service 9](#_Toc480203525)

[Location Data Web Service 10](#_Toc480203526)

[Initial Android Application Class Diagram 10](#_Toc480203527)

[Web application 11](#_Toc480203528)

[AnomalyNotification Service 12](#_Toc480203529)

[Implementation 12](#_Toc480203530)

[Development methodology 12](#_Toc480203531)

[Testing & Evaluation 13](#_Toc480203532)

[Reflection 13](#_Toc480203533)

[References 13](#_Toc480203534)

[Appendices 13](#_Toc480203535)

# Introduction

## Abstract

An augmented reality Android application that displays information relevant to the user depending on their location. The application was originally developed for a client that operated on an industrial site, so the data collected would be relevant to their use case, such as the temperature of a pipe. However, the app is generalised enough that it can work with any numerical data set, for example the energy usage of buildings at the university. The data is displayed on a graph to show how it changes over time and statistical analysis is applied to highlight any anomalies in the data.

To find the location of the user, the Android device’s GPS is utilised to allow the acquisition of the device’s latitude and longitude to find the position and the bearing to find which direction it is facing.

A separate Google Maps web application has been developed to allow the mapping of locations against data in an SQL database. Both the Android app and web app communicate with the data sources using Node JS web services. The web services are used to store and retrieve location points as well as pull the data for each location wherever it is stored.

## Rationale

The project proposal was initially provided by a local company, Sabisu, who develop reporting tools for customers within the oil and gas industry. They were looking for an augmented reality Android application that would allow a user on an industrial site to walk around with a tablet and view information on what was around them. Sabisu also asked for a web application that would allow points of interest to be plotted on a map, this would be used by admin users.

There were not many technical limitations put in place within the project proposal from Sabisu, all they asked for was an augmented reality Android application, and an admin web application to go along with it. This meant that the project could be approached with any methods or technologies that would best suit the given task.

Furthermore, this project allows for a range of technologies to be used including Android, Node JS and SQL. This would help with employability as both web applications and mobile applications are what the current market is moving towards.

## Ethical, Legal and Social Issues

Since this project involves a real client it is important that the British Computer Society code of conduct (British Computing Society, 2017)is adhered to. The section that would apply specifically would be “professional competence and integrity” meaning that work should only be undertaken that I think I am competent of and I should be willing to accept criticisms and alternative viewpoints.

As my application is intended to be used within the chemical industry it could be reporting some critical information to the user. However, as the application is only reading the information from a data source that has been inputted by another system the application does not have any ethical issues in this regard.

# Research & Analysis

## Justification of choices

### Android Development Language

For Android development there are a number of language and libraries that can be used. After some research there were a number of languages that seemed to be popular and potentially suitable for the project, these included:

* C/C++ (Android Native Development Kit (NDK))
* Java (Android Software Development Kit (SDK))
* Kotlin (Android SDK)
* C#/Vuforia (Unity)

Since the Android operating system is built in C and C++ it is possible to create applications for Android in these languages. Applications built using the NDK are often faster than those written in a Java based language as they do not need to run on the Java Virtual Machine (JVM). However, for this project the speed limitations of the JVM are not going to be an issue and the lack of support for the NDK compared to the Java SDK mean that C and C++ were ruled out for the choice of Android development language.

During the initial research period the head of Sabisu suggested that the Unity engine could provide a solid method for creating augmented reality applications. With some research, it was found that there is a library for Unity called Vuforia which provides an easy way to add augmented reality and image recognition to an application. However, for the image recognition to work it had to have knowledge of the shape beforehand and shapes had to be complex to enable more accurate recognition. As the application would be looking at 3D shapes such as tanks and pipes it was clear that using Vuforia would not work.

The most popular language for Android development is Java, it is the language used by Google’s Android SDK. Therefore, a Java based language was chosen as the language for the creation of the project. Kotlin was chosen as it complies down to Java bytecode so it will run on the JVM and it can also work alongside any existing Java libraries. The advantage of Kotlin for Android development is that it removes some of the potential drawbacks of Java such as, null pointer exceptions and having to reference every Android component by using the findViewById method. Due to the advantages that Kotlin provides it is the language that has been chosen for the development of the Android application.

### Web Service Development Language

The project contains two separate web services, one to store and retrieve the geo-location points for both the web application and the Android application. The other web service retrieves the data about a given location, this data will be stored in an SQL database.

