University of
South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2018
Industrial IT and Automation

Environmental Public Health Information Management System

Ola Anton Grytten

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

Course: FMH606 Master's Thesis 2018
Title: *Environmental Public Health Information Management System*
Pages: *108*
Keywords: *MVC Core 2.0, Android, Azure, Database, API*

Student: *Ola Anton Grytten*
Supervisor: *Hans-Petter Halvorsen*
External partner: *Tel-Tek, Porsgrunn kommune, Sykehuset Telemark*
Availability: *Open*

Summary:
This project was carried out to develop a website and an Android app to display different pollutions in the Grenland area. The background for the project is increased focus on environmental health and performing measures to improve the environment. The Android app was mainly developed to allow push notifications, which eliminates the need for checking the website to detect pollutions.

Both the website and the mobile app was implemented. The website was hosted at Microsoft Azure. Push notifications worked as intended.

The website was developed by using the ASP.NET Core 2.0 Framework. The mobile app was developed by using Xamarin.Android. Microsoft SQL Server was used as database storage. An external API from NILU was used for gathering different types of air pollution data.

Preface

This master thesis is my final work of my master program Industrial IT and Automation at The University College of South-Eastern Norway. I am grateful for all that I have learned during this period, and for getting the chance to be a part of this exciting project.

I would like to thank my supervisor Hans-Petter Halvorsen for good guidance and feedback during the project period. I would also like to thank Porsgrunn kommune, Tel-Tek, Sykehuset Telemark and my co-supervisor Nils-Olav Skeie.

The resulting website can be found at:
`https://environmentalpublichealth.azurewebsites.net/`
An OneDrive folder containg relevant files can be found at:
`https://1drv.ms/f/s!AreA9velRQIgib1ZK_jOYnlJ9OSAsA`

The reader is expected to have some knowledge about programming and database development. However, much of the content in the report can be understood without this.

The task description can be found in appendix A.

Porsgrunn, 15th May 2018

Ola Anton Grytten

Contents

List of Figures

List of Tables

Nomenclature

Symbol	**Explanation**
IDE	Integrated Development Environment
API	Application Programming Interface
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
CSS	Cascading Style Sheets
RAD	Rapid Application Development
FCM	Firebase Cloud Messaging
PHP	Hypertext Preprocessor
NILU	Norwegian Institute for Air Research
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language
LINQ	Language Integrated Query
IIS	Internet Information Services
AXML	Article XML Markup Language
XML	eXtensible Markup Language

1 Introduction

This master thesis will be written as an answer to the project proposal ”Environmental Public Health Information Management System” within the course ”FMH606-1 18V Master‘s Thesis”, given by The University College of South-Eastern Norway. The project is a collaboration between USN, Tel-Tek, Porsgrunn municipality and Telemark hospital-Department of Occupational Medicine.

1.1 Background

Government, companies and citizens have in later time increased their focus on environmental health. There are many sources that expose the environment for different kinds of emissions like air pollution, toxic released in water etc. Some may only affect local environment, while other may affect the environment globally. How the emissions affect the environmental can be a complex question. Wear and tear on car tire for example, will release plastic as particulate matter into the air. This plastic can be inhaled while breathing, or it may end up in the ocean and later end up into the food chain. Furthermore, green house gases, which are threatening to increase earth temperature, may not be any immediately threat to the human and animal health, but it’s widely known that they can cause global warming which may lead to climatic changes.

It is important for the government and companies to take measures for reducing emissions both locally and globally. The decision makers, for instance the government, need to be provided with good information regarding the topic. Citizens may also be interested in everyday pollution in their local environment, which for example will be helpful for asthmatic people who want to stay inside on days with problematic pollution levels. Today there are a lot of environmental data available, but they are typically spread around on different websites. By gathering available data into one solution, it would be easier to get an overview and read the available data. Therefore, USN has collaborated with Tel-Tek, Porsgrunn municipality and Telemark hospital-Department of Occupational Medicin to develop a prototype to make environmental data publicly available at Grenland area.

1.2 Report Structure

Chapter 2 gives a brief description about the system and available data.
Chapter 3 tells more about how the system is planned to behave.
Chapter 4 gives an overview of framework, tools and services that could be used.
Chapter 5 tells more about the development method used, and which frameworks, tools, and services that are used.
Chapter 6 tells about how the database was implemented.
Chapter 7 tells more about the website implemented.
Chapter 8 tells more about the mobile app implementation.
Chapter 9 tells more about how the services is implemented and how data is gathered.
Chapter 10 show the result of the implementations.
Chapter 11 discusses the results.
Chapter 12 gives some suggestions for future work.
Chapter 13 gives the final conclusion.

2 System Overview

This chapter gives the reader an introduction about the need of an information management system, an overview of existing systems and available emission and pollution data. An overview of the system will be given.

2.1 The need of an Information Management System

Focus on global warming and health gives increased focus on accessing environmental data. The importance of performing measures to lower emissions strengthens the need of relevant pollution data. Also, the public interest in health in peoples local areas would benefit from getting live pollution data. For example, this is beneficial for asthmatic people who should stay inside during the worst periods of air pollution.

Such a system might enlighten citizens that are not really aware of the effects that pollution might have on health and environment. People might see the benefits of taking public transport instead of driving their own car, or staying inside during the worst periods.

Regarding Grenland, there are several large factories releasing emissions, which increases the need of pollution measurements.

Furthermore, an information management system can expose other types of measured data like temperature, number of cars passing, wind speed, wind direction etc. Such information can be correlated to the amount of pollution, especially air pollution, and can be used to see the bigger picture, like how the different factors may affect each other.

2.2 Existing Solutions

There are some existing websites that gives information about pollution. This section will cover two of them.

- Luftkvalitet.info
- Norskeutslipp.no

Luftkvalitet.info is developed and maintained by NILU to give the public access to real time monitoring of air pollution within different cities in Norway [1]. Air pollution is given by measured values and a color code to indicate the health risk of air exposure. The different color codes are explained with text where they are addressing different population groups that possibly should stay inside. Asthmatic for example will tolerate less air pollution than the rest of the population. There is some limited functionality for accessing trending data for the different air pollution components. For the larger cities it is possible to receive SMS notifications during the winter. [2]

Norskeutslipp.no offers emission data based on reports from the industry. Smaller emission releases are not reported, but estimated based on activity data and emission factors. The website offers functionality to narrow down emission based on location. There is not real time monitoring as the emissions are reported at annual reports from the industry. [3]

2.3 System Description and Available Data

The system is built up around 4 different components, which is illustrated in figure 2.1. The website and the mobile app are displaying all data within the system. The pollutions within the system are air pollution gathered from NILIs API, which is covered in section 2.3.1. One of the main concept is to generate status for the sources, for example indicates how much the current air pollution is by text and color. The changes will trigger the system to send out push notifications to any users that are subscribing on these through the mobile app.

Both the website and the mobile app consists of a main page intended to be used as a dashboard, were the most important data is displayed. The dashboard on the website is organized by showing the last day historical data in addition to information about the statuses within the system. The mobile app dashboard shows all the status, but only the last measured values, not the last day historical data.

Furthermore, the website and mobile app contain functionalities for getting more details and show trending data for the last month. Moreover, the website provide additional functionalities for data analytics and report generations.

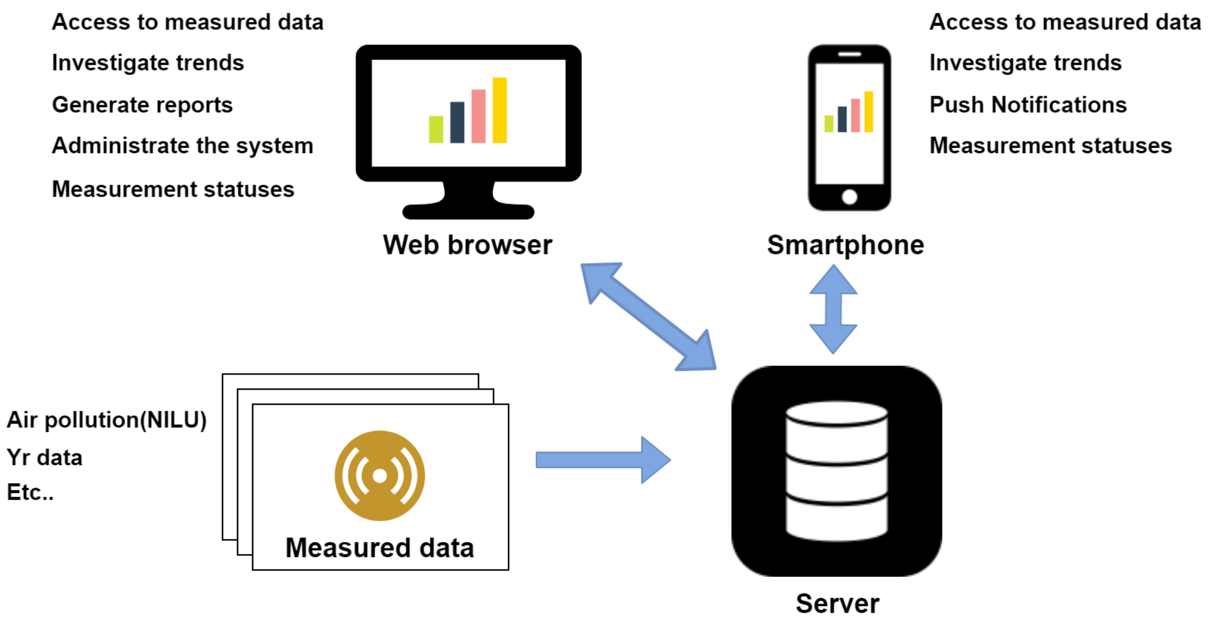


Figure 2.1: System Description

2.3.1 Available Data - Grenland Area

NULI is a service which measure air pollution on different locations in Norway. The measurements happen automatically and are available to third parties through an API, which gives access to four different stations in the Grenland area [4].

- Sverresgate - Porsgrunn
- Lensmannsdalen - Skien
- Furulund - Brevik
- Haukenes - Skien

The stations currently measure five different components:

- **NO2**
  NO2 is the short term for Nitrogen Dioxide, and belongs to a group of reactive gases known as oxides of nitrogen oxides (NOx). The primary sources of this gas are emission from cars, trucks, buses or engines that burn fuel. [5]
- **PM10**
  PM10 is the short term of particulate matter, which indicates the particle size of an air pollutant. Particles emitted into the atmosphere will eventually be taken down by the gravity or rain. Larger particles will quickly be taken out from the atmosphere. However, finer particles will stay in the atmosphere for a longer time and can be transported over longer distances.
  The size is defined as the fraction particles with an aerodynamic diameter smaller then the given size, which for PM10 is 10 µm. [6]
- **PM2.5**
  PM2.5 is the same as PM10, but with smaller particle matter size, which is 2.5 µm. [6]
- **O3**
  O3, the short term for Ozone, is the most important photo chemical oxidant in the troposphere, where it filters dangerous light from the sun. Because of photo chemical reactions with NOx gases, the concentrations of ozone are often lower in urban centres and higher in suburban areas. [7]
- **CO2**
  CO2, the short therm of Carbon dioxide, is a common and naturally gas. CO2 has an important role for life on earth, as humans and animals exhale it while plants absorb it in order to grow. CO2 is a naturally greenhouse gas because it traps energy from the sun in the atmosphere. However, emission of this gas from human activity may increase the greenhouse effect. The main emission source comes from engines that burn fuel. [8]

3 Planned System Behavior

The content of this chapter will explain more about system requirements, specifications and planned behavior.

