Environmental Sensor Monitoring tablet application designed using cross-platform design patterns and frameworks

Tomasz Cielecki - s083134

Kongens Lyngby 2015

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Applications for mobile devices, including smart phones and tablets have become widely popular. The respective market places each contain hundreds of thousands of apps, the major ones contain millions of applications and each day these numbers increase. There are several ways to create these applications, either using the native development kits or using one of the several other methodologies which allow to target more than one operating system at once.

There are several problems and paradigms to address when creating applications using a cross-platform development kit, which this thesis will address. One of the important paradigms, is how to reuse code across different platforms, an often difficult task, due to reasons such as differences in programming languages, differences in how to construct User Interface code, differences in how to interact with hardware on the individual platforms.

This thesis focuses on how to create cross-platform applications using design patterns, which enable a great amount of code to be shared across platforms. This is shown by implementing a couple of tablet applications, used for environmental sensor monitoring, for Windows 8.1 and Android.

Resumè

Applikationer til mobile enheder, inklusive smart phones og tablets er fakta at de er blevet meget populære. De respektive markeder indeholder hver især hundrede af tusinder af apps, mens de største markeder indeholder millioner af applikation, hvor hver dag øges antallet. Der findes forskellige måder at lave disse application, enten ved at benytte sig af native udviklings værktøjer eller ved at benytte sig af en af de mange andre metoder, som muligør at målrette en application til mere end et operativsystem ad gangen.

Der er mange problemer og paradigmer at addressere, når man laver applikationer som benytter sig af disse cross-platform udviklings værktøjer, hvilket denne afhandling vil addressere. Et af de vigtigste paradigmer er, hvordan kode kan genbruges på tværs af forskellige platform, en ofte besværlig opgave at udføre på grund af forskelligheder i udviklingssprog, forskelligheder i brugergrænseflade kode og forskelligheder i hvordan interaktion foregår med hardware på de enkelte platforme.

Denne afhandling fokuserer på, hvordan cross-platforms applikationer kan udvikles, ved brug af design patterns, som mulliggør at en stor del af kode kan deles på tværs af platforme. Dette bliver vist ved at implementere nogle tablet applikationer, til brug til at overvåge miløj-sensorer, til platformene Windows 8.1 og Android.

Preface

This thesis is completed at DTU Compute in partial fulfillment of the requirements for acquiring the Master of Science degree in Digital Media Engineering.

The thesis addresses design patterns, in a cross-platform context, such that a larger part of a code base can be shared across applications.

The thesis is composed of a product, in form of several mobile device applications and a report to document them.

Lyngby, 19-January-2015

Tomasz Cielecki

Acknowledgements

This thesis would not have been possible without Brüel & Kjær Enivronmental Management Solutions (EMS). Hence, I would like to thank my supervisors Niels Bruun Svendsen and the Christian Bækdorf, for invaluable advice during the project and guidance towards understanding the subject. I would likewise thank Stig Høgh my supervisor from Technical University of Denmark (DTU) for guidance and supervision.

Additionally I would like to thank Brüel & Kjær EMS for providing office space, work computer and a Windows 8 Tablet, software licenses; time and knowledge.

I would also like to thank my family and friends, for understanding my dedication of time and supporting me during completion of this thesis.

Contents

Abbreviations

Abbreviation	**Explanation**
EMS	Brüel & Kjær Environmental Management Solutions
NMT	A Noise Monitoring Terminal or a location
RTNC	Real-Time Noise Control
NS	Noise Sentinel
UI	User Interface
OOP	Object Oriented Programming
SDK	Software Development Kit
IoC	Inversion of Control
DI	Dependency Injection
MVVM	Model-View-ViewModel
PCL	Portable Class Library
ACS	Azure Access Control
LOC	Lines of Code
IL	Intermediate Language
IDE	Integrated Development Environment

Chapter 1

Introduction

This chapter will focus on the background and history of Brüel & Kjær. This chapter will describe prior work done on similar projects to this dissertation. This chapter will describe the problems with device fragmentation and differences in platforms. It will then proceeded to the problem and thesis definition and the motivation behind it. Finally it will describe the delimitations and the methods, which are used in this thesis as well as an outline of this report's structure.

1.1 Brüel & Kjær Sound & Vibration Measurement A/S

> "Brüel & Kjær Sound & Vibration Measurement A/S supplies integrated solutions for the measurement and analysis of sound and vibration. As a world-leader in sound and vibration measurement and analysis, we use our core competences to help industry and governments solve their sound and vibration challenges so they can concentrate on their primary task: efficiency in commerce and administration." [Kjæ14a]

The company was founded in 1942 by Per Villhelm Brüel and Viggo Kjær and has since then developed and created over 3000 devices and solutions, which the company can offer to its customers [Kjæ14b]. It was acquired by Spectris in 1992 due to economical difficulties and the company was split into several separate companies.

- Brüel & Kjær Sound and Vibration Measurement A/S (the core sound and vibration market)
- Brüel & Kjær Vibro (machinery condition monitoring)
- B-K Medical (ultrasonic medical diagnostic instruments)
- Innova Air Tech Instruments A/S (gas analysis instrumentation)
- Danish Pro Audio (studio microphones)

1.1.1 Environment Management Solutions

In 2009 Brüel & Kjær acquired Lochard Ltd.[in09], this has since become the department Environment Management Solutions (EMS) [Kjæ14c]. EMS specializes EMS is where this thesis was made in collaboration with, specializes in Urban and Industrial environment management along with Airport environment management.

1.1.2 Noise Sentinel

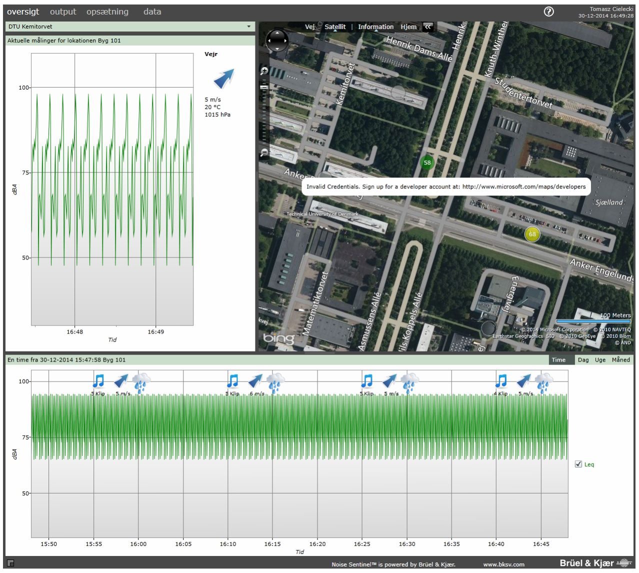
One of the products they provide is called Noise Sentinel [Kjæ14d], which is a solution where Noise Monitoring Terminals (NMT) are deployed on a site, where the customer (tenant) needs it. It then monitors that site 24/7 and records noise and weather conditions. Other sensors can also be mixed in, for vibration and dust particles.

The tenant gets several web interfaces, to control and manage their purchased product.

1.1.2.1 Noise Sentinel client

One of the interfaces is called Noise Sentinel client. This interface allows the tenant to set up rules for each individual NMT. These rules dictate when an

Figure 1.1: Screenshot of the Noise Sentinel client



oversigt
output
opsætning
data
Tomasz Cielecki
30-12-2014 16:49:28
DTU Kemitorvet
Aktuelle målinger for lokationen Byg 101
Vejr
5 m/s
20 °C
1015 hPa
dBA
Tid
Vej
Satellit
Information
Hjem
Invalid Credentials. Sign up for a developer account at: http://www.microsoft.com/maps/developers
En time fra 30-12-2014 15:47:58 Byg 101
Time
Dag
Uge
Måned
Leq
Noise Sentinel™ is powered by Brüel & Kjær.
www.bksv.com
Brüel & Kjær

alarm in the system is triggered for that location/NMT. Subsequently the alarms are used to notify the tenant, if set up in the rule, about this through e-mail and/or SMS. Additionally it is used in the reports the tenant generates from the system, which can be handed to the authorities, which require these.

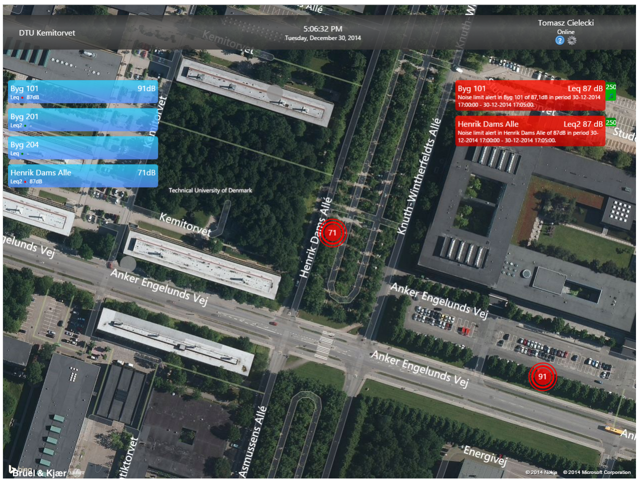
Apart from being able to set up NMT's in through the Noise Sentinel client, it is also possible to see the NMT's current location, data and historical data as well.

A screenshot of said client can be seen in Figure 1.1, where it is possible to see a map with several locations, historical data at the bottom and current data at the upper left part. For a bigger version see Appendix B.

1.1.2.2 Real-Time Noise Control client

When a location is exceeding the limits defined in the rules, they can be seen and managed in another web interface called the Real-Time Noise Control (RTNC) client. This client can be seen in Figure 1.2. The interface allows for looking at all the NMT's and the alerts for them. NMT's are shown on the map just like in the Noise Sentinel client, clicking on a NMT on the map, will draw a line to the name and current value inside of the associated blue box on the left hand side

Figure 1.2: Screenshot of the Real-Time Noise Control client



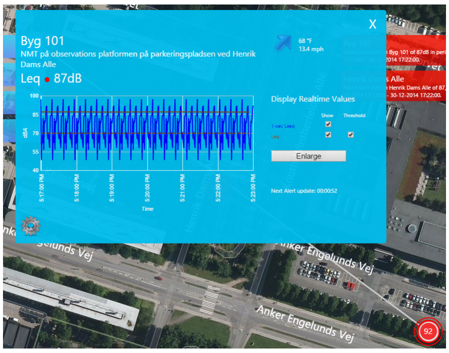
Byg 101 Leq 87 dB
Noise limit alert in Byg 101 of 87,1dB in period 30-12-2014 17:00:00 - 30-12-2014 17:05:00.

Henrik Dams Alle Leq2 87 dB
Noise limit alert in Henrik Dams Alle of 87dB in period 30-12-2014 17:00:00 - 30-12-2014 17:05:00.

see Figure 1.3. Clicking one of the red boxes on the right, which are alerts, will do the a similar thing, just by drawing a line to the NMT with the exceedance, see Figure 1.4 (for bigger versions of figures see Appendix B).

If an Alert has a sound clip associated, the person using the interface, has the ability to listen to said clip and further inspect, whether the cause of the exceedance, was indeed related to noise generated by the subject being monitored. If that was not the case, a comment can be added to the Alert, which will be added to the reports generated by the system.

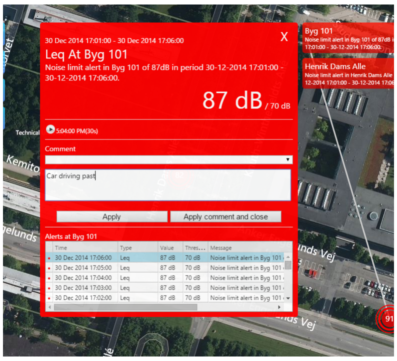
Figure 1.3: Screenshot of NMT information



Byg 101
NMT på observations platformen på parkeringspladsen ved Henrik Dams Alle
Leq ● 87dB

Next Alert update: 00:00:52

Figure 1.4: Screenshot of an Alert


30 Dec 2014 17:01:00 - 30 Dec 2014 17:06:00

Leq At Byg 101

Noise limit alert in Byg 101 of 87dB in period 30-12-2014 17:01:00 - 30-12-2014 17:06:00.

87 dB / 70 dB

5:04:00 PM(30s)

Comment

Car driving past

Alerts at Byg 101

Time	Type	Value	Thres...	Message
30 Dec 2014 17:06:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:05:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:04:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:03:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:02:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101

Byg 101

Noise limit alert in Byg 101 of 87dB i

17:01:00 - 30-12-2014 17:06:00.

Henrik Dams Alle

Noise limit alert in Henrik Dams Alle

12-2014 17:01:00 - 30-12-2014 17:06

1.1.2.3 OpenAPI

Recently EMS have been developing an API, which allows for developers to easier make applications, which communicate with parts of Noise Sentinel. This API provides several interfaces to i.e. identify clients, getting information about access to a tenants sites and locations, information to access resources such as Alerts, Real-Time data etc. This is an attempt to reuse parts of previous applications for internal use in EMS and other departments interested in using parts of Noise Sentinel in their applications.

1.2 Prior work

This section describes projects previously made at EMS, which this thesis will use as an underlying basis.

1.2.1 Cross-platform mobile notification system for noise monitoring system

The first project is *"Tablet Interface For Environment Monitoring"* by Jonas Lund [Lun13], which is about finding a suitable user interface for Tablet devices for Environment Monitoring with resulting in a product in form of a Mock-up, which can be seen in Figure 1.5. This was done with an emphasis on usability and user experience, and all the tests were made on mocked up data. So no

Figure 1.5: Screenshots of Jonas's Tablet Interface

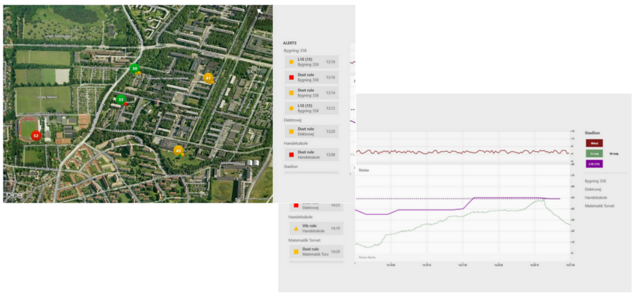
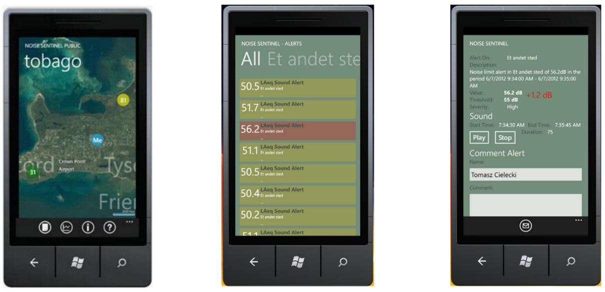


Figure 1.6: Screenshots of RTNC mobile App (Windows 7)



50.5 LAeq Sound Alert Et andet sted
51.7 LAeq Sound Alert Et andet sted
56.2 LAeq Sound Alert Et andet sted
51.1 LAeq Sound Alert Et andet sted
50.5 LAeq Sound Alert Et andet sted
50.4 LAeq Sound Alert Et andet sted
50.2 LAeq Sound Alert Et andet sted
LAeq Sound Alert

NOISE SENTINEL
Alert On: Et andet sted
Description:
Noise limit alert in Et andet sted of 56.2dB in the period 6/7/2012 9:34:00 AM - 6/7/2012 9:35:00 AM
Value: 56.2 dB +1.2 dB
Threshold: 55 dB
Severity: High
Sound
Start Time: 7:34:30 AM End Time: 7:35:45 AM
Duration: 75
Play Stop
Comment Alert
Name:
Tomasz Cielecki
Comment:

actual functionality was made for this project in terms of communicating with Noise Sentinel to get real data.

1.2.2 Internship & Bachelor project

The other project is the one made during my internship at EMS, which was a study on making a cross-platform applications across Windows Phone 7, Android and iOS. Taking a web application called Noise Sentinel and cramming it down into a series of native applications. The applications featured real-time data in form of the current dB level, from Noise Monitoring Terminals along with historical data for the last hour, day, week and month. This project was expanded on in my bachelor, *"Cross-platform mobile notification system for noise monitoring system"* [Cie12], project where these Apps were extended with RTNC functionality, which allowed them to receive notifications when a NMT had rule was broken. Additionally you could see details about the alerts, listen to the associated sound clips and comment on them. Screenshots of this App can be seen in Figure 1.6.

1.3 Device Fragmentation

There is a big variety of smart phone and tablet devices on the market, which have different screen sizes, performance, operating systems, market places, hardware features and software development kits (SDK), along with a multitude of other parameters. Additionally the amount of devices and features are ever increasing, with new and better hardware and design improvements, in both their hardware and software implementations. As an example one can look at a report OpenSignal has conducted on Android fragmentation [Rob12], where one can see that already back in 2012, there were 3997 distinct Android device models from 599 distinct brands. That number is increasing.

Since there is such a big device fragmentation along with different operating systems, there are several problems to address, when developing applications for those devices.

The main problems from a developers perspective are as follows.

- Difference in SDK's between the major platforms. Due to this fact, when accessing device specific features on one platform, the code is in fact not the same on the other platform.
- Difference in programming languages between the major platforms. Apple uses Objective-C and Swift, Google uses Java and Microsoft uses C# for their devices. This increases the difficulty in creating an application targeting all platforms as it has to be written one time for each platform.

There has been attempts to solve these problems, which each have their own unique solution. Among them are Cordova [Cor14] (formerly know as Phone-Gap), RhoMobile [Rho14] and Xamarin [Xam14a], which are the most popular ways of solving these problems. These frameworks will be discussed more in depth in Chapter 3.

1.4 The Problem

The problem, for Brüel & Kjær, reveal itself through the following sub problems. Firstly Brüel & Kjær, wishes to allow their customers to access their products from everywhere and on most devices. As it is right now Noise Sentinel is only accessible through modern Desktop browsers and not on mobile devices. Some things work works on a mobile devices, however, only partially. The second problem is that, Brüel & Kjær wishes to be effective in creating mobile applications and create code, such that parts can be reused in future applications.

1.5 Thesis Definition

The idea for this thesis is to take the two projects described in Sections 1.2.1 and 1.2.2 and combine them into a working application. The idea is to make this application work on at least two platforms. The aim will be to focus on design patterns and attempt to reuse as much code as possible between the two platforms. Then the idea is to measure the code to see how much of it was possible to be reused, how much was UI code and how much was platform specific code.

1.6 Report structure

The report will be structured as a application development report and will contain the following sections.

- **Introduction** - an introduction of the problem and project structure
- **Analysis** - an analysis of the project, use cases, requirement specification
- **Design** - an analysis of various design patterns and proposal of which to use and how the overall architecture of the Apps will look
- **Implementation** - details about the actual implementation, code snippets, actual implementation of design patterns
- **Tests** - description of tests made on the code and how it was achieved
- **Metrics** - code is measured to identify how much code was platform code, reusable code and view code
- **Discussion** - discussion of the findings
- **Conclusion** - evaluation of the project

1.7 Methodology

An agile approach will be used in the process of making this project. Daily scrum meetings were held with the development team in EMS, integrating with their agile process, allowing me to discuss problems, explain my progress and plan tasks.

The agile iterative model consisted of the following stages:

- **Analysis** - analysis of the component
- **Design** - design of the component
- **Implementation** - implementation of the design
- **Tests** - code tests: black box and integration tests

This was done in several stages of the project, attempting to not use excessive amounts of time for each part of the project, and not getting stuck on something; attempting to follow the agile manifesto [agi01].

Chapter 2

Analysis

This chapter prior work, which lead up to this project will be analyzed. Functional and non-functional requirements will be defined and the project will be delimited to that set of requirements. A domain analysis will be conducted to identify the key concepts which are relevant to this project.

2.1 Prior Work

As introduced in Chapter 1 this project is using two previous projects as a reference point. This section will analyze them and describe them in more detail, to clear up what they each do and not do.

2.1.1 Tablet Interface For Environment Monitoring

Briefly introduced in Section 1.2.1, Jonas Lund designed a Tablet Interface for the Windows RT platform. All data was mocked up. However, a lot of excellent thoughts and decisions were made during that project, which will be very useful and considered during this project.

The Windows RT platform, has matured since Jonas made his project and is now Windows 8.1 Modern Apps. Some of the guidelines have been changed, new visual controls have been added to the platform etc. Hence, some of the things designed might be different in this project.

To get an overview of the features mocked-up in his project see the following list:

1. Map with locations (NMTs) and Alerts to right of it, also described as overview mode (see Figure A.1)
2. Investigation mode, with Alerts to the left, graphs and legends on the right (see Figure A.2)
3. In Investigation mode, switch to a real-time graph
4. Switching mode on NMTs on the map between different environmental sensors. Noise, vibration and dust, indicated by different icons.
5. Context driven buttons in the bottom App bar (see Figure A.3 and (see Figure A.4))
    - (a) When an Alert is selected:
        - i. Pick a predefined comment
        - ii. Write your own comment
        - iii. Listen to associated sound clip
        - iv. Show Alert on graph in Investigation mode
    - (b) When no Alert is selected:
        - i. Control map
            - A. Lock map
            - B. Center map
            - C. Change map type
        - ii. Sorting Alerts according to Location, Type and Time

Looking at Figure A.4 from Appendix A, the flow of commenting a selected Alert can be seen. The visual representation of commented Alerts in the bottom right corner of the screen, seem off compared to all other visual elements and is in fact not part of any guidelines how to overlay the map. Chapter 4 will discuss how this will be implemented differently.

2.1.2 Cross-platform mobile notification system for noise monitoring system

Also briefly mention in Section 1.2.2, this thesis is based on the project, *"Cross-platform mobile notification system for noise monitoring system"* [Cie12], which implements some of the RTNC functionality on top of the product I made during my internship at EMS. The main focus in that project was to create the infrastructure required to push notifications from Noise Sentinel to Android, iOS and Windows Phone 7 devices notifying the user about Alerts happening on their site. The RTNC functionality of the project was built on top of what was made in my internship project.

The application featured the following for a single hard coded site.

1. Displaying a map with NMTs and their real-time noise value
2. Details for a selected location, displaying
   (a) Name
   (b) Image
   (c) Current noise level using a gauge visualization
   (d) Historic data
3. Page with graphs displaying historic data for all the locations stacked for comparison
4. Login with Azure Access Control
5. Subscription to Push Notifications
6. Displaying an unordered list of Alerts
7. Displaying details for an Alert including
   (a) Comments
   (b) Sound clip

Looking at the feature set in Section 2.1.1, they are very similar to the ones above. However, none of them are capable of logging a user into Noise Sentinel, none are able to get the tenants sites which they have access to.

The project is also not made with design patterns in mind. Nor was the code structured in ways to help it being shareable between platforms. The code itself uses file linking to include it in each project. UI behavior is duplicated between platforms along with the UI itself. Overall the code is far from optimal from both a architectural and code sharing perspective.

2.2 Domain Analysis

This section will concern the process of making a domain analysis. Which is described as follows.

> "The process of identifying, collecting, organizing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain"[KCH+90]

Brüel & Kjær EMS seeks to create an application they can offer to their customers. The purpose of the application is an application, which they can use when they are out in the field or in the office to monitor NMTs, to see whether they are inside the allowed levels dictated by the government for noise, vibration and dust pollution.

2.2.0.1 General Knowledge of the Domain

Tenant

- A tenant is one customer
- A tenant can have multiple users
- A tenant can have multiple sites

Site

- A site is associated with one tenant
- A site is associated to one or multiple NMTs

NMT

- A NMT contains the environmental sensors, which continuously monitor the environment
- The NMT is associated with a geographical location, which is also why it is called a location sometimes.
- A NMT is normally connected to the Internet, which allows it to send data to Noise Sentinel

Alert

- Generally most Alerts, when regulations are broken, are when construction is made, blasts are made in a mine or whatever the NMTs are monitoring. So during work hours.
- An Alert is associated to a Location/NMT
- Alerts can be associated multiple Alert Rules

Alert Rule

- A Alert Rule is a decision tree
- I.e. a rule could be the noise level cannot exceed 70dB during 8am to 5pm
- A Alert Rule decides to generate a sound clip for the alert or not
- It decides the length of the sound clip
- It can collapse Alerts
- and much more...

2.2.0.2 Users and Clients

The users of Noise Sentinel are managers who are responsible for keeping the site within the allowed values. Lets call them Site Managers. They are not the only users of the system. The Noise Sentinel support staff is also a user of the system, as they are providing support to customers to get them set up.

Noise Sentinel also allows for a public web site, which is the Noise Sentinel client mentioned in Section 1.1.2.1. However, it is stripped of its administration pages.

2.2.0.3 Environment

The environment Noise Sentinel is usually run in, is on modern browsers such as Chrome, Firefox and Internet Explorer. The Noise Sentinel Client requires Silverlight to run, which is available for Windows based systems. The RTNC client is a HTML5 web site, which runs in most modern desktop browsers without any additional plugins required.

Brüel & Kjær EMS wishes to support Windows 8.1 tablets, Android and iOS as well, covering a larger part allowing the customer to bring their own device and run Noise Sentinel anywhere they want. This also makes it easier for the site manager to inspect sites when out in the field, not having to lug around a laptop, checking the status on a smaller and lighter device.

2.2.0.4 Tasks and Procedures

When an Alert occurs, the site manager investigates whether or not it was the site that produced the exceedance, or whether it was exceedance coming from outside. This can be done by listening to the sound clip associated with the Alert, if the Alert Rule permits. The site manager can comment on the Alert either through a set of pre-defined comments or a new one. Other tasks include active surveillance of the site monitoring the RTNC client or passively by waiting for an SMS or e-mail.

2.3 Requirements Specification

The main problem in this thesis is to create applications using design patterns well suited for cross-platform applications to maximize code sharing. Hence, the non-functional requirements will reflect this.

The project will also be scoped in this specification and limited to a set of requirements, which is deemed fit to be able to implement during the time frame associated to the project.

2.3.1 Functional Requirements

The functional requirements will be split into two parts, as there will be implementation both on a server and an implementation client side.

2.3.1.1 Mobile Client

MFR1
The mobile client must identify itself with Noise Sentinel, such that a pairing between Noise Sentinel and the client is made, so each client can be uniquely identify itself.

MFR2
Client should have the possibility of presenting a collection of the sites the tenant has access to.

MFR3
If the user has an elevated role, he should have access to all the tenants and associated sites accessible for their role.

MFR4
The client should be able to give the user an overview of the geographical location of each NMT associated with a site.

MFR5
The client should be able to present details about a NMT, in form of name, current noise level, image and description. If a weather station is attached additional details should be available in form of wind direction, wind speed, temperature and current pressure.

MFR6
The client should be able to fetch real-time data for the NMTs associated to a site and feed it back to the user.

MFR7
The client should be able to fetch Alerts for a site and display them to the user.

MFR8
The client should be able to display details about an Alert in the form of noise level, rule, comments and description.

MFR9
The client should be able to play a sound clip associated to an Alert.

MFR10
Historical data should be accessible for a site in the form of last hour, last day, last week and last month historical data

2.3.1.2 Server

SFR1
An interface for allowing callbacks from Azure Access Control Service should be implemented, to allow devices to identify themselves and receive a token for future calls to the server

SFR2
The server should allow the client to get a collection of tenants accessible to the user

SFR3
The server should allow the client to access a collection of sites available to the user

SFR4
The server should implement an interface, to get a collection of tokens and information, which allow the client to access Azure services, which provide Alerts and Real-time data

2.3.2 Non-Functional Requirements

NR1
Tools are Windows 8.1 SDK, Visual Studio, Xamarin.Android and Xamarin.iOS. Due to EMS have heavily invested in this technology, switching to something else is not desired.

NR2
Programming language because of NR1 is C#

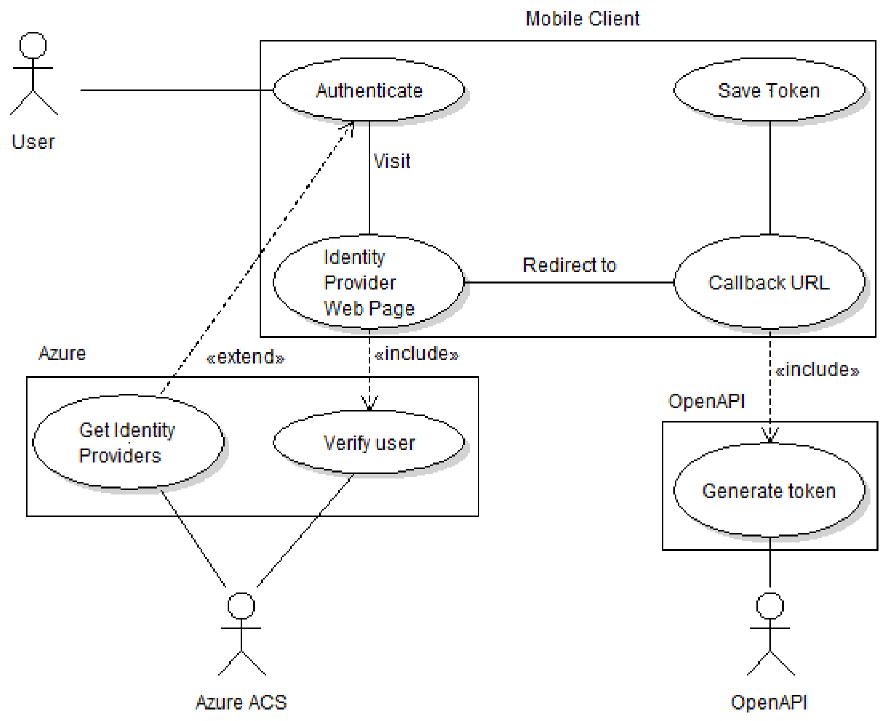
NR3
The design of the application has to made with good programming principles and design patterns

NR4
The priority of implementing the application is: Windows 8.1, Android then iOS in that order

2.3.3 Use Cases

Use cases provide a list of steps an Actor can do in a system. They can provide a short summary of what the system will offer and not offer, depending on the granularity in the use case. The following uses cases serve to provide an overview of the offered features of the system in this project.