When developing a web service there is a large range of languages that could have been used. However, the two that came to mind first were C# .NET and Node.js.

C# .NET was a potential candidate as it has all the features that would be required and being a Microsoft product it had good integration with SQL server using Entity Framework. Furthermore, since C# has been around for seventeen years it is a mature and widely used language with plenty of support.

Node.js is a more modern development language, being initially released in 2009 and only coming into more widespread use within recent years. So, due to the more modern technology being used it could be more appealing to the industry. Furthermore Node.js is a very light weight solution and as both web services are just reading and writing data to databases it appears something heavier like .NET would be too much as most its features would not be utilised.

### Integrated Development Environments (IDEs)

For Android development there really is only one IDE to use, Android Studio. Android Studio is the official IDE supported by Google and it provides all the tools necessary for creating an Android application. Although most Android development is done in Java, Android Studio provides support for the Kotlin plugin which allows code to be translated from Java to Kotlin, as well as providing code completion for Kotlin.

For the web development aspects of the project, Visual Studio was used. Visual Studio was chosen because it is widely used within in the industry and university so is therefore very familiar. Visual Studio provides support for a lot of different languages including the ones needed for this project which are HTML, CSS and JavaScript.

### Data Storage

Storing the data that the project used came down to the choice between two different paradigms, SQL (Structured Query Language) or No SQL. However, this project would not take need nor take advantage of any of the features of a No SQL data storage mechanism. For this reason, SQL Server was used for storing any data that the applications will access.

A lot of industrial data that Sabisu accesses is stored in a data historian called InfoPlus21 (IP21), having access to this system would allow the app to return live data as it is updated by the sensors on the industrial site. However, access to this would require a virtual private network (VPN) as well as credentials to log into IP21. Furthermore, after deciding to make the application more generalised it would not make sense to spend the time adding such a niche feature. After speaking to Sabisu they agreed that just connecting to a SQL database would be suitable.

## Requirements

The initial proposal provided contained several requirements, some were necessary and some were just “nice to haves”. After analysing the requirements provided, it was clear that it would not be possible to complete them all in time and some of them would not be possible to work on outside of the Sabisu offices. Therefore, it was necessary to remove any requirements that would not be feasible as well as anything that would not be implemented in time and would not affect the end product too much. To categorise and prioritise the requirements the MoSCoW (Must have, Should have, Could have, Will not have) system was used, this ensured that if all of the minimum requirements were met then a shippable product would be produced. Requirements that come under “must have” are requirements that would make the project useless if they were not included. Requirements that come under “should have” are requirements that are important to the project but are not critical to success. Requirements that come under “could have” are “nice to haves”, they may add more polish or functionality to the project but are less important than “should have” requirements. Anything that falls under “will not have” will not be done in this release, they are items that are feasible but will not be able to be included in the given time period.

For example, one of the requirements was to have it connect with the Sabisu platform to integrate with some of their APIs. However, this would require having a VPN for their network during development and it is not a feature that is necessary for the application to work.

### Must

* The web application must allow the user to create, edit and delete locations
* The web application must display all created locations on the Google map interface
* The web service must return all created locations that have not been deleted
* The web service must be able to return all locations within one-hundred meters of a given latitude and longitude
* The web service must allow locations to be created, edited and deleted in the database
* The web service must be able to return all data associated with a given location
* The Android application must use the device’s GPS to get the current location
* The Android application must access the device’s camera to create an augmented reality application.
* The Android application must interact with the web service to get the locations near the device
* The Android application must retrieve the data associated with a given location
* The Android application must display the retrieved information on a graph

### Should

* The Android application should perform analytics on the data retrieved to identify any anomalies in the data
* The Android application should use the device’s compass to get the direction the device is facing

### Could

* The Android application could implement QR code scanning to retrieve information about the location associated with that QR code
* The Android application could have offline capabilities to store the data if internet connectivity is lost

### Will not have this time

* The Android application will not have connectivity to enable Sabisu log ins
* The Android application will not use the Sabisu API’s
* The web services will not connect to IP21

## Estimates



Figure Y – estimates

Before development work began it was important to break down the requirements into smaller tasks, then estimate how long it would take to complete each task. Doing this makes it possible to check if every requirement can be completed during the allotted development time, if it is not possible then a re-evaluation of the requirements would be necessary.