3.1 Requirements and Specifications

A website should be able to give the users a quick overview of different sources and give a short indication of the current values and the condition of the data. The home page should contain a dashboard giving a graphical presentation of the most important data. By logging in and further investigating the data, it should be possible to get a graphical presentation of the data by time, which could be for days, weeks and years. It should be possible to compare data from different time periods, for example compare the present year with previous years etc.

Further, the website should be able to generate reports based on the gathered data, but this functionality should not be available to ordinary users.

The data should also be available through a Mobile-App which should feature most of the same possibilities as the website when it comes to user functionality. In addition, it should also be possible to generate push notifications if the data goes over or under a specific limit. The users should be able to choose to subscribe on the sources they want to receive push notifications for.

The system should be able to create services that will let the sensor system send data to the system. It should be possible to generate services that will collect data from external data sources which could be presented in the system.

Finally, it should be easy to install the system on a server, and move it between servers.

3.2 Website - Planned Behavior

A plan for the behavior of the website is made based on the requirements and specifications given in section 3.1. A navigation bar should be present on the top of the page for easy access to all its functionally. Figure 3.1 shows an overview of the planned functionality. The section below will explain different web pages which is planned.

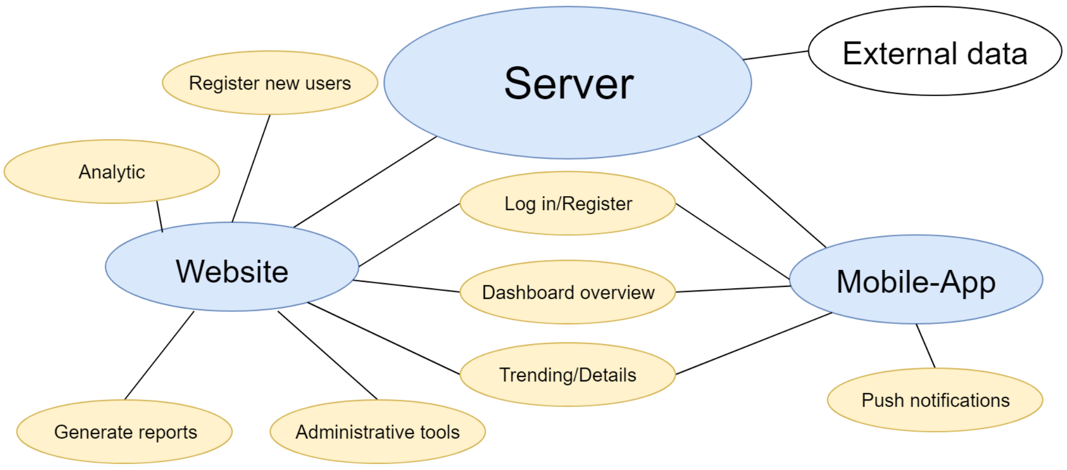


Figure 3.1: Planned functionality - Overview

3.2.1 User Log In and Registration

Some of the website's functions are more resource demanding, like for example analytic investigation and report generation. Users are required to log in to get access to these features. In this way it is easier to restrict or limit their usage, in case of shortage in server resources.

3.2.2 Dashboard

The dashboard web page will act as the home page of the website. The most important functionality is to give the user a quick overview of the measured parameters and associated statuses. The most important sources should be graphically represented by plotting historical data from the last day. Logged in users should be able to make personal configurations for the most important sources on the dashboard. In addition, all the sources should also be represented in a map for easy location. Furthermore, a table should contain more details about the sources and their measured parameters should be presented at the bottom of the page.

The status for each parameter will be illustrated by a color and a text indicating the severity. The status of a source will be equal to the highest parameter severity associated to the source.

3.2.3 Trending

A web page should provide more details about the sources within the system. When the user selects a source the web page should return details and corresponding parameters, where the last historical data should be plotted to detected recently trendings. Moreover, options for fixed time intervals should be available, like for example last day or last week. Each parameter should also have their status limits specified. Further, data from external sources may come with a license for usage, so the web page must be able to inform the users about any potential licenses.

3.2.4 Analytic

A web page should give the user the ability to investigate sources and parameters more in depth. The most important concept is to compare parameters against other parameters and detecting trends. In term of air pollution it might be possible to determine if any area is more problematic then other areas, and perform measures to, for instance, lower air pollution. It should also be possible to compare periods of the same parameters by plotting them side by side. This way, it is easier to determine if any period stands out as problematic or good. Moreover, a logged in user should be able to save the setup for selected parameters. This would be an advantage if an user continuously is investigating the same sources.

If choosing a long time period, there might be to many measurements which make the plot unreadable. Therefore, it should be possible to scale the data, which means combining measurements within a time period into only one measurement.

3.2.5 Report Generation

It should be possible to generate reports in a pdf format, which should include some of the same data as the analytic functionality provides. These reports are intended to be used as a basis for decision making. An example could be to perform measures and actions for improving air quality. Like the analytic web page, logged in users should be able to save the setup for later use.

3.2.6 Administrative functionality

The system should be dynamically. This means that administrative users should be able to add sources to the system by logging into an administrative page. It should also be possible to edit sources in the system.

3.3 Mobile App - Planned Behavior

The mobile app will share some of the functionalities as the website, except analytic investigation and report generation. In addition, it should receive push notifications if status changes happens. Moreover, the user should be able to choose which sources he/she wants to subscribe on. Due to this, the user must be logged in to be allowed to use the app functionalities.

3.3.1 Dashboard

Like the website, the mobile dashboard should give the user a quick overview of the measured parameters and associated statuses. However, it will contain fewer details, like plotting, due to smaller screen size.

3.3.2 Trending

The mobile trending page should have the same functionalities as the web page.

3.3.3 Push Notifications

A mobile app is ideal for receiving push notifications on the go. The users should be able to subscribe on push notifications for each source they feel is relevant for them. The user should also be able to disable push notifications at configurable time intervals, typically at night.

3.3.4 Register/Log in

It should be possible to register new user directly from the app. There should be no system administrative functions on the mobile app.

4 Technical Analysis

This chapter will mainly cover topics regarding requirements for different tools, framework and hardware, and which of them that might be best suited to be used in the solution.

4.1 Needed Requirements

The functionality given in chapter 3 will give the base line of what the system should do. The functionality can be divided into three independent sections, which use the same database.

- **Website front and back-end**
  Typically using a server side framework which generates HTML/JavaScript web pages populated with data from a database.
- **Services**
  Typically background related tasks. Based on the desired functionality this would include fetching external data and sending push notifications as it is not directly tied to the website. However, it would use the same database.
- **Mobile App**
  Independent from website-front end and services, but need access to the same database. The mobile app can access data through APIs on the server.

A decision diagram was made to illustrate the most important decisions regarding frameworks, tools and services, which can be seen in figure 4.1.

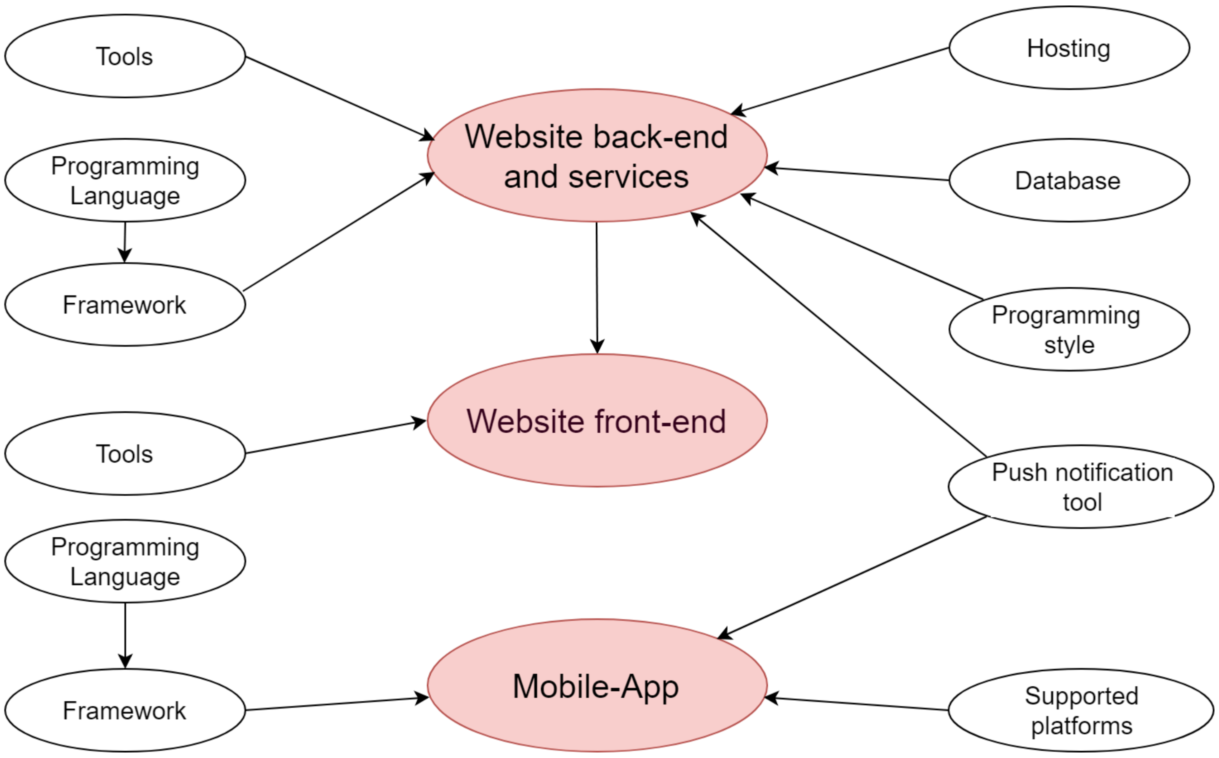


Figure 4.1: Decisions

4.2 Frameworks and programming languages

Finding the best frameworks and programming language is an important decision, and it often comes down to previous experience.

4.2.1 Web solution - Back-end

Considering suggestions and previous experience, there were mainly three frameworks alternatives for web development, all of them capable.

- ASP.NET
- ASP.NET Core
- PHP

The main difference between ASP.NET and ASP.NET Core is that the core version is an open source, cross platform implementation of the .NET framework, which can run on windows, Linux and macOS. ASP.NET can only run on windows[9].

PHP is a popular open source general-scripting language mainly used for developing websites. [10][11].

4.2.2 Mobile App

There are two main smart phone operation systems, which is iOS and Android. There are many framework alternatives for mobile app development, where three of them are listed as alternatives for this thesis.

- Android Studio
- Xcode
- Xamarin

Android Studio is the official IDE for android apps, which supports programming languages like java and C++, where Java is the most used and most supported language. [12]

Apple offers Xcode IDE for iOS app development, and supports both switch and objective-C[13].

Xamarin, the third option, is no official IDE or framework, but instead targets both iOS and Android. I can build native apps supporting multiple platforms with a shared codebase written in C#. [14]

4.3 Database

There are many options for database storage, but based on suggestions and previous experience there were two alternatives, both of them suitable.

Microsoft SQL Server

Microsoft SQL server is a well used database technology made by Microsoft. However, it is not an open source, but Microsoft has recently made it available for Linux OS. [15]

MySql

MySql database is an open source database technology made by Oracle. [16]

4.4 Website front-end tools

4.4.1 Bootstrap

Bootstrap is a HTML, JavaScript and CSS library made by twitter, which comes with predefined design for creating websites. It also scales the content related to screen size, which is an great advantage when browsing the website from a smart phone. Furthermore, Bootstrap allows less time spent on design development, and more time on the website functionality itself.