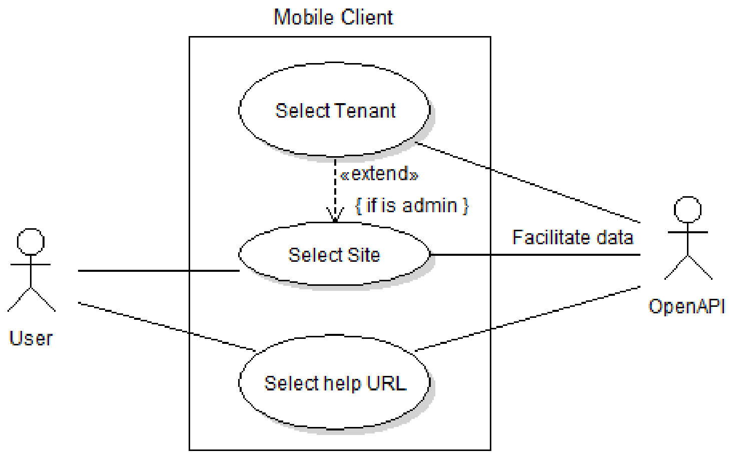
Figure 2.1: Use Case Diagram 1: Authentication



The first use case is concerning itself with Authentication. As OpenAPI requires all requests to contain a special token to communicate with it, the mobile client must be authenticated. Figure 2.1 describes how Authentication happens and is also described in Use Case 1.

Use Case 1	**Authentication**
Primary Actors:	• User • Azure ACS • OpenAPI
Preconditions:	None.
Postconditions:	User is authenticated.
Main Success Scenario: 1. Azure ACS provides a collection of Identity Providers 2. User picks an Identity Provider to authenticate with 3. The user is presented with a form to input their credentials 4. Credentials are verified with Azure ACS 5. The application is redirected to a Callback URL pointing at OpenAPI 6. The URL again redirects to a schema containing a token 7. Token is saved and user is now authenticated	
Extensions: 3.a Invalid credentials: 1. System shows failure message	

Figure 2.2: Use Case Diagram 2: Dashboard



Use case 2, is when the user is logged in, then they should be presented with sites they have access to. Additionally help links should be presented as well. If the user is an admin or have access to multiple tenants, they should be presented with the accessible tenants, before they are able to select a site. As seen in Figure 2.2 all the data is fetched from *OpenAPI* and is based on who the user is.

Use Case 2	**Dashboard**
Primary Actors:	• User • OpenAPI
Preconditions:	User is authenticated.
Postconditions:	• Site is selected or • Help URL is selected
Main Success Scenario:	

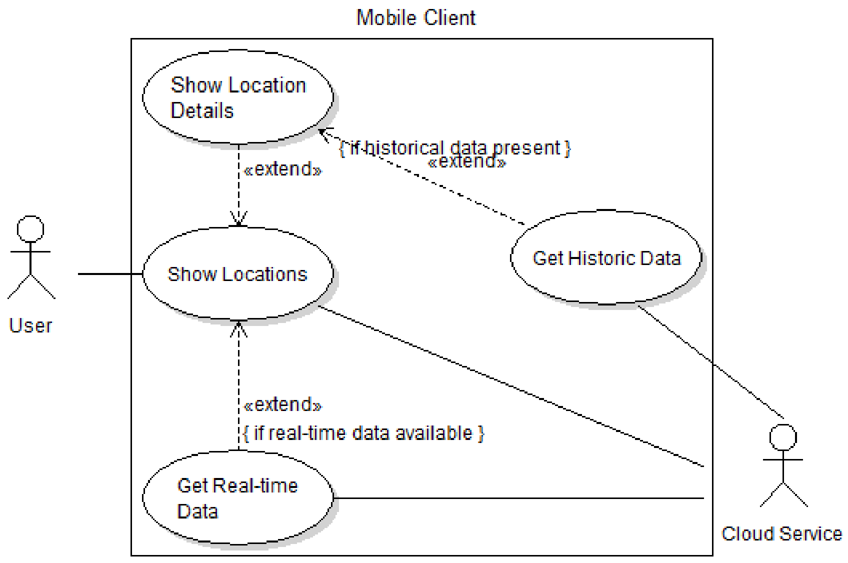
1. OpenAPI provides data for sites, tenants and help URLs
2. User selects a Site

Extensions:

2.a User is an admin or has access to multiple tenants:

1. User needs to select a tenant

Figure 2.3: Use Case Diagram 3: Locations and historical data



After the user has selected a site, as seen in Jonas's mockup, the user will be presented with the locations on a map. If there is real-time data available this is loaded. Location details can be shown along with historical data for that location. As seen in Figure 2.3 the responsibility of the Cloud Service is to provide the historical data, locations and real-time data.

Use Case 3	**Locations and historical data**

Primary Actors:	• User • Cloud Service
Preconditions:	• User is authenticated. • Site is selected.
Postconditions:	None.

Main Success Scenario:

1. OpenAPI provides data for associated locations to the selected site
2. User sees locations

Extensions:

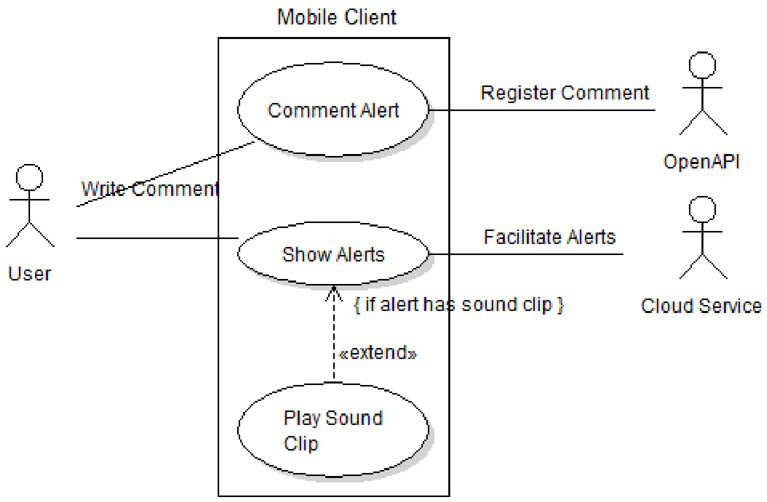
2.a If real-time data is available:

    1. Real-time data is continuously fetched for each location and displayed

2.b User shows details for location:

    1. If historical data is present for the location, it is displayed to the user

Figure 2.4: Use Case Diagram 4: Alerts



Use Case 4	**Alerts**
Primary Actors:	• User • OpenAPI • Cloud Service
Preconditions:	• User is authenticated. • Site is selected.
Postconditions:	Alert has been commented.
Main Success Scenario: 1. Cloud Service provides Alerts for the site 2. User shows an Alert 3. User writes a Comment on an Alert (a) Comment is sent to OpenAPI where it is registered	

Extensions:

2.a If sound clip is available:

1. User can play sound clip associated with Alert

2.4 Chapter Summary

To sum up this chapter it can be told that prior work was described in more depth. A brief domain analysis was conducted, followed by a requirements specification. Finally uses cases were presented to describe the desired behavior.

Chapter 3

Technology Analysis

In this chapter the technologies used in this project will be analyzed. They are an integral part of the solution and the design. Xamarin's approach to cross-platform programming will be explained. Along, with some very useful design patterns, which help decouple code along with providing a framework to easily share code between different platforms. Among them is Inversion of Control (IoC).

3.1 Xamarin

Ximian was founded in 2000, back when Microsoft announced their .NET framework [ZDN00], to create an Open Source counterpart, Mono [dI03]. Since then it was bought by Novell and in 2009 the first version of MonoTouch was released [ZDN09]. MonoTouch, was the forerunner for Xamarin.iOS and allowed writing native applications using C# for iOS. In 2011 Novell was bought by Attachmate [MW10], which later proceeded to lay off over 800 employees, including majority of the Mono team as it did not believe in the future of MonoTouch [GC11]. Later that year Xamarin was founded [dI11] continuing where it left off at Novell with the MonoTouch and Mono for Android project, which since have become Xamarin.iOS and Xamarin.Android and more products have been added to the lineup.

What Xamarin.iOS and Xamarin.Android offer is to write the same language across two platforms, where applications normally are written in Objective-C/Swift for iOS and Java for Android. This language is C# and builds on top of Mono. Without thinking much about it, it is easy to see that opens up for a lot of code that can be shared across platforms, which was not as easy to do before. What is unique about this the solutions they provide is that all the code, which is written is turned into native applications, using the native visual controls on each of the two platforms and providing access to the native API's and last but not least, provides apps with native performance. Since the Windows platform also uses C# as language it opens up for the possibility to target a lot of different platforms with the same piece of code.

Additionally Xamarin also offers plugins for Visual Studio, such that C# and .NET developers, which are used to that integrated development environment (IDE), will feel at home when creating applications using Xamarin.

3.1.1 Xamarin in Comparison

There are three approaches to develop mobile applications. Either writing a mobile web-page, which runs in the browser of the mobile device; writing a hybrid application, a run "write once, run everywhere" approach, which in short terms is a web-page wrapped in a native application. Making it possible to leverage more of the built in capabilities of the devices. Lastly there is the native approach, using the native SDK for the respective platforms. The latter approach is where Xamarin rests, with an abstraction on top.

3.1.1.1 Mobile Web-pages And Hybrid Applications

Writing mobile and responsive web-pages, which work on all mobile devices and their different browsers can be quite some work. You also get a limited subset of available features you can access on the phone. Browsers are also fragmented and have different feature sets they support and some require different CSS tricks to make applications look the same across these different browsers. This is also a problem in Hybrid Applications, most of the time the Hybrid Application frameworks such as the ones mentioned in Section 1.3, mitigate this and make these tricks and tweaks for the developer, such that they should not handle these situations. However, there are several other problems, which can be listed in the following pros and cons.

Pros:

- The library takes care of the platform specific differences and the same language is used across all platforms. You write code once and it works on all devices.
- Many frameworks use JavaScript or some other well known language, which many developers are already familiar with. Makes for less of a learning curve.
- The interface of the application can be debugged in a web browser.

Cons:

- Apps are dependent on the web browser component the SDK provides, which can vary in performance.
- Interface look and feel are not the same as in the native applications.
- Only a subset of device specific functionality is available and depends on the framework, which features are available.

3.1.1.2 Native Applications

Writing native applications using the respective SDK's gives a lot more power to the developer, compared to the web-app or hybrid application approach. This is due to the developer has access to the full API the SDK provides. However, these SDK's have different languages. Hence, there are inherit problems with writing native applications. Such as if you want to cover all major platforms, it will require developers to know each of these platforms or having multiple teams doing one platform each. Due to differences in languages there will be multiple code bases. So the pros and cons for this approach is as follows.

Pros:

- The developer has access to the entire Native API for the respective platform.
- The developed apps will have native performance.
- Applications will inherently have native look and feel.

Cons:

- Applications will have to be built multiple times, as code cannot be shared between platforms.
- There is a potential need for multiple teams handling their own platform.
- Tools differ from platform to platform.

3.1.1.3 Xamarin Applications

Writing applications using Xamarin, does not mean that you get a black magic box, like when writing hybrid apps that spits out applications for each platform. The developers still have to know each platform and their API's, as Xamarin is a wrapper around the native API, such that the developer can write C# code. However, as mentioned before this greatly increases the amount of sharable code, while having full access to the native API keeping most of the pros from the native approach, while eliminating a lot of the cons.

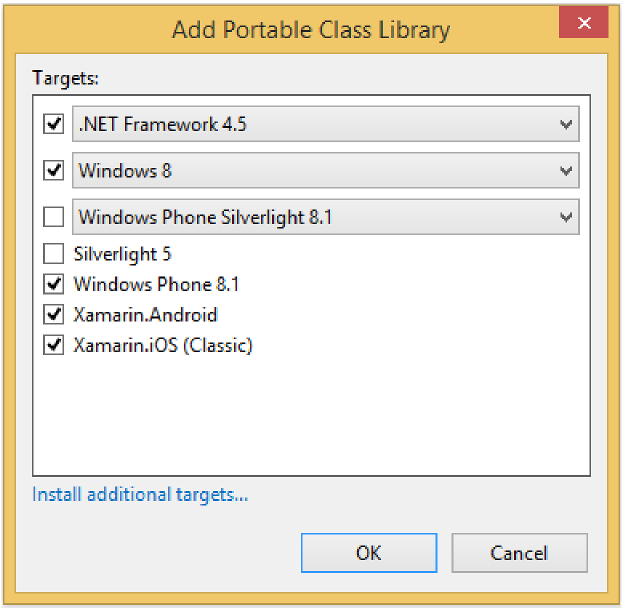
Pros:

- Same pros as for Native Applications 3.1.1.2.
- Same language across all supported platforms, which means more potential code sharing between them.
- Modern language features, such as language-level asynchronous programming, which eliminates bookkeeping, callbacks and frameworks for doing asynchronous tasks. Support for lambda expressions and much more.
- Use the same tools to create Windows, iOS and Android applications.

Cons:

- Additional yearly cost to allow develop with Xamarin tools.
- Not as big community as the native counterparts, but it is growing.
- Added size of applications, due to the overhead of Mono libraries.

Figure 3.1: Creating a new Portable Class Library project


Add Portable Class Library

Targets:

- [✓] .NET Framework 4.5
- [✓] Windows 8
- [ ] Windows Phone Silverlight 8.1
- [ ] Silverlight 5
- [✓] Windows Phone 8.1
- [✓] Xamarin.Android
- [✓] Xamarin.iOS (Classic)

Install additional targets...

OK Cancel

3.2 Portable Class Libraries

Portable Class Libraries (PCL) have been a part of C# and .NET since 2011 [Mic11] and allow developers to create assemblies, which work on more than one .NET platform. It was officially added to Xamarin late 2013 in version 4.10.1 and Mono 3.2.3. What this means is that when you create a new project of the PCL type, you get the choice of which platforms you want to support as seen in Figure 3.1. No additional configuration or tweaks are needed to reference the PCL into another project being one of the targeted platforms.

Depending on which platforms the developer targets in the PCL project, it unlocks a subset of the .NET framework, which the developer can use in the code written in the PCL. This in turn means that platform specific code in the PCL is not supported. Although only a subset of the .NET framework is supported, it gives the developer a chance to create a lot of the model code used through the various applications and platforms referencing this library. Meaning, a larger part of the code can be shared between them. Using design patterns such as described in Section 3.3, it is possible to use platform specific code inside of the PCL through an abstraction.

3.3 Inversion of Control

Inversion of Control (IoC) is a characteristic of a framework and not a design pattern or a specific implementation. Inversion of Control comes in various forms of which Dependency Injection and Service Locator will be described in this section. What Inversion of Control is great for and used for in many frameworks is to look up an implementation of an interface in a plugin. It is a way of removing dependencies from a class.

When not using the principle of IoC, it is very common to hard code the classes and objects that you want to interact with inside a class. The objective of IoC is to remove that responsibility from the class and instead provide it a way to get these objects and classes, either through Dependency Injection or using a Service Locator.

The objectives of IoC are as follows.

- Decoupling of the usage of a class from the actual implementation.
- Focusing a class on a specific task.
- Remove assumptions on what classes do and how they do it; instead rely on contracts.
- Minimizing side effects of replacing a class with something else.

IoC is also often referred to as the *"Hollywood Principle: Don't call us, we'll call you"*, which in a nutshell describes what Ioc is all about.

3.3.1 Dependency Injection

Dependency Injection is a design pattern, where dependencies are injected or passed as reference to a dependent object or class. It separates dependencies from their own behavior.

Listing 3.1: Class without dependency injection

```
public class Lamp {
    private LightBulb _lightSource;

    // Constructor
    public Lamp() {
        _lightSource = new LightBulb();
    }

    public void TurnOn() { _lightSource.TurnOn(); }
}
```

Listing 3.1 describes how code would look like when no Dependency Injection is used. The *Lamp* class itself hard codes which light source it uses and needs to know that the *LightBulb* class has a method *TurnOn()* in order to interact with it. In other words, this class only knows how to use *LightBulb* instances and swapping that out with another light source, such as a LED or a candle is not possible.

Listing 3.2: Class with dependency injection

```
public interface ILightSource {
    void TurnOn();
}

public class Lamp {
    private ILightSource _lightSource;

    // Constructor
    public Lamp(ILightSource lightSource) {
        _lightSource = lightSource;
    }

    public void TurnOn() { _lightSource.TurnOn(); }
}
```

Listing 3.2 shows how Dependency Injection through constructor injection works. Here the *Lamp* class has no knowledge about what light source it gets, the responsibility is up to the framework to pass in an instance implementing the *ILightSource* interface. This makes the *Lamp* class free of knowledge about its surroundings and in turn leaves it a clean class.

There are also other types of Dependency Injection, where code is injected through either a setter method, where a method that the injector uses is exposed by the client. There is also interface injection, where the dependency defines an interface, which needs to be implemented by the dependent class, where just like in setter injection, a setter method needs to be implemented and exposed.

It is apparent that using Dependency Injection requires a component where instances of dependencies are created and where instances of dependent classes are created. These are often referred to as IoC containers and are either stand alone libraries or part of a framework [Cla13, p. 28].

The pros and cons of Dependency Injection are as follows.

Pros:

- Can be introduced as refactoring as it does not require change in behavior.
- Classes are easier to test as they are more independent. Because of this, objects can easily be mocked or stubbed.
- Knowledge about concrete implementations of dependencies can be removed entirely from dependent classes, as all they need to know is the interface they implement.
- System configuration can be externalized, and depending on the situation different implementations of the interface can be injected.

Cons:

- Code can be harder to read because behavior is moved away from construction of dependent classes. More files need to be referred to, to find out what the system does.
- Decreases encapsulation as the developer needs to know how it works and not just what it does [Mar11].
- Increases coupling to the IoC container or framework and can create pain when wanting to change the way injection works.

3.3.2 Service Locator

Just like Dependency Injection the Service Locator design pattern's objective is to remove the dependency of a dependent class. This is done by having a single object knowing how to get hold of all the dependencies. So that object will have a method, which helps to return an instance of a dependency when it is needed.

One approach to the Service Locator could be a singleton which holds instances of dependency classes, like in Listing 3.3. This of course needs to be created somewhere and populated with actual instances of the dependencies. Whether this is done through a configuration file or through code is up to the developer and the *ServiceLocator* implementation.

Listing 3.3: Singleton Service Locator

```
public class ServiceLocator {
    public static ILightSource LightSource() {
        return Instance._lightSource;
    }

    private static ServiceLocator Instance;
    private ILightSource _lightSource;
}
```

A more dynamic approach to the *ServiceLocator* class would be to define a generic method, which takes the interface type as argument and returns an instance of that. This would have an internal collection of all the instances of dependencies as seen in Listing 3.4.

Listing 3.4: Generic Service Locator

```
public class ServiceLocator {
    public static T Resolve<T>() {
        return (T)_instances[typeof(T)];
    }

    private static ServiceLocator Instance;
    private Dictionary<Type, object> _instances;
}
```

Either way there is a setup process where these instances need to be registered which could look as seen in Listing 3.5, which also shows an example of the usage of the *ServiceLocator*.

Listing 3.5: Service Locator Setup and Usage

```
public class ServiceLocator {
    public static void Register<T>(T instance) {
        _instances[typeof(T)] = instance;
    }
}

// Registering
public void SetupServiceLocator {
    ServiceLocator.Register<ILightSource>(new LightBulb());
}

public class Lamp {
    private ILightSource _lightSource;

    public Lamp() {
        // Usage of ServiceLocator
        _lightSource = ServiceLocator.Resolve<ILightSource>();
        // or
        _lightSource = ServiceLocator.LightSource();
    }

    public void TurnOn() { _lightSource.TurnOn(); }
}
```

The pros and cons of using the Service Locator design pattern is as follows.

Pros:

- Code can be added without needing to re-compile the application.
- Large parts of code can be separated and the only link between them is the Service Locator.

Cons:

- Hides dependencies of dependent classes as compared to Dependency Injection it does not expose its dependencies.
- Harder to maintain code, because it is unclear when breaking changes are introduced due to the hidden dependencies.
- All tests need to interact with the global service locator needing to register mocked dependencies under test, opposed to Dependency Injection where dependent classes can be tested in isolation.
- Can potentially become a bottleneck in concurrent applications.

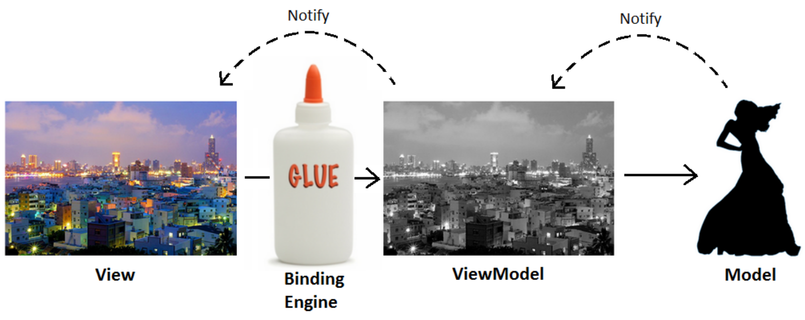
3.3.3 IoC Summary

If a generic approach to IoC is used such as shown in Listing 3.5 or an equivalent to Dependency Injection, it makes it possible to load platform specific code, which implements the interface defined for the dependent class, into a PCL. As long as the dependency class implements the defined interface, any piece of code can be registered at run-time which makes it very flexible.

3.4 Model-View-ViewModel

Model-View-ViewModel (MVVM) is a popular design pattern, used and loved by many .NET developers. It was first presented as the Presentation Model (PM) by Martin Fowler back in 2004 [Fow04]. Fowler, presented a pattern where an abstraction of the a view was represented in a Presentation Model. This PM updates the view and stays in sync with it. In late 2005 John Grossman presented the MVVM pattern [Gro05], which is identical to Fowler's PM.

Figure 3.2: Illustration of the MVVM design pattern



Although, being identical, it was still a generalization and specialization of PM intended for WPF. Compared to PM, where the synchronization logic, which updates the UI is contained in the PM, MVVM relies on a binding engine. This binding engine comes with WPF for free and allows the developer to create expressions to for example bind a *Label* and its text property to a property in a *ViewModel*. So in short terms the *ViewModel* does not need to know about the view itself. Creating *ViewModels* with Dependency Injection can make them entirely platform independent and be put into a PCL, which in turn can be used across different platforms.

Figure 3.2 describes the flow and dependencies of the MVVM design pattern. Left most is the View, which receives input from the user and displays values from the *ViewModel*. In Windows applications normally the XAML provides the glue or binding engine. In Xamarin projects, the developer has to provide this themselves or through a 3rd party framework. The binding engine's responsibility is to marshal bound properties from the *View* to the *ViewModel*. *ViewModels* in turn must implement an interface, *INotifyPropertyChanged*, which is used to trigger updates on the *View*. Hence, it is also the binding engine's responsibility to listen to these triggers and update the *View* accordingly.

When the *ViewModel* has its properties modified by user interaction, it can in turn can update one or several *Models* which it uses as data source. User interaction on buttons and alike, happens through the *ICommand* interface, which the binding engine also knows of. This *ICommand* implementation can update properties on the *ViewModel*, interact with *Models* etc.

In other terms the *ViewModel* is an abstraction of the *View*, its purpose is to format the necessary data in a way so it is suited for presentation, as a *Model* is not always representative of how the values in it should be shown in a visual representation.

3.5 Frameworks

There are several frameworks which work for both Xamarin and Windows 8.1 as are the target platforms of this project. These framworks include MVVM implementations, UI abstractions, IoC in various forms and much more. This section will describe what they each do and in the end it will compare the frameworks to each other.

3.5.1 ReactiveUI

> "A MVVM framework that integrates with the Reactive Extensions for .NET to create elegant, testable User Interfaces that run on any mobile or desktop platform. Supports Xamarin.iOS, Xamarin.Android, Xamarin.Mac, WPF, Windows Forms, Windows Phone 8 and Windows Store apps." [tea14]

ReactiveUI was started by Paul Betts back in 2010 and is a MVVM framework. What makes it unique compared to other solutions, is that it builds on top of Reactive Extensions [Mic12] (Rx). Rx makes it very easy to create applications which are event based, as Rx handles this very elegantly for the developer. Additionally, Rx provides a simple binding engine to wire up *Views* with *ViewModels*. Lastly, it also provides a way to navigate the application from a *ViewModel* and a simple implementation of the *Service Locator* design pattern.

3.5.2 MvvmLight

> "The main purpose of the toolkit is to accelerate the creation and development of MVVM applications in WPF, Silverlight, Windows Store (RT) and for Windows Phone." [Bug09]

MvvmLight was created by Laurent Bugnion in 2009. As the name indicates is also a MVVM framework. Not only does it work for the platforms mentioned, it recently also got support for Xamarin projects.

MvvmLight does more than just being an MVVM framework. It is a framework, which encourages the usage of MVVM and design patterns. This includes Dependency Injection, Service Locator and Publisher Subscriber messaging pattern. It also provides *ViewModel* navigation and a simple binding engine like Rx.

3.5.3 MvvmCross

Another very popular MVVM framework is MvvmCross [Lod10], which was started by Stuart Lodge in 2011. It was specially developed for the Xamarin platform and implements many of the same features as MvvmLight does along with some extra very useful things.

In addition to MvvmLight, it also provides a Plugin framework which builds around the built in IoC functionality. It provides a more advanced binding engine, which allows for creating bindings in the AXML markup used on Xamarin.Android similar to what you do with bindings in XAML. The language used for these bindings is advanced an allows for composite bindings, inline value conversion, if statements and more. This can be useful if the developer wishes to move some of the behavior out of a *ViewModel* and into a *View*. It also provides a large amount of plugins for various functionality such as GPS, sound effects, getting accelerometer data, file system interaction and many more. There is also a big community which creates plugins for MvvmCross.

3.5.4 Xamarin.Forms

Xamarin.Forms [Xam14b] is a new addition to the product lineup from Xamarin. It is a framework which abstracts creation of UI, such that it can be written once and create native UI on the supported platforms. The supported platforms are Xamarin.Android, Xamarin.iOS and Windows Phone 8. While not supporting Windows 8.1 tablet applications it is still a very interesting framework, because of its ability to write UI in a PCL and have it working on the supported platforms.

Xamarin.Forms also has a binding engine allowing MVVM frameworks to have their *ViewModels* bound to the Forms UI. While only having a limited default UI control types so far, it is still possible to extend that repertoire by adding your own and wrapping them in classes that Xamarin.Forms can interpret. This potentially will grow a community creating more UI controls for Xamarin.Forms. Xamarin.Forms also implements IoC through the Service Locator pattern.

3.5.5 Frameworks Comparison

To compare each framework a matrix, which can be seen in Table 3.1, has been made. Features, each framework implement are listed and marked with ✓if present in the framework and marked with ✘if not present.

	ReactiveUI	MvvmLight	MvvmCross	Xamarin.Forms
Service Locator	✓	✓	✓	✓
Dependency Injection	✘	✓	✓	✘
Subscriber/Publisher Messages	✓	✓	✓	✓
INotifyPropertyChanged	✓	✓	✓	✘
ICommand	✓	✓	✓	✘
ValueConverters	✓	✓	✓	✘
Navigation Service	✓	✓	✓	✓
Simple Bindings	✓	✓	✓	✓
Advanced Bindings	✘	✘	✓	✘
Plugins	✘	✘	✓	✘
View Abstractions	✘	✘	✘	✓

Table 3.1: Framework feature comparison

At a glance it can be seen that the MVVM frameworks are very similar in features implemented with few features not being implemented by ReactiveUI and MvvmLight.

3.6 Chapter Summary

This chapter covered what Xamarin is and compared it to other approaches to mobile development. It covered Portable Class Libraries, which is a means to create Library projects which target multiple platforms. It covered the Inversion of Control principle with specific design patterns, which use this covering Dependency Injection and Service Locator. This chapter also covered the Model-View-ViewModel design pattern, which is a means to decouple Views from behavior and Model code using an abstraction and a binding engine. Lastly this chapter covered some MVVM frameworks including ReactiveUI, MvvmLight and MvvmCross, but also Xamarin.Forms, which is not a MVVM framework, but rather an UI abstraction.

Chapter 4

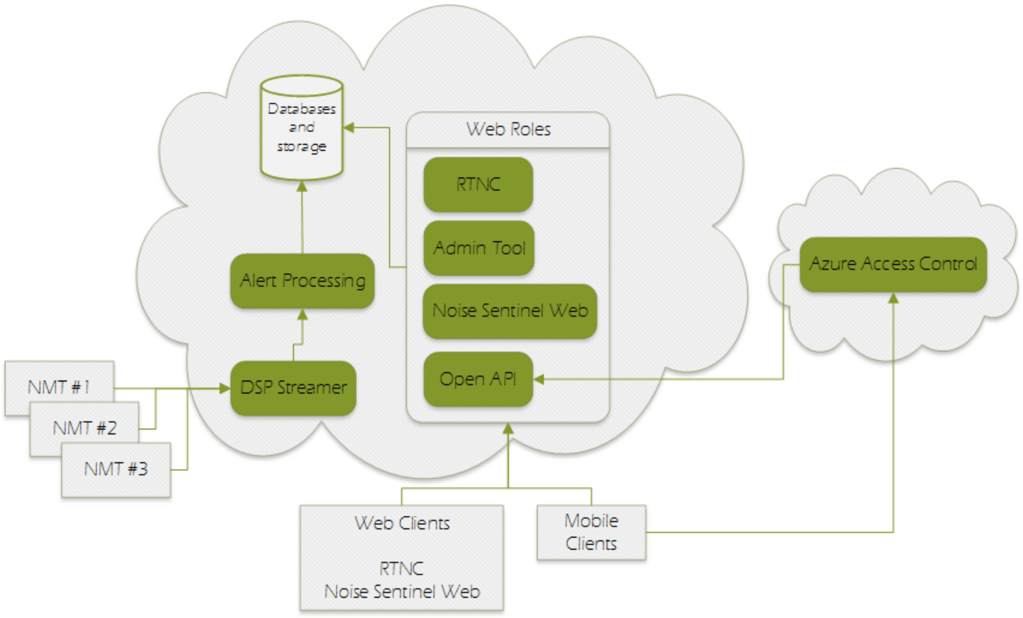
Design

This chapter covers the general design of the applications made in this thesis. It describes how components interact with each other and how the structural design of the applications are.

The design decisions of this chapter are all based on the design patterns described in the Chapter 3 and the attempt is to maximize the amount of code that goes into the Portable Class Library projects, such that code is shareable between the mobile platforms.