Due to the short development time, each individual task could only be estimated to a maximum of two days, if it was estimated to be longer than two days then it should be broken down into separates tasks that would each be less than two days work. A single day was the equivalent of one work day, so about 7-8 hours.

# Design

## Mock-up designs



Figure 1 initial mock-up of main application screen

When designing the user interface, it was important to keep in mind the use case of the application. As it was initially intended to be used in an industrial setting where the user would be walking around outside, the user interface needed to be simple to use.

To meet these criteria the entire user interface consists of large buttons that could be accurately pressed even when wearing gloves. Furthermore, since this is an augmented reality application it is important not to clutter up the interface too much as it could obscure the camera view.

As well as being functional, a user interface should also look appealing. To achieve this, Google’s material design guide lines were followed. Material design is used to help create a flat modern look and use shapes and shadows to help the user understand how the application works. In this application material design is used to help create the impression of a stacked set of cards, this is used to convey the idea that clicking on the stack will allow the user to see each card individually.

Finally, another important item to consider for good user interface design is the choice of colour. As the top button is the only one that can be clicked it has been coloured green. Green typically conveys ideas of safety and correctness so therefore it was chosen for the only clickable button. Red is normally used as the inverse of green, however it didn’t seem suitable to use red for the other buttons. Therefore, blue was chosen for the other buttons as it comes across as a much more neutral colour compared to red and green.

## Architecture

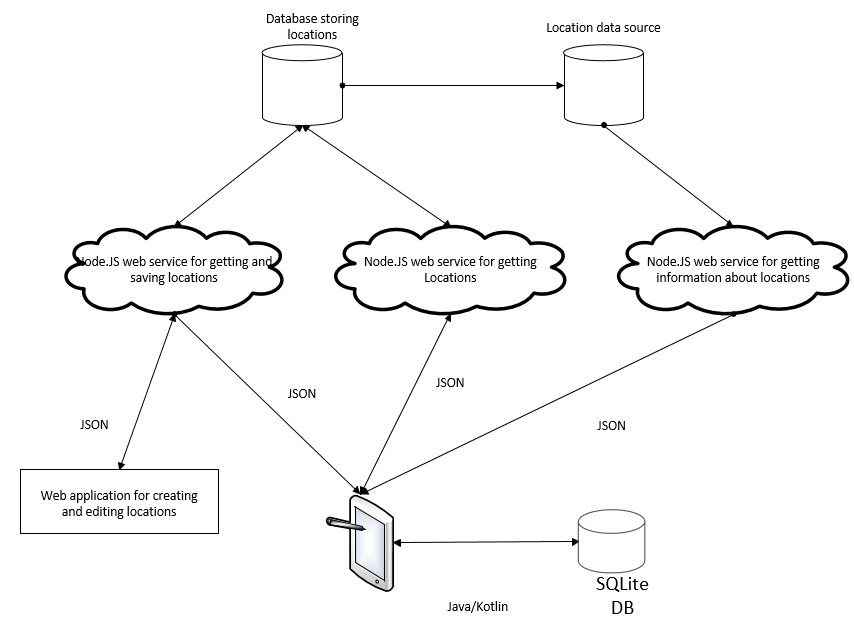


Figure 2 – The architecture of the full system

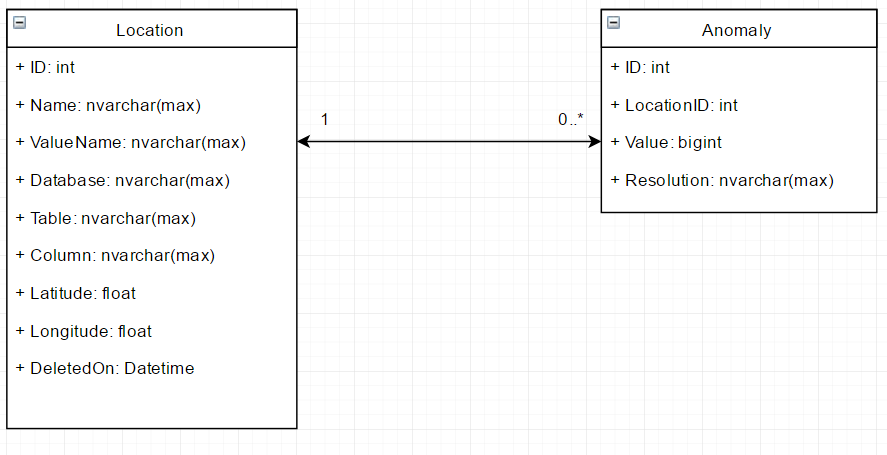
Figure two shows the architecture of the system as well as how data flows between each of the separate applications. The main structure of the system is using microservices to enable connectivity and data transmission between the different layers. Martin Fowler describes a microservice architecture as “an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms”. The microservice approach was chosen as it improves both maintainability and scalability. By keeping a piece of functionality inside a single container if anything goes wrong with that feature then it can be immediately isolated to that single service. Having a maintainable architecture is especially important when shipping the application out to a customer or client. Furthermore, once the application starts to be used by multiple users it will be clear which services are put under the most load and a microservice architecture would allow each service to be scaled up or down with demand.