4.4.2 Charting Library

Typically, charting library for web application uses JavaScript to draw charts. There are many alternatives which might be suitable. It is important that any charting library supports formatting of the date axis in order to get the date correctly scaled.

ChartJs is an open source JavaScript charting library released under the MIT license [17]. It offer different kinds of charts like line, bar, scatter charts etc. [18]

Google Chart, an other option, is a free charting library made by Google. It offer support for line, bar, pie chart etc. [19]

4.5 Back-End Tools and Services

4.5.1 Pdf Generation

There are many options for generating pdf files. Rotativa is one option, which convert HTML and JavaScript to pdf files [20]. There is also an online version of Rotativa which generates pdf files through APIs using HTTP requests, but this requires paid subscription [21].

4.5.2 Push notifications

Push notification library needs to be supported in both the framework for the server and for the mobile-app framework. There are many factors to take into account. Does the notification appear if the the app runs in the background, or does the app need to be open in order to get notifications? How much data will it consume? How much delay could be tolerated?

It should be possible to dynamically create notification services for data sources were users can choose to subscribe on relevant data sources.

Another important concept is the that it must be supported on both Android and iOS. If the Xamarin framework seems to work properly it should work on both platforms. If using Android Studio to develop an Android app, it is important that the library also is supported in an iOS development framework in case of expanding.

Firebase Cloud Messaging - FCM

Firebase Cloud Messaging is a service made by Google which can be used to send out push notifications. It supports both Android and iOS which makes it suitable for cross platform development. [22]

Apple Push Notification Service - APNs

Apple Push Notification Service features push notification to iOS devices. It offers no support for cross platform push notifications. [23]

Azure Notification Hubs

Azure Notification Hubs is an service which allows cross platform notifications to be sent out using just one API call. Azure subscription is required in order to use this service. However, a free Azure subscription include 1 million free push notifications. Using Azure Notification Hubs will give an advantage if developing cross platform apps. [24]

PushSharp

PushCharp is a library that will send push notifications through FMC and APNs. The App need to be registered within the FMC and APNs services in order to work. It is written in C# and can be used with the asp.net framework. This library is released under the Apache License version 2.0 [25] [26]

4.6 Hosting

For hosting there are two alternatives, either rent an external server or host the system on a private server. Both Porsgrunn municipality and USN have servers that might be alternatives.

4.6.1 Private hosting

Both Windows and Linux can host web solutions given that the frameworks support the operation systems.

Regarding ASP.NET Core 2.0 it can be hosted on both Windows and Linux [27] [28].

4.6.2 External hosting

There are many companies that offers rental servers. One alternative is Microsoft Azure which offers both website hosting and functionality for background service.

4.6.3 Cost Estimation

Costs depend on the server usage that the solution might require. There will be different costs regarding private and external hosting. Choosing private hosting might require some investment costs both on hardware and internet access.

Moreover, an cost estimate was made choosing Microsoft Azure as external hosting service [29]. Pay-As-You-Go subscription was chosen using the cheapest alternative [30]. Table 4.1 shows the cost estimate.

Azure hosting cost(NOK)	Year 1	Year 2	Year 3	Year 4	Year 5	Total
Azure functions	0	0	0	0	0	0
Web App	929	929	929	929	929	4643
SQL database	1437	1437	1437	1437	1437	7183
Custom domain	97	97	97	97	97	486
Total	2462	2462	2462	2462	2462	**12312**

Table 4.1: Cost estimate using Azure Pay-As-You-Go subscription

The cost estimate is clearly manageable, costing around 2500 NOK each year. Furthermore, if the solution needs more resources, it is possible to scale up the resources available by choosing a different subscription. It will increase the costs, but comes as a result of increased usage, which could defend spending more money.

4.7 Scheduling

Scheduling is important as the solution needs some kind of services running in the background. The services must be able to perform tasks at known intervals. Regarding private hosting, both Windows and Linux are able to run services in the background given that the framework is supported.

If choosing an external hosting service it is important that background tasks can be performed and run the desired code.

4.8 Maintenance

Estimating the amount of hours that might be used on maintenance and improvement may be difficult. Unknown bugs might occur, or most likely occur, after commissioning. Changes in external data that the system depends on may also lead to maintenance need. An inaccurate estimation was made taking into account both maintenance and the need of some minor improvements. The estimate can be seen in table 4.2.

Hour usage	Year 1	Year 2	Year 3	Year 4	Year 5	Total
Maintenance	60	30	30	30	30	180
Improvement	50	0	0	0	0	50
						230

Table 4.2: Maintenance - hour usage

4.9 ASP.NET MVC - Theory and Example

MVC is an architecture that separates an application into three components, which is model, view and controller. The model handles the logic for the applications, the view handles the UI and the controller handles user interactions. Requests will be handled by a routing component which will direct the request to the desired controller and action. A great advantage of the architecture is the loose coupling between the three components. Due to this, it is possible to develop each component more or less separately. Moreover, this also makes testing of the application easier. [31]

Figure 4.2 shows the flow in an MVC application. When a browser requests a page, the routing component will direct the traffic to the correct controller, which then will return a view containing HTML/JavaScript code and model data.

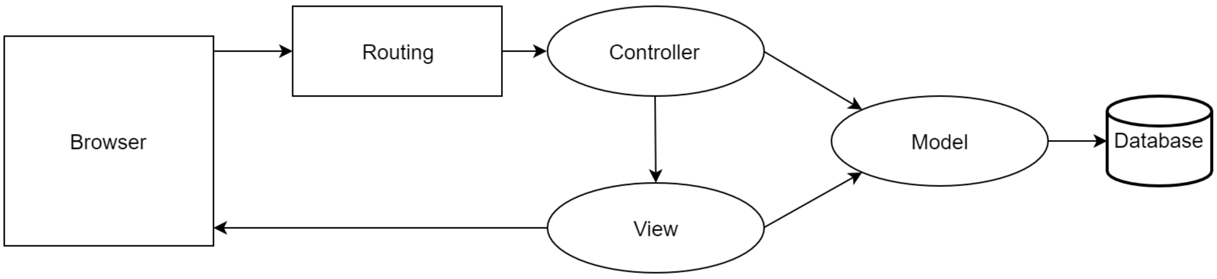


Figure 4.2: MVC block diagram

4.9.1 Example

This section covers a basic MVC example. The web page will return a view populated with some simple user data. Figure 4.3 shows the routing class which will route the traffic to desired controller. It also defines the default controller if no one is selected, which in this case is the home controller.

The example was created as an empty MVC project with the ASP.NET framework and uses bootstrap as styling.

```
public class RouteConfig
  {
      public static void RegisterRoutes(RouteCollection routes)
      {
          routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

          routes.MapRoute(
              name: "Default",
              url: "{controller}/{action}/{id}",
              defaults: new { controller = "Home", action = "Index", id =
                  UrlParameter.Optional }
          );
      }
  }
```

Figure 4.3: Routing

Figure 4.4 shows the model which will generate names and age using a for loop in the constructor. If the names were to be fetched from a database it should happen in the model. However, they could also be fetched in the controller, but that may reduce the readability of the code.

```
public class User
{
    public string name { get; set; }
    public string age { get; set; }
    public User(string name_, string age_)
    {
        name = name_;
        age = age_;
    }
}

public class Users
{
    public List<User> users = new List<User>();
    Users()
    {
        for (int i = 0; i < 5; i++)
        {
            User user = new User("Name" + i.ToString(), i.ToString());
            users.Add(user);
        }
    }
}
```

Figure 4.4: Model

The view is shown in Figure 4.5. The razor engine, which enables C# inside the view, will populate a table containing all the user names and age. No server code is executed on the client. The client will only receive the generated HTML code.

```
@model reportExample.Models.Users
<table class="table table-hover table-condensed col-md-12">
    <thead>
        <tr>
            <th>Name</th>
            <th>Age</th>
        </tr>
    </thead>
    @foreach (var item in Model.users)
    {
        <tr>
            <td>@item.name</td>
            <td>@item.age</td>
        </tr>
    }
</table>
```

Figure 4.5: View

Figure 4.6 shows the home controller. In this example, it will create a new model of the type Users and return it with the view.

```
 public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            Users users = new Users();
            return View(users);
        }
    }
```

Figure 4.6: Controller

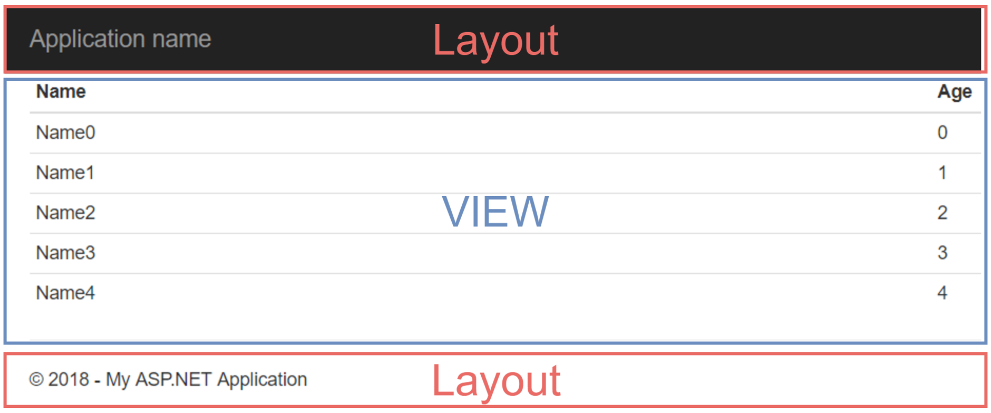
The layout of the page is placed in a separate file. When creating a view it is possible to set the layout instead of write the whole layout into the view. The layout is shown in figure 4.7 and contains the code line ”@RenderBody()”, which is where the view will be inserted. For readability most of the HTML code drawing the layout is removed.