Figure 4.1 describes the overall architecture of how Noise Sentinel works from the Mobile Client's point of view. Starting from the left, NMT's connect to a streamer service in the cloud, which processes all the sound data. This data is sent to an alert processing component, which applies rules to the data the NMT's provided. Alerts and data is then saved to databases. Sound files generated from Alerts are stored in Azure Blob Storage and Alerts themselves are stored in Azure Table Storage. Several Web Services and Web Roles are hosted in the same environment, which serve Web Clients such as RTNC, administration tools, Noise Sentinel client and OpenAPI and many more. On the rightmost side of the figure, Azure Access Control can be seen. It provides the ability to allow customers to use their preferred Identity Provider, being either Google, Facebook, Yahoo or others. The association with OpenAPI, which is also described in Figure 2.1, is due to a callback, which also helps pair a device to

Figure 4.1: Overview of Noise Sentinel architecture

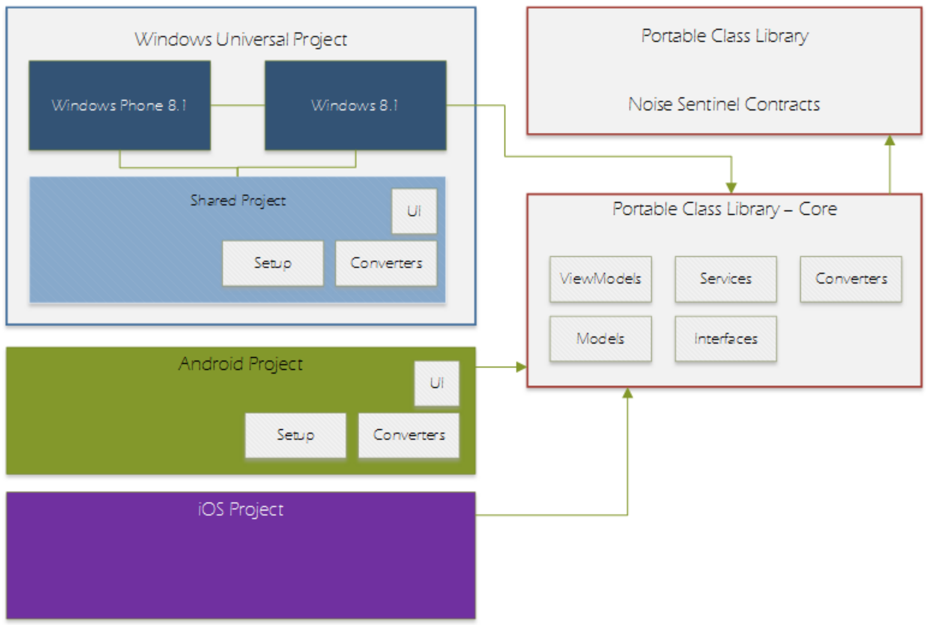


Noise Sentinel. It will be described in more depth in Section 4.1.3.1.

4.1 Mobile Applications

The overall structure of the mobile applications can be seen in Figure 4.2. It has been chosen to split the project up such that all platform specific code is isolated from code that is cross-platform. Hence, in this case there is a Windows Universal project containing a Windows Phone 8.1 project along with a Windows 8.1 project. They both share code through a Shared Project. This shared project contains the UI, the setup process for registering services and platform specific code in the IoC container along with native value converters for formatting values in *ViewModel* bindings. The same goes for the Android project. The iOS project is there to validate that the code also builds on that platform as well and validates that the referenced code from the PCL is in fact cross-platform.

Figure 4.2: Overall Structure of Applications



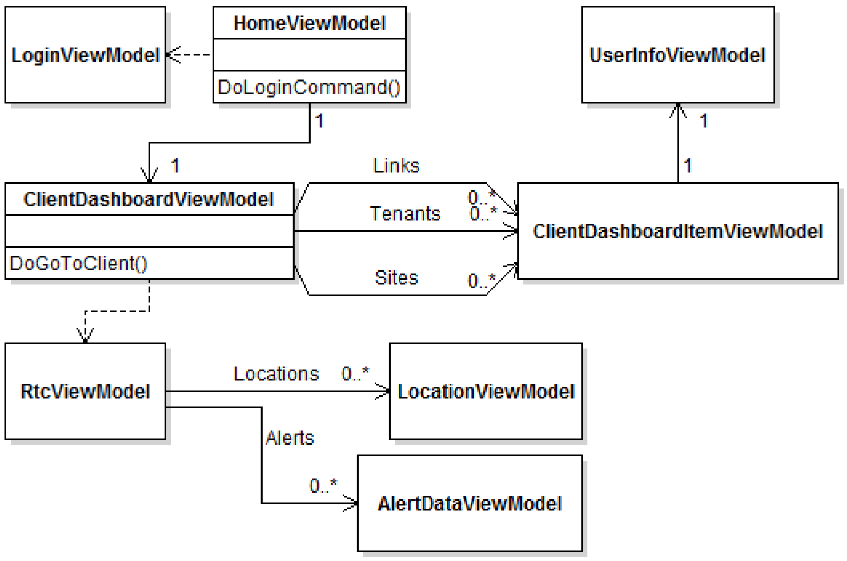
4.1.1 Core

The Core library is a Portable Class Library, which contains most of the application logic, including *ViewModels*, *Interfaces*, *Services* which implement these Interfaces, Models and Converters. The choice of using a PCL instead of a Shared Project is due to the fact that, Shared Projects do not compile to a DLL file, which means that distribution of a Shared Project is more cumbersome as you will have to distribute a project with several files instead of a single DLL.

4.1.1.1 ViewModels

There are several *ViewModels* contained inside the Core PCL. Each *ViewModel* represent an abstraction of a *View* and as described in Chapter 3 dependency injection is used where applicable through constructor injection. This makes it a lot easier to take a *ViewModel* and test it in isolation as when instantiating it will make it clear to the tester what dependencies it needs. A class diagram describing the associations between the *ViewModels* can be seen in Figure 4.3.

Figure 4.3: ViewModel Class Diagram



AccountSettingsViewModel - The responsibility of this *ViewModel* is to help authenticate a user. It uses the plugin described later in Section 4.1.3.1, to load the available Identity Providers to be displayed in the associated *View*. Additionally when the user is logged in it presents the ability to log the user out. This *ViewModel* is in fact only used in the Windows 8.1 project as the navigation flow differs from what is possible and available on the other platforms. The equivalent *ViewModel* on the other platforms is the *LoginViewModel*.

This *ViewModel* is highly dependent on the *HomeViewModel*, which will be described later. This is due to how it is visually represented on Windows 8.1, using a *Flyout* in the right side, meaning that the *HomeViewModel* is still loaded, while the *AccountSettingsViewModel* is presented. When a Identity Provider is selected from the list, the *HomeViewModel* is notified and calls the Authentication plugin described in Section 4.1.3.1, which in turn presents a modal web browser. This is the reason of the dependency, because the *Flyout* cannot present the modal web browser.

AlertDataViewModel - This *ViewModel* is simply prepares values for visual representations from an Alert. This is done from the two models *AlertData* which contains all the necessary information about the Alert, but also *SoundRequestData* which contains information about the sound clip associated to the Alert, if any.

ClientDashboardItemViewModel - This *ViewModel* is a representation of the *ClientDashboardItemV1*, which is a contract used in communication with OpenAPI. *ClientDashboardItemV1* contains an internal collection of itself, making it recursive. The *ClientDashboardItemV1*, which the name also hints is used to describe an item in a Dashboard. This model is used to give the mobile client a list of Tenants, which each has a collection of accessible Sites. It is also used to give the mobile client a collection of URL's for help documents.

ClientDashboardViewModel - This *ViewModel* orders the *ClientDashboardItemViewModels* such that they are divided into lists of Tenants, Sites and Links to make it easier to represent these lists visually on the different platforms. This *ViewModel* also helps searching through Tenants, as in cases where an admin is using the (see Use Case 2, extension 2.a) application several Tenants will be displayed.

HomeViewModel - This *ViewModel* is what the application starts with visiting. If the user is not logged in a welcome message is presented it contains a *Command* for directing the app to the *LoginViewModel* or on Windows 8.1 to the *AccountSettingsViewModel*. If logged in a *ClientDashboardViewModel* is fetched using the *AppClientService* described in Section 4.1.1.2. When it is fetched the bound view presents the *ClientDashboardViewModel*.

LocationViewModel - This is a representation of an NMT, and it contains properties to display the values it can display. It also contains historical data for graphs.

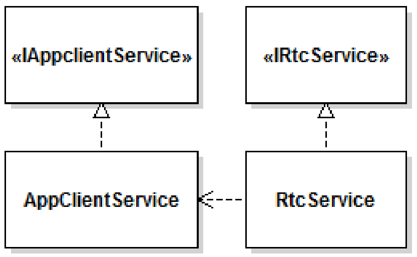
LoginViewModel - Similar to the *AccountSettingsViewModel* this *ViewModel* is simply used on the non-Windows platforms to allow the user to log in their selected Identity Provider.

RtcViewModel - or Real-time control *ViewModel* has most of the core functionality in this project besides *HomeViewModel*. This *ViewModel* interacts with all the services described in Section 4.1.1.2 to fetch NMT's, their real-time data, historic data and alerts for each NMT or location.

4.1.1.2 Services

There are two interfaces in the core, which are shown in Figure 4.4, one for each Service. These are used for the dependency injection. As already described, each dependency each *ViewModel* holds are injected through the constructor, hence there is a need to have a contract for each dependency in order to inject it. This also means the *ViewModels* don't know which exact implementation

Figure 4.4: Services class diagram



they get.

The two services in this project are **AppClientService** which is a helper service to communicate with OpenAPI. It's job is to facilitate Use Case 2 shown in Figure 2.2 in Chapter 2. It helps fetching a *ClientDashboardSetV1* model and serve it to the interested *ViewModels*. It also fetches access information for the Noise Sentinel Real-Time Data service and fetches Alerts from Azure Table Storage and Sound data from Azure Blob Storage.

The other service is the *RtcService*, which has the responsibility of fetching all the locations from the Real-Time Data Service, it fetches Real-Time data for the locations along with historical data.

How the two services interact the *RtcViewModel* can be seen in Figure 4.5, which is a sequence diagram. The *RtcViewModel* calls the asynchronous *Start()* method, which makes the *RtcService* start fetching data for the Site that was selected on the dashboard. First it figures out whether it has the access data to the web services it needs to call. If not, this information is fetched from *OpenAPI* using the token from the authentication process described in Section 4.1.3.1. When it has all the access information, it can proceed fetching the inventory for the site from the Real-Time Data Service. This inventory is a collection of all the locations that are associated with the Site. Afterwards it starts fetching real-time data, which it keeps looping over and over again until the *RtcViewModel* calls the *Stop()* method on the *RtcService*.

4.1.2 Application Projects

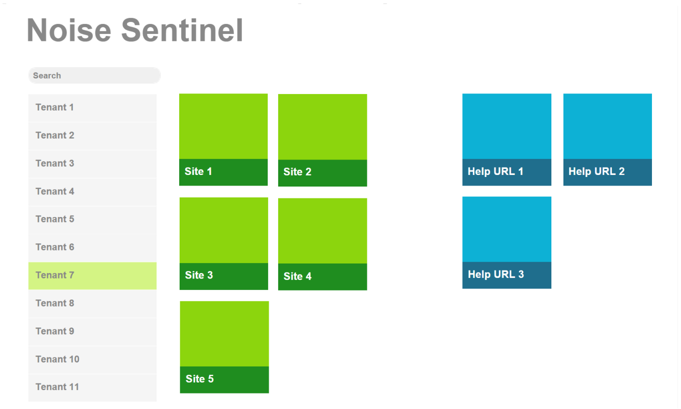
As seen in Figure 4.2 there are three application projects, two for each of the Windows platforms, which is a Windows Universal project containing a project for both Windows Phone 8.1 and Windows 8.1, along with a shared project.

Then one project for Android and iOS each. Since the Windows Universal projects contain the same API for both the phone and non-phone platform, it uses Shared projects to share platform specific code between the two project.

Each of the platforms are responsible for their own UI code, set up of the IoC container mapping the interface to the actual implementation and lastly for special platform specific converters to use to convert properties in *ViewModels* which need to be converted to platform specific models. For instance when wanting to present colors on the different platforms, a converter must be made for that platform to be able to bind it to the model the *View* expects to see.

4.1.2.1 User Interface

Figure 4.6: Mock up of the Dashboard

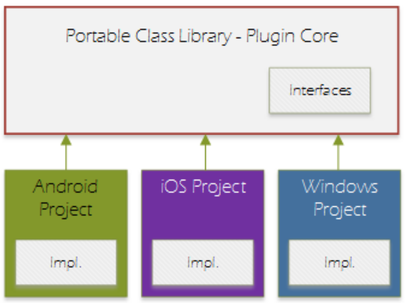


The mock-up from introduced in Chapter 1 and discussed in Chapter 2, which Jonas made had a couple of design elements, overlayed the map with menus. There are no official guidelines telling that this is prohibited or bad practice. However, looking at many of the other apps that have been made since back when Jonas made his mock-ups, there is a general trend showing that this is not something that is practiced a lot. However, there is a general trend where Windows 8.1 applications which use Hub controls, expand in sections which are also used to display details about items there. Hence, a decision was made to reflect these trends and move the UI a bit around. Looking at Appendix C, it can be seen in Figure C.5 how it has been chosen to add a detail section for a location, when it is selected on the map. It expands between the map and the Alert section. More important is moving away from the proposal of having the overlay menu with comments for Alerts that was in Jonas's mock-up, which has been made into a section which expands on the right of the Alerts, when an Alert is selected. This can be seen in Appendix C Figure C.6. This provides a UI more true to the trends and is what a Windows 8 user would expect when using the application.

Additionally a mock-up was made for Use Case 2, which describes the action of selecting a Tenant and Site. This can be seen in Figure 4.6, see Appendix E for a larger version. Leftmost is a list of Tenants, when the user is an admin, as they normally have access to all the Tenants created in the system. A search box above has been made to be able to filter through them. When selecting a Tenant it reveals the Sites that Tenant has access to. If the user is not an admin, the sites are simply shifted to the left simply showing the Sites the user has access to.

4.1.3 Plugins

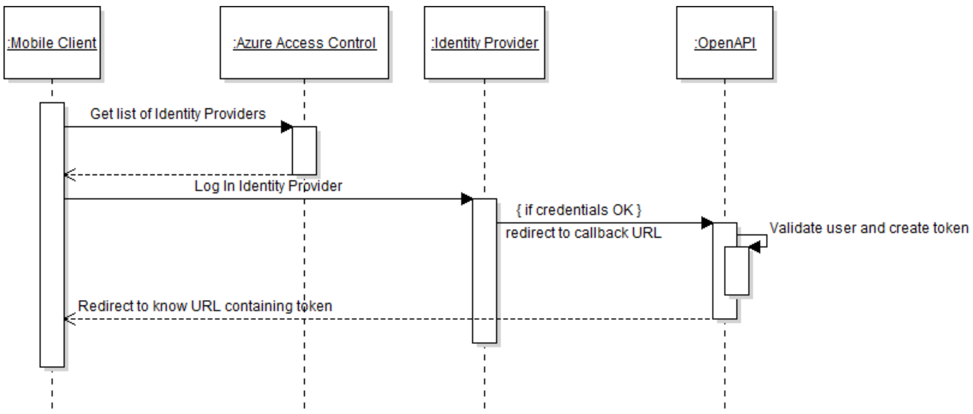
Figure 4.7: Plugin Structure



Unlike the core library where most of the code is application specific, plugins are a place where you can put in code, which you want to share across different applications. In this project several of 3rd party plugins were used, for things such as handling persistent storage for settings, generating application ID's by getting information about the hardware, better HttpClient handler implementation and more. However, one plugin was created for this project, which is an Authentication plugin, which helps with the workflow of how Azure Access Control handles authentication. What is common about these plugins how they are generally structured. All plugins start with a PCL library containing all the interfaces describing what the plugin does. Cross-platform implementations also go in this PCL. However, the as seen in Figure 4.7, it is not uncommon to have a project for each platform you want to support with a platform specific implementation. This way in the setup process in the different application projects the actual implementations can be registered in the IoC container for use through the application.

4.1.3.1 Authentication

Figure 4.8: Azure Access Control Sequence Diagram



Authentication using Azure Access Control (ACS) works by using the OAuth protocol. The sequence diagram in Figure 4.8 describes how it works along with Noise Sentinel using the OpenAPI. The Mobile application requests the available Identity Providers it has access to from ACS. This information is gathered from a *Realm* and *Name Space* value generated in the ACS control panel. The user can now pick either of the Identity Providers return in the list. Upon doing so a web browser needs to be opened to authenticate with the Identity Provider. After the user is logged into the Identity Provider, the web browser is redirected to a callback URL, which in this case is an URL for the OpenAPI along with a unique application identifier, such that a device can be paired to the authenticated user. This information is stored by OpenAPI and a token is generated for that user to use OpenAPI in future API calls. Which is retrieved by the client when the web browser again is redirected to a URL, which the web browser component knows is the end of the process and the URL is parsed to retrieve the token and it is then usable by the client.

4.2 OpenAPI

OpenAPI is a ASP.NET MVC project which follows a RESTful [Fie00, p. 76] approach to designing the web service. This means it follows a set of guidelines on the architecture used to design the web service. Applied to OpenAPI, this means that there is a uniform interface for identification of resources. Meaning, that the URLs and methods to access resources are descriptive of what you get and how you get it.

There are two methods needed for the application in the OpenAPI. The ability to retrieve a Dashboard and also very important, the ability to retrieve access information, for how and which signature to use when calling various services in Noise Sentinel and when fetching Alerts from Azure Table storage and Sound clips from Azure Blob storage.

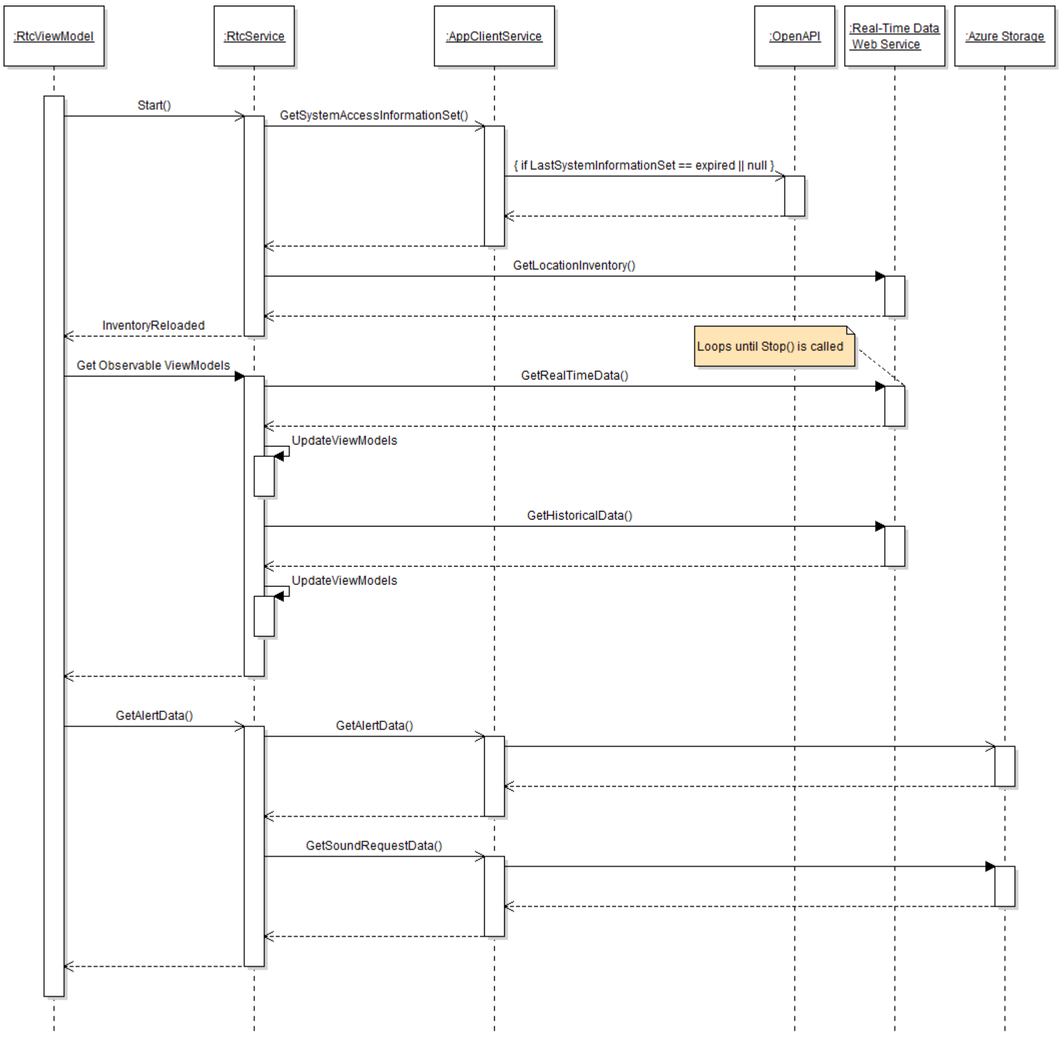
All communication between OpenAPI and clients calling it is designed to use contracts which are serialized at the OpenAPI to be transferred to the client. These can in turn be deserialized by the client into an C# class instance.

All requests to OpenAPI need to have the correct header, which contains a Simple Web Token (SWT), which is generated from the token retrieved from the Authentication process described in Section 4.1.3.1. If it is not present the OpenAPI will return with a *Unauthorized* HTTP status, not delivering any data. By using this header with the token, the user can be identified and the correct resources can be fetched for that user

4.3 Chapter Summary

This chapter described the overall architecture of the project. It described how the applications are split into a core part and several application projects, which are designed to split shareable code from platform specific code. The services in the core PCL were described along with their roles and their behavior. User interface was described using mock ups and a change to Jonas's proposal was decided to follow trends of applications already released in the market. Plugin structure and design of the Authentication plugin was described. Lastly the RESTful OpenAPI web-service was described and which calls can be made and how a user identifies himself with it.

Figure 4.5: Services Sequence diagram



Chapter 5

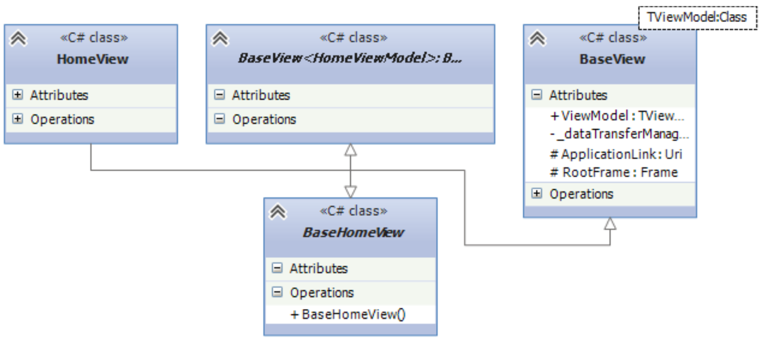
Implementation

This chapter will concern itself with the implementation of the applications described in the previous Chapter 4. It describes how generic constructs are used to conveniently help the developer do less casting of types. It describes how services are using asynchronous programming with the async/await keywords.

Looking at Table 3.1 in Chapter 3 it shows that MvvmLight and MvvmCross are both very feature rich frameworks with features that can be useful when creating cross-platform applications. However MvvmCross has an edge as it supports bindings in the Android markup and has a plugin architecture, which enhances breaking out code into smaller tasks, which can be generalized and used across different applications. Hence, the choice has been made to implement the applications using this framework.

5.1 BaseView

Figure 5.1: Class Diagram showing BaseView and usage



MvvmCross did not have generic *View* implementations at the time of writing the code, for the Windows platforms. Hence, instead a *BaseView* implementation was made to accommodate this. *BaseView* is a convenience class, which helps casting the *ViewModel* property of the View to the type passed into the generic implementation. This removes the need to cast it manually every time the property is accessed. Looking at Figure 5.1 it can be seen how the *BaseView* class is inherited all the way down to the actual view implementation, in this case the *HomeView*. Although, a bit hard to see from the class diagram, the *HomeView* depends on the *HomeViewModel*, but not the other way around. This is a general characteristic of *ViewModels*, as they never know anything about a View in the MVVM pattern. Listing 5.1 shows the generic implementation of *BaseView*, which is marked *abstract* such that an instance of it can never be instantiated directly.

Listing 5.1: BaseView implementation

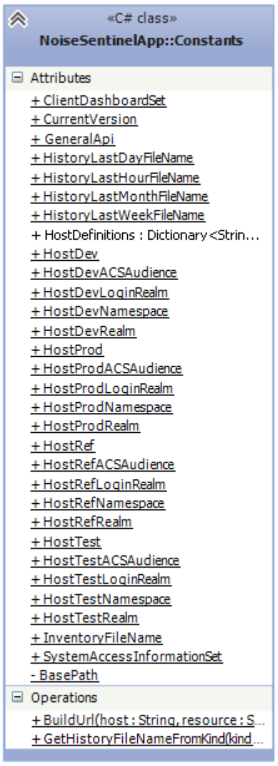
```
public abstract class BaseView<TViewModel>
    : MvxWindowsPage where TViewModel : MvxViewModel
{
    public new TViewModel ViewModel
    {
        get { return (TViewModel) base.ViewModel; }
        set { base.ViewModel = value; }
    }

    // other code here
}
```

5.2 Constants

The applications need to know how to contact the various services involved. There are also several environments, which Noise Sentinel uses for development, test, reference and production, which are deployed in that order. Hence, a class *Constants*, which is depicted in Figure 5.2, has been made which handles all the URL's for these systems. As the name indicates it has been chosen to hard code these into the application.

The class has a single Dictionary where the keys are the mentioned environments above. When needing the information for the development environment, they key is simply *dev*, and returns an object with all the necessary information to contact that environment, but also the necessary information on which *Realm* and *Name Space* values to use for Azure Access Control in order to get the correct list of Identity Providers. Additionally it has a method *BuildUrl* which takes a host name, and one of the defined API names also defined in the class, which builds the exact URL to communicate with an API.

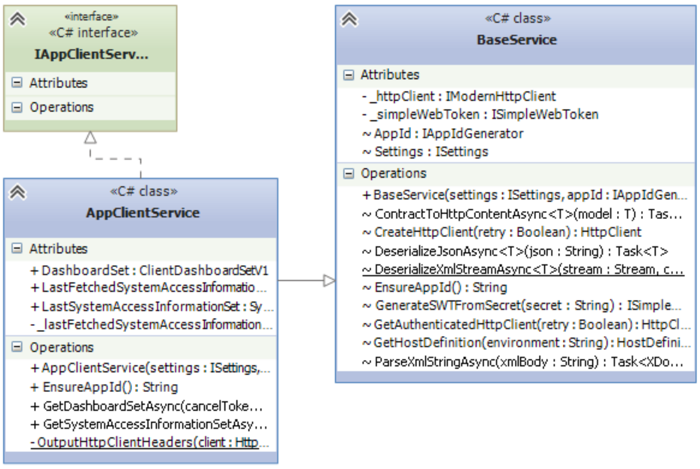
Figure 5.2: Class Diagram showing the Constants class



5.3 AppClientService

AppClientService's responsibility is to fetch access information, which provides signatures and URLs for the services containing real-time data and alerts. It also has the responsibility of fetching a dashboard set from OpenAPI. Seen in Figure 5.3 it inherits from *BaseService* which is a common class for *AppClientService* and *RtcService* as both the services communicate with OpenAPI.

Figure 5.3: Class Diagram of AppClientService


«interface»
«C# interface»
IAppClientServ...
Attributes
Operations
«C# class»
AppClientService
Attributes
+ DashboardSet : ClientDashboardSetV1
+ LastFetchedSystemAccessInformatio...
+ LastSystemAccessInformationSet : Sy...
- _lastFetchedSystemAccessInformation...
Operations
+ AppClientService(settings : ISettings,...
+ EnsureAppId() : String
+ GetDashboardSetAsync(cancelToke...
+ GetSystemAccessInformationSetAsy...
- OutputHttpClientHeaders(client : Http...
«C# class»
BaseService
Attributes
- _httpClient : IModernHttpClient
- _simpleWebToken : ISimpleWebToken
~ AppId : IAppIdGenerator
~ Settings : ISettings
Operations
+ BaseService(settings : ISettings, appId : IAppIdGen...
~ ContractToHttpContentAsync<T>(model : T) : Tas...
~ CreateHttpClient(retry : Boolean) : HttpClient
~ DeserializeJsonAsync<T>(json : String) : Task<T>
~ DeserializeXmlStreamAsync<T>(stream : Stream, c...
~ EnsureAppId() : String
~ GenerateSWTFromSecret(secret : String) : ISimple...
~ GetAuthenticatedHttpClient(retry : Boolean) : HttpCl...
~ GetHostDefinition(environment : String) : HostDefini...
~ ParseXmlStringAsync(xmlBody : String) : Task<XDo...

In short *BaseService* serves the purpose of creating an instance of *HttpClient* which has all the correct headers set, such that when calling OpenAPI, it will not get a *Unauthorized* return message.

The two methods *GetDashboardSetAsync* and *GetSystemAccessInformationSetAsync* are using the async/await keywords, which were added in a recent C# version. This allows for easily executing code asynchronously without using callbacks. All awaited methods in the core library are using *ConfigureAwait(false)* as this prevent switching of context when the asynchronous operation has finished. Normally this is set to true, and what happens is that before executing an async method, the code normally remembers which context it was running on before the execution of the code. Then when the operation finishes it switches back to that context. Since the method is contained inside a library which does not have a notion of main threads etc. switching context is not important. When doing many small async operation the switching of context can also impose an overhead on the application since it needs to switch the context every time, unless using *ConfigureAwait(false)*. This also solves the problem of deadlocks when calling the asynchronous methods in a synchronous fashion by requesting the *Result* property of a *Task*.

The way the two methods work is that they each call OpenAPI with the authenticated *HttpClients* they get from *BaseService*. The response data by *OpenAPI* is simply deserialized into models described in the Noise Sentinel contracts, which are shown in Figure 4.2 in Chapter 4. After the deserialization of the response the objects can now be used throughout the application just like any other C# class instance.