Both the web application and Android application communicate with the web services using HTTP requests, sending and receiving JSON data. Although the applications could communicate with the databases directly, the web services are there to apply any business logic and shape the data in a way that can be directly used to by the applications without further modification.

The database for storing locations is the database specific for the application. It contains information about locations and the data that is mapped against them, as well as a list of locations with anomalous data against them. It also has stored procedures used to retrieve and insert data from and to the database as well as retrieve data from another database.

The Android application also has a database associated with it, it is used to cache the data retrieved from the web services so that if the Android device loses connectivity to the network it will still be able to show some data.

## Database Diagrams



The database used by the web services only includes two tables. The Location table stores information about locations, this includes the position of the location in the physical world and the location of the data that is mapped against that location in a separate database or table. The location table contains a nullable column called “DeletedOn” this column is used to implement “soft deletion” of records. Doing this means that records marked as deleted can be easily filtered out in queries by including “where DeletedOn is null”. However, this clause can be removed by always selecting data from views where deleted data has already been filtered out, this removes repetition of code and stops deleted records from accidently being returned. By having the column as a datetime field it is also possible to see when the data was marked as deleted, this can be useful when going back see what happened and when.

The database also contains a table for the storing of anomalies, the table is related to the Location table using LocationID as a foreign key. Anomalies are not intended to be deleted there for there is no field for marking them as deleted.

## Class Diagrams

### Location Web Service

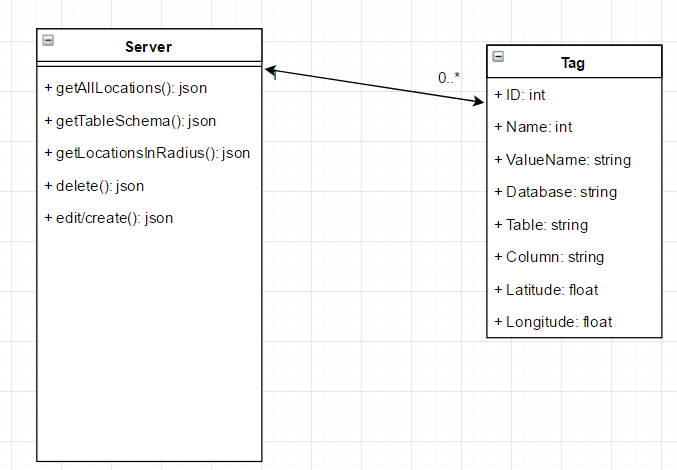


Figure 2

The web service that allows locations to be saved, edited, deleted and retrieved from the database consists of just two classes. The Tag class is used as a data transfer object (DTO), mapping the data that goes into and from the database. Doing this allows the data to be easily manipulated in the form of a JavaScript object.

The Server class is where the logic and functionality is implemented. The getAllLocations method returns all locations from the database that have not been deleted, this method is used for the web based application.

The getTableSchema method takes the name of a database and table, then returns the names of the columns for the given table. This is also used in the web application to help with the creation of locations and mapping data to the selected location.

The getLocationsInRadius method takes a latitude and longitude value for a location and returns all locations from the database that are within one hundred meters of the given location. This functionality is only used by the Android application to show the user all points within their vicinity.

The delete function is used to mark locations in the database as deleted so that they will not be displayed in any of the applications. The function takes the ID of the location to be deleted.