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
        @RenderBody()

</body>
</html>
```

Figure 4.7: Layout

Figure 4.8 shows the result when running the application. The view and layout are marked in the figure.



Name	Age
Name0	0
Name1	1
Name2	2
Name3	3
Name4	4

Figure 4.8: Layout

5 Development: Methods and Design

This chapter will give the reader an overview of working methods to achieve the desired functionality given in chapter 3. Framework, tools and services will be chosen based on the analysis in chapter 4.

5.1 Working Methods

There are many approaches for developing a piece of software. Unified process is one of them, which focuses more on designing the software before it is written. By collecting functional requirements based on software specifications and requirements, it is possible to divide the software into pieces. These functional requirements will then be iterated through, where the software mostly will be designed before it is written. However, this requires more experience and knowledge about how to write the software in advance. Instead, RAD was chosen as working method as it does not focus as much on pre-design of the software, but more on fast prototype development. Then the customer can give feedback on the prototype, which then will be used for further development. The software prototype will be improved rapidly by iterating through the stages from design to prototype. [32]

5.2 Frameworks, tools and services

Frameworks, tools and services will be chosen based on the analyses in chapter 4. Open source, cross platform and previous experience laid the basis of the decisions.

5.2.1 Database

Microsoft SQL Server was chosen as database for the project due to previous experience. As Microsoft SQL Server now supports Linux in addition to Windows, the database can be hosted on a Linux server [15].

5.2.2 Website

It was decided to use ASP.NET Core 2.0 MVC for website development due to previous experience programming in C#. MVC architecture itself also clearly separates code logic, which makes it more suitable for readability and maintenance. Visual Studio 2017 was chosen as IDE. Libraries are installed using the NuGet Package Manager [33].

The front-end tools in the context of web site means HTML, JavaScript and CSS libraries. Bootstrap was decided to be used for web site design as it is widely used and intuitive [34].

Chart.js was chosen as charting library for the web site as it seems intuitive and elegant in use [18]. Moreover for generating reports it was decided to use Google Charts for plotting [19].

5.2.3 Mobile App

Due to work load and access to hardware it was decided to only implement an app for Android. Xamarin.Android was chosen as framework using Visual Studio 2017 as IDE [35].

5.2.4 Services

The services should use the open source and cross platform .NET Core implementation of the .NET framework [36]. Then it is possible to target both Windows and Linux as Server.

FCM was chosen as service for sending push notifications. [37].

5.2.5 Hosting

Azure Cloud Hosting was chosen to host the solution for development and testing [29]. However, cross platform is emphasized, so it should be possible to host on different platforms.

5.3 Initial Software Design

Some general use cases and sketches was initially made to act as base for the software design. As mentioned in section 5.1 RAP was chosen as working methods for software development, and the software will therefore be continuously improved as the customer gives feedback.

Figure 5.1 shows a detailed overview of how the different parts is planned to interact with each others. Interactions between the server and the mobile app should happen using the HTTP protocol.

Some early sketches were made to give a brief indication on how the website and mobile app might look like. These can be seen in Appendix B. The use cases can be seen in Appendix C

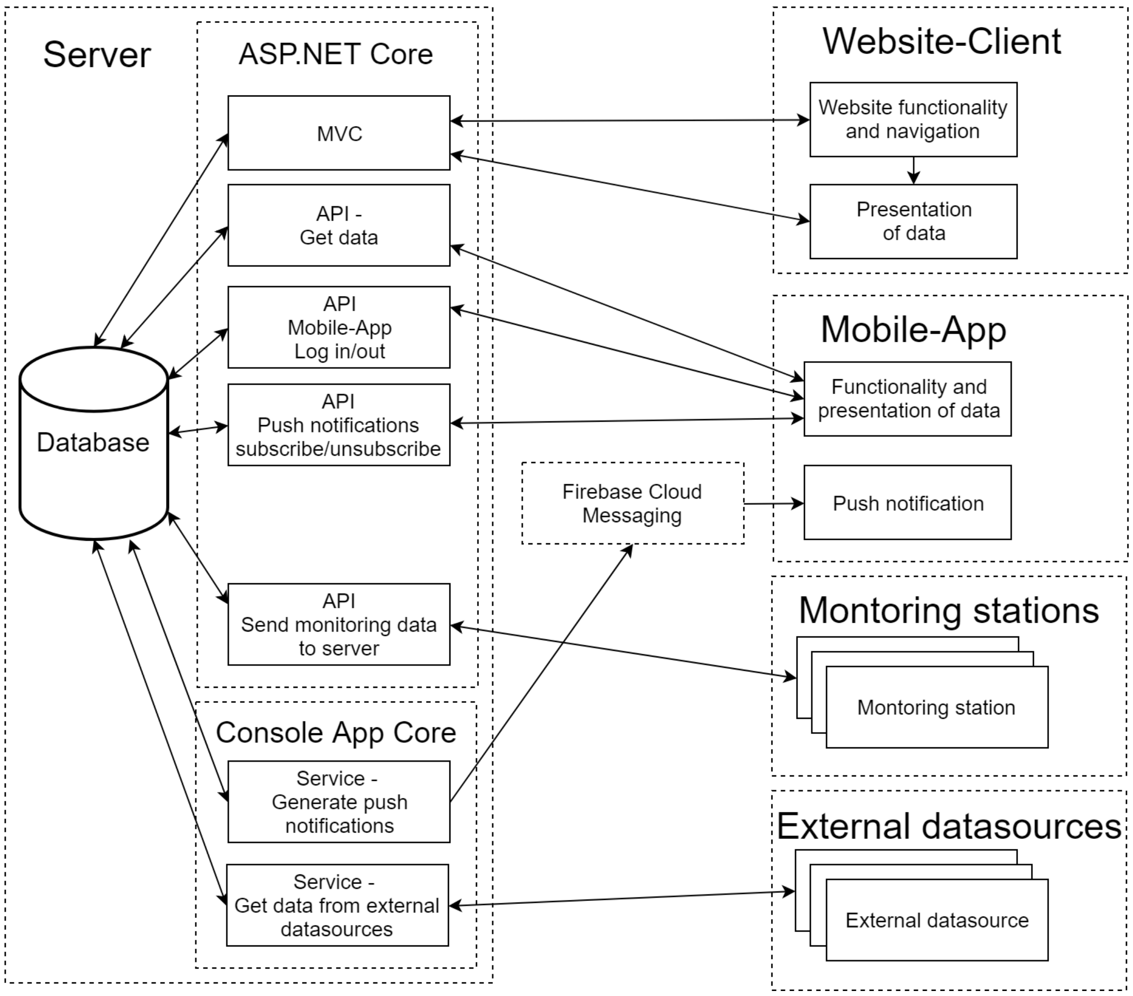

Server
ASP.NET Core
MVC
API - Get data
API Mobile-App Log in/out
API Push notifications subscribe/unsubscribe
API Send monitoring data to server
Database
Console App Core
Service - Generate push notifications
Service - Get data from external datasources
Firebase Cloud Messaging
Website-Client
Website functionality and navigation
Presentation of data
Mobile-App
Functionality and presentation of data
Push notification
Montoring stations
Montoring station
External datasources
External datasource

Figure 5.1: System overview

6 Database - Implementation

This chapter will cover design and implementation of the database.

6.1 Design

The database design was made using erwin Data Modeler [38]. The implementation can be seen in figure 6.1. The tables can roughly be divided into five groups with different responsibilities:

- User information
- Source and parameter information
- Status information
- Push notification configurations
- API configurations

The table USERS holds information about the users like for example email addresses, user names and passwords. The password is hashed with a random salt value. In order to authenticate users, the salt value needs to be stored in the database, which is done in the column UserGuid. Furthermore, any user configuration is related to the USERS table through the table USERCONFIG. Also, in order to limit or grant user access to different functionalities, the USERS table is related to a table called USERROLES, which relates roles to an specific user by the roles stored in the table ROLE.

The actual storage of any sources within the system happens in the table DATASOURCE. It stores information like names, descriptions, areas, positions etc. In order to give a source a location inside a geographical area, the tables AREALEVEL1 and AREALEVEL2 were made. For example Grenland area will be stored in AREALEVEL1 and Porsgrunn in AREALEVEL2. This will make the locations of the sources easier to understand. A source can have many parameters, which is stored in the PARAMETER table. All the measurements related to the parameters are stored in the table MEASUREMENTS. In addition the table MEASUREMENTSBUFFER was made to allow increased loading speed. PARAMETERTYPE indicates the type of the parameter, for example NO2 with corresponding description.

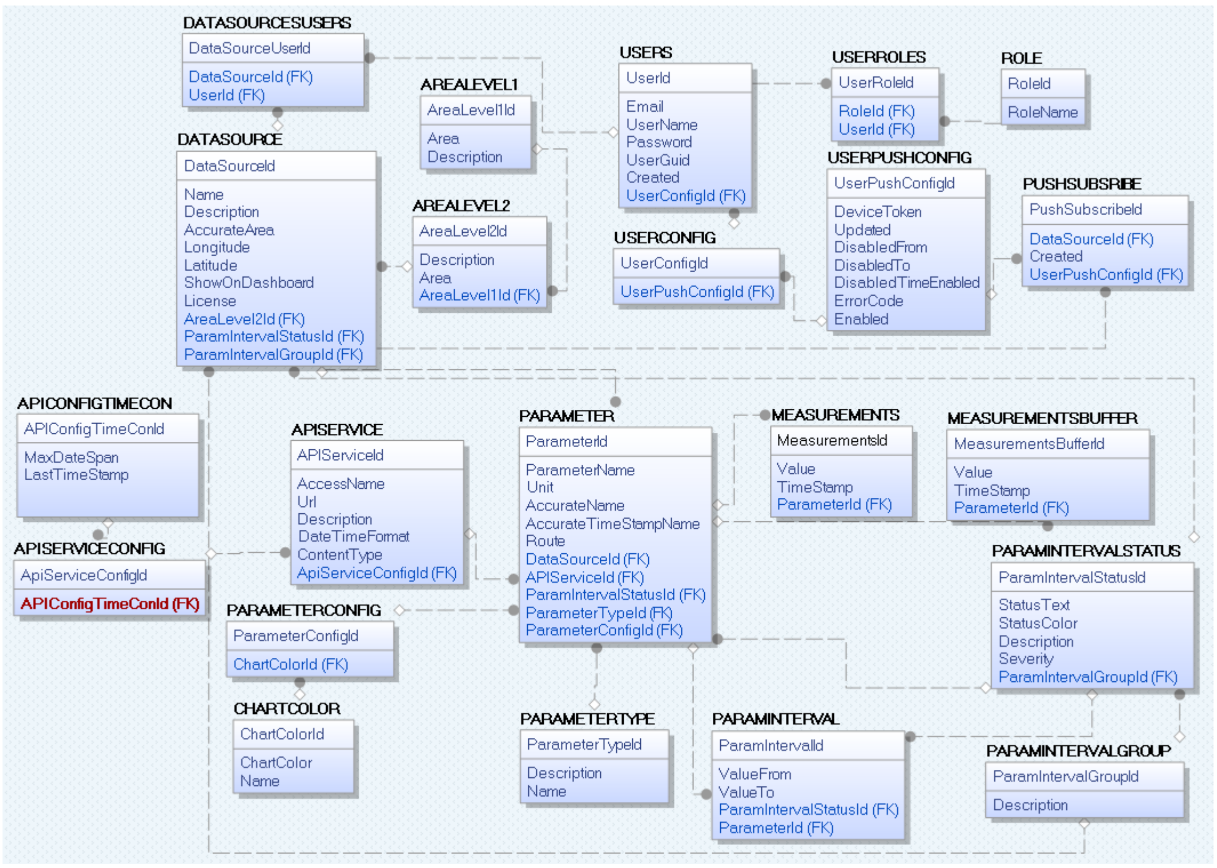


Figure 6.1: Database - design

Each source and parameter has a corresponding status containing information about the health of the last measurement, which is stored in the table PARAMINTERVAL-STATUS. For example, a status can indicate medium air pollution by text and color. The PARAMINTERVALSTATUS must be grouped together, which is done by the table PARAMINTERVALGROUP. For instance, all statuses about NO2 should be grouped together. The actual status limit are stored in the table PARAMINTERVAL, and each parameter has their own status limit regardless of the parameter type. This might lead to redundant information in the database and could be solved differently. The idea behind is that parameters of the same type do not necessarily have the same status limits. Describing number of cars passing on a stretch of road, for instance, might be different depending on the car volume the road is designed for.

To be able to implement push notifications, the database needs to store the device token which is related to a user. Through the table USERCONFIG, users are related to the USERPUSHCONFIG table, which holds configurations regarding push notifications. The actual subscriptions happen in the PUSHSUBSCRIBE table, where it will bind the subscription to a source.

The table APISERVICE stores any information regarding collecting data from external APIs. APICONFIGTIMECON is a configuration table which enables the API queries to be updated with dynamically changing date time fields.

6.2 Entity Framework

Entity Framework was chosen in order to work with the database using .NET objects. [39]

6.2.1 LINQ

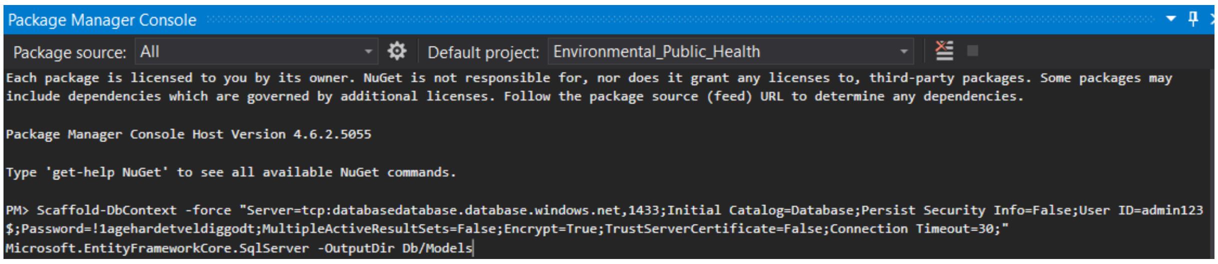
LINQ was used to query the database. One of the advantage with LINQ is that most the code is written in one solution, which makes it easier to perform maintenance. Figure 6.2 shows an example were the database is asked to return all sources and associated parameters.