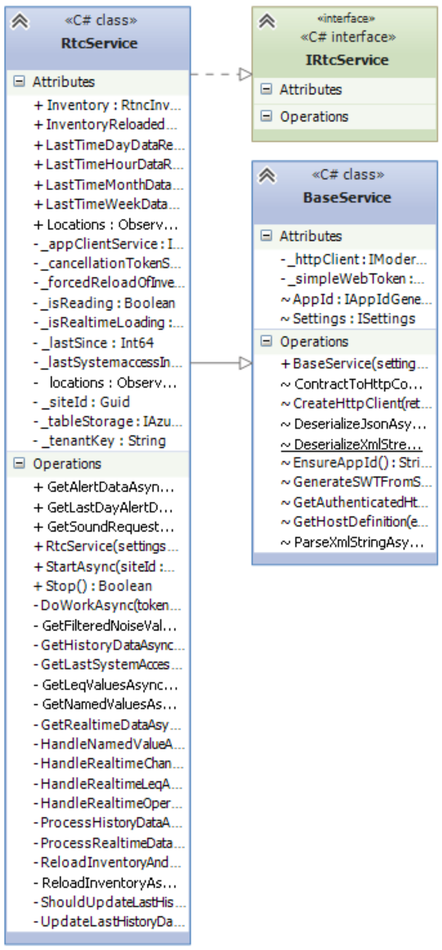
5.4 RtcService

Exactly like the *AppClientService*, *RtcService* also inherits from *BaseService* in order to get an authenticated *HttpClient*. It also takes the *IAppClientSerivce* as one of its arguments in the constructor as it needs the *SystemAccessInformationSet* which *AppClientService* helps fetching. This dependency is resolved by the Dependency Injection MvvmCross provides.

RtcService is slightly more complex than the *AppClientService*. Its purpose is to do the heavy lifting of fetching Real-time data and Alerts for the *RtcViewModel* and the associated *LocationViewModels*. It contains two methods which starts and stops the fetching of Real-time data for a specific site. Namely *StartAsync()* and *Stop()*. As the first method indicates, it starts something asynchronously. What it exactly does is to set up a *CancellationTokenSource* for when *Stop()* is called to shut down gracefully. It also awaits the private method *DoWorkAsync*, which can be seen in Listing 5.2. This method, runs every half second to fetch Real-time data and historical data. The *Stop()* method calls *Cancel()* on the *CancellationToken*, which has been passed into *DoWorkAsync* and all the methods it awaits.

The *GetLastSystemAccessInformationSetAsync()* method checks whether a

Figure 5.4: Class Diagram showing the RtcService



System Information Set has been fetched and that it has been less than 20 minutes since last time it fetched it from the server. This is due to the keys contained in the set are regenerated and only give temporary access to the application to access the resource. If this is the case, it asks the *AppClientService* to fetch a new set.

Listing 5.2: BaseView implementation

```
private async Task DoWorkAsync(CancellationToken token)
{
    _isReading = true;
    _forcedReloadOfInventory = true;

    while (_isReading)
    {
        await GetLastSystemAccessInformationSetAsync(token)
              .ConfigureAwait(false);
        await ReloadInventoryAndFireIfReloadedAsync(token)
              .ConfigureAwait(false);
        await GetRealtimeDataAsync(token)
              .ConfigureAwait(false);
        await GetHistoryDataAsync(HistoryDataKind.Hour, token)
              .ConfigureAwait(false);
        await GetHistoryDataAsync(HistoryDataKind.Day, token)
              .ConfigureAwait(false);
        await GetHistoryDataAsync(HistoryDataKind.Week, token)
              .ConfigureAwait(false);
        await GetHistoryDataAsync(HistoryDataKind.Month, token)
              .ConfigureAwait(false);
        await Task.Delay(500, token).ConfigureAwait(false);
        token.ThrowIfCancellationRequested();
    }
}
```

5.4.1 Inventory Reloading

ReloadInventoryAndFireIfReloadedAsync(), like the name indicates reloads the inventory of Locations. This only happens if there is no current inventory or the boolean *_forceReloadOfInventory* is set to true. If the inventory was reloaded the event *InventoryReloaded* is fired to notify the *RtcViewModel* about this change.

5.4.2 Real-time Data

GetRealtimeDataAsync(), is where the most of the magic happens. What it does is to download an XML file which on the web service is updated every half second. This XML file contains all the real-time data information for the past 2 minutes, just to make sure that if a hiccup occurs in the system somehow, the past 2 minutes can be recovered and still be displayed in the application. The available data from this XML file is the current noise level, pressure, wind speed, temperature and wind direction. It also contains notifications from the web service about reloading the Inventory of location, this is in cases where locations are added or removed from the site that is currently monitored.

The *RtcService* has an *ObservableCollection* of *LocationViewModels* which is where all real-time data is populated and inventory changes are made to. This is used by the *RtcViewModel* as source for its locations, and it keeps a reference it. The smart thing about *ObservableCollections* is that MVVM implementations understand how they are updated as they implement the *INotifyCollection-Changed* and *INotifyPropertyChanged* interfaces, which are integral to MVVM. So when the reference to the *ObservableCollection* is bound in a *View* and the collection is modified. The MVVM binding engine knows that it needs to change or update the visual representation.

5.4.3 Historical Data

RtcService also handles fetching historical data, which is primarily used to display in graphs. Historical data is simply a key/value collection of time stamps and noise values. Similarly to the Real-time service historical data is an XML file which is deserialized and merged into the corresponding *LocationViewModel*, when there are changes to the data. A method *ShouldUpdateLastHistoryData* was made to handle when the historical data should be updated. Historical data is divided into hourly, daily, weekly and monthly data. Hourly data is updated every minute and should only be fetched every minute, daily data is updated every 10 minutes, and weekly and monthly data is updated every 12 hours and hence, should only be re-fetched using that interval.

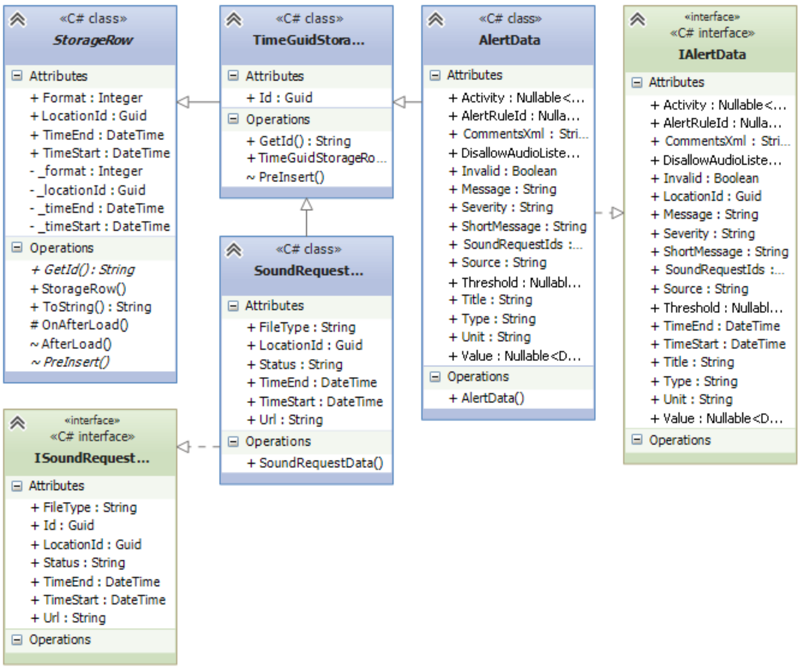
5.4.4 Alerts

Alerts and sound clips are fetched using the two methods *GetAlertDataAsync()* and *GetSoundRequestDataAsync()* both create their own queries to Azure Table storage to get data for a specific location within a specific interval. These queries are executed through the Windows Azure Storage plugin made for this project, see Section 5.5.1 for more information about that.

5.5 Plugins

5.5.1 Windows Azure Storage

Figure 5.5: Class Diagram of the return types from Azure Storage

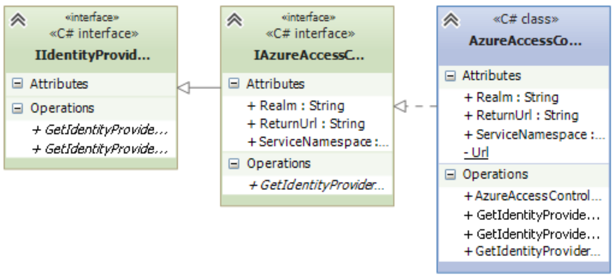


Since the Windows Azure Storage library cannot be used in a Portable Class Library, a plugin had to be made to enable the usage. However, the library is not even made for the Xamarin.Android and Xamarin.iOS platforms to begin with, so this had to be done first. Two new library projects for the two platforms were created and in a similar fashion as in the WindowsPhone project the same files were linked into the project and compiled. This resulted in class libraries for each platform. However, since the type of result you get from Azure Table storage is of type *StorageRow* which has platform specific code inside of the class, it cannot be used inside of a Portable Class Library. Therefore the classes inheriting from the *StorageRow* class were duplicated inside of the PCL, and Interfaces were made for them, which the classes needing to implement *StorageRow* also had to implement, such that translation between the platform specific and the portable types was easy due to the contract they both implement, this can be seen in

Figure 5.5. Methods in all three class libraries for the plugins (refer to Figure 4.7 in Chapter 4 for a refresh of the structure of plugins), were made to translate from one type to the other and for queries to the Azure Storage was made in the plugins. This makes the Azure Storage Plugin very specific to the return data types the plugin supports. In this case for Alerts and Sound Request Data (sound clips). However, all other clients needing this information would be able to reuse the plugin as the task of contacting the Azure Storage is encapsulated.

5.5.2 Authentication

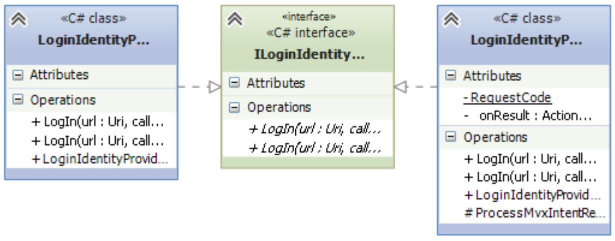
Figure 5.6: Class Diagram of the Identity Provider Client



The authentication plugin consists of two parts. The first is a means of acquiring a collection of Identity Providers. Seen in Figure 5.6 a couple of interfaces and an implementation specifically for Azure Access Control has been made. *IIdentityProviderClient* is an interface, which defines a couple of methods for getting an *IEnumerable<IdentityProviderInformation>*. *IdentityProviderInformation* is simply a class with a name and a couple of URL's on how to log in and log out of an identity provider. The *IAzureAccessControlIdentityProviderClient* is a specialization of the previous interface, which is suited for Azure Access Control, as ACS requires a couple of values in order to identify which account and which identity providers it needs to return to the requesting client. The actual fetching happens using an instance of *HttpClient* which gets a JSON document in the response which in turn is deserialized to the return type of the methods defined in the interfaces. All this happens in the PCL and no platform specific code is needed for this.

As for the login process in it self, showing up the platform specific web browser component, the implementation is in the platform specific library of the plugin. Looking at Figure 5.7, on the left hand side is the Windows 8.1 implementation, which internally uses the *WebAuthenticationBroker* class, which already implements everything needed. This is not the case on Android, as this class is

Figure 5.7: Class Diagram of the Login Tasks



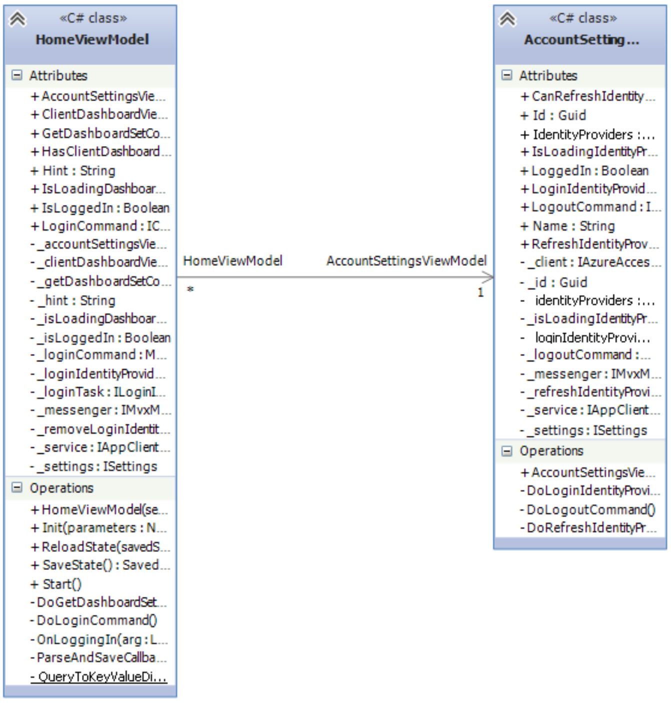
not present on that platform. Instead it implements an Android *Activity*, which displays a *WebView* and implements a custom *WebViewClient* which handles the redirection until the token is received as described in the sequence diagram in Figure 4.8 in Chapter 4.

5.6 ViewModels

Most of the implemented *ViewModels* are very trivial apart from the couple of *ViewModels* handling the authentication flow on Windows 8.1. What is common for all of the *ViewModels* is that every dependency is injected through the constructor. Every value that needs to be bound to a view is a public property implementing the *INotifyPropertyChanged* pattern. All user interaction happens through *ICommand* implementations, which are bound to *Click* events or *Command* properties views.

The relationship between *HomeViewModel* and *AccountSettingsViewModel* can be seen in Figure 5.8. *HomeViewModel* instantiates an instance of *AccountSettingsViewModel* for use when used in an *Flyout* on Windows 8.1. This behavior is previously described in Section 4.1.1.1. The problem in this case is that the *Flyout* is not able to launch the modal *WebAuthenticationBroker* described in Section 5.5.2. This must be done through the *HomeViewModel*. The way this is achieved, is to use the *Subscriber/Publisher* pattern, which is implemented in MvvmCross also called Messaging. This allows any part of the system to subscribe or listen to specific method of a specific type. Hence a class *LoginIdentityProviderMessage*, which simply has a property for the selected Identity Provider, which is chosen in the *Flyout* from the displayed collection of Identity Providers. This message with the Identity Provider is published by the *AccountSettingsViewModel*. In turn *HomeViewModel* subscribes to get these messages, and when such one is received, the *LoginIdentityProviderTask* from the authentication plugin is invoked with the selected *IdentityProvider* and the

Figure 5.8: Class Diagram of HomeViewModel and AccountSettingsView-Model



modal view is displayed correctly.

5.7 Custom Views

A couple of custom view implementations were necessary in this project. The first one is a wrapper of the *MapFragment* at is has issues when contained in a *ScrollView*, the second is a *Graph* view component to display historical data.

5.7.1 Map View

On Android, to mimic the navigation of the Windows 8.1 application, in the *RtcView*, a special implementation had to be made for the *MapView* to allow scrolling the map when it is inside a *ScrollView* or in this case a *Hori-*

zontalScrollView. This is due to the *ScrollView* intercepting all the touches made, and thinking the touches are made on itself, rather than letting them pass to the *MapView*. The solution here is to call *RequestDisallowIntercept-TouchEvent(true)* on the *ScrollView* whenever you want to pan around the *MapView*, hence a wrapper is needed to be made around the *MapView*. The actual implementation of this can be seen in Listing F.1 in Appendix F. The essence of this is to create a *View* around the *MapFragment*, lets call it *Touch-ableWrapper*, which is a *FrameLayout* which in its most basic form listens for touch events. It then exposes two actions, which it is possible to attach a method which needs to be executed when touch down or up is detected. This is used in a class inheriting from *MapFragment*, where in the *OnCreateView* method, the inflated *View* is wrapped in the created *TouchableWrapper*. This in turn also exposes two events, corresponding to the ones of the *TouchableWrapper*. Now when using the *Fragment* the two events that it exposes can be used to call the *RequestDisallowInterceptTouchEvent(true)*, when touches down are detected, and the same call, with a false argument when touch up is detected. An example of the usage of this wrapped *TouchableMapFragment* can be seen in Listing 5.3.

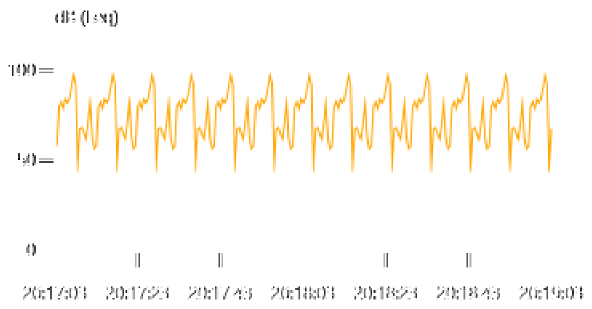
Listing 5.3: Usage of TouchableMapFragment

```
var frag = FragmentManager
            .FindFragmentById<TouchableMapFragment>(Resource.Id.map);
frag.TouchUp += (sender, args) =>
    _hsv.RequestDisallowInterceptTouchEvent(false);
frag.TouchDown += (sender, args) =>
    _hsv.RequestDisallowInterceptTouchEvent(true);
```

5.7.2 Graphs

The Open Source library OxyPlot was investigated to begin with to help displaying graphs in the applications. The great thing about OxyPlot is that it already has implementations for all the platforms covered by this project. For some reason, the performance of the OxyPlot View was very bad on all the platforms, which in many cases made the application freeze for 2 or more seconds. Hence, it was decided to fall back on an implementation made during the Internship Project described in in Section 1.2.2 in Chapter 1. It is a simple implementation where line and bar graphs were implemented for Windows Phone 7, Android and iOS. This implementation was made with file linking in mind and a lot of compiler directives were used to swap out platform specific methods. This project was cleaned up and made into a set of libraries, not plugins as there is no need for Dependency Injection with the views in this case. The libraries consists of a core PCL with all the classes that can be shared, and one for each platform, very similar to how a plugin is structured. Referencing these libraries from each project, it is not possible to use these graphs. The views are called *MiniChart*. A screenshot of a *MiniChart* can be seen in Figure 5.9.

Figure 5.9: Screenshot of a MiniChart



5.8 Custom Bindings

There are some views that do not support creating bindings from it out of the box on Android. Hence, there is a need to create some custom implementations to make that work. The entire code for this can be seen in Listing G.1 in Appendix G.

The idea for this custom binding is to be able to give it an *ObservableCollection* of *LocationViewModels*, which are then bound to the *MapView* appearing as circles with a value inside indicating the current noise level.

The steps for creating bindings to a collection are as follows.

1. Assign the *ObservableCollection* to the *ItemsSource* property of the binding. In this case it is called *LocationMarkerSet*, which provides the binding capabilities.

2. When the *ObservableCollection* is assigned, remove the previous *WeakSubscription* to handle changes in the collection, if present. Otherwise continue with subscribing to the *INotifyCollectionChanged* implementation in *ObservableCollection*.

3. When *ItemsSource* is assigned remove all items on the map, then re-add the new items from *ItemsSource* to the map, and then center the map around the items.

The process of adding a new item from the *ItemsSource* to the *MapView* are as follows.

1. Create a new *Marker* from the coordinates in the *LocationViewModel*.
2. Create a new instance of *LocationViewModelWrapper*, which is a helper class to handle changes in the *LocationViewModel* and reflect them on the *Marker*.
3. The *LocationViewModelWrapper* creates a initial *Bitmap* for the circle and assigns it to the *Marker*.
4. Then the *LocationViewModelWrapper* creates a weak subscription to the *INotifyPropertyChanged* implementation of the *LocationViewMode* to listen for changes in the properties for the *Coordinate* and the *CurrentNoiseLevel*. When the *Coordinate* property is updated, the handler for the weak subscription reflects the changes by setting a new *Position* for the *Marker*. If the *CurrentNoiseLevel* changes, a new *Bitmap* is created reflecting those changes and assigned to the *Marker*.

5.9 Chapter Summary

This chapter covered a *BaseView* implementation. How URLs are stored in a *Constants* class. How the two services *AppClientService* and *RtcService* are implemented. It covers the implemented plugins for this project for using Azure Storage and Authentication. It covers the general implementation of *ViewModels* and a special case for the *AccountSettingsViewModel* on Windows 8.1. It covers a custom wrapper around the *MapFragment* on Android and what was used for displaying graphs in the applications. Lastly it covers how custom bindings were used to bind items on the Android *MapView* to a collection of *LocationViewModels*

Chapter 6

Testing

This chapter concerns itself on how the Apps were tested using a Unit Test framework. The code in this project is designed with design patterns, which decouple and encapsulate code, into smaller pieces orienting a certain task or functionality and additionally split application logic away from UI code. It makes the code more testable. This will be demonstrated in this section.

To carry out these tests, a Unit Test framework called NUnit [NUn14] is used. This runs as a .NET 4.5 library project, where any Unit Test runner that supports NUnit can run the tests. Even though the framework is a Unit Testing framework, it is chosen not to be used to carry out unit tests in their intended fashion, where normally code is divided into small units of code, to be able to test them in isolation. Data and services are also usually mocked up to gain greater control over the flow and the actual testing cases. This way you can for instance force errors occurring in the code and test error cases. This is obviously more difficult with a live system. However, mocking up all aspects of the system also takes a lot of time as there are several API's this Application connects to, with several methods each to get data. Hence, in this the tests carried out, would be better described as integration tests. Using a Unit Test framework is still very useful for this and it also shows that the code can run on an entirely other platform than the mobile devices, which strengthens the portability aspect of it.

6.1 Mocking

Several aspects of the system were mocked up, as it is difficult to automate log-in flows in a web browser etc. Mocking also allows for greater control in the code that was mocked up.

In this case the following things were mocked up:

- **Application Identification** - The Application Client requires the Application to identify itself with a unique identifier. This identification comes from a plugin. However, this plugin was not made for unit tests, so a very simple mock was made identifying the test as *UnitTest* and a unique *Guid*.
- **Application Settings** - Another plugin provides persistent storage for the application. Since this is also not available for Unit Tests either, this was also mocked up, using a *Dictionary<string, object>* as the backing storage.
- **Log in** - Opening up a browser and automating a log in procedure is not a part of the *NUnit* testing framework, although there might be some implementations doing so. This was also mocked up hard coding two tokens, that would normally have been returned in the callback URL when logging in. These were issued for two dedicated Unit Test users in EMS's development environment. These have access to data recorded for a specific time span, which can be validated against.
- **View Dispatcher** - *MvvmCross* provides a View Dispatcher for each platform, which uses the platform specific ways of loading visual elements on the screen. These are obviously not implemented for Unit Tests, and these are mocked up. The mock View Dispatcher simply stores all the requests, such that navigation can be tested against those collections.

6.2 Service Tests

Using the mocked classes, it is possible to test the Services made for the application. A class *BaseServiceTests* has been created to handle the set up process of instantiating all the mocked up classes and creating instances of the services with all the necessary parameters. *BaseServiceTests* also implements methods generally used throughout the tests, to provide instances of the Application ID generator and a authenticated instance of the *AppClientService*. All the test classes related to the Service Tests inherit from *BaseServiceTests*. These tests are split into two classes. One which tests the *AppClientSerivice* and another testing the *RtcService*.

Test Case	Test Data	Result
#1 Receive InformationSet First Call	tenantKey: tomasz00 userGuid: b9fec904(..) siteGuid: 752c1ebc(..)	OK
#2 Receive InformationSet First Call	tenantKey: tomasz01 userGuid: c9fec904(..) siteGuid: 1a3429fd(..)	OK
#3 Receive InformationSet Two Calls	tenantKey: tomasz00 userGuid: b9fec904(..) siteGuid: 752c1ebc(..)	OK
#4 Receive InformationSet Two Calls	tenantKey: tomasz01 userGuid: c9fec904(..) siteGuid: 1a3429fd(..)	OK

#5 Receive DashboardSet First Call	tenantKey: tomasz00
	userGuid: b9fec904(..)
	siteNames: {
	"DTU Kemitorvet",
	"DTU Kollegiebakken"
	}
		OK

#6 Receive DashboardSet First Call	tenantKey: tomasz01
	userGuid: c9fec904(..)
	siteNames: {
	"Microsoft byggeri"
	}
		OK

Table 6.1: Test cases, data and results for AppClientService

Tests, test cases and results can be seen in Table 6.1, for tests of the *AppClientService*. Looking at the tests, they each require certain information to carry out their task. In most cases, they require a *tenantKey*, which identifies the owner of the site. A *userGuid*, which identifies the actual user logging in to the system and invoking the API. Lastly for the *InformationSet* tests, a *siteGuid*, which is used to check if that particular *siteGuid* is contained in the *InformationSet* returned from the API. All tests are only testing for positive scenarios, which is OK in this case, as these are not Unit Tests, but rather integration tests with a live API. Similarly tests for fetching the *DashboardSet*, which contains information about which sites are accessible to the APP are made, which are validated against the *siteNames*, to see if the result returns the expected values.

For the *RtcService* tests, cases and results can be seen in Table 6.2. Similar test values for the test cases are used as for fetching the *InformationSet* in the *AppClientService* tests. However instead of validating against the *siteGuid*, the returning result it checked for whether it contains locations inside of the inventory. If it does it passes. The Real Time Data test which fails is due to issues when running asynchronous code in the testing environment. When stepping into the code using the debugger the correct values are received. However, for some reason the event, which normally fires when the collection of *ViewModel*s is updated, does not fire on the thread the tests run. So the subscription to that event never gets notified about the change and eventually the test times out and fails.

Test Case	**Test Data**	**Result**
#1 Load Inventory	tenantKey: tomasz00 userGuid: b9fec904(..) siteGuid: 752c1ebc(..)	OK
#2 Load Inventory	tenantKey: tomasz00 userGuid: b9fec904(..) siteGuid: c2ee5b91(..)	OK
#3 Load Inventory	tenantKey: tomasz01 userGuid: c9fec904(..) siteGuid: 1a3429fd(..)	OK
#4 Real Time Data	tenantKey: tomasz00 userGuid: b9fec904(..) siteGuid: 752c1ebc(..)	FAIL
#5 Alert Data	tenantKey: tomasz00 userGuid: b9fec904(..) siteGuid: 752c1ebc(..)	OK

Table 6.2: Test cases, data and results for RtcService

6.3 ViewModel Tests

The testing done for the *ViewModels* is mainly testing navigation between the *ViewModels*. The only navigation in done in this application is between the *HomeViewModel* and *RtcViewModel*. Both of them use *AppClientService*, which has already been tested, while the latter *RtcViewModel* also uses *RtcService*, which also has been tested.

In a similar fashion a base class *ViewModelTestBase*, has been created for these tests, setting up all necessary classes and IoC to allow the *ViewModels* to function properly.

6.4 Chapter summary

This shows that the *ViewModels* and *Services* used in the Application are truly platform independent. It also shows that they are testable using a Unit Test framework. It is obvious that several more tests could be made, and the code would most likely increase in quality with additional testing. It also shows that development of Applications can be done using Test Driven Development method, as most of the Application logic is contained and separated from UI and platform specific code. Even with platform specific code, Unit Test frameworks such as XUnit could also be deployed to test platform specific code contained in the main Applications and in plugins to strengthen the quality of the code.

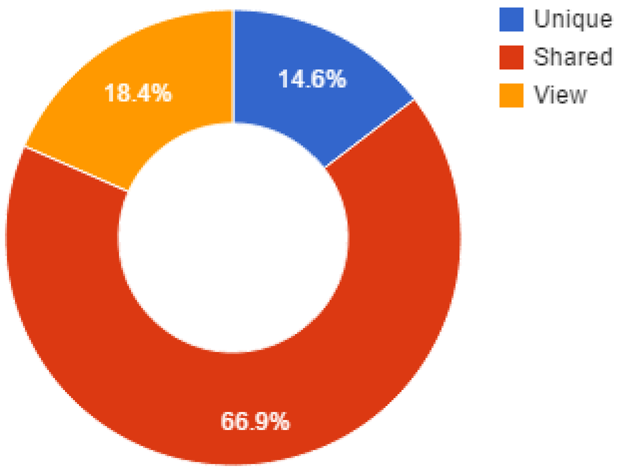
Chapter 7

Metrics

This chapter concerns itself with metrics about how much code was reused and shared between the Apps. How much code was platform specific, but also estimations about how much time was spent developing the Apps.

To measure how much code was shared between the two Applications, some code metrics are deployed. Visual Studio, which is the Integrated Development Environment (IDE) used to write the Apps, has measures built in to create code metrics. However, these are based on Intermediate Language (IL), which is compiled and optimized code. What is interesting here is how much code the developer ends up writing, hence IL metrics are not a good measure. Another problem is that the IL the .NET compiler and Mono compiler is potentially different and not comparable. Hence, a script was made to create some metrics about the code written in the project, which is based on Lines of Code (LOC) instead.

Figure 7.1: Code metrics for Android



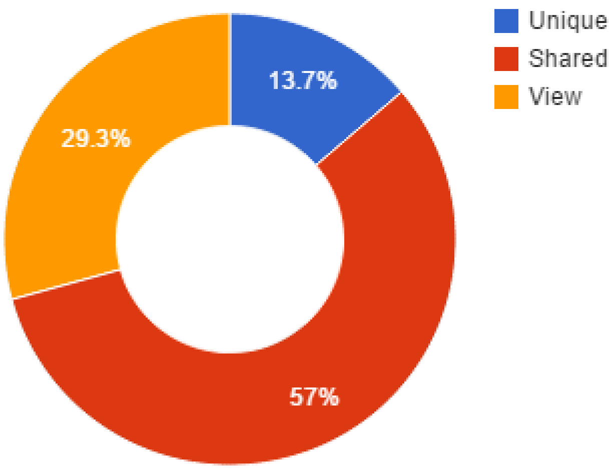
Using LOC instead of IL is simple, although a bit problematic. This is due to the contents of the files can potentially differ in style. This is especially true if multiple developers work on the project. In this case there has only been one developer, but to be able to compare the files, it is good practice to normalize them - rather format them according to some rules. In order to do that a tool called ReSharper [Jet14], which is a plug-in for Visual Studio has been used to clean code according to the rules found in Appendix H and the default rule set that comes with it. That makes sure that compiled C# code gets formatted correctly. Windows 8 and Android both makes use of XML type of files, with different schema to describe the UI. These files have been formatted according to the following rules and an example can be see in Listing 7.1.