The creation and editing of a location is wrapped up into one function. Traditionally both actions would have their own function, however since the action for editing and creating a location uses the same stored procedure in the database, there was no point splitting them up. The function takes a Tag object when creating and editing a location. The function is only used by the web application to create and edit locations.

### Location Data Web Service

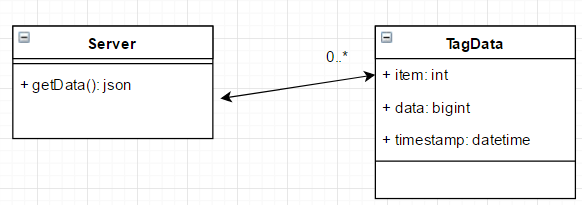


Figure 3

The web service for getting data related to a given location consists of two classes, TagData is the DTO class describing what the data should look like. Server is where all the functionality and connectivity to SQL server is handled.

The getData method retrieves all data for a given location, it takes the ID of a location and returns a JSON array of TagData objects.

### Initial Android Application Class Diagram

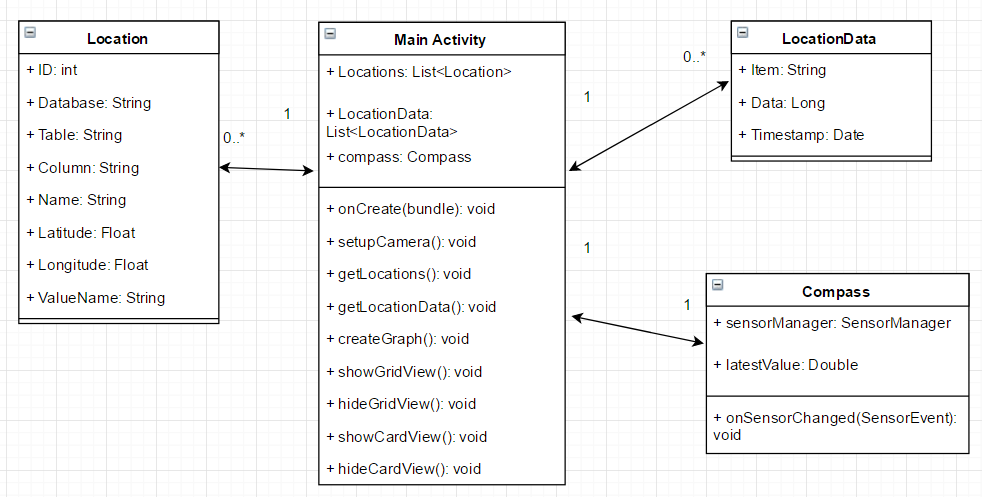


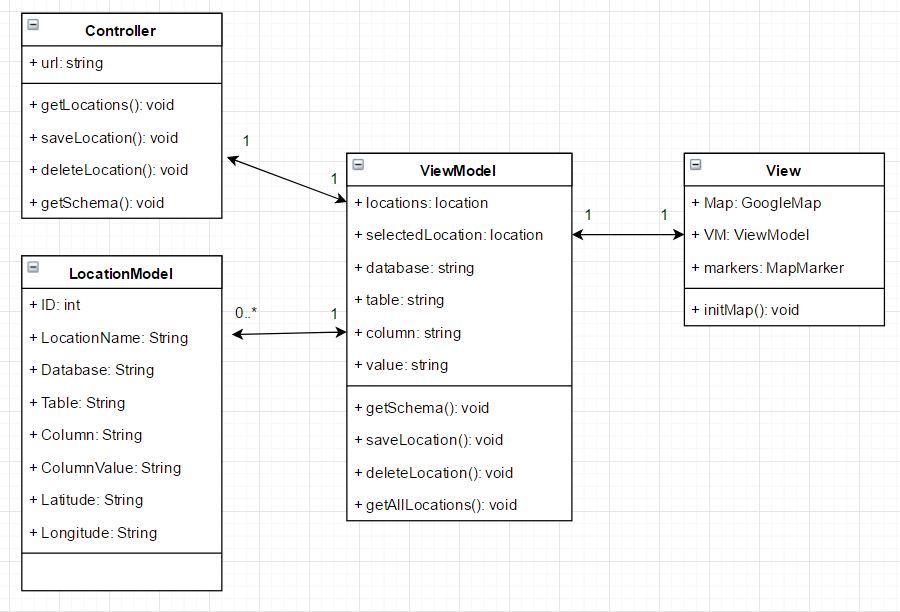
Figure 4

Figure 4 shows the initial class diagram for the Android application before any development had taken place. The Android application makes use of the same DTOs as the Node JS web services, this is done to help keep the shape of the data consistent as it moves between applications.