```
using (var db = new DbContext())
            {
                var qryResult = (from a1 in db.Arealevel1
                                 join a2 in db.Arealevel2
                                 on a1.AreaLevel1Id equals a2.AreaLevel1Id
                                 join d in db.Datasource
                                 on a2.AreaLevel2Id equals d.AreaLevel2Id
                                 select new
                                 {
                                     areaLevel1 = a1.Area,
                                     areaLevel2 = a2.Area,
                                     area = d.AccurateArea,
                                     d.Name,
                                     d.DataSourceId,
                                     parameters = (from p in db.Parameter
                                                   where p.DataSourceId ==
                                                         d.DataSourceId
                                                    select new
                                                    {
                                                        p.ParameterId,
                                                        p.ParameterName
                                                    })
                                     });
                }
```

Figure 6.2: Example - query the database using LINQ

6.2.2 Generating C# Models from Database

All tables in the database must be known by the entity framework in order to use LINQ. This is done by creating classes of all the tables, which could be done manually or automatically using a script. Figure 6.3 shows how this is done by running a script using the Nuget Package Managar Console.



```
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may
include dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 4.6.2.5055

Type 'get-help NuGet' to see all available NuGet commands.

PM> Scaffold-DbContext -force "Server=tcp:databasedatabase.database.windows.net,1433;Initial Catalog=Database;Persist Security Info=False;User ID=admin123
$;Password=!1agehardetveldiggodt;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Db/Models
```

Figure 6.3: Generating C# Models from the database

6.3 Views

Four views are created in order to make the LINQ queries easier to perform. For example, one view will group together data from the tables DATASOURCE, AREALEVEL1 and AREALEVEL2.

7 Website - Implementation

This chapter will cover topics regarding design of software both on server and the website. It will not cover the code itself in depth, but give a overview of how the system works, like important functions calls and interaction between server and clients.

As explained in chapter 4, the website front-end and back-end use the ASP.NET MVC Core 2.0 framework. These are therefore both explained in this chapter. The mobile app also depends on data from the server, but will be explained more in detail in chapter 8. Only the communication between server and mobile app will be covered in this chapter as it doesn't require detailed knowledge about how the mobile app is implemented.

7.1 Server side - code patterns

Code design is important, both for readability and maintenance. Therefore the code was divided into different sections with different responsibilities.

- **Controller**
  The controller's responsibilities is mainly to read and send data between the system and the users. No logic should be handled in the controller.
- **Model logic**
  The model will handle all the business logic and containing class definitions of all the data in the system. If the UI somehow were to be changed it is possible to reuse the model logic.
- **Database logic**
  This section will only handle communication back and forth between the database and the program. The model logic will not directly query the database for data, but instead ask the database logic, which then will fetch the data. If the database were to be replaced the model logic also can be reused.

These sections can be seen in figure 7.1 indicating the data flow on the server. In terms of MVC, it can be said that the model logic and the database logic belongs to M - Model.

MVC return views containing HTML and JavaScript code populated with model data. However, it is possible to only return the model data without views which makes it possible to reuse code for both the website and the mobile app. This was in focus when developing the code.

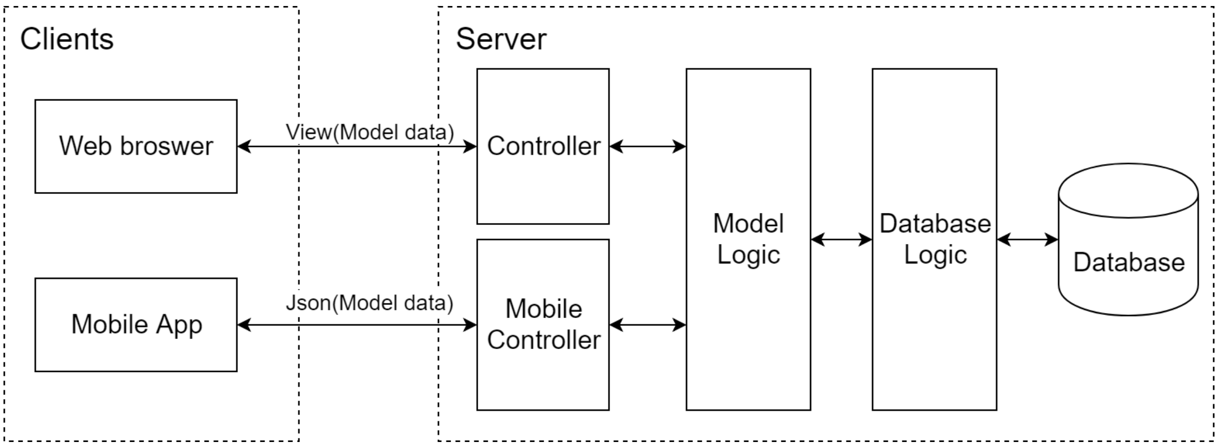


Figure 7.1: Data flow between server and clients

7.2 Interactions between server and clients

The controllers, which interact with the clients, are divided into two groups, one for the website and one for the mobile app. The main difference is that the controllers for the website return views containing model data, while the controllers for the mobile app only return model data as JSON. The controllers have methods called actions, and the routing mechanism is set up with controller names and action names. Figure 7.2 was made to illustrate what happens when typing addresses in the web browser. Listing 7.3 shows the action method which is called that will return a view containing model data. The concept of separating controller and model logic can clearly be seen as the action will get model data from the model logic.

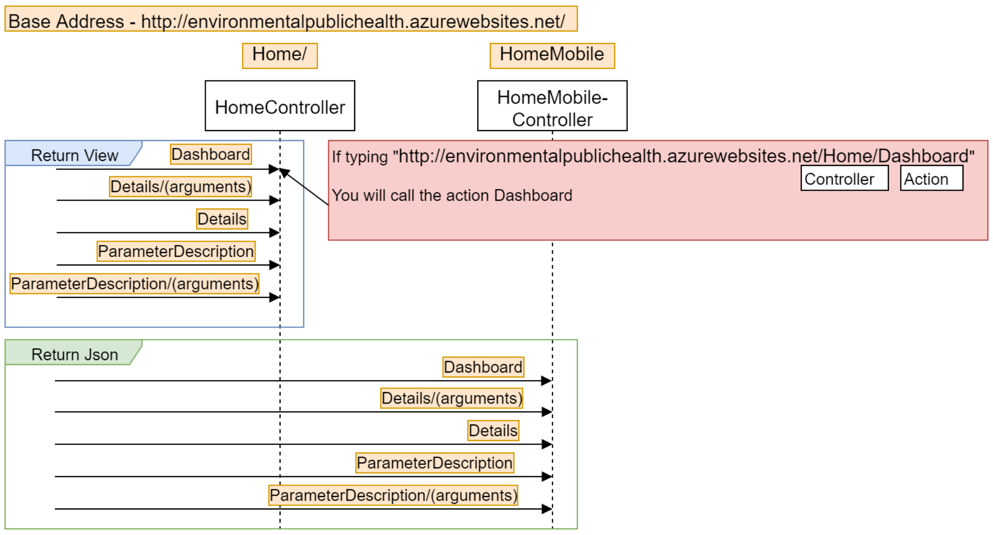

Base Address - http://environmentalpublichealth.azurewebsites.net/
Home/
HomeMobile
HomeController
HomeMobile-Controller
Return View
Dashboard
Details/(arguments)
Details
ParameterDescription
ParameterDescription/(arguments)
If typing "http://environmentalpublichealth.azurewebsites.net/Home/Dashboard"
Controller
Action
You will call the action Dashboard
Return Json
Dashboard
Details/(arguments)
Details
ParameterDescription
ParameterDescription/(arguments)

Figure 7.2: Controller and addressing

```
[AllowAnonymous]
public class HomeController : Controller
{
    [HttpGet]
    public ActionResult Dashboard()
    {
        DashboardContainerView modelData = new DashboardContainerView()
            ;
        modelData.Dashboard =  HomeLogic.GetDashBoard(true,
            ConstantConfigurations.GetchartJsStringFormat());
        if (modelData.Dashboard.error.error)
        {
            // Handle error
        }
        return View(modelData);
    }
}
```

Figure 7.3: Dashboard Actions - Website

Listing 7.4 show how the dashboard data is returned as JSON data instead of view containing data.

Figure 7.5 was made to show the data flow between the different code layers both for the website and the mobile app. The controllers will ask the model logic data, which then again will ask the database logic for data stored in the database.

```
[Route("api/[controller]/[action]")]
    public class HomeMobileController : Controller
    {
        [HttpGet]
        public JsonResult Dashboard()
        {
            DashboardContainerViewMobile modelData = new
                DashboardContainerViewMobile();
            modelData.Dashboard = HomeLogic.GetDashBoard(false,
                ConstantConfigurations.GetAndroidChartStringFormat());
            if (modelData.Dashboard.error.error)
            {
                // Handle error
            }
            return Json(modelData);
        }
    }
```

Figure 7.4: Dashboard Action - Mobile App

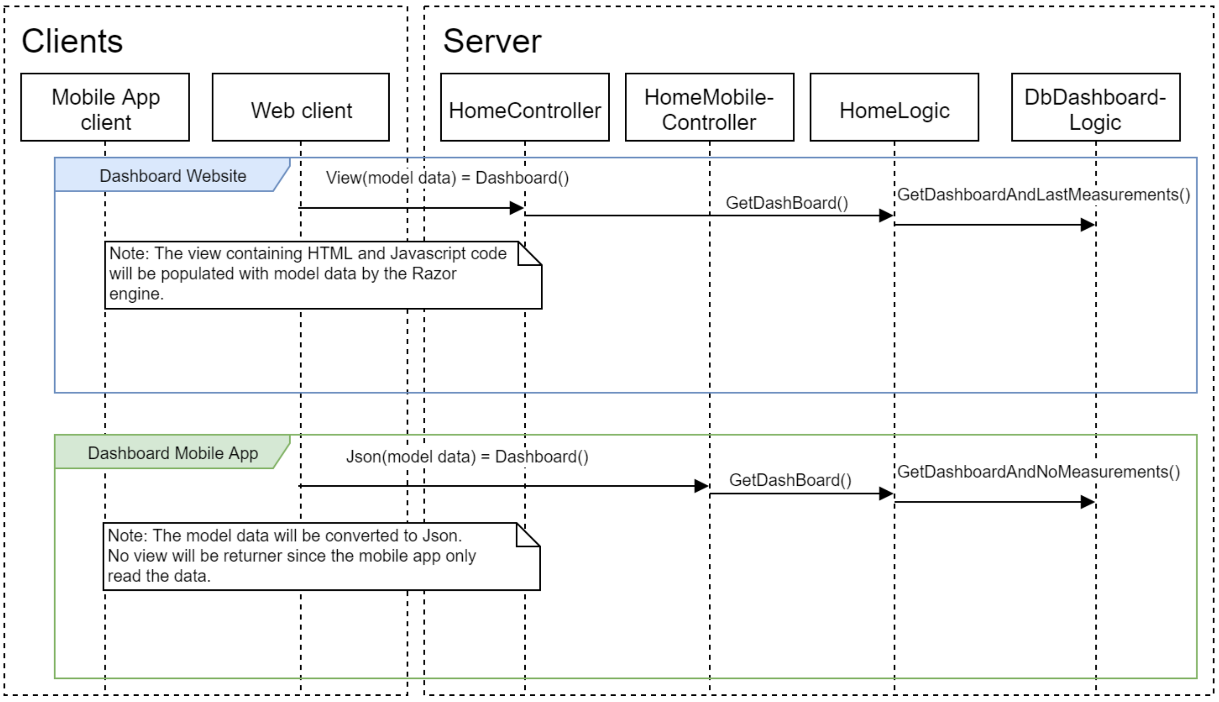

Clients
Server
Mobile App client
Web client
HomeController
HomeMobile-Controller
HomeLogic
DbDashboard-Logic
Dashboard Website
View(model data) = Dashboard()
GetDashBoard()
GetDashboardAndLastMeasurements()
Note: The view containing HTML and Javascript code will be populated with model data by the Razor engine.
Dashboard Mobile App
Json(model data) = Dashboard()
GetDashBoard()
GetDashboardAndNoMeasurements()
Note: The model data will be converted to Json. No view will be returner since the mobile app only read the data.

Figure 7.5: Data flow with function names

7.2.1 Middleware

A middleware is used to handle requests and responses [40]. Default routing and authorization check is handled in the middleware. Listing 7.6 shows how the default routing is added, where the main page is the dashboard page.