- Each element starts on its own line
- Each element ends on
    1. Its own line if it contains nested children
    2. In line with itself or an attribute if it has no children
- Each attribute and value starts and ends on its own line

Listing 7.1: Example of XML code formatting

```
<LinearLayout
  android:orientation="vertical"
  android:id="@+id/locationcontainer"
  android:layout_width="300dp"
  android:layout_height="match_parent">
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Location Details"
    android:layout_margin="10dp"/>
</LinearLayout>
```

Figure 7.2: Code metrics for Windows 8



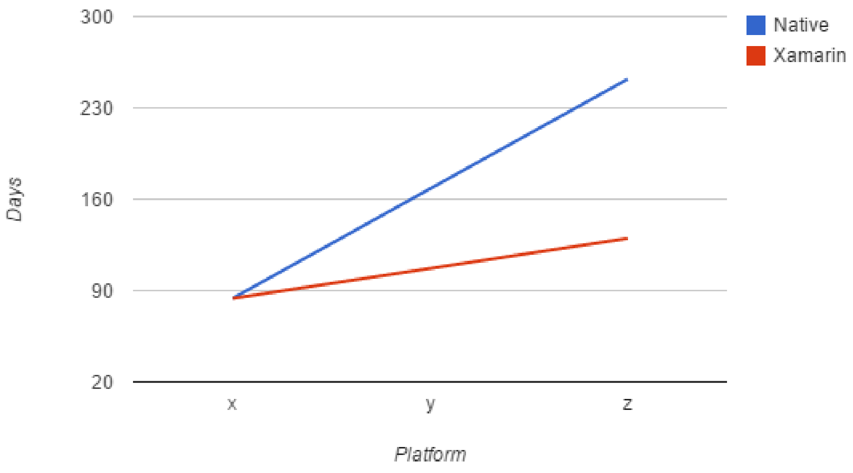
The script takes advantage of the .csproj files, Visual Studio creates, when creating a new project. These files are MSBuild [Mic13] project files, containing information about targets, tasks, but more important properties and items. The latter provides information about, which items are compiled with the project, which files are in the project. The files themselves are XML files, which means they can easily be parsed.

Files such as the *Resource.Designer.cs*, *App.xml* and *App.xaml.cs* are filtered out before lines are counted, as they are not representative of code written by the developer. The former file is auto generated, and can contain thousands of lines of code, which are auto generated ID's for Views in Android. *App.xml* and *App.xml.cs*. Images or any other non-code files are not line counted either.

Looking at Figure 7.1, showing the results from gathered from the script for the Android project. It can be seen that 66.9% of the code written comes from shared code. That shared code is what is contained in the core PCL of the project along with common model code in the Model PCL and Custom View code for the *MiniChart* View. The rest of the code is made out of 18.4% view code and 14.6% unique, which amounts to approximate third of the code which is specific to that platform.

Looking at Figure 7.2, which similarly shows results gathered for Windows 8, it has slightly more platform specific code, which amounts to 43% of the code. The reason for this is most likely the sharing functionality, which is not implemented on Android, which takes up some of those lines of code.

Figure 7.3: Estimated development days spent on Native vs. Xamarin



Taking a look at commits in the Git repository used throughout the project, it can be gathered that amount of days used developing the Windows 8 App was 84 days while the Android took 23 days. The Windows 8 App was developed first, and it was also when all the *ViewModels*, Services and all the other application logic was defined. If assumed that the development of the platform specific part of the Windows 8 App took approximately as long as the Android App, it can be approximated that it took 60 days to write the Application logic. Taking it further and assuming a third implementation for iOS, would take as long as it took for Android. Also assuming that writing the applications using the native SDK would then take 84 days for each platform, as none of the code can be shared. Then a conclusion can be drawn that as soon as you reach the third platform, you have spent 50% less time on those 3 Apps writing them with Xamarin, than you would have writing them natively. This can be seen in Figure 7.3.

7.1 Chapter summary

The metrics show that applications made with the traditional native approach described in Chapter 3, take longer time to create as you have to create a significant bigger amount of code to achieve the same as when creating native applications using a single language and sharing a lot of code. The metrics also show that depending on the platform more approximately 60% of the code can be shared between all platforms.

Chapter 8

Conclusion

This chapter concerns itself with the discoveries made during the project, an overall conclusion and lastly the project is put into perspective by discussing the future work needed in order to make a release of it on the different app markets.

8.1 Findings

In Chapter 1 several problems were discovered with how developers see developing for several platforms. These problems manifest themselves as follows.

- Differences in SDK's between platforms.
- Differences in languages between platforms.
- Increase in difficulty and time required to cover several platforms

The problems from Brüel & Kjær's perspective is to allow their customers to access their products from any device from anywhere. They also want their developers to be effective in creating applications targeting multiple platforms.

Chapter 2 went into further detail on which existing solutions were already made. It detailed the domain of the project and found the functional and non-functional requirements. Lastly it described several use cases to cover the requirements.

Chapter 3 went into detail with various design patterns and frameworks along with different ways to develop applications and described the pros and cons of each of them. Based on the analysis MvvmCross was chosen along with heavy use of Dependency Injection to help encapsulate code into smaller task and due to the DI promote the ease of testing the encapsulated code in isolation.

Chapter 4 described the architecture and structure of the applications, along with descriptions on behavior of the important components in the system.

Chapter 5 went into details, describing how the most important parts of the applications were implemented, it discussed obstacles encountered and how they were solved with specific solutions and examples.

Chapter 6 described how the NUnit unit testing framework was used to create integration tests of the project, showing the ease of testing parts of code in isolation and describing how parts of the application could be mocked up to stimulate certain values and states.

Chapter 7 discussed how code was measured to find out how much code was shared across platforms, how much of it was UI code and how much of it was unique platform specific code.

8.2 Conclusion

The overall conclusion based on the findings is that it is possible to create cross-platform applications with great amount of shared code. It was discovered that for the applications developed in this project the average shared code was 61.9%. When approximated in time used to develop, using the cross-platform tools and design patterns, compared to traditional native development, the amount of time saved in comparison was approximately 50%, and linearly increasing for each additional platform added to the equation. The resulting applications can be seen in Appendix C and Appendix D. It can also be concluded that the two applications made both cover all of the functional and non-functional requirements and the tests prove that the design patterns used, provide a greater amount of testablity of the code produced.

8.3 Future Work

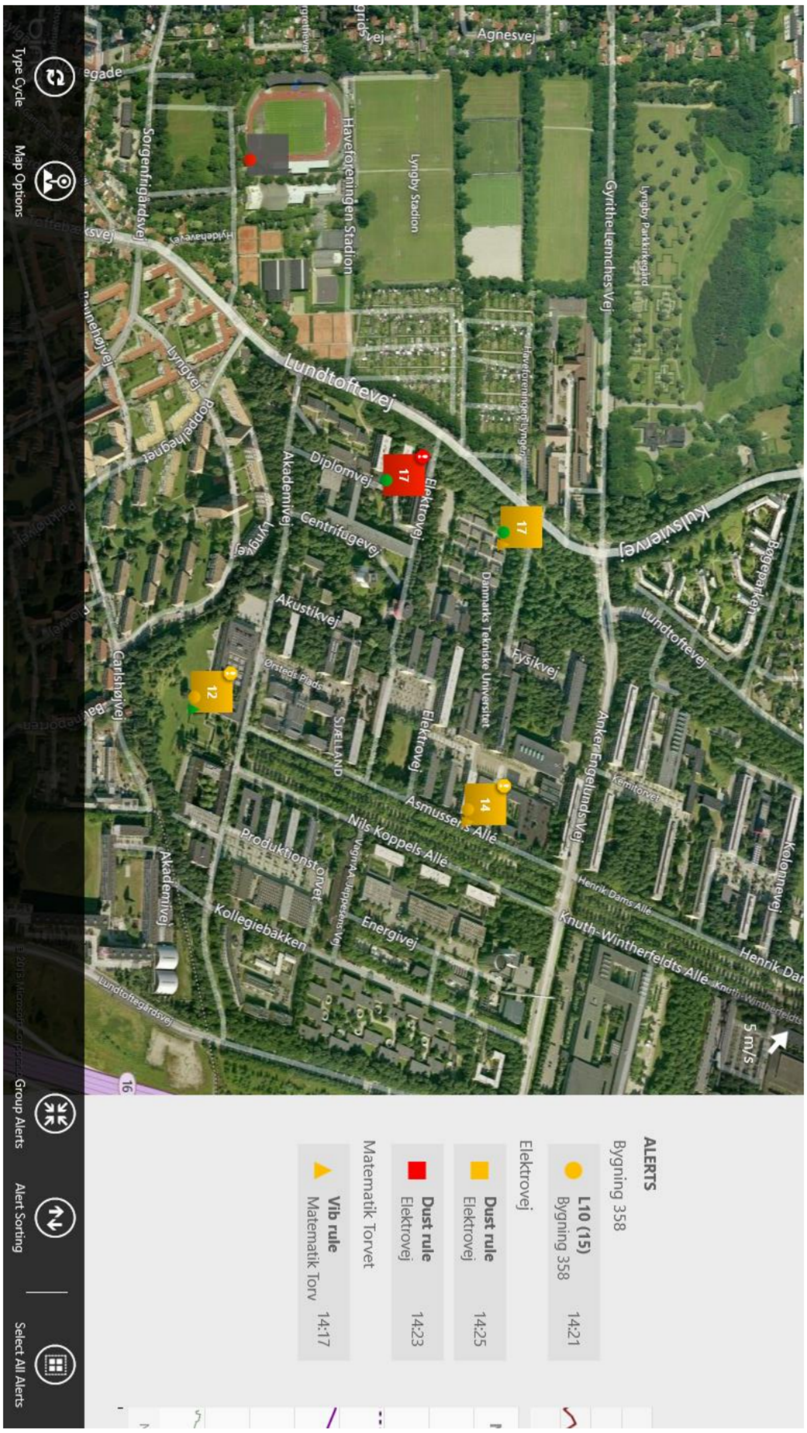
While the Android application served to prove that the code created during the project indeed was cross-platform compatible. Only minor things would be needed to be done in order to bring the UI and functionality up on the same level as the Windows 8.1 application. Additional work on the navigation flow also needs to be made as a bug was discovered when logging in the Android application, sometimes it does not recognize the log-in before the application has been closed and opened again. Back navigation away from the *RtcView* is also broken.

If a future release of Xamarin.Forms will support Windows 8.1 and tablet idioms, it will could prove to be a very good tool to increase the amount of shared code, as it then could be abstracted away into the portable class library.

The custom bindings made for the Android *MapView*, should be investigated more to see if it would be possible to generalize it and adapt it into being able to be used with arbitrary *ViewModels* and push it upstream to MvvmCross. This would greatly decrease the platform specific code on the Android platform and the need to create a new implementation every time you need to show something different on a map.

Appendix A

Screenshots from Tablet Interface For Environment Monitoring



ALERTS

Bygning 358

●	**L10 (15)** Bygning 358	14:21

Elektrovej

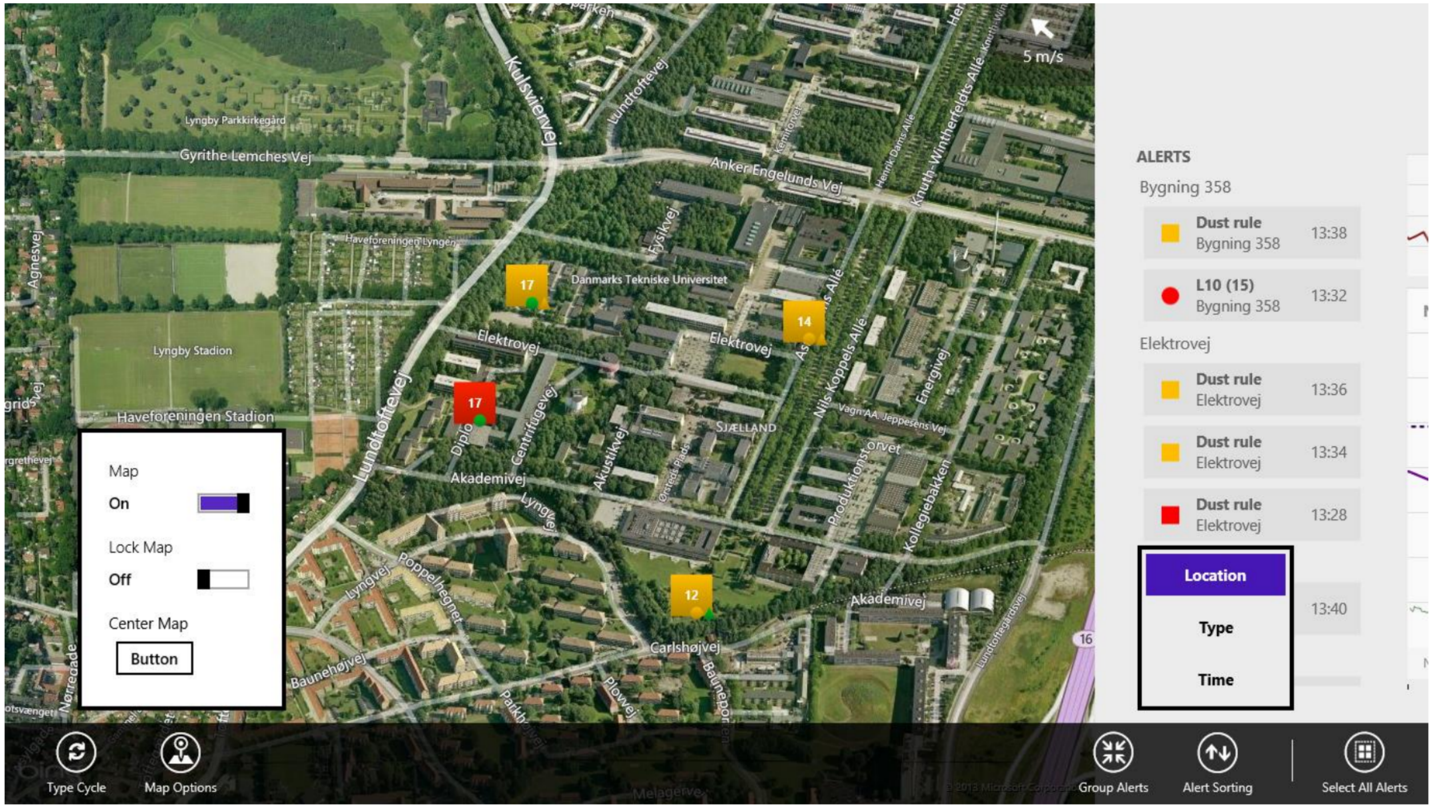
🟨	**Dust rule** Elektrovej	14:25
🟥	**Dust rule** Elektrovej	14:23

Matematik Torvet

▲	**Vib rule** Matematik Torv	14:17

Figure A.1: Screenshot of the Map Interface

Figure A.2: Screenshot of the Investigation Interface



ALERTS

Bygning 358

🟨	Dust rule Bygning 358	13:38
●	L10 (15) Bygning 358	13:32

Elektrovej

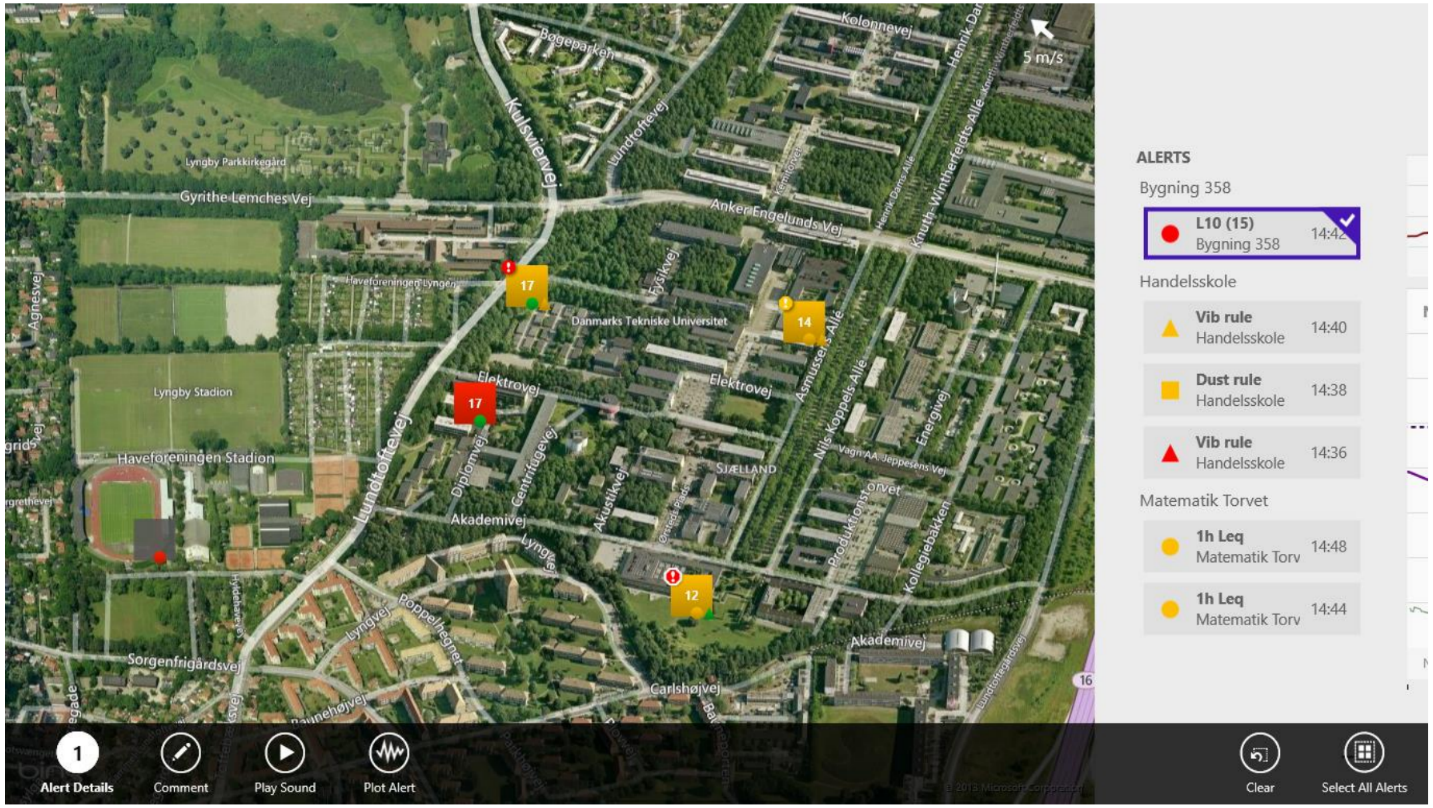
🟨	Dust rule Elektrovej	13:36
🟨	Dust rule Elektrovej	13:34
🟥	Dust rule Elektrovej	13:28
		13:40

Map
On

Lock Map
Off

Center Map
Button

Figure A.3: Screenshot of the Contextual App Bar



ALERTS

Bygning 358

●	L10 (15)	Bygning 358	14:42

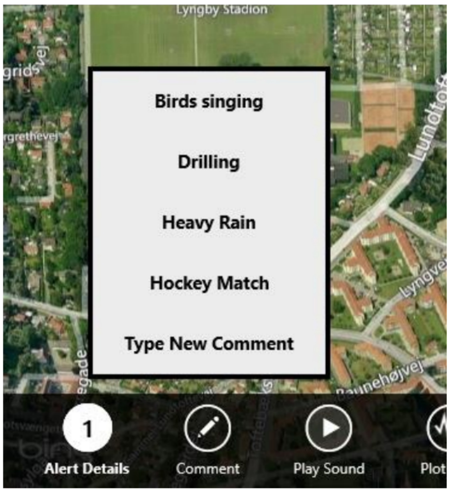
Handelsskole

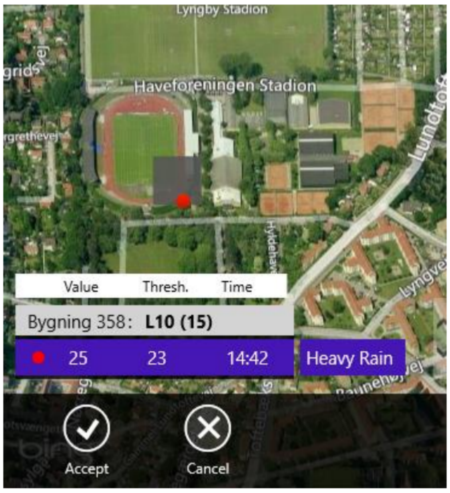
▲	Vib rule	Handelsskole	14:40
■	Dust rule	Handelsskole	14:38
▲	Vib rule	Handelsskole	14:36

Matematik Torvet

●	1h Leq	Matematik Torv	14:48
●	1h Leq	Matematik Torv	14:44

Figure A.4: Screenshot of commenting flow (upon Alert selection)





	Value	Thresh.	Time	
Bygning 358: **L10 (15)**				
●	25	23	14:42	Heavy Rain

Appendix B

Noise Sentinel screenshots

Figure B.1: Screenshot of the Noise Sentinel client

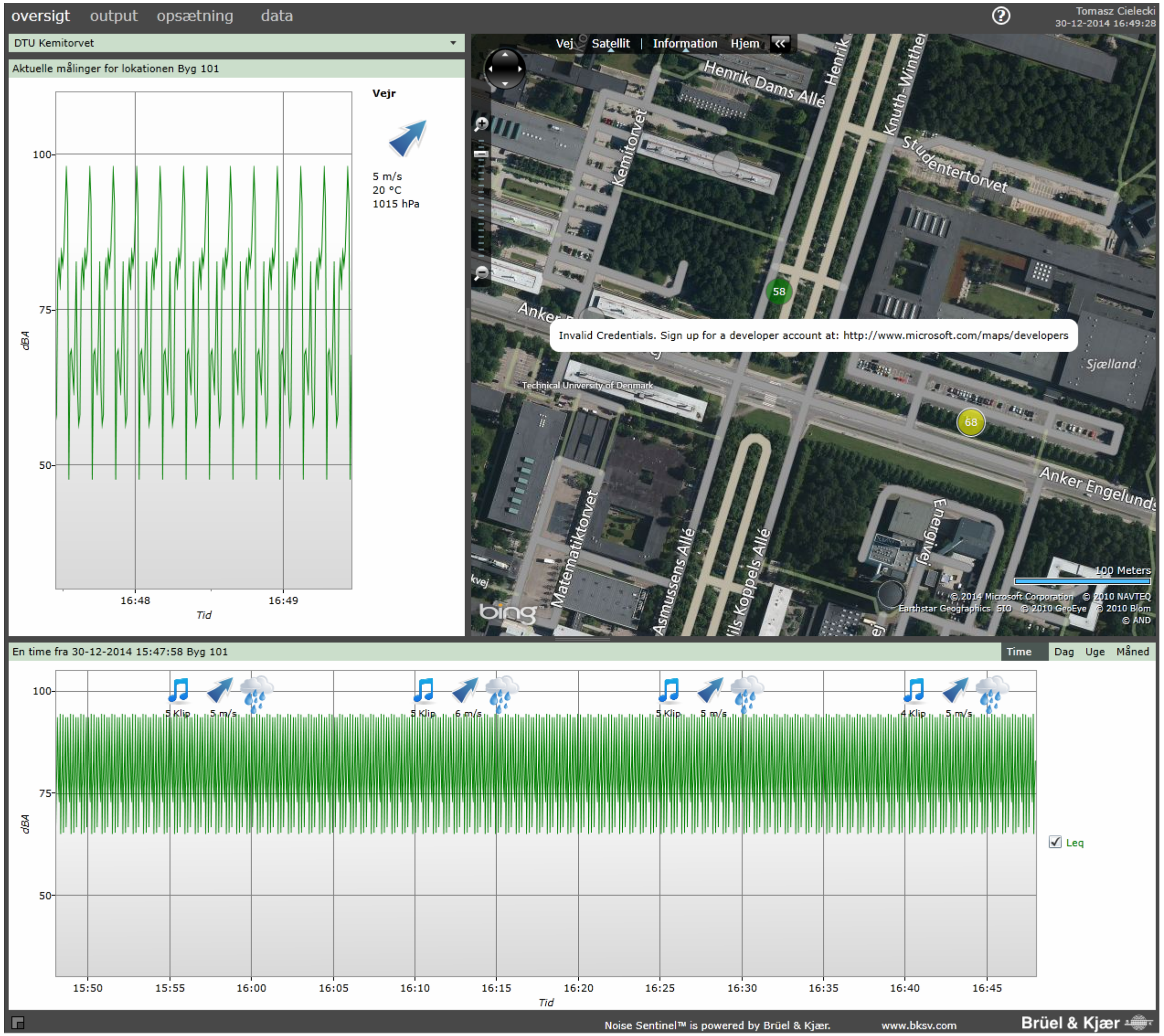
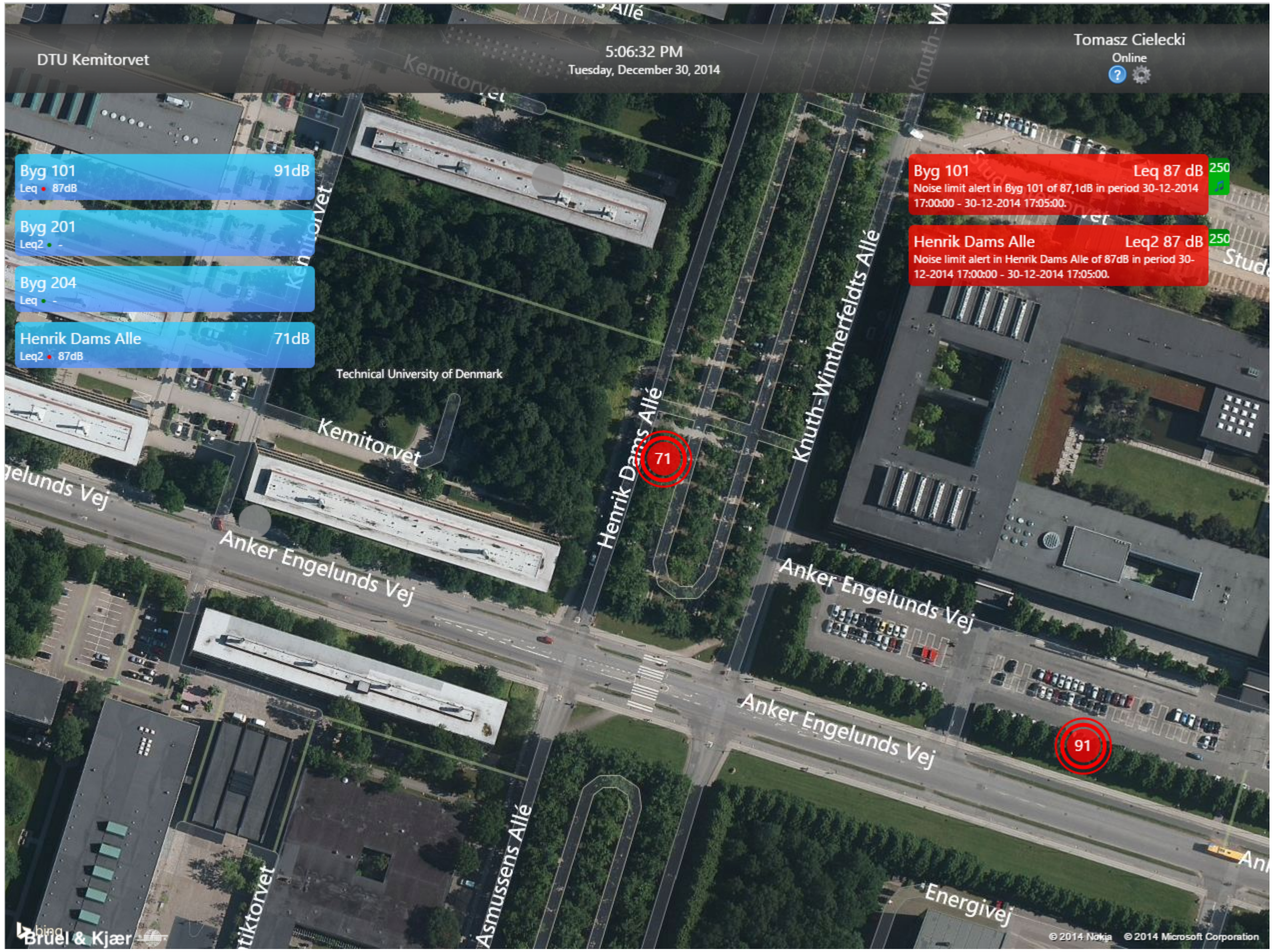


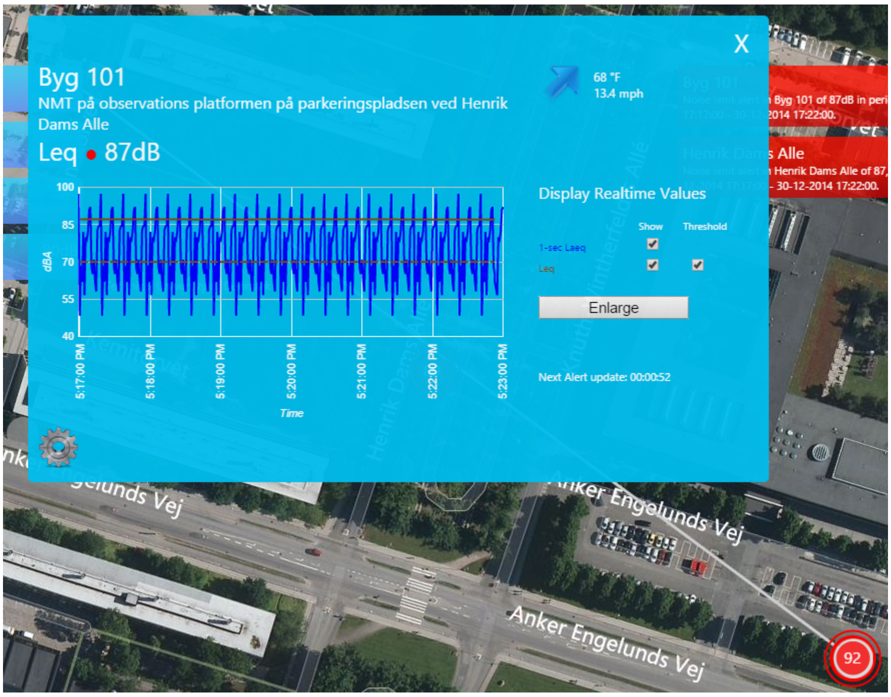
Figure B.2: Screenshot of the RTNC client



Byg 101 Leq 87 dB
Noise limit alert in Byg 101 of 87,1dB in period 30-12-2014 17:00:00 - 30-12-2014 17:05:00.