In Android applications each screen is known as an activity, by default the first activity that is opened in an application is the main activity. The Main Activity class is where most the application logic and functionality is contained. The onCreate method is called by the Android operating system when the activity is loaded up, here all the initialisation of events and listeners is set up. The setupCamera method is called from onCreate and is used to initialise the device’s camera and show the camera preview within the application. getLocations calls the location web service to retrieve the locations near the user and stores the locations in the Locations list. getLocationData gets the data related to a given location, it takes the ID of a given location and stores the data in the LocationData list. createGraph uses the data in the LocationData list to create a graph from each of the items in the list using the timestamp and data value. Since the application only has one activity the rest of the functions are used to show and hide different aspects of the user interface. By having everything in one activity it removes the need to pass data around different activities. The drawback to having one activity is that a lot of code can end up in the one class.

In order to detect what direction the user is facing the device’s compass needed to be utilised. This has been moved into its own class to keeps functionality contained in a single class instead of mixing with other operations, this is known as separation of concerns. To access the device’s, compass a sensor manager needs to be used, this can be used to acquire access to the compass values. The onSensorChanged method is fired every time the orientation or angle of the device is changed. This method then gets the latest value for the compass and stores it in latestValue.

### Web application

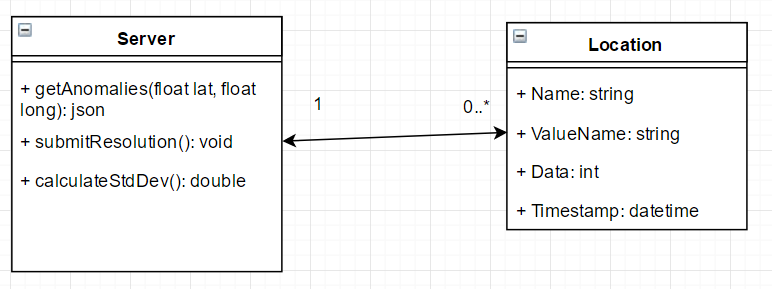


The web application used to manage locations and the data associated with them has a model-view-viewmodel (MVVM) architecture. The view contains references to the Map, the markers on the map and the viewmodel. All the view is intended to do is initialise the viewmodel.

The LocationModel acts as the DTO, mapping what is received and transferred to and from the web service.

The ViewModel class takes the LocationModels and applies the logic to display them as points as points on the map. The viewmodel sits in between the controller and the view, taking data from the controller, transforming it into models and displaying it with the view.

### AnomalyNotification Service



The anomaly notification web service is used by the Android application to retrieve any locations that has anomalous data associated with it. The getAnomalies function takes a latitude and longitude and returns any locations within the vicinity of that given position that have anomalous data. The getAnomalies function makes use of the calculateStdDev to take the data from a location and find out if there are any anomalous data points. The submitResolution function is used to put a resolution or reason next to an anomalous data point in the database.

The location class is used as a DTO for this web service, it mirrors the anomaly table in the database and is used for getting anomalies and submitting anomalies with resolutions.

# Implementation

## Development methodology

This project was developed using an agile methodology. Each week of development was a sprint, where a single feature would be implemented then it can be fully tested and any rework can be applied in the same sprint. There are many different approaches to agile development such as Scrum, Extreme Programming and Feature-Driven Development. The flavour of Agile used for this project was Scrum, in Scrum there are three “actors” these are: Product Owner, Scrum Master and the Development Team. The product owner is the person who has the vision of what the end product should be, in this case the product owner is Sabisu. The Scrum Master acts as an intermediary between the product owner and development team, they help remove any obstacles encountered by the development team and clear up any questions with the product owner. In this case the Scrum Master was the project manager from Sabisu. The development team in Scrum is responsible for creating the product but they are also entirely self-managed, they are responsible for managing time and resources.