```

app.UseMvc(routes =>
    {
        routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Dashboard}/{id?}");
            });
```

Figure 7.6: Middleware default routing

Authorization information is saved in cookies on the client. However, this not practical when authorizing requests from mobile apps. Therefore JWT bearer authentication and authorization was implemented for requests from the mobile app[41]. This means the JWT token must be sent with each request. To be able to use two authorization methods, a custom authorize filter was made[42]. Figure 7.7 shows the custom filter. If the controller name contain "Mobile", JTW bearer authorization will be used. The default is cookie based authorization.

```
public class AddAuthorizeFiltersControllerConvention :
    IControllerModelConvention
    {
        public void Apply(ControllerModel controller)
        {
            if (controller.ControllerName.Contains("Mobile"))
            {
                controller.Filters.Add(new AuthorizeFilter("apipolicy"));
            }
            else
            {
                controller.Filters.Add(new AuthorizeFilter("defaultpolicy")
                    );
            }
        }
    }
```

Figure 7.7: JWT Bearer

7.3 Website

This section will mostly focus on how the website is built based on model data, HTML and JavaScript. Bootstrap is used for designing the layout of the website. There are four main pages containing the most important functionalities.

- Dashboard.
- Trending.
- Analytic.
- Report Generation.

7.3.1 Dashboard

The dashboard contains multiple plots for each source. The razor engine is used to generate HTML and JavaScript code for each plot, which happens inside a loop iterating through the model containing all sources and associated parameters. The razor engine will give all HTML elements an id which is used by the corresponding JavaScript code. This is done by incrementing an integer for each iteration. This integer is used at the end of the id tags for generating both HTML elements and JavaScript variables. When generating the table razor engine will iterate through the sources and add them as rows. This tables contains all the parameters within the sources. These parameters are clickable in order show the parameter description inside a pop up, The parameter deceptions are not initially loaded, but loaded on click using Ajax requests [43].

7.3.2 Trending

In difference to the dashboard page, the trending page contain one large plot, which is by default loaded with the last day historical data from the selected source. A selector is used to chose time period for the historical data. A drop down menu allow the source to be changed. When changing the selected source, a get request to the controller will be sent including source id and selected time period, which allow the selected time period to be saved between the requests. The table containing source details will be populated by the razor engine iterating through the associated parameters included in the model.

7.3.3 Analytic

The analytic web page is based on JavaScript. Initially, the razor engine populates the selector containing all the sources, while the associated parameters are loaded into a

JavaScript array. After the view is generated, JavaScript will populate the selector containing the parameters. This also happens when the selected source changes. Two date time selectors are used in order to select a time span [44]. A add button is used to add parameters and time span to a JavaScript array. In addition, for each added parameter, JavaScript will add a row to a table containing associated information. The table also include a remove button, which would remove the parameter both from the table and from the array. Furthermore, Ajax is used to load measurements for the selected parameters. This is done by converting the array to a JSON string and add it as payload to the Ajax request. The JSON string is then decoded on the server before it will query the database for the selected measurements and return the data as JSON, which then is used to populate the plot.

A second analytic page is used to enable side by side plotting. Most of the same implementations are used as shown above beside the date time picker. It is possible to choose week, month or year as time period, were JavaScript will dynamically update the date time pickers to allow valid date formats. jQuery UI is used as date time picker [45]. A plugin for allowing week selections was also used [46]. The date format is indicated as a number in the array which is converted to JSON when sending the request to the server. The date time format on the returned measurements are moved to fit into one specific week, month or year depending on the selected time span option.

7.3.4 Report Generation

The report generation web page use some of the same implementation as the analytic web page regarding JavaScript and setup. However, it is only possible to select sources, and years as time periods. The report is generated using HTML, which means that Ajax cant be used. Instead, the JavaScript array which contain the source and date selection will be posted using an ordinary post submitting. This is done by converting the array to a JSON string which is then inserted into a hidden HTML input field. The input field is then submitted to the controller, which then decode the string and query the database. All measurements within a month is then scaled down into one average measurement in order to get a readable plot. Google chart is used for plotting [19] Min, max and average values are also calculated.

8 Mobile App - Implementation

This chapter will cover the implementation of the mobile app. It was decided to only target the Android platform, due to the work load to implement apps in both iOS and Android. Xamarin.Android was chosen as framework [35]. Visual Studio 2017 was used as IDE.

8.1 Overview

Android uses AXML for drawing user interface. So called activities contain all the C# code, which can utilize the AXML layout. The app is built up using three different activities.

- Log in
- Register User
- Main activity

The main activity contains all the functionality for accessing data from the server. When logged in, the users can navigate through different sub pages under the main activity containing the different functionalities. These sub pages are implemented as fragments, which contain relevant code functionality. The different fragments are linked to corresponding AXML layouts. As the user navigates to another fragment the main activity will replace the current fragment with the new fragment.

A simple logic overview can be seen in figure 8.1 for how the mobile app works.

8.2 Communications with Server

Communication with the server happens using the HTTP protocol. As mentioned in chapter 7, the data flowing between the server and app uses JSON formatted strings. The NuGet package "Newtonsoft.Json" is used to serialize and deserialize JSON formatted data. The deserialize method will format the received JSON string into an object. In order to do this a class definition of the target object must be defined. As both the server

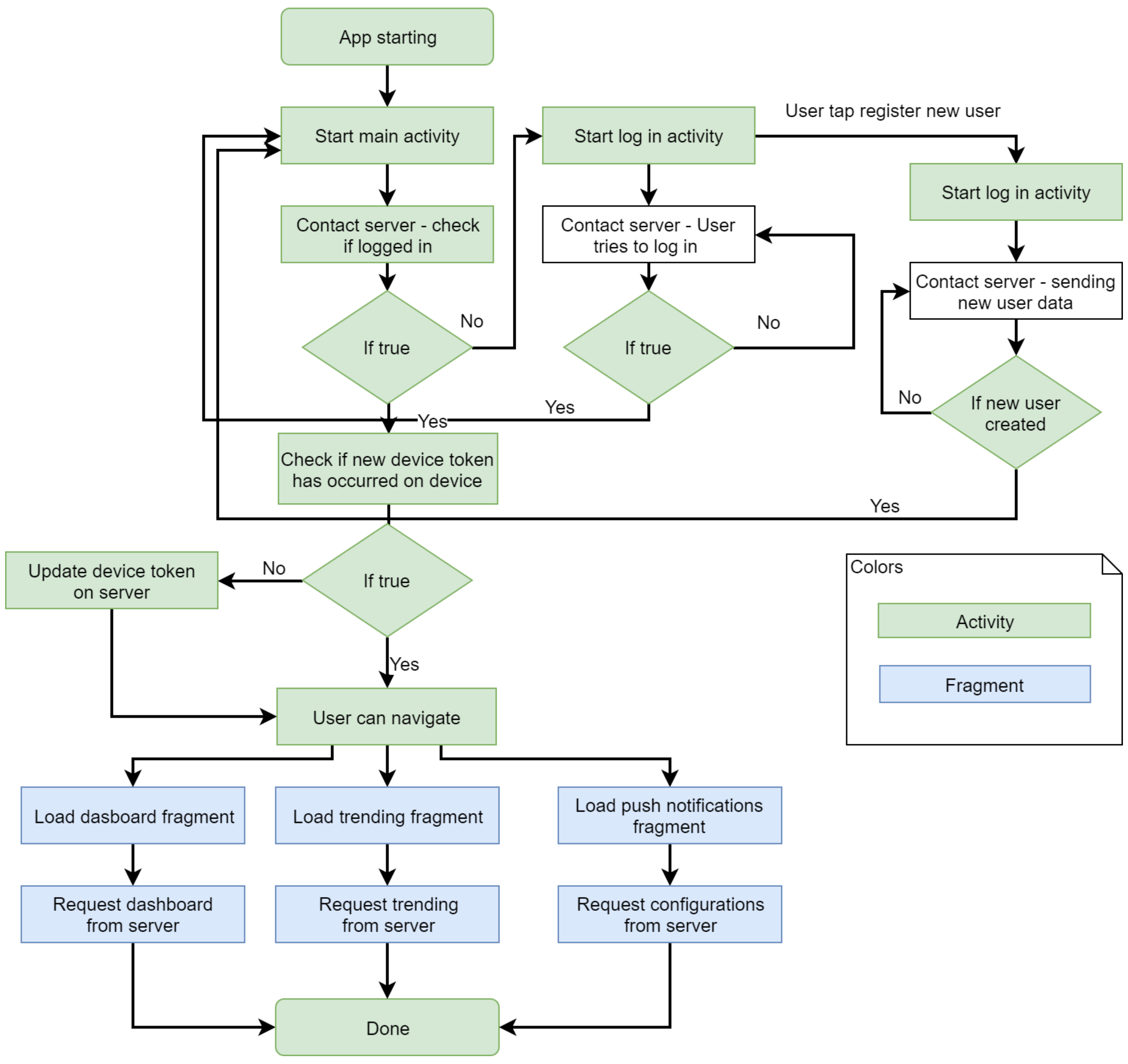

App starting
Start main activity
Contact server - check if logged in
If true
No
Yes
Start log in activity
Contact server - User tries to log in
If true
No
Yes
User tap register new user
Start log in activity
Contact server - sending new user data
If new user created
No
Yes
Check if new device token has occurred on device
If true
No
Yes
Update device token on server
User can navigate
Load dasboard fragment
Load trending fragment
Load push notifications fragment
Request dashboard from server
Request trending from server
Request configurations from server
Done
Colors
Activity
Fragment

Figure 8.1: Android App - Logic Overview

and mobile app are programmed in C#, it's possible to just copy the class definitions from the server source code into the mobile app source code.

The NuGet package "System.Net.Http" are used to send data by the HTTP protocol.

8.2.1 Authorization and Authentication

As mentioned in chapter 7, the mobile app is using JWT Token bearer as authentication and authorization. When the user is authenticated, the server will create an encrypted

JWT token, which then will be saved locally on the mobile app. This token needs to be sent on each request to the server in order to get access to the data.

8.2.2 Push Notifications

Push notification is implemented by using the NuGets ”Xamarin.Firebase.Messaging” and ”Xamarin.GooglePlayServices.Base”. The mobile app requires Google Play Service to be installed in order to receive push notifications. Each Android device has an unique token for each app, and is given by the server managing the FCM service. This unique token must be forwarded to the server as it is needed for sending push notifications. The token is updated at a regular basis, so a method for detecting changes was implemented. Each time the app is forwarding a new token to the server, a local copy will also be saved on the device. When the app starts, or the user logs in, it will ask the FCM service for the current token. If this token differs from the local copy, it will forward the new token to the server. In this way the server always has an updated token.

8.3 User Interface and functionality

The user interface uses a library called Android Support Libraries, which is used to draw the navigation menu and toolbar [47]. The mobile app has five pages containing all its functionality.

- **Dashboard**
  Will request dashboard data from the server. As a mobile screen is small it will not load any historical data, only the last measurements, time stamp and status.
- **Trending**
  This page will load more details about the chosen source, like status intervals, license data and information about the source. Like the website it will load historical data in order to get an understanding of trending.
- **Notifications**
  Will load configuration from the server regarding push notifications. New configurations will be sent to the server at once when the user make any changes.
- **Log in**
  Provides the log in functionalities. When the user is authenticated, the server will return a new JWT Bearer token which will be saved locally on the device. The use can navigate to the register new user page.
- **Register new user**
  The user must enter user name, email and password, where the password must be confirmed twice to be able to create a new user. Only valid information is accepted,

like valid email and valid password. The password must contain 8 chars, minimum one number and one upper case character. This will be checked on the server.

Dedicated classes are taking care of the communication with the server, which increases readability and maintenance.

8.3.1 Plotting

OxyPlot is used for plotting, which is a cross platform library released under to MIT license [17] [48].

9 Services - Implementation

This chapter will cover implementation of all the services running in the background at fixed time intervals. They are not tied directly to the website or the mobile app, but rather read and write to the database only. It was decided to use Azure Function since they can target the open source .NET Core. [49].

9.1 Overview

There are three tasks that need to be performed as services.

- **Fetch External Data**
  Will get data from external APIs and copy them into the database.
- **Status Updates and Push Notifications**
  Will update statuses for all the source and parameters in the database according to the predefined limits in the database. If status changes has occurred, push notifications will be sent out to all subscribing users.
- **Update Buffer Table**
  The buffer table will only store measurements from the last month. The service will put any new measurements into the buffer table, and deleting old ones.

An overview can be seen in figure 9.1, which also indicates how the services run when Azure Function is triggered. The individual services will be explained more in detail in the sections below. The business logic and database logic was separated in order to increase readability and maintenance.

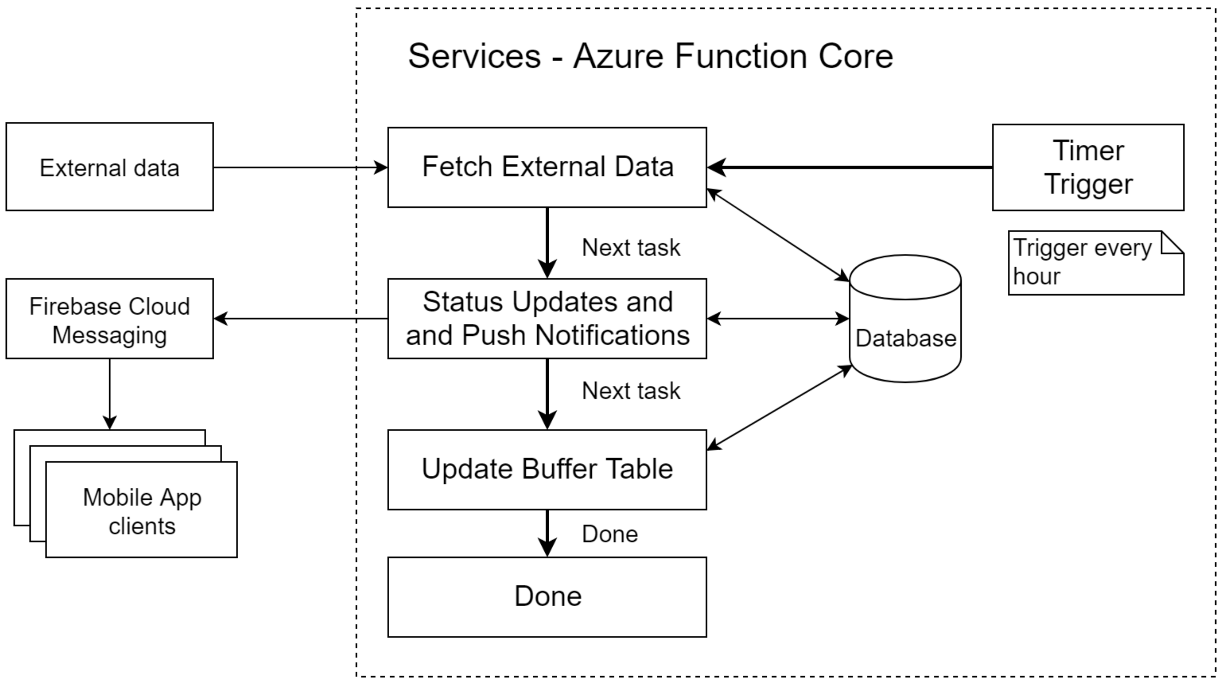


Figure 9.1: Services Overview

All the functionality is written as C# classes which makes it more suitable to use them in multiple .NET applications. They were first written in a .NET CORE console application for local development and testing. Figure 9.2 shows how the Azure Function was implemented and use the C# classes. ”0 20 * * * *” means how often the function will trigger, which is called a CRON expression [50]. In this case the timer will trigger 20 minutes past every hour due to that NULI's API will be updated 10-20 minutes past every hour.