Henrik Dams Alle Leq2 87 dB
Noise limit alert in Henrik Dams Alle of 87dB in period 30-12-2014 17:00:00 - 30-12-2014 17:05:00.

Figure B.3: Screenshot of NMT details



Byg 101

NMT på observations platformen på parkeringspladsen ved Henrik Dams Alle

Leq ● 87dB

68 °F
13.4 mph

Display Realtime Values

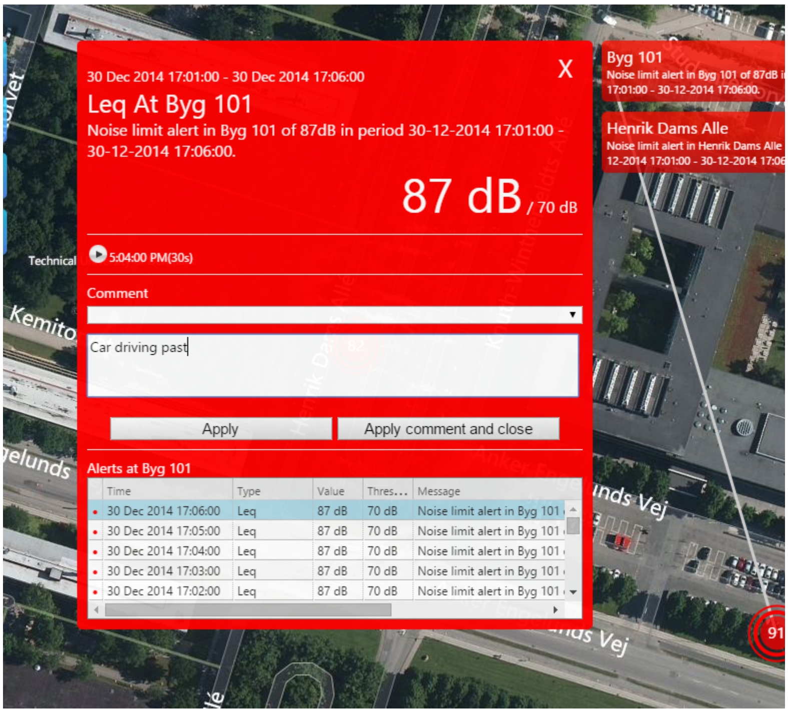
	Show	Threshold
1-sec Laeq	✓	
Leq	✓	✓

Next Alert update: 00:00:52

Byg 101 of 87dB in peri
2014 17:22:00.

s Alle
Henrik Dams Alle of 87,
- 30-12-2014 17:22:00.

Figure B.4: Screenshot of Alert details



30 Dec 2014 17:01:00 - 30 Dec 2014 17:06:00

Leq At Byg 101

Noise limit alert in Byg 101 of 87dB in period 30-12-2014 17:01:00 - 30-12-2014 17:06:00.

87 dB / 70 dB

5:04:00 PM(30s)

Comment

Car driving past

Time	Type	Value	Thres…	Message
30 Dec 2014 17:06:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:05:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:04:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:03:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101
30 Dec 2014 17:02:00	Leq	87 dB	70 dB	Noise limit alert in Byg 101

Byg 101

Noise limit alert in Byg 101 of 87dB i
17:01:00 - 30-12-2014 17:06:00.

Henrik Dams Alle

Noise limit alert in Henrik Dams Alle
12-2014 17:01:00 - 30-12-2014 17:06

Appendix C

Windows 8.1 Client Screenshots

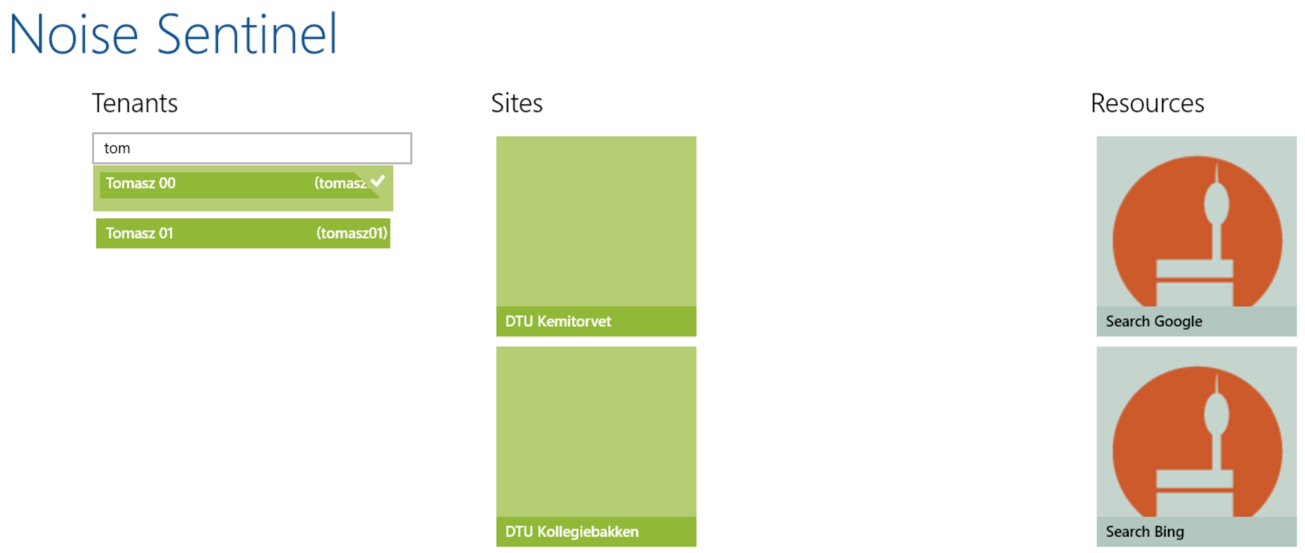
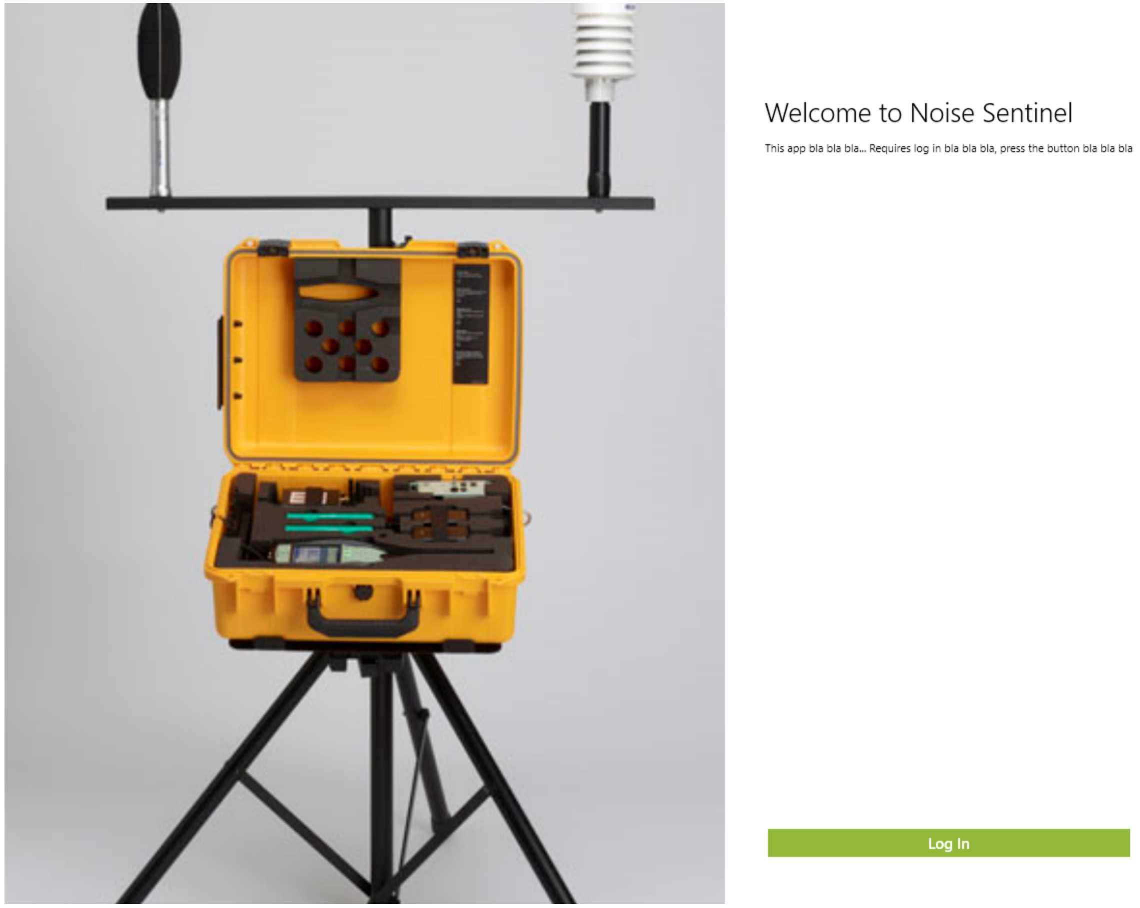


Figure C.1: Welcome screen when opening the application (HomeView + HomeViewModel)

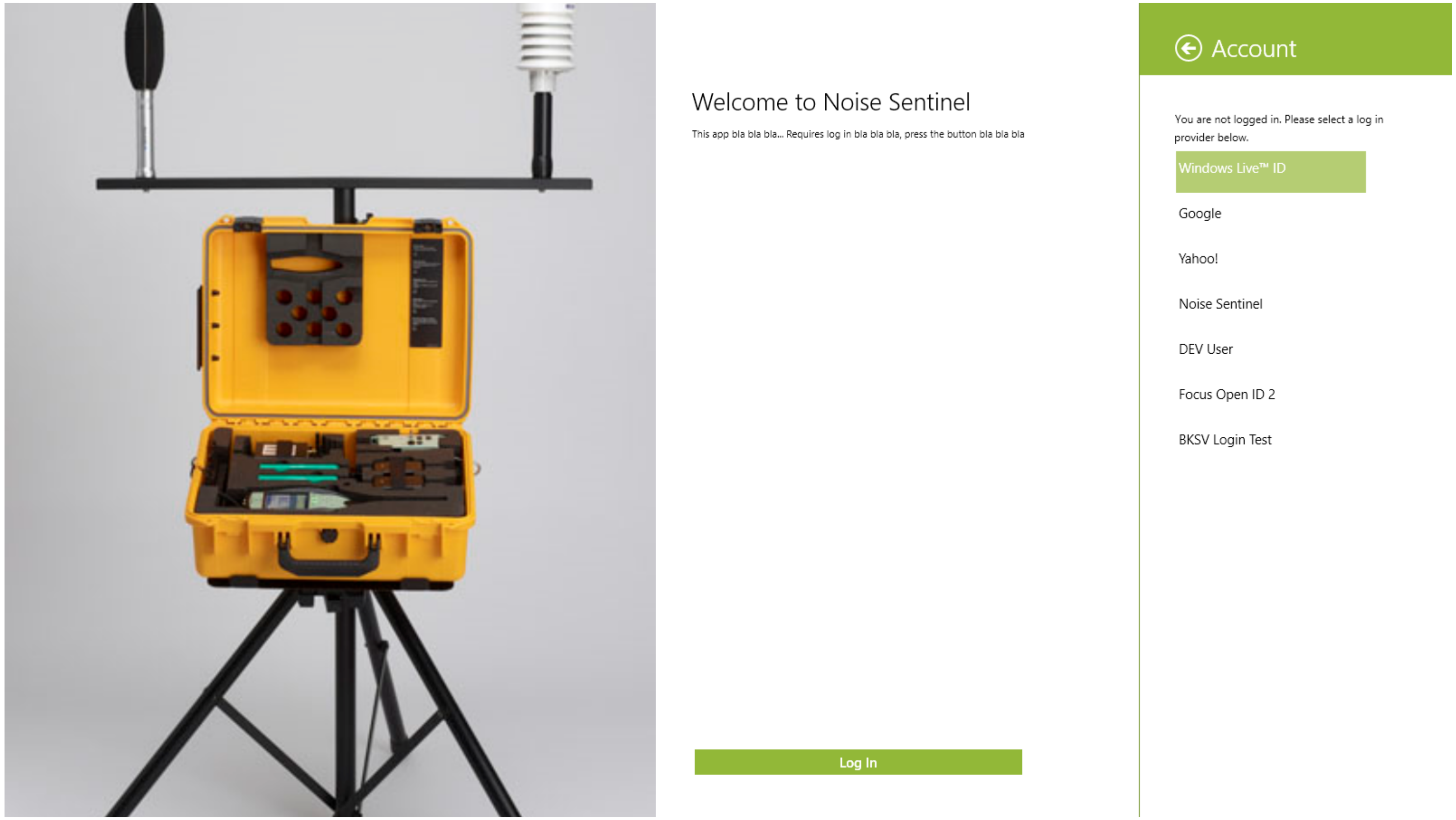
Figure C.2: AccountSettingsFlyout on the right side when pressing the login button



Welcome to Noise Sentinel

This app bla bla bla... Requires log in bla bla bla, press the button bla bla bla

Figure C.3: WebAuthenticationBroker showing up modally to allow logging in user

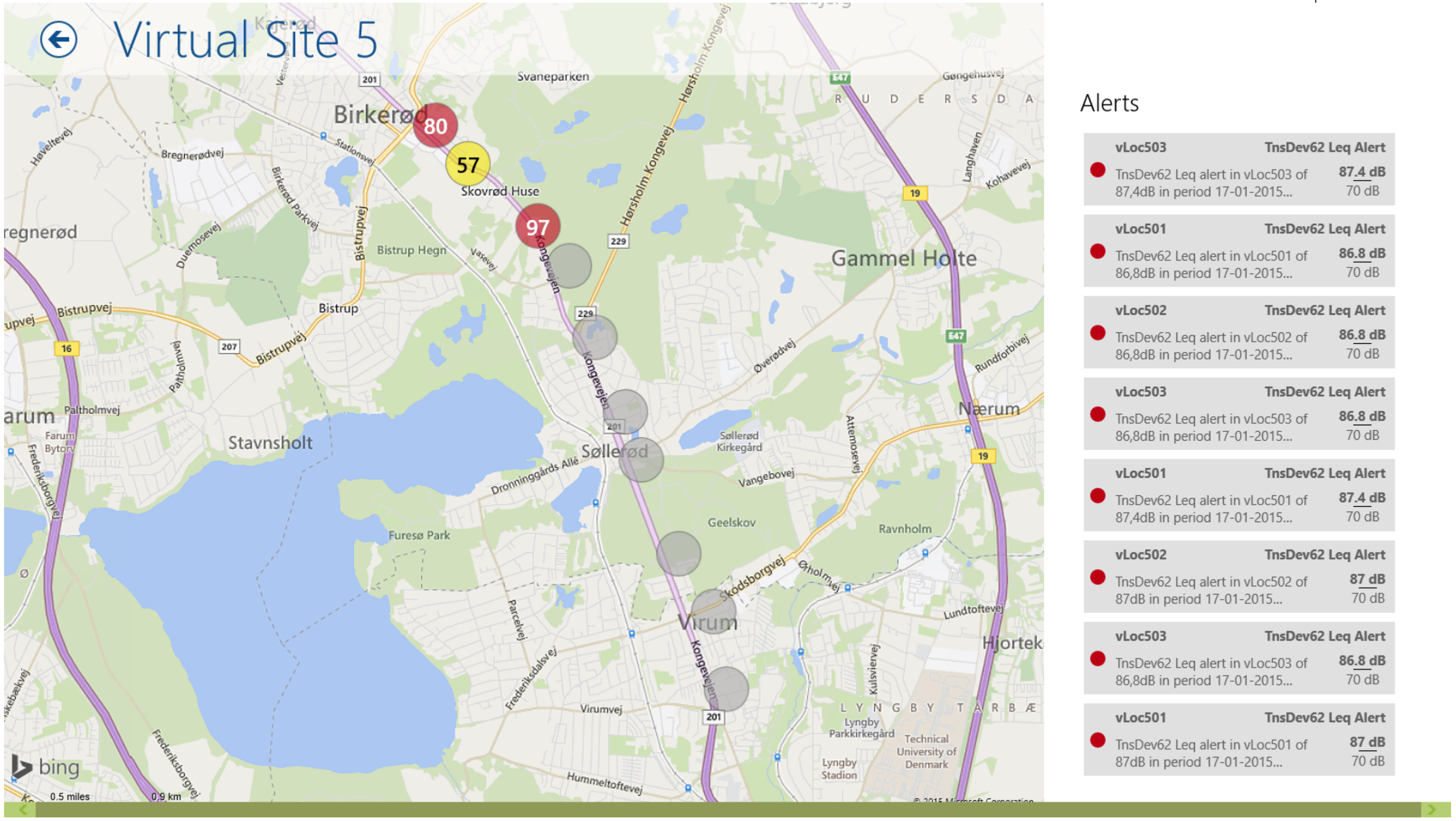


Welcome to Noise Sentinel

This app bla bla bla... Requires log in bla bla bla, press the button bla bla bla

You are not logged in. Please select a log in provider below.

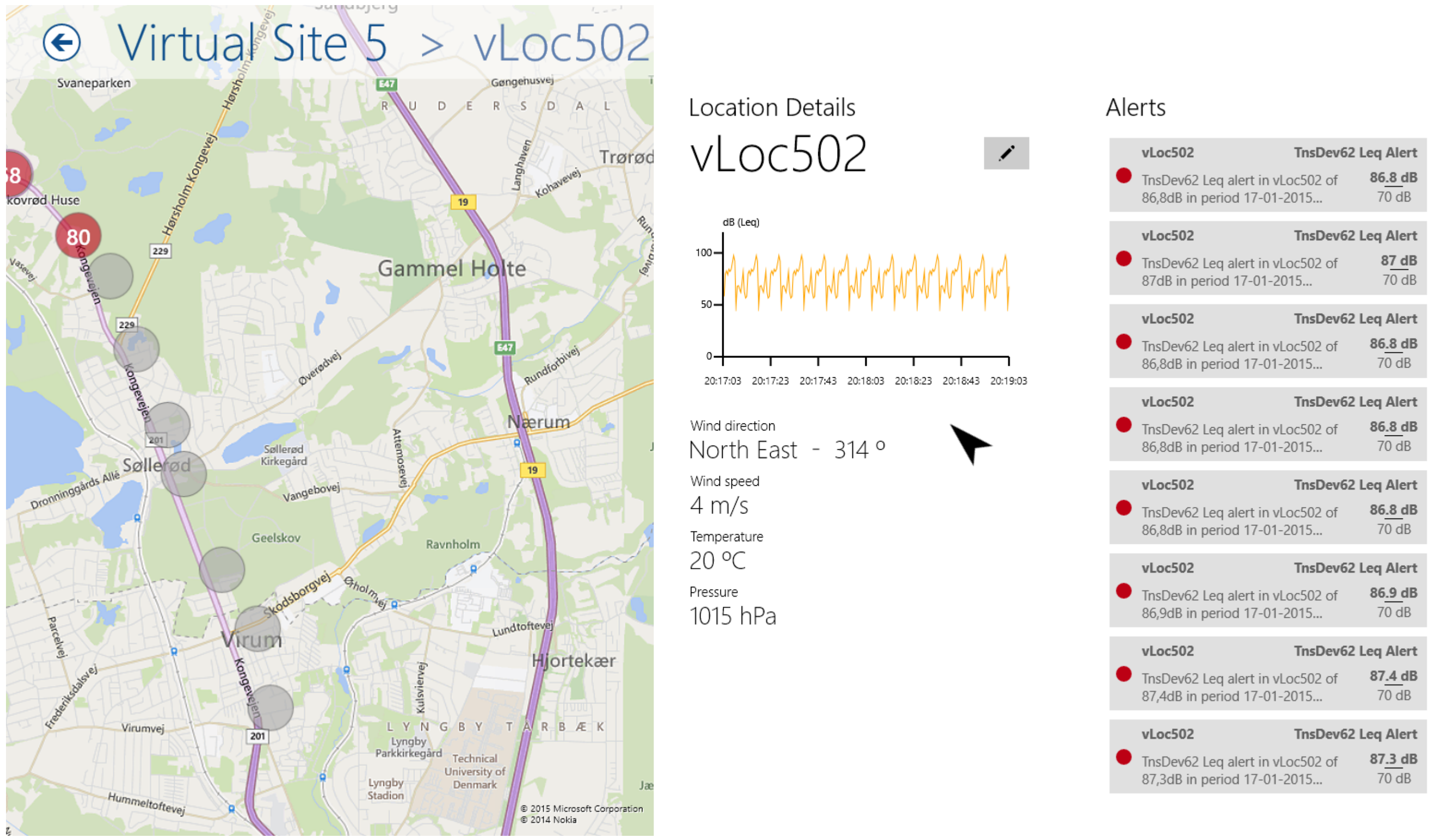
Figure C.4: View of map with locations and Alerts on rightmost side



Alerts

vLoc503		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc503 of 87,4dB in period 17-01-2015...	87.4 dB / 70 dB
vLoc501		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc501 of 86,8dB in period 17-01-2015...	86.8 dB / 70 dB
vLoc502		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015...	86.8 dB / 70 dB
vLoc503		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc503 of 86,8dB in period 17-01-2015...	86.8 dB / 70 dB
vLoc501		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc501 of 87,4dB in period 17-01-2015...	87.4 dB / 70 dB
vLoc502		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc502 of 87dB in period 17-01-2015...	87 dB / 70 dB
vLoc503		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc503 of 86,8dB in period 17-01-2015...	86.8 dB / 70 dB
vLoc501		TnsDev62 Leq Alert
●	TnsDev62 Leq alert in vLoc501 of 87dB in period 17-01-2015...	87 dB / 70 dB

Figure C.5: Location selected, showing details about it



Location Details

vLoc502

Wind direction
North East - 314 º

Wind speed
4 m/s

Temperature
20 ºC

Pressure
1015 hPa

Alerts

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015... 86.8 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 87dB in period 17-01-2015... 87 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015... 86.8 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015... 86.8 dB / 70 dB

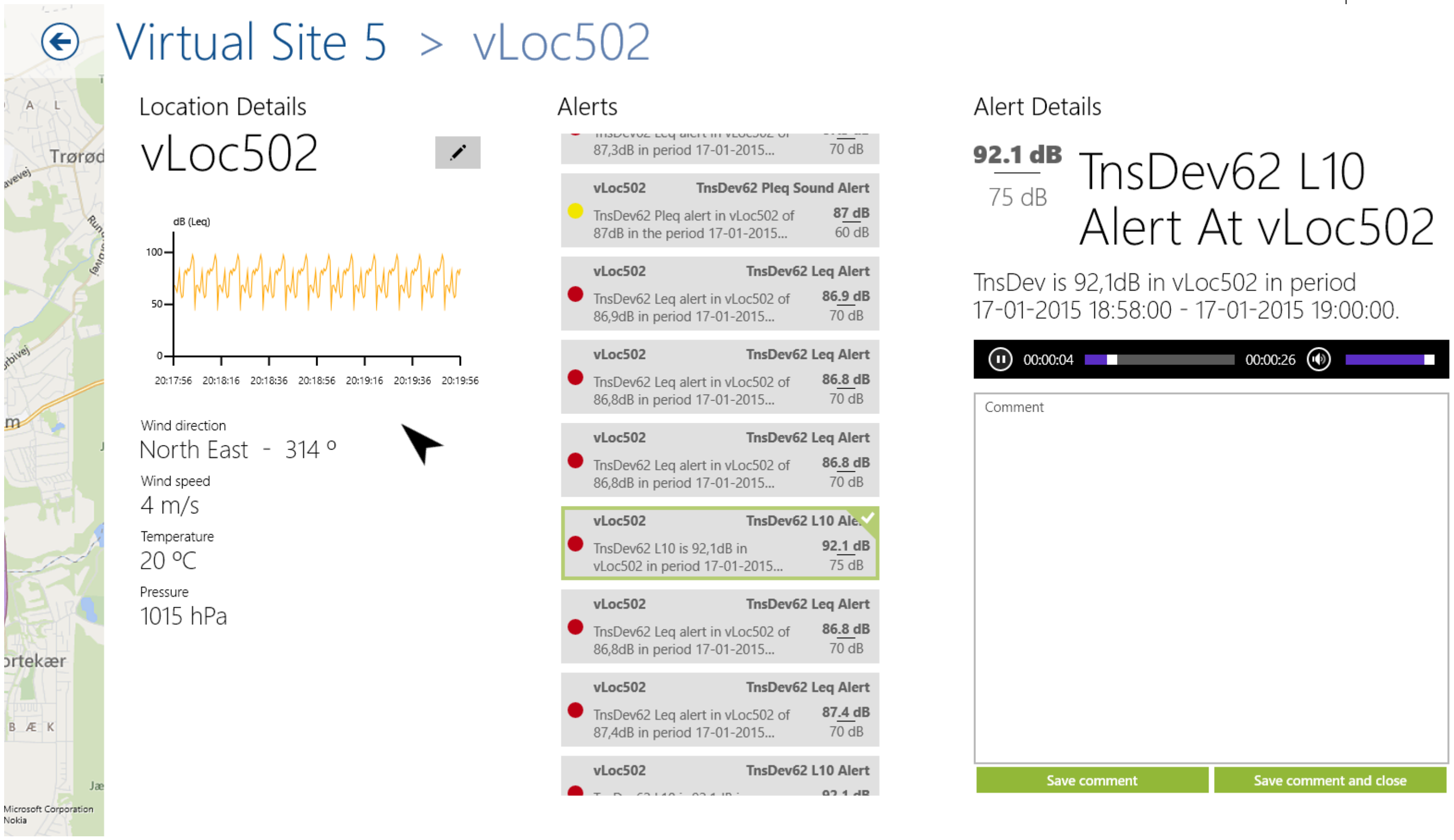
vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015... 86.8 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,9dB in period 17-01-2015... 86.9 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 87,4dB in period 17-01-2015... 87.4 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 87,3dB in period 17-01-2015... 87.3 dB / 70 dB

Figure C.6: Alert selected, showing details about Alert and sound clip playing



Virtual Site 5 > vLoc502

Location Details

vLoc502

Wind direction
North East - 314 º

Wind speed
4 m/s

Temperature
20 ºC

Pressure
1015 hPa

Alerts

87,3dB in period 17-01-2015... 70 dB

vLoc502 TnsDev62 Pleq Sound Alert
TnsDev62 Pleq alert in vLoc502 of 87dB in the period 17-01-2015... 87 dB / 60 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,9dB in period 17-01-2015... 86.9 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015... 86.8 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015... 86.8 dB / 70 dB

vLoc502 TnsDev62 L10 Ale
TnsDev62 L10 is 92,1dB in vLoc502 in period 17-01-2015... 92.1 dB / 75 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 86,8dB in period 17-01-2015... 86.8 dB / 70 dB

vLoc502 TnsDev62 Leq Alert
TnsDev62 Leq alert in vLoc502 of 87,4dB in period 17-01-2015... 87.4 dB / 70 dB

vLoc502 TnsDev62 L10 Alert

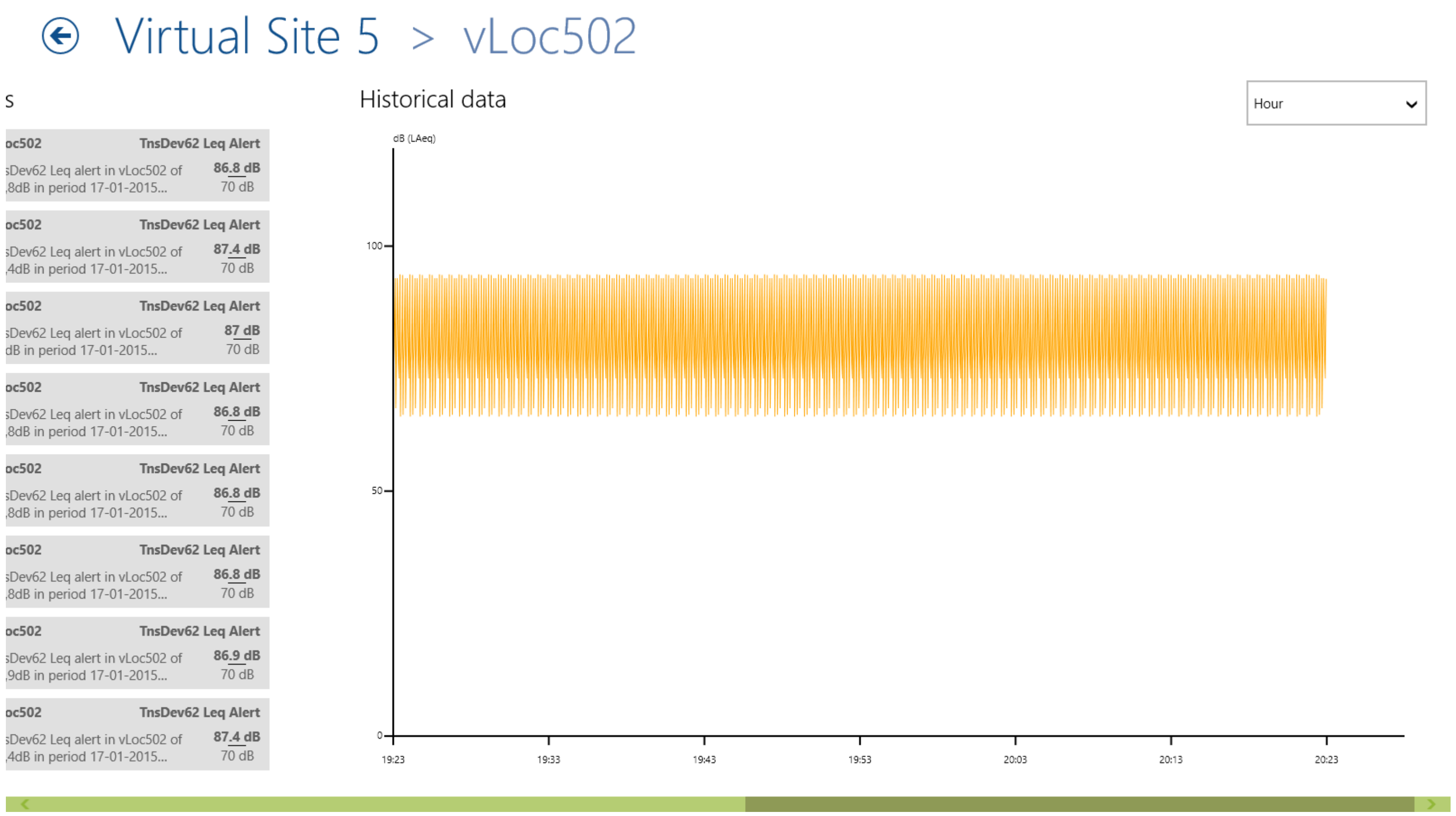
Alert Details

92.1 dB / 75 dB

TnsDev62 L10
Alert At vLoc502

TnsDev is 92,1dB in vLoc502 in period 17-01-2015 18:58:00 - 17-01-2015 19:00:00.