At the beginning of each sprint there were meetings with a member of Sabisu to discuss the progress of the previous weeks work and what should be worked on in the upcoming week. This methodology was effective since there was a live client. It allowed for changes to be made and features to be added during development so that the client knows what they are getting before development is finished.



## Database

The database behind all of the applications created is a SQL Server database, SQL Server Management Studio was used to create the database, it’s tables and stored procedures. Using Management Studio meant that tables and views could be easily created with the user interface and stored procedures could be written using SQL.

The database was created first as that is what all of the applications rely on so there would be no point in developing anything else until the database was complete. The first table that needed to be implemented was the one used to store the locations.

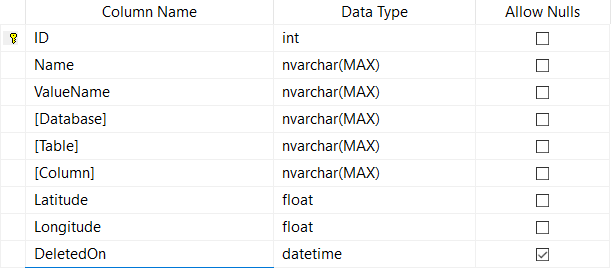


Figure x – design of the Location table

The Location table consists of the columns shown in figure x. All of the columns apart from DeletedOn are marked as not allowing nulls, this ensures that no data can be omitted when a row is inserted. The DeletedOn column is not null when a record is marked as deleted. Each record has a unique ID which auto-increments every time a record is inserted. The ID also doubles as the primary key for the table, having a primary key is important as it helps with the indexing of rows which makes the data faster to search once the table gets large.

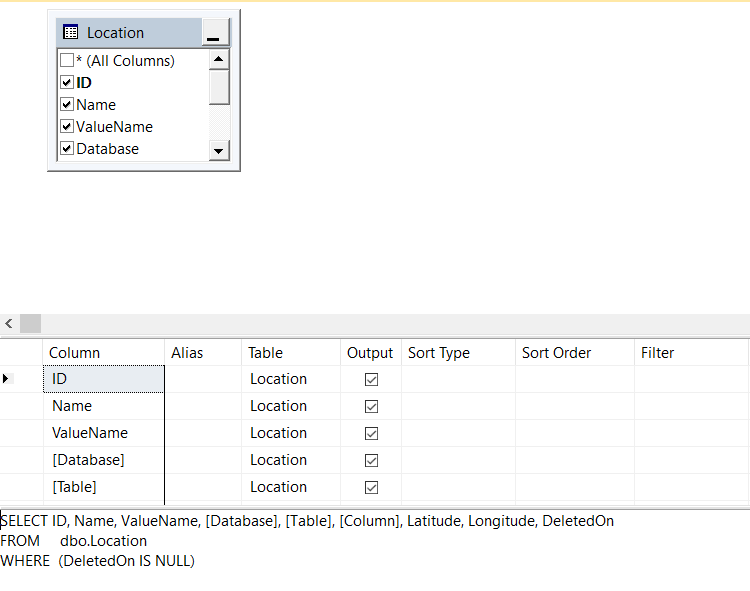


Figure x – active locations view

To make sure that only locations that were not deleted were returned, all stored procedures that read from the Location table go to a view that filters out any deleted records.

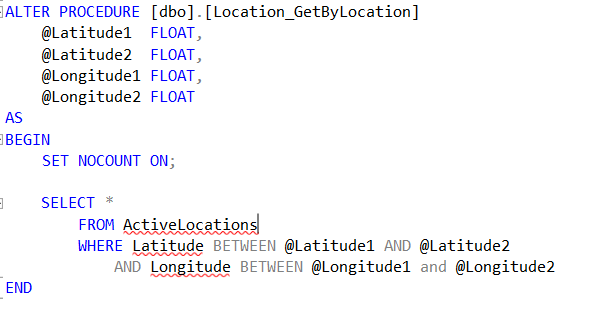


Figure x – stored procedure returning certain locations

Figure x shows the SQL code used to create the stored procedure that returns all the locations within the vicinity of the user. Two sets of latitude and longitude points are passed into the procedure and these are used to return all the points that lay in between. Stored procedures were used as having the logic in the database makes it a lot easier to make changes rather than having it in a server side application. The logic of the stored procedure can be changed without having to make any re-deployments. This method was also chosen as it is similar to the approach that applications at Sabisu use, having a similar architecture will make it easier for the client to pick up and maintain.