```
public static class ServiceCollectExternalData
{
    [FunctionName("ServiceCollectExternalData")]
    public static void Run([TimerTrigger("0 20 * * * *")]TimerInfo
        myTimer, TraceWriter log)
    {
        log.Info(\$"C# Timer trigger function executed at: {DateTime.
            Now}");
        ApiExternalDataHandler ApiWorker = new ApiExternalDataHandler()
            ;
        ApiWorker.Run();
        UpdateStatusHandler statusWorker = new UpdateStatusHandler();
        statusWorker.Run();
        DataBufferHandler dataBufferWorker = new DataBufferHandler();
        dataBufferWorker.Run();
    }
}
```

Figure 9.2: Azure Function

9.2 Fetch External Data

The database was designed to store all the information used to request the external APIs for data. This means that no information is hard coded, which makes it possible to dynamically add API calls for relevant sources.

Implementation of the database design and code was inspired by how NILU's API works, which can return the data formatted as JSON strings [4][51]. According to NULI API documentation it has two relevant methods for returning data.

- **Up to date data**
  Return data within the three last hours.
- **Historical data**
  Return historical data from within a time interval which need to be defined in the query.

It is not possible to retrieve historical data when querying the up to date method. However, data after commissioning will of course be saved. Requesting the API every hour, which returns data from within the last three hours, means that some of the data may be redundant. To make sure that no redundant data is stored into the database, no data older then newest data point will be saved.

The API can return data from multiple stations at once. This means only one query is needed to request all the data within the three latest hours. The table APISERVICE in the database is used to hold configuration used for querying.

Furthermore, the API for returning historical data can also return the newest data. Due to this fact, it is possible to use this API method for collecting both historical and newest data. However, only one parameter can be selected, which means that each parameter needs a dedicated query. Therefore, each parameter needs one row in the APISERVICE table. As the API needs a time interval for querying, the configuration table APICONFIGTIMECON is used to hold information about the time interval. It has two columns, where one holds the time for the last relevant record in the database. The historical API has a 6 month limitation for the time interval in the query, so the other column is used to configure this length. When building the query the last recorded time stamp will be used as start date, and the column holding the interval length will be added to the start date and set as end date. When using this configuration the query string saved in the APISERVICE must contain "{Config.dateStart}" and "{Config.dateEnd}".

Figure 9.3 shows an overview of the logic for fetching data from external sources. First all configurations will be loaded from the database, and then used to query the API. Section 9.2.1 explains more about how the code is accessing the relevant data.

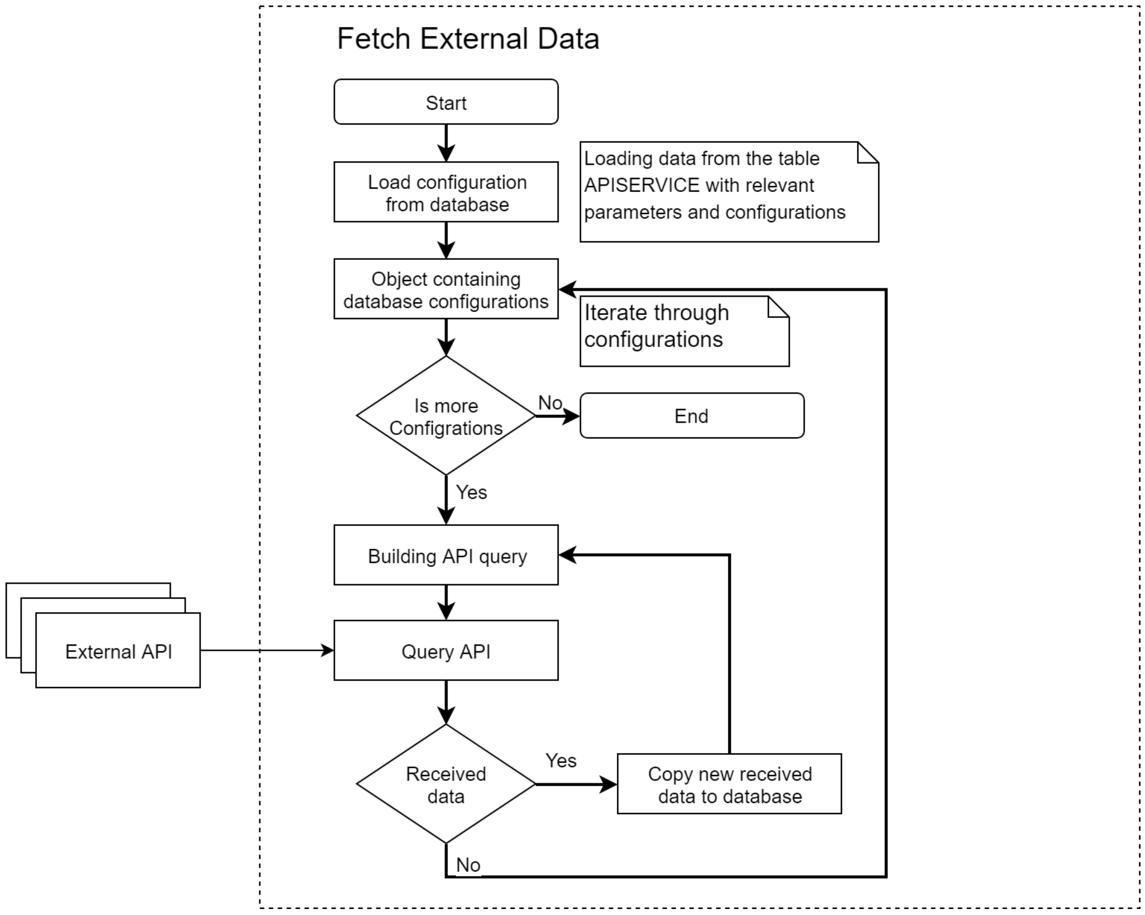

Fetch External Data
Start
Load configuration from database
Loading data from the table APISERVICE with relevant parameters and configurations
Object containing database configurations
Iterate through configurations
Is more Configrations
No
End
Yes
Building API query
External API
Query API
Received data
Yes
Copy new received data to database
No

Figure 9.3: Flow diagram for fetching external data

9.2.1 Working with JSON data

Figure 9.4 shows a JSON string results when using an example query.