Figure C.7: Historical data about selected location, drop down lets the user select different views



Virtual Site 5 > vLoc502

s

oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of ,8dB in period 17-01-2015... 86.8 dB 70 dB

oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of ,4dB in period 17-01-2015... 87.4 dB 70 dB

oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of dB in period 17-01-2015... 87 dB 70 dB

oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of ,8dB in period 17-01-2015... 86.8 dB 70 dB

oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of ,8dB in period 17-01-2015... 86.8 dB 70 dB

oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of ,8dB in period 17-01-2015... 86.8 dB 70 dB

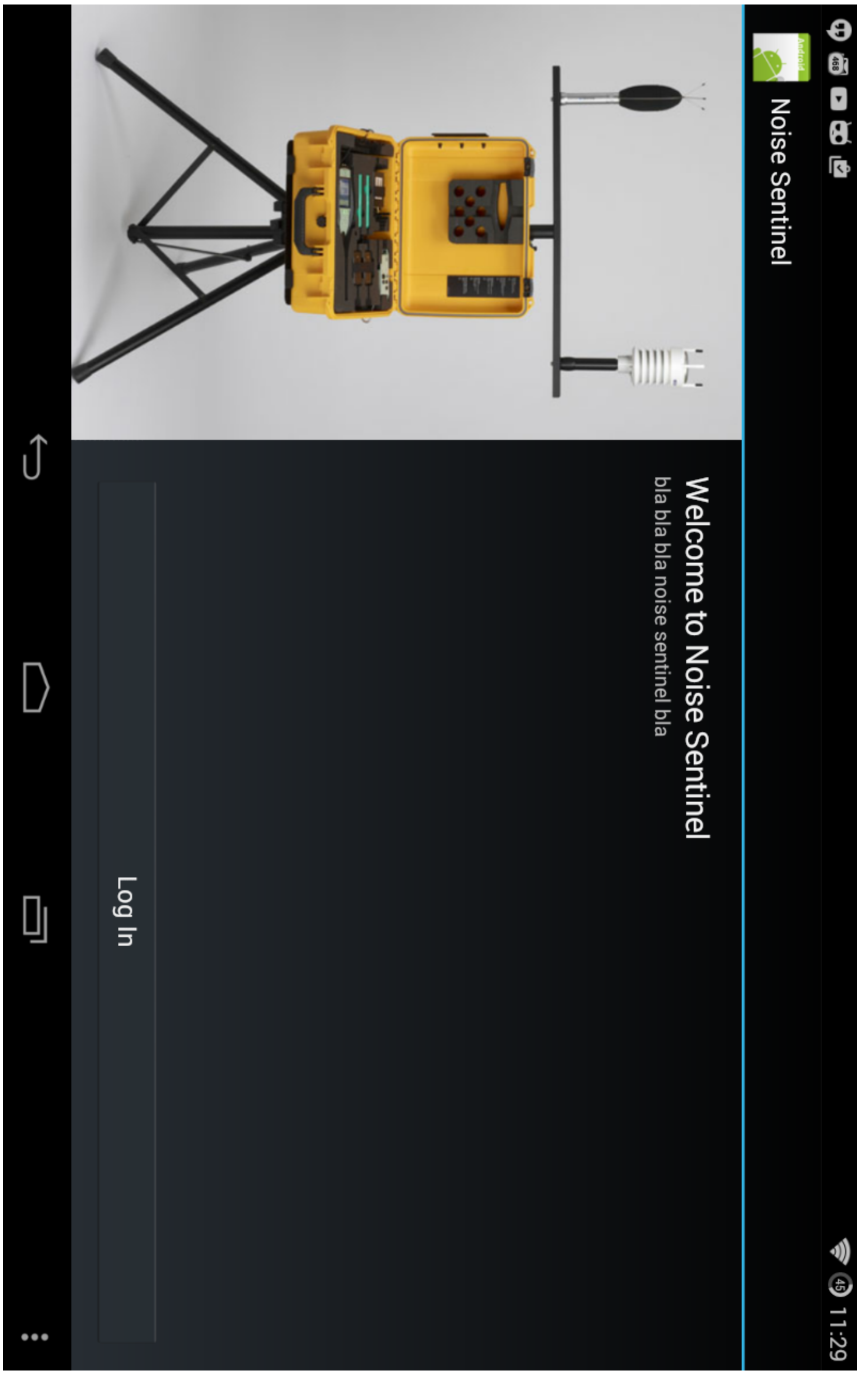
oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of ,9dB in period 17-01-2015... 86.9 dB 70 dB

oc502 TnsDev62 Leq Alert
sDev62 Leq alert in vLoc502 of ,4dB in period 17-01-2015... 87.4 dB 70 dB

Historical data

Appendix D

Android Client Screenshots

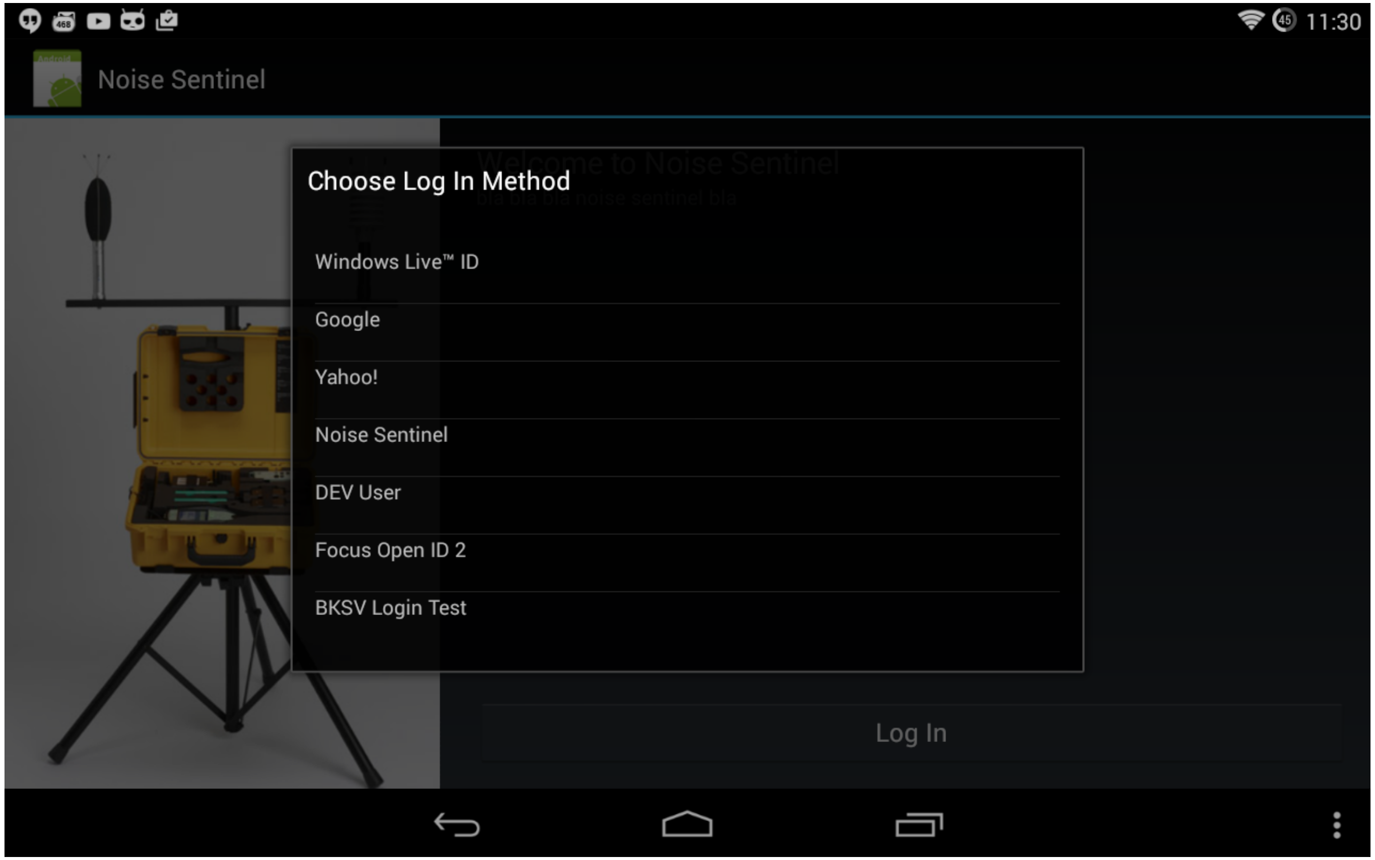


Welcome to Noise Sentinel

bla bla bla noise sentinel bla

Figure D.1: Welcome screen when opening the application

Figure D.2: Dialog appearing with the list of Identity Providers



Choose Log In Method

Windows Live™ ID

Google

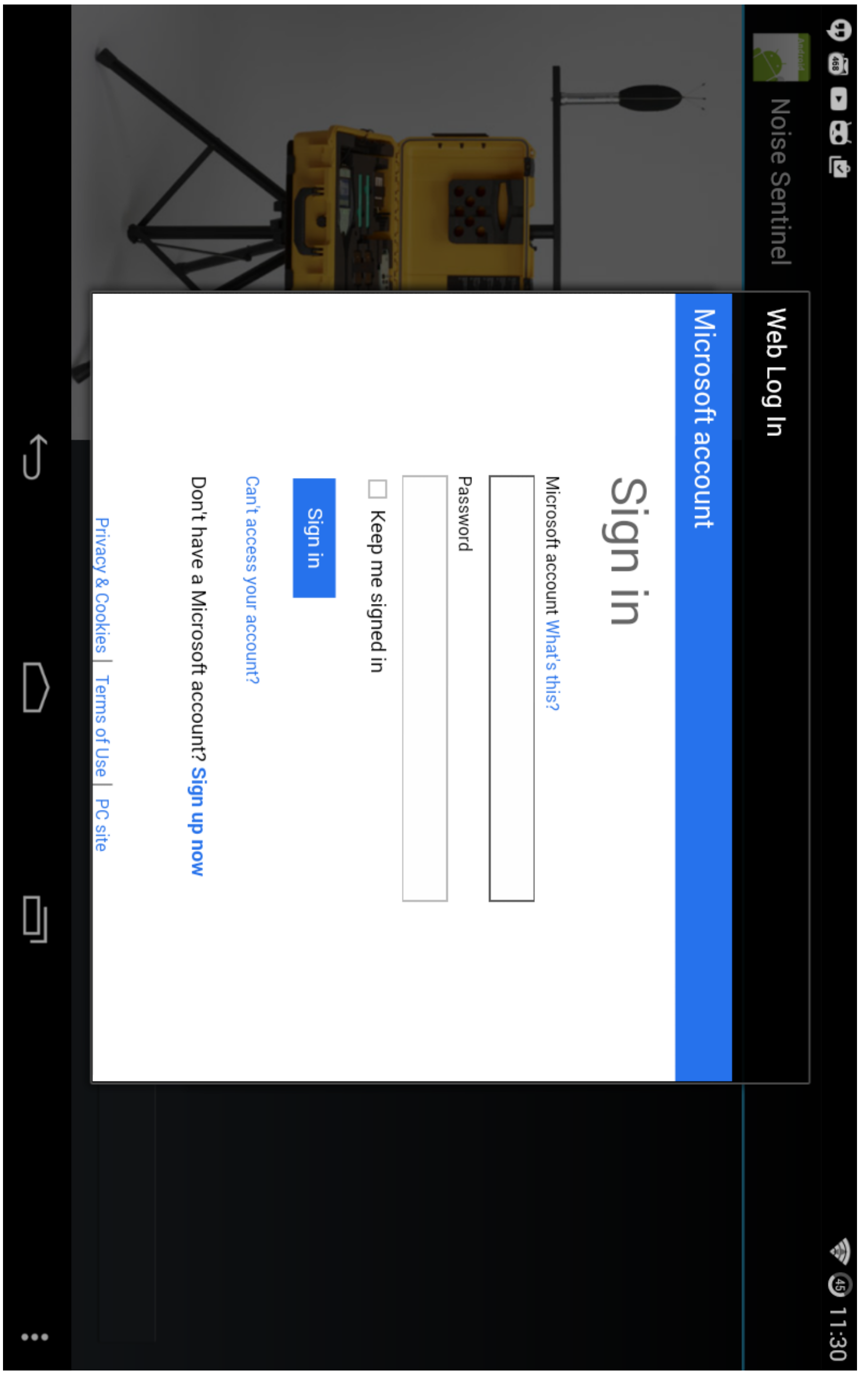
Yahoo!

Noise Sentinel

DEV User

Focus Open ID 2

BKSV Login Test



Noise Sentinel

Web Log In

Microsoft account

Sign in

Microsoft account What's this?

Password

Keep me signed in

Sign in

Can't access your account?

Don't have a Microsoft account? **Sign up now**

Privacy & Cookies | Terms of Use | PC site

11:30

Figure D.3: WebView showing up in a dialog to allow logging in user

Figure D.4: View of map with locations and Alerts on rightmost side

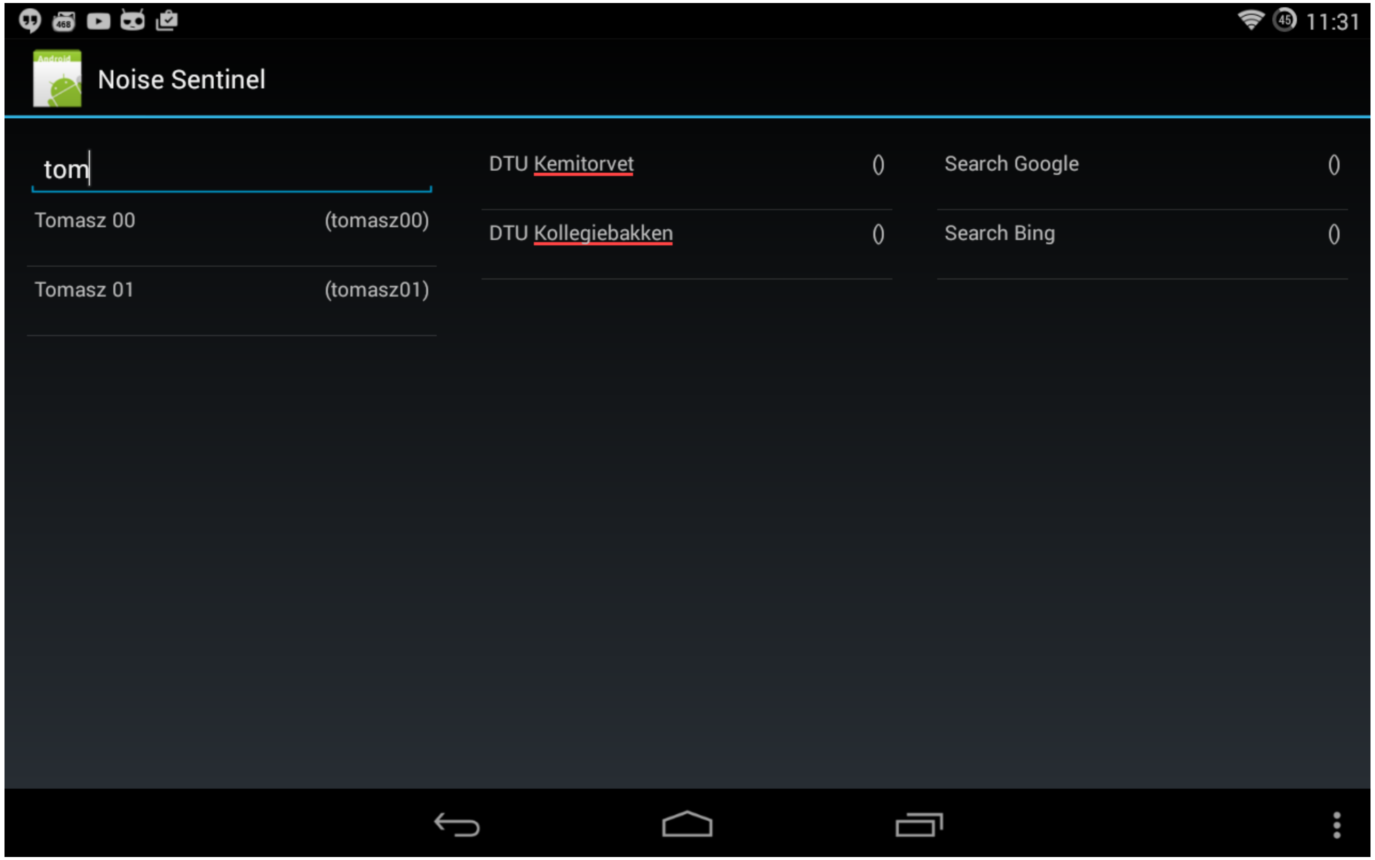
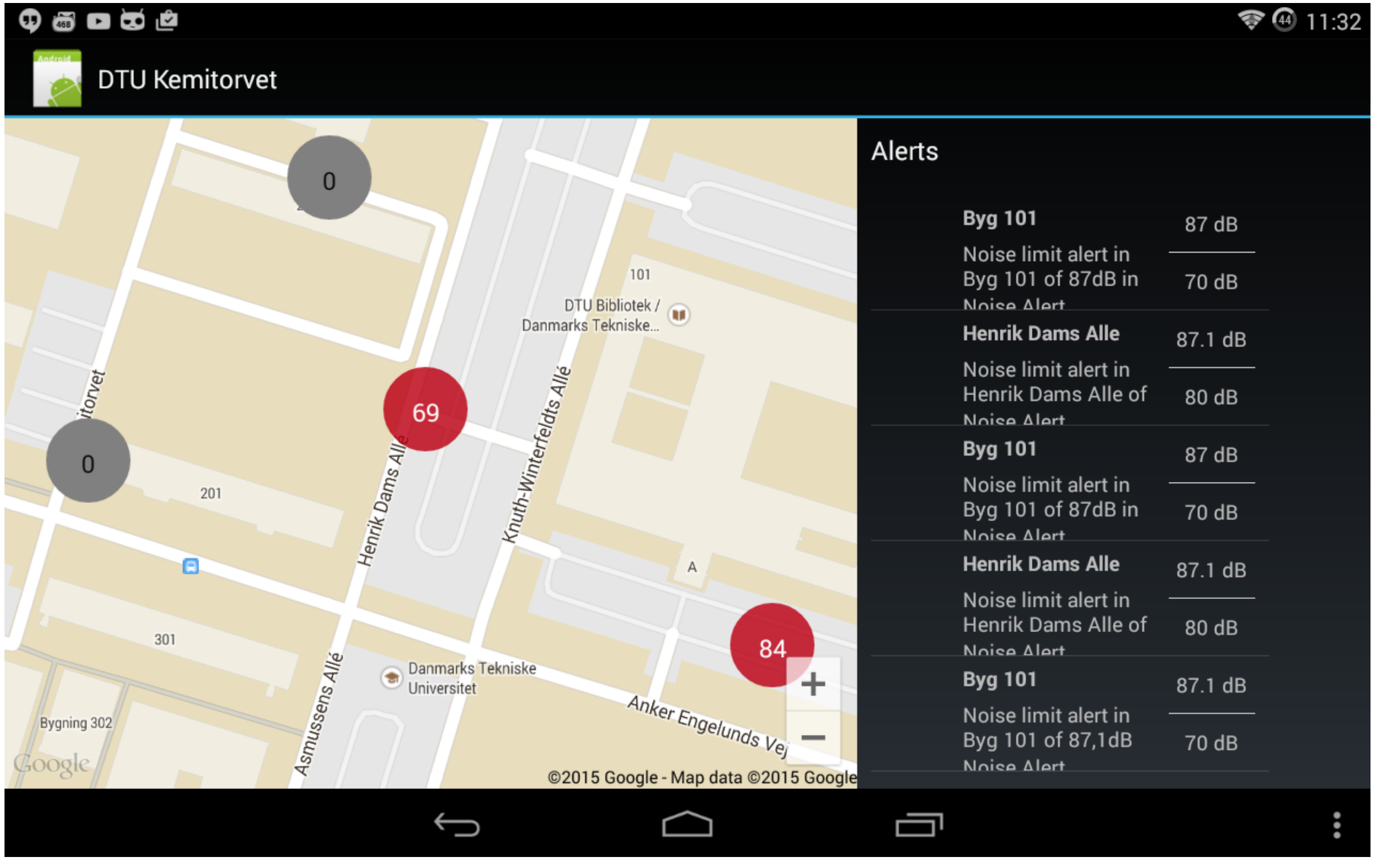
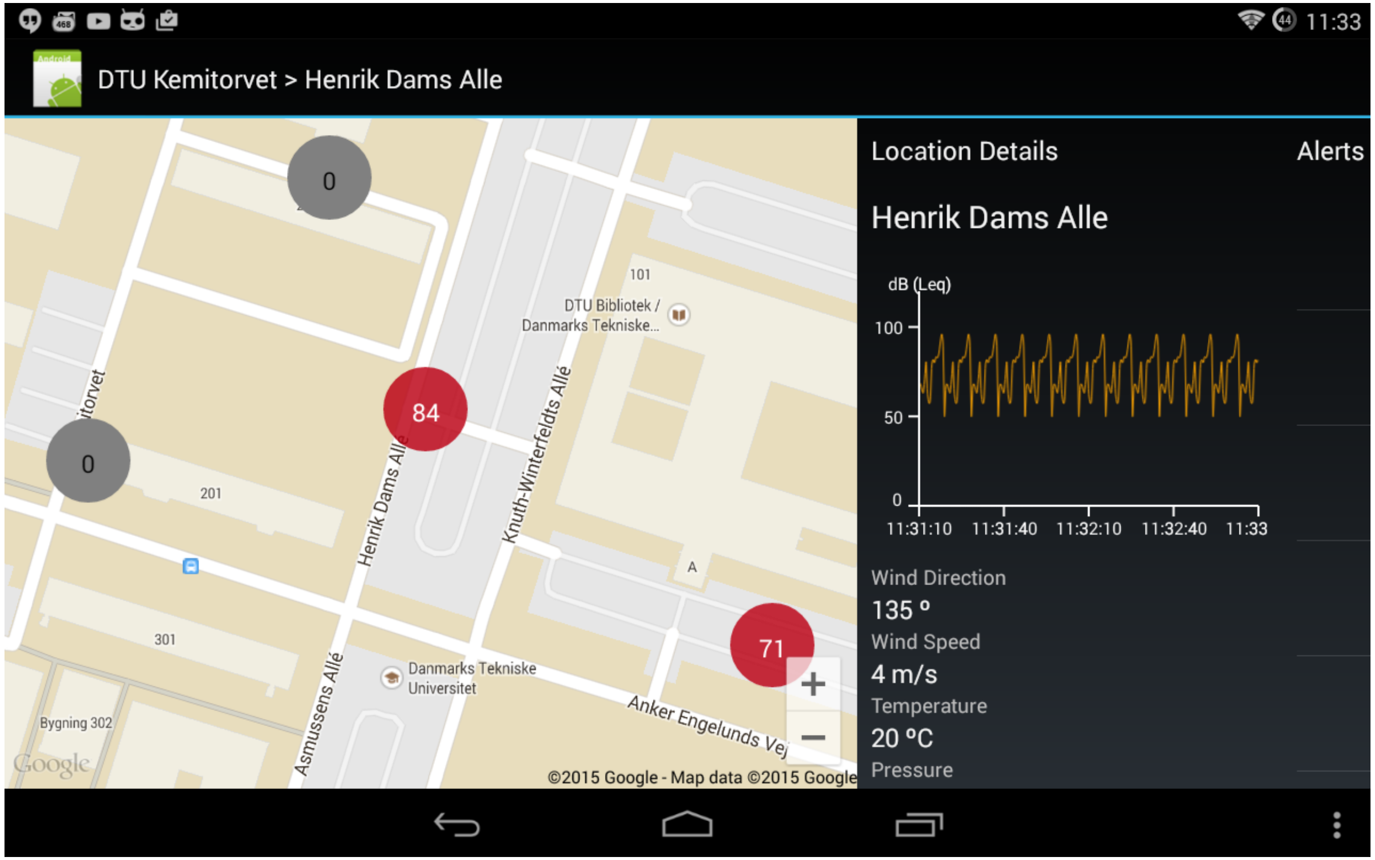


Figure D.5: Location selected, showing details about it



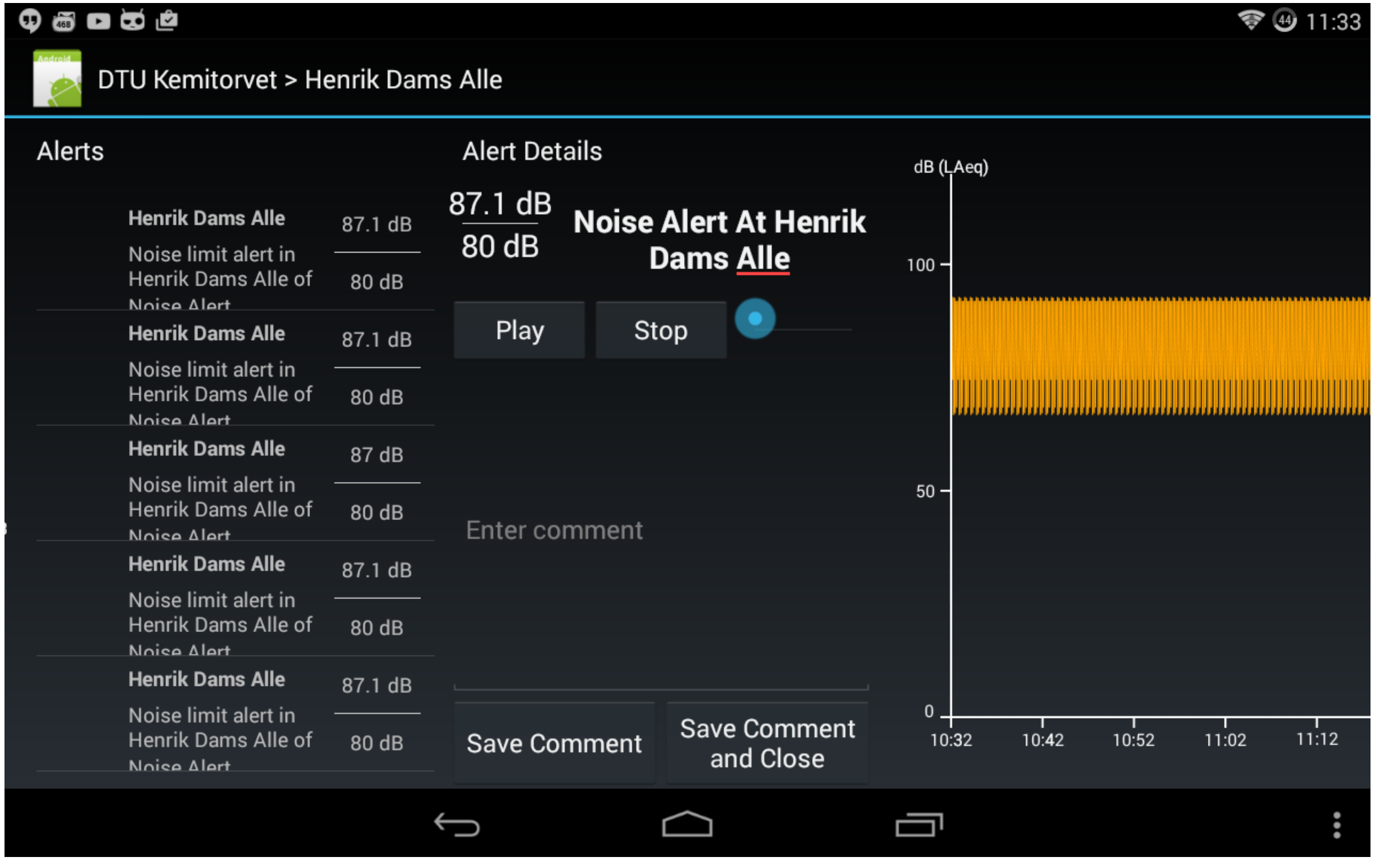
Byg 101	87 dB
Noise limit alert in Byg 101 of 87dB in Noise Alert	70 dB
Henrik Dams Alle	87.1 dB
Noise limit alert in Henrik Dams Alle of Noise Alert	80 dB
Byg 101	87 dB
Noise limit alert in Byg 101 of 87dB in Noise Alert	70 dB
Henrik Dams Alle	87.1 dB
Noise limit alert in Henrik Dams Alle of Noise Alert	80 dB
Byg 101	87.1 dB
Noise limit alert in Byg 101 of 87,1dB Noise Alert	70 dB

Figure D.6: Alert selected, showing details about Alert and sound clip playing



Wind Direction
135 º
Wind Speed
4 m/s
Temperature
20 ºC
Pressure

Figure D.7: Historical data about selected location, drop down lets the user select different views



11:33

DTU Kemitorvet > Henrik Dams Alle

Alerts

Henrik Dams Alle Noise limit alert in Henrik Dams Alle of Noise Alert	87.1 dB / 80 dB
Henrik Dams Alle Noise limit alert in Henrik Dams Alle of Noise Alert	87.1 dB / 80 dB
Henrik Dams Alle Noise limit alert in Henrik Dams Alle of Noise Alert	87 dB / 80 dB
Henrik Dams Alle Noise limit alert in Henrik Dams Alle of Noise Alert	87.1 dB / 80 dB
Henrik Dams Alle Noise limit alert in Henrik Dams Alle of Noise Alert	87.1 dB / 80 dB