## Text File Parser

To get some realistic data, Teesside University was approached to see if there was any data that was numerical, had a timestamp, historical and was tied to a location. Initially data such as the number of people logged into a lab over a period of time was discussed. However, having this data could allow members of staff to be tracked even if the data was anonymised and this was against the universities policy. A dataset that was possible to obtain was the energy usage of seven university buildings over twenty days, this was ideal as it was measured every ten minutes, fluctuated throughout the day and each set of readings was related to a location.

However, when the data was obtained it was given as a series of text files, one file for each building. Since this was not the ideal format for the application the files needed to be parsed and inserted into a database table. The data was in the format: building/meter ID, date-time, reading value, the end of each reading was then terminated by a carriage return character.



Figure x – original data from text file

To get the data from the text files and into an SQL database a C# program was created. C# was used as it provided ways to read from files and insert into a SQL database without the need for any external libraries.

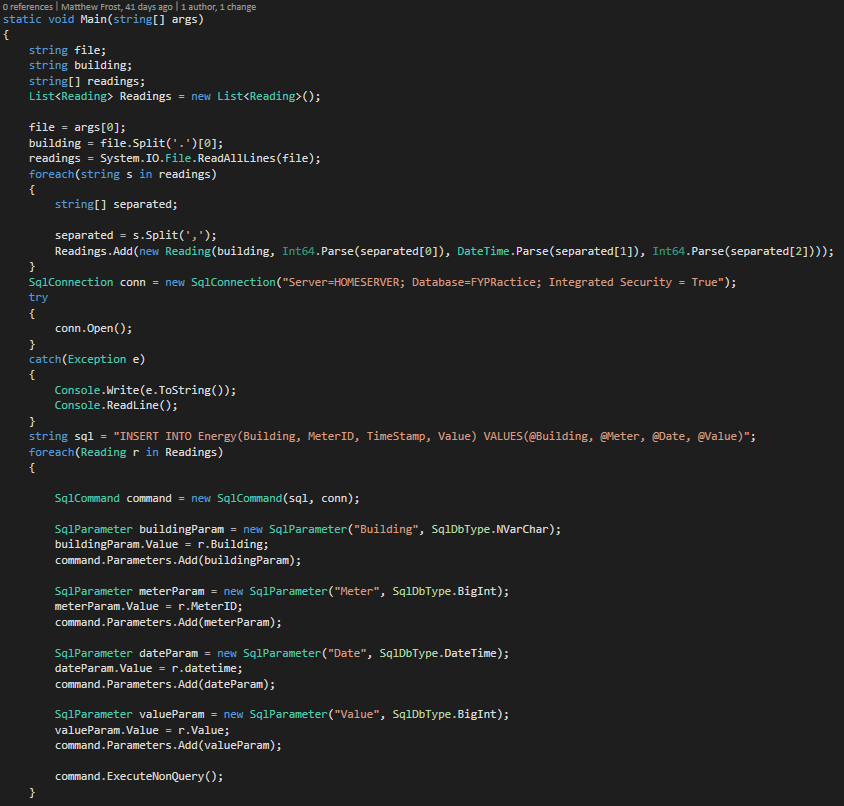
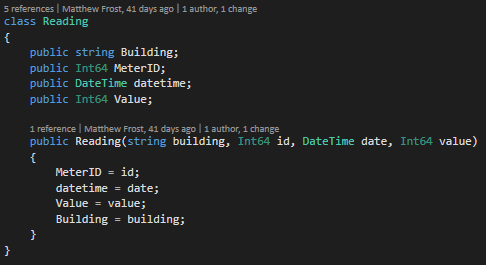


Figure x – Text file parser program

Figure x shows the full source code for the program used to parse the text files. The program is a console application that can be started from the command line with the name of a text file passed in as a parameter. The program gets the name of building from the name of the file and then starts to parse the file line by line. Each line is split into: meter ID, Timestamp and meter reading value. Each line is then used to create a “Reading” object, which is then inserted into a list of “Reading” objects. Once the entire file has been parsed, a connection to the SQL database and the list of “Reading” objects is then iterated over and inserted one by one.



## Deployment

# Testing & Evaluation

# Reflection

# References

# Appendices