Alert Details

87.1 dB / 80 dB **Noise Alert At Henrik Dams Alle**

Enter comment

Appendix E

Dashboard Mockup

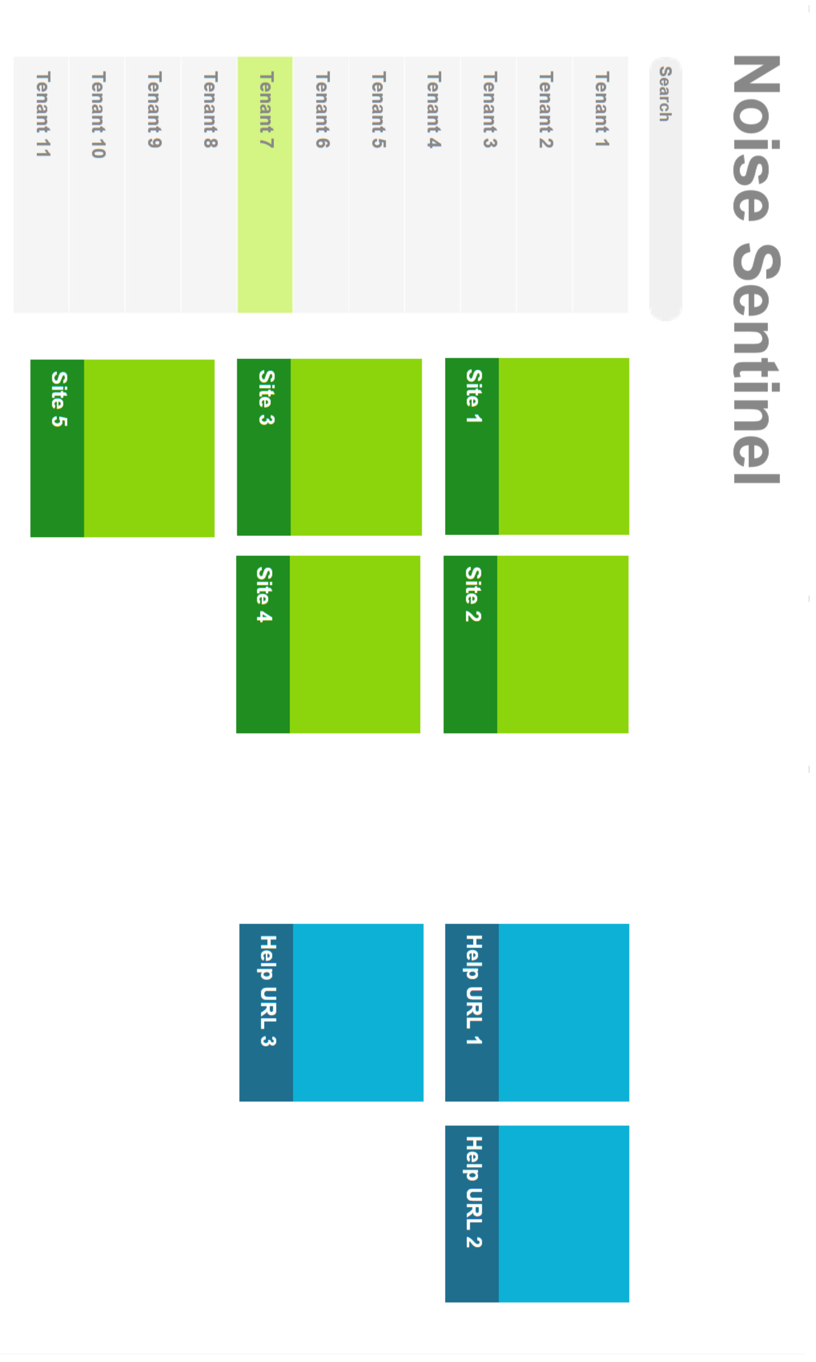


Figure E.1: Mock up of the Dashboard

Appendix F

Touchable MapFragment

Listing F.1: Touchable MapFragment implementation

```
public class TouchableMapFragment : MvxMapFragment
{
    public event EventHandler TouchDown;
    public event EventHandler TouchUp;

    public override View OnCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState)
    {
        var root = base.OnCreateView(inflater, container, savedInstanceState);
        var wrapper = new TouchableWrapper(Activity);
        wrapper.SetBackgroundColor(
            Resources.GetColor(Android.Resource.Color.Transparent));
        ((ViewGroup) root).AddView(wrapper,
            new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.MatchParent,
                ViewGroup.LayoutParams.MatchParent));

        wrapper.TouchUp = () =>
        {
            if (TouchUp != null)
                TouchUp(this, EventArgs.Empty);
        };
        wrapper.TouchDown = () =>
        {
            if (TouchDown != null)
                TouchDown(this, EventArgs.Empty);
        };

        return root;
    }

    private class TouchableWrapper : FrameLayout
    {
        public Action TouchDown;
        public Action TouchUp;

        #region ctors

        protected TouchableWrapper(IntPtr javaReference, JniHandleOwnership transfer)
            : base(javaReference, transfer) {}

        public TouchableWrapper(Context context)
            : this(context, null) {}

        public TouchableWrapper(Context context, IAttributeSet attrs)
            : this(context, attrs, 0) {}

        public TouchableWrapper(Context context, IAttributeSet attrs, int defStyle)
            : base(context, attrs, defStyle) {}

        #endregion

        public override bool DispatchTouchEvent(MotionEvent e)
        {
            switch (e.Action)
            {
                case MotionEventActions.Down:
                    if (TouchDown != null)
                        TouchDown();
                    break;
                case MotionEventActions.Cancel:
                case MotionEventActions.Up:
                    if (TouchUp != null)
                        TouchUp();
                    break;
            }

            return base.DispatchTouchEvent(e);
        }
    }
}
```

Appendix G

Custom Binding for MapView

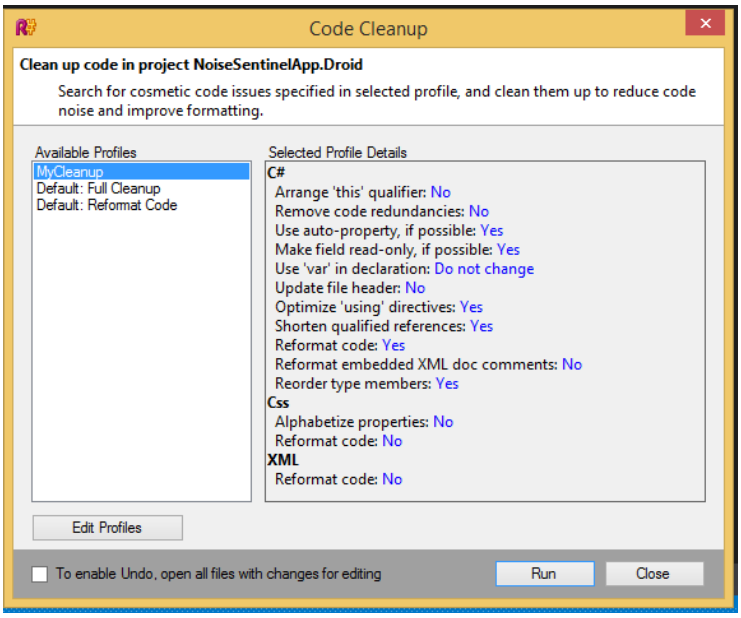
Listing G.1: Custom Binding for MapView

```
public class LocationMarkerSet
{
    private readonly GoogleMap _map;
    private readonly int _mapWidth;
    private readonly int _markerSize;
    private readonly int _textSize;
    private readonly List<LocationViewModelWrapper> _wrappers
        = new List<LocationViewModelWrapper>();
    private IEnumerable _itemsSource;
    private IDisposable _token;

    public LocationMarkerSet(GoogleMap map, int markerSize, int textSize, int mapWi
    {
        _map = map;
        _markerSize = markerSize;
        _textSize = textSize;
        _mapWidth = mapWidth;
    }

    public IEnumerable ItemsSource
    {
        get { return _itemsSource; }
        set
        {
            if (_itemsSource == value)
                return;
            if (_token != null)
            {
                _token.Dispose();
                _token = null;
            }
            _itemsSource = value;
            var notify = _itemsSource as INotifyCollectionChanged;
            if (notify != null)
                _token = notify.WeakSubscribe(HandleChangedMessage);
            ReloadAll();
        }
    }

    public IList<LocationViewModelWrapper> Wrappers
    {
        get { return _wrappers; }
    }

    private void ReloadAll()
    {
        RemoveAll();
        AddAll();

        CenterMap();
    }

    private void AddAll()
    {
        if (_itemsSource == null)
            return;
        foreach (var loc in _itemsSource.Cast<LocationViewModel>())
        {
            AddLocation(loc);
        }
    }

    private void AddLocation(LocationViewModel loc)
    {
        var options = new MarkerOptions();
        options.SetPosition(new LatLng(loc.Coordinate.Latitude,
            loc.Coordinate.Longitude));
        options.SetTitle(loc.Name);
        var marker = _map.AddMarker(options);
        var markerWrapper = new LocationViewModelWrapper(
            loc, marker, _markerSize, _textSize);
        _wrappers.Add(markerWrapper);
    }

    private void RemoveAll()
    {
        foreach (var wrap in _wrappers)
        {
            wrap.Remove();
            wrap.Dispose();
        }
        _wrappers.Clear();
    }

    private void RemoveLocation(LocationViewModel loc)
    {
        var wrap = _wrappers.FirstOrDefault(mw => mw.Location == loc);
        if (wrap == null)
            throw new MvxException("Zombie not found");
        _wrappers.Remove(wrap);
        wrap.Dispose();
    }

    private void HandleChangedMessage(object sender,
        NotifyCollectionChangedEventArgs e)
    {
        switch (e.Action)
        {
            case NotifyCollectionChangedAction.Add:
                foreach (var loc in e.NewItems.Cast<LocationViewModel>())
                {
                    AddLocation(loc);
                }
                break;
            case NotifyCollectionChangedAction.Remove:
                foreach (var loc in e.OldItems.Cast<LocationViewModel>())
                {
                    RemoveLocation(loc);
                }
                break;
            case NotifyCollectionChangedAction.Replace:
            case NotifyCollectionChangedAction.Move:
                throw new MvxException("Zombies should not be moved");
            case NotifyCollectionChangedAction.Reset:
                ReloadAll();
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }
    }

    private void CenterMap()
    {
        var builder = new LatLngBounds.Builder();
        foreach (var wrap in _wrappers)
        {
            builder.Include(wrap.Marker.Position);
        }
        var bounds = builder.Build();
        var padding = ((_mapWidth*10)/100);

        var cu = CameraUpdateFactory.NewLatLngBounds(bounds, padding);
        _map.AnimateCamera(cu);
    }
```

```
}

public sealed class LocationViewModelWrapper
    : IDisposable
{
    private readonly LocationViewModel _location;
    private readonly Marker _marker;
    private readonly int _markerSize;
    private readonly int _textSize;
    private readonly IDisposable _token;

    public LocationViewModelWrapper(
        LocationViewModel location, Marker marker, int markerSize, int textSize)
    {
        _location = location;
        _marker = marker;
        _markerSize = markerSize;
        _textSize = textSize;

        using (var bitmap = CreateMarkerView())
            _marker.SetIcon(BitmapDescriptorFactory.FromBitmap(bitmap));

        _token = _location.WeakSubscribe(OnLocationChanged);
    }

    public Marker Marker
    {
        get { return _marker; }
    }

    public LatLng Position
    {
        get { return _marker.Position; }
        set { _marker.Position = value; }
    }

    public LocationViewModel Location
    {
        get { return _location; }
    }

    public void Dispose()
    {
        _token.Dispose();
    }

    private void OnLocationChanged(object sender, PropertyChangedEventArgs e)
    {
        if (e.PropertyName == "Coordinate")
            Position = new LatLng(
                _location.Coordinate.Latitude, _location.Coordinate.Longitude);
        if (e.PropertyName == "CurrentNoiseLevel")
        {
            using (var bitmap = CreateMarkerView())
                _marker.SetIcon(BitmapDescriptorFactory.FromBitmap(bitmap));
        }
    }

    public void Remove()
    {
        _marker.Remove();
    }

    private Bitmap CreateMarkerView()
    {
        var currentNoiseLevel = Location.CurrentNoiseLevel;
        var backgroundColor = Color.Gray;
        var foregroundColor = Color.Black;
        if (currentNoiseLevel > 1 && currentNoiseLevel <= 40) // green
        {
            backgroundColor = Color.Argb(210, 146, 185, 56);
            foregroundColor = Color.White;
        }
        else if (currentNoiseLevel > 40 && currentNoiseLevel <= 55)
        {
            backgroundColor = Color.Argb(210, 242, 230, 0);
            foregroundColor = Color.Black;
        }
        else if (currentNoiseLevel > 55)
        {
            backgroundColor = Color.Argb(210, 188, 0, 20);
            foregroundColor = Color.White;
        }

        var px = _markerSize;
        var bitmap = Bitmap.CreateBitmap(px, px, Bitmap.Config.Argb8888);
        using (var canvas = new Canvas(bitmap))
        {
            using (var paint = new Paint(PaintFlags.AntiAlias)
                {Color = backgroundColor})
            {
                var centerXy = px/2f;
                canvas.DrawCircle(centerXy, centerXy, centerXy, paint);
                paint.Color = foregroundColor;
                paint.TextAlign = Paint.Align.Center;
                paint.TextSize = _textSize;
                canvas.DrawText(string.Format("{0}", Location.CurrentNoiseLevel),
                    centerXy,
                    centerXy + _textSize/2f, paint);
            }
        }
        return bitmap;
    }
}
```

Appendix H

ReSharper rules



Clean up code in project NoiseSentinelApp.Droid

Search for cosmetic code issues specified in selected profile, and clean them up to reduce code noise and improve formatting.

Selected Profile Details

C#
Arrange 'this' qualifier: No
Remove code redundancies: No
Use auto-property, if possible: Yes
Make field read-only, if possible: Yes
Use 'var' in declaration: Do not change
Update file header: No
Optimize 'using' directives: Yes
Shorten qualified references: Yes
Reformat code: Yes
Reformat embedded XML doc comments: No
Reorder type members: Yes
Css
Alphabetize properties: No
Reformat code: No
XML
Reformat code: No

To enable Undo, open all files with changes for editing

Figure H.1: ReSharper Clean Code settings

Appendix I

Line counting script

```
// based on https://gist.github.com/praeclarum/1608597

void Main()
{
  var projects = new List<Solution> {
    new Solution {
      Name = "Windows 8",
      ProjectFiles = new List<string> {
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp.Store\\
            NoiseSentinelApp.Store.Windows\\NoiseSentinelApp.Store.Windows.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp.Store\\
            NoiseSentinelApp.Store.Shared\\NoiseSentinelApp.Store.Shared.projitems"
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp.Models\\
            NoiseSentinelApp.Models.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp\\
            NoiseSentinelApp.csproj",

        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\Custom Views\\MiniChart.
            WindowsCommon\\MiniChart.WindowsCommon.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\Custom Views\\MiniChart.Cor
            MiniChart.Core.csproj",
      },
    },

    new Solution {
      Name = "Android",
      ProjectFiles = new List<string> {
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp.Droid\\
            NoiseSentinelApp.Droid.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp.Models\\
            NoiseSentinelApp.Models.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp\\
            NoiseSentinelApp.csproj",

        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\Custom Views\\MiniChart.Droid\\
            MiniChart.Droid.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\Custom Views\\MiniChart.Core\\
            MiniChart.Core.csproj"
      },
    },

    new Solution {
      Name = "Core",
      ProjectFiles = new List<string> {
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp.Models\\
            NoiseSentinelApp.Models.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\NoiseSentinelApp\\
            NoiseSentinelApp.csproj",
        "C:\\vcs\\git\\MastersThesis\\NoiseSentinelApp\\Custom Views\\MiniChart.Core\\
            MiniChart.Core.csproj"
      },
    },
    };

  var excludedFiles = new[] { "App.xaml", "NoiseSentinelTheme.xaml", "OxyplotStyles.xam
      , "Resource.Designer.cs",
    "AssemblyInfo.cs", };

  new CodeShareReport().Run(projects, excludedFiles);
}

class CodeShareReport
{
  Dictionary<string, FileInfo> _files = new Dictionary<string, FileInfo>();

  void AddCodeRef (string path, Solution sln)
  {
    if (_files.ContainsKey(path))
    {
      _files[path].Solutions.Add(sln);
      sln.CodeFiles.Add(_files[path]);
    }
    else
    {
      var info = new FileInfo { Path = path, };
      info.Solutions.Add(sln);
      _files[path] = info;
      sln.CodeFiles.Add(info);
    }
  }

  void AddViewRef (string path, Solution sln)
  {
    if (_files.ContainsKey(path))
  {
    _files[path].Solutions.Add(sln);
    sln.ViewFiles.Add(_files[path]);
  }
  else
  {
    var info = new FileInfo { Path = path };
    info.Solutions.Add(sln);
    _files[path] = info;
    sln.ViewFiles.Add(info);
  }
}

const string SharedProjectPrefix = "$(MSBuildThisFileDirectory)";
static string[] AndroidViewFileEndings = new[] { ".axml", ".xml" };
static string[] ViewCodeBehindFileEndings = new [] { "Fragment.cs", ".xaml.cs", "
    cs" };
  public void Run (List<Solution> solutions, IEnumerable<string> excludedFiles)
{
  //
  // Find all the files
  //
  foreach (var sln in solutions)
  {
    foreach (var projectFile in sln.ProjectFiles)
    {
      var dir = Path.GetDirectoryName(projectFile);
      var projectName = Path.GetFileNameWithoutExtension(projectFile);
      var doc = XDocument.Load(projectFile);

      var q = GetPaths(doc, "Compile", excludedFiles: excludedFiles);
      foreach (var inc in q)
      {
        var inc1 = inc;
        if (inc.StartsWith(SharedProjectPrefix)) {
          inc1 = inc.Remove(0, SharedProjectPrefix.Length);
        }
        var path = Path.GetFullPath(Path.Combine(dir, inc1));

        if (ViewCodeBehindFileEndings.Any(x => path.ToLowerInvariant().Contains(x
            ToLowerInvariant())))
          AddViewRef(path, sln);
        else
          AddCodeRef(path, sln);
      }

      // xaml (very naive implementation)
      q = GetPaths(doc, "Page", excludedFiles: excludedFiles);
      foreach(var inc in q)
      {
        var inc1 = inc;
        if (inc.StartsWith(SharedProjectPrefix)) {
          inc1 = inc.Remove(0, SharedProjectPrefix.Length);
        }
        var path = Path.GetFullPath(Path.Combine(dir, inc1));
```

```
          AddViewRef(path, sln);
        }

        //axml
        q = GetPaths(doc, "AndroidResource",
          allowedFileEndings: AndroidViewFileEndings, excludedFiles: excludedFiles);
        foreach(var inc in q)
        {
          var path = Path.GetFullPath(Path.Combine(dir, inc));
          AddViewRef(path, sln);
        }
      }
    }

    //
    // Get the lines of code
    //
    foreach (var f in _files.Values)
    {
      try
      {
        f.LinesOfCode = File.ReadAllLines(f.Path).Length;
      }
      catch (Exception) { }
    }

    //
    // Output
    //
    var table = new ConsoleTable("Sln", "Total [l]", "Unique [l]", "Shared [l]", "View
        ]", "Unique [%]", "Shared [%]", "View [%]");
    foreach (var sln in solutions)
    {
      table.AddRow(
        sln.Name,
        sln.TotalLinesOfCode,
        sln.UniqueLinesOfCode,
        sln.SharedLinesOfCode,
        sln.ViewLinesOfCode,
        string.Format("{0:p}", sln.UniqueLinesOfCode / (double)sln.TotalLinesOfCode),
        string.Format("{0:p}", sln.SharedLinesOfCode / (double)sln.TotalLinesOfCode),
        string.Format("{0:p}", sln.ViewLinesOfCode / (double)sln.TotalLinesOfCode));
    }

    Console.WriteLine(table.ToString());
    Console.WriteLine("\tLegend:\r\n\t\tl = lines");
  }

  private IEnumerable<string> GetPaths(XDocument doc, string elementName, string
      attribute = "Include", IEnumerable<string> allowedFileEndings = null,
    IEnumerable<string> excludedFiles = null)
  {
    if (allowedFileEndings == null)
      allowedFileEndings = new[] {".cs", ".xaml", ".xaml.cs", ".axml", ".xml"};

    if (excludedFiles == null)
      excludedFiles = new[] { "App.xaml", "Resource.Designer.cs" };

    var items = from x in doc.Descendants()
      let e = x as XElement
      where e != null
      where e.Name.LocalName == elementName
      where e.Attributes().Any(a => a.Name.LocalName == attribute)
      where allowedFileEndings.Any(a => e.Attribute(attribute).Value.ToLowerInvarian
          Contains(a.ToLowerInvariant()))
      where !excludedFiles.Any(a => e.Attribute(attribute).Value.ToLowerInvariant()
          Contains(a.ToLowerInvariant()))
      select e.Attribute(attribute).Value;
    return items;
  }
}

class FileInfo
{
  public string Path = "";
  public HashSet<Solution> Solutions = new HashSet<Solution>();
  public int LinesOfCode = 0;

  public override string ToString ()
  {
    return Path;
  }
}

class Solution
{
  public string Name = "";
  public List<string> ProjectFiles = new List<string>();
  public List<FileInfo> CodeFiles = new List<FileInfo>();
  public List<FileInfo> ViewFiles = new List<FileInfo>();

  public override string ToString ()
  {
    return Name;
  }

  public int UniqueLinesOfCode
  {
    get
    {
      return (from f in CodeFiles
          where f.Solutions.Count == 1
          select f.LinesOfCode).Sum();
    }
  }

  public int SharedLinesOfCode
  {
    get
    {
      return (from f in CodeFiles
          where f.Solutions.Count > 1
          select f.LinesOfCode).Sum();
    }
  }

  public int LinesOfCode
  {
    get
    {
      return (from f in CodeFiles
          select f.LinesOfCode).Sum();
    }
  }

  public int ViewLinesOfCode
  {
    get
    {
      return (from f in ViewFiles
          select f.LinesOfCode).Sum();
    }
  }

  public int TotalLinesOfCode
  {
    get { return LinesOfCode + ViewLinesOfCode; }
  }
}

#region ConsoleTable
// https://github.com/khalidabuhakmeh/ConsoleTables/blob/master/ConsoleTables.Core/
    ConsoleTable.cs
public class ConsoleTable
{
  public IList<string> Columns { get; protected set; }
  public IList<object[]> Rows { get; protected set; }

  public ConsoleTable(params string[] columns)
  {
    Columns = new List<string>(columns);
    Rows = new List<object[]>();
  }

  public ConsoleTable AddColumn(string[] names)
  {
    foreach (var name in names)
      Columns.Add(name);

    return this;
  }

  public ConsoleTable AddRow(params object[] values)
  {
    if (values == null)
    throw new ArgumentNullException("values");

  if (!Columns.Any())
    throw new Exception("Please set the columns first");

  if (Columns.Count != values.Length)
    throw new Exception(string.Format("The number columns in the row ({0}) does no
        match the values ({1}",
      Columns.Count, values.Length));

  Rows.Add(values);
    return this;
}

public static ConsoleTable From<T>(IEnumerable<T> values)
{
  var table = new ConsoleTable();

  var columns = typeof(T).GetProperties().Select(x => x.Name).ToArray();
  table.AddColumn(columns);

  foreach (var propertyValues in values.Select(value => columns.Select(column => t
      (T).GetProperty(column).GetValue(value, null))))
    table.AddRow(propertyValues.ToArray());

  return table;
}

public override string ToString()
{
  var builder = new StringBuilder();

  // find the longest column by searching each row
  var columnLengths = Columns
    .Select((t, i) => Rows.Select(x => x[i])
      .Union(Columns)
      .Where(x => x != null)
      .Select(x => x.ToString().Length).Max())
      .ToList();

  // create the string format with padding
  var format = Enumerable.Range(0, Columns.Count)
    .Select(i => " | {" + i + ", -" + columnLengths[i] + " }")
    .Aggregate((s, a) => s + a) + " |";

  var longestLine = 0;
  var results = new List<string>();

  // find the longest formatted line
  foreach (var result in Rows.Select(row => string.Format(format, row)))
  {
    longestLine = Math.Max(longestLine, result.Length);
    results.Add(result);
  }

```

```
    // create the divider
    var line = " " + string.Join("", Enumerable.Repeat("-", longestLine - 1)) + " ";

    builder.AppendLine(line);
    builder.AppendLine(string.Format(format, Columns.ToArray()));

    foreach (var row in results)
    {
      builder.AppendLine(line);
      builder.AppendLine(row);
    }

    builder.AppendLine(line);
    builder.AppendLine("");

    return builder.ToString();
  }

  public void Write() { Console.WriteLine(ToString()); }
}
#endregion
```

Bibliography

[agi01] The agile manifesto. http://www.agilemanifesto.org/principles.html, 2001. [Online; Last seen 1-19-2015].

[Bug09] Laurent Bugnion. Mvvmlight. http://www.mvvmlight.net/, 2009. [Online; Last seen 1-19-2015].

[Cie12] Tomasz Cielecki. Cross-platform mobile notification system for noise monitoring system. diploma thesis, Technical University of Denmark (DTU), 2012. IMM-B.Eng-2012-1.

[Cla13] Jeremy Clark. Dependency injection: A practical introduction. http://www.jeremybytes.com/downloads/dependencyinjection.pdf, 2013. [Online; Last seen 1-19-2015].

[Cor14] Cordova. http://cordova.apache.org/, 2014. [Online; Last seen 1-19-2015].

[dI03] Miguel de Icaza. Mono early history. http://lists.ximian.com/pipermail/mono-list/2003-October/016345.html, 2003. [Online; Last seen 1-19-2015].

[dI11] Miguel de Icaza. Announcing xamarin. http://tirania.org/blog/archive/2011/May-16.html, 2011. [Online; Last seen 1-19-2015].

[Fie00] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures. Master's thesis, University Of California, Irvine, 2000.

[Fow04] Martin Fowler. Presentation model. http://martinfowler.com/eaaDev/PresentationModel.html, 2004. [Online; Last seen 1-19-2015].

[GC11] The Register Gavin Clarke. Novell mono layoffs. http://www.theregister.co.uk/2011/05/03/novell_mono_layoffs/, 2011. [Online; Last seen 1-19-2015].

[Gro05] John Grossman. Introduction to model/view/viewmodel pattern for building wpf apps. http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx, 2005. [Online; Last seen 1-19-2015].

[in09] i newswire. Brüel & kjær and lochard join forces. http://www.i-newswire.com/br-el-kj-r-and-lochard-join-forces/a252100, 2009. [Online; Last seen 1-19-2015].

[Jet14] JetBrains. Resharper::the most intelligent extension for visual studio. https://www.jetbrains.com/resharper/, 2014. [Online; Last seen 1-19-2015].

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-21, ESD-90-TR-222, November 1990.

[Kjæ14a] Brüel & Kjær. About brüel & kjær. http://bksv.com/AboutUs/AboutBruelAndKjaer, 2014. [Online; Last seen 1-19-2015].

[Kjæ14b] Brüel & Kjær. Brüel & kjær company presentation video. http://bksv.com/video/CompanyPresentation/bn0455.wmv, 2014. [Online; Last seen 1-19-2015].

[Kjæ14c] Brüel & Kjær. Environment management solutions. http://bksv.com/Products/EnvironmentManagementSolutions.aspx, 2014. [Online; Last seen 1-19-2015].

[Kjæ14d] Brüel & Kjær. Noise sentinel. http://www.bksv.com/Products/EnvironmentManagementSolutions/UrbanEnvironmentManagement/Type7871NoiseSentinel, 2014. [Online; Last seen 1-19-2015].

[Lod10] Stuart Lodge. Mvvmcross. https://github.com/MvvmCross/MvvmCross, 2010. [Online; Last seen 1-19-2015].

[Lun13] Jonas Lund. Tablet interface for environment monitoring. diploma thesis, Technical University of Denmark (DTU), 2013. IMM-B.Eng-2013-12.

[Mar11] Tony Marston. Dependency injection is evil. http://www.tonymarston.net/php-mysql/dependency-injection-is-evil.html, 2011. [Online; Last seen 1-19-2015].

[Mic11] Microsoft. Portable class libraries. http://msdn.microsoft.com/en-us/library/vstudio/gg597391(v=vs.100).aspx, 2011. [Online; Last seen 1-19-2015].

[Mic12] Microsoft. Reactive extensions. https://rx.codeplex.com/, 2012. [Online; Last seen 1-19-2015].

[Mic13] Microsoft. Msbuild. http://msdn.microsoft.com/en-us/library/dd393574.aspx, 2013. [Online; Last seen 1-19-2015].

[MW10] Novell Mass Waltham. Novell agrees to be acquired by attachmate corporation. https://www.novell.com/news/press/2010/11/novell-agrees-to-be-acquired-by-attachmate-corporation.html, 2010. [Online; Last seen 1-19-2015].

[NUn14] NUnit. Nunit is a unit-testing framework for all .net languages. http://www.nunit.org/, 2014. [Online; Last seen 1-19-2015].

[Rho14] RhoMobile. http://rhomobile.com/, 2014. [Online; Last seen 1-19-2015].

[Rob12] James Robinson. http://opensignal.com/reports/fragmentation.php, 2012. [Online; Last seen 1-19-2015].

[tea14] ReactiveUI team. Reactiveui. http://reactiveui.net/, 2014. [Online; Last seen 1-19-2015].

[Xam14a] Xamarin. http://xamarin.com/, 2014. [Online; Last seen 1-19-2015].

[Xam14b] Xamarin. Xamarin forms. http://xamarin.com/forms, 2014. [Online; Last seen 1-19-2015].

[ZDN00] Steven Bonisteel ZDNet. Microsoft sees nothing but .net ahead. https://web.archive.org/web/20111105195731/http://www.zdnetasia.com/microsoft-sees-nothing-but-net-ahead-10028684.htm, 2000. [Online; Last seen 3-29-2012 (archive.org)].

[ZDN09] Rupert Goodwins ZDNet. Monotouch lets .net coders build iphone apps. http://www.zdnet.com/article/monotouch-lets-net-coders-build-iphone-apps, 2009. [Online; Last seen 1-19-2015].