```
Query: "https://api.nilu.no/aq/historical/NO2?fromtime=2018.02.24 23:00&
    totime=2018.02.25 01:00&areas=Grenland&stations=Sverresgate"
Result:
{
        "zone": "Øst og Sørlandet",
        "municipality": "Grenland",
        "area": "Grenland",
        "station": "Sverresgate",
        "eoi": "NO0106A",
        "component": "NO2",
        "latitude": 59.13812,
        "longitude": 9.65212,
        "values": [
            {
                "fromTime": "2018-02-24T23:00:00+01:00",
                "toTime": "2018-02-25T00:00:00+01:00",
                "value": 18.2448174081,
                "unit": "µg/m³",
                "index": 1,
                "color": "6ee86e"
            },
            {
                "fromTime": "2018-02-25T00:00:00+01:00",
                "toTime": "2018-02-25T01:00:00+01:00",
                "value": 14.6143858682,
                "unit": "µg/m³",
                "index": 1,
                "color": "6ee86e"
            }
        ]
    }
```

Figure 9.4: Json data from NILU's API

To be able to read and iterate through JSON strings the nuget package Newtonsoft.Json was installed. Figure 9.5 shows how the JSON string in figure 9.4 is converted to a JSON object and then iterated through. Note how the path names shown in figure 9.4 is used when accessing the data in the JSON object. In this case the JSON will hold two items in an array called "values". Inside this array measurement and time stamp can be found with the corresponding names "value" and "toTime". These names need to be saved in the database as configurations in order to know what data to pick.

```
JObject jsonObject = JObject.Parse(jsonString);

int length = data["values"].Count();
            if (length != 0)
            {
                for (int i = 0; i < length; i++)
                {
                    string value = (string)data["values"][i]["value"];
                    DateTime timeStamp = (DateTime)data["values"][i]["toTime
                        "];
                }
            }
```

Figure 9.5: Converting JSON string to JSON Object

9.3 Status Updates and Push Notifications

Push notifications is tied to status changes. When calculating new statuses, the old ones will be kept in memory in order to detect changes. These changes will trigger push notifications. An overview for the logic is shown in figure 9.6.

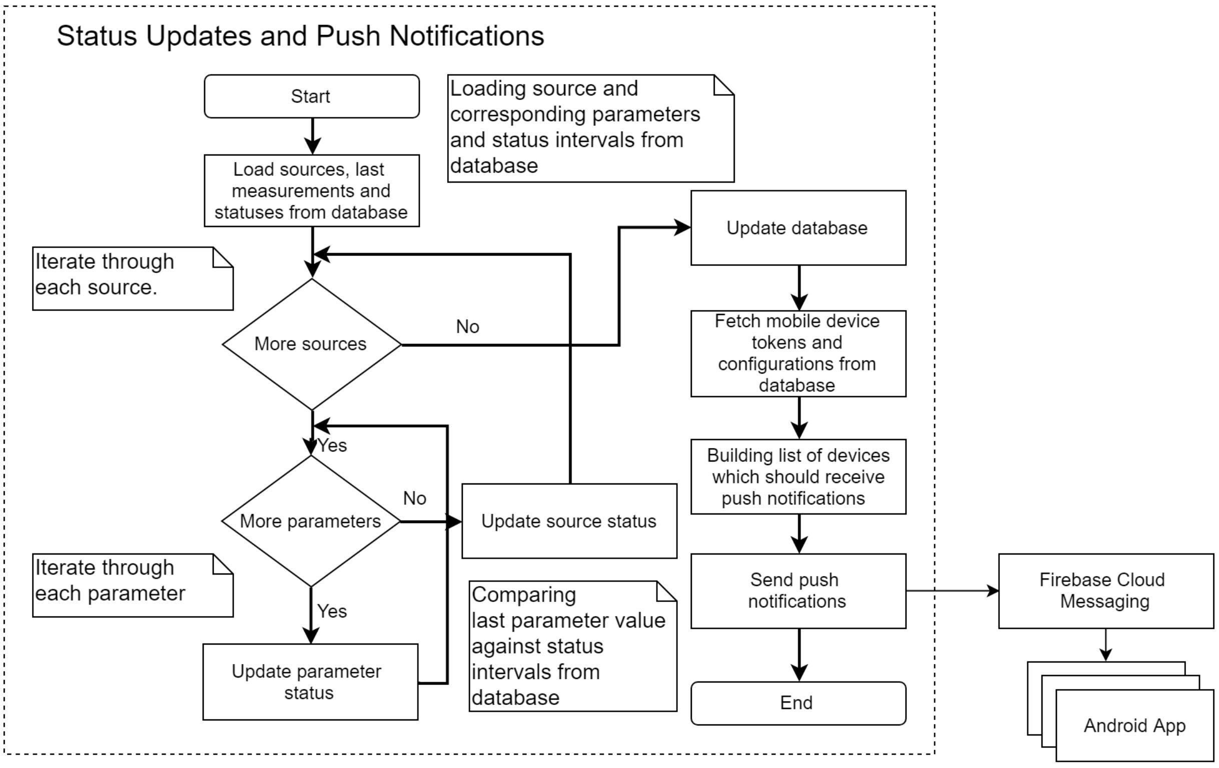

Status Updates and Push Notifications
Start
Loading source and corresponding parameters and status intervals from database
Load sources, last measurements and statuses from database
Iterate through each source.
More sources
No
Yes
Update database
Fetch mobile device tokens and configurations from database
Building list of devices which should receive push notifications
More parameters
No
Yes
Update source status
Iterate through each parameter
Update parameter status
Comparing last parameter value against status intervals from database
Send push notifications
Firebase Cloud Messaging
End
Android App

Figure 9.6: Flow diagram for updating statuses and sending push notifications

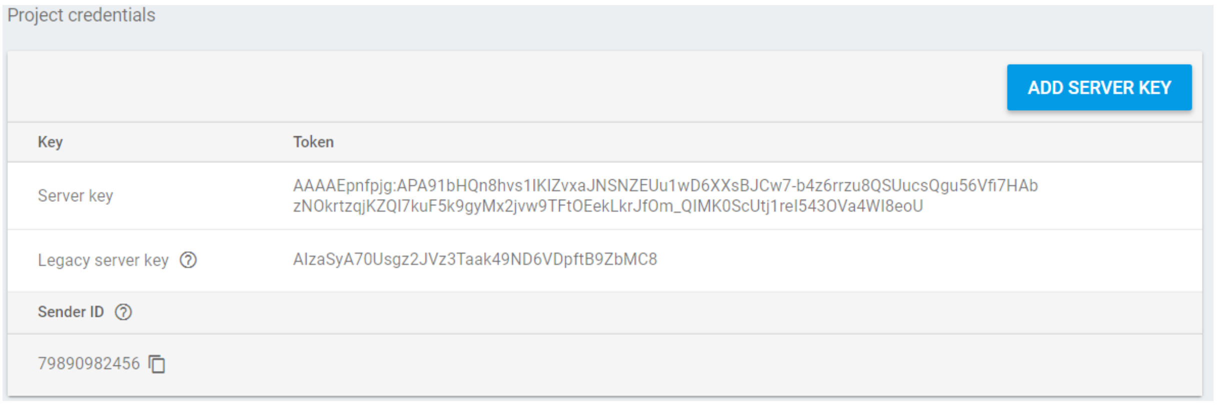
9.3.1 Status updates

Each parameter has corresponding configurations tables holding information about the different status intervals, which tell the user how bad or good the values are. It will load all the sources and corresponding parameters and status intervals, and then find the relevant status interval for each parameter based on the last measured value. Each status is given with a number indicating severity, where lower is good and higher is bad. The parameter that holds the status with the highest severity will set the status for the source. Furthermore, if the last measurement is older then 5 hours, it will be indicated as no data and stored as null in the database. Moreover, it will not influence the source status unless all parameter statuses are indicating no data.

The table PARAMINTERVAL holds all the status intervals for each parameter, which is linked to PARAMINTERVALSTATUS that holds information about the severity. In case of air pollution, one row in PARAMINTERVAL can hold information about measurement values that indicates low air pollution, which is then linked to PARAMINTERVALSTATUS that holds the text and color indicating low air pollution. PARAMINTERVALSTATUS is grouped together by the table PARAMINTERVALGROUP.

9.3.2 Push Notifications

Each register user has their own table that holds information about their device token, which will be updated each time the user logs into the app. To be able to send push notifications through the mobile app, the app must be registered within the service Google Firebase, and be set up with Firebase Cloud Messaging. [37]. Figure 9.7 shows the information needed to send push notifications using Google Firebase Cloud Messaging in addition the device tokens.



Key	Token
Server key	AAAAEpnfpjg:APA91bHQn8hvs1IKIZvxaJNSNZEUu1wD6XXsBJCw7-b4z6rrzu8QSUucsQgu56Vfi7HAbzNOkrtzqjKZQI7kuF5k9gyMx2jvw9TFtOEekLkrJfOm_QIMK0ScUtj1reI543OVa4WI8eoU
Legacy server key	AIzaSyA70Usgz2JVz3Taak49ND6VDpftB9ZbMC8

Sender ID

79890982456

Figure 9.7: Firebase Cloud Messaging

If any status changes have occurred, all subscribing users and corresponding device tokens will be fetched from the database, which will then be sent to Firebase Cloud Messaging. This happens using ordinary http post requests. The concept can be seen in figure 9.8.

```
WebRequest tRequest = WebRequest.Create("https://fcm.googleapis.com/fcm/send");
            var applicationID = "Server key"
            var senderId = "Sender ID";
            tRequest.Method = "post";
            tRequest.ContentType = "application/json";

            var dataMessage = new
            {
                to = "Device Token",
                notification = new
                {
                    body = "Body",
                    sound = "default",
                    title = "Title",
                },
                priority = "high"
            };

            var json = JsonConvert.SerializeObject(dataMessage);
            Byte[] byteArray = Encoding.UTF8.GetBytes(json);
            tRequest.Headers.Add(string.Format("Authorization: key={0}", applicationID));
            tRequest.Headers.Add(string.Format("Sender: id={0}", senderId));
            tRequest.ContentLength = byteArray.Length;

            // Code for sending the request is removed due to readability.
```

Figure 9.8: Sending data to FireBase Cloud Messaing

9.4 Update Buffer Table

Database is fast, even with large tables, but the performance will be slower as the tables grow. Dashboard and trending functionality is loaded frequently, and only needs the newest measurements from the database. Therefore, it did make sense to implement a buffer table only containing the newest measurements.

The buffer table will hold measurements from the last month. Figure 9.9 shows the logic for updating the buffer table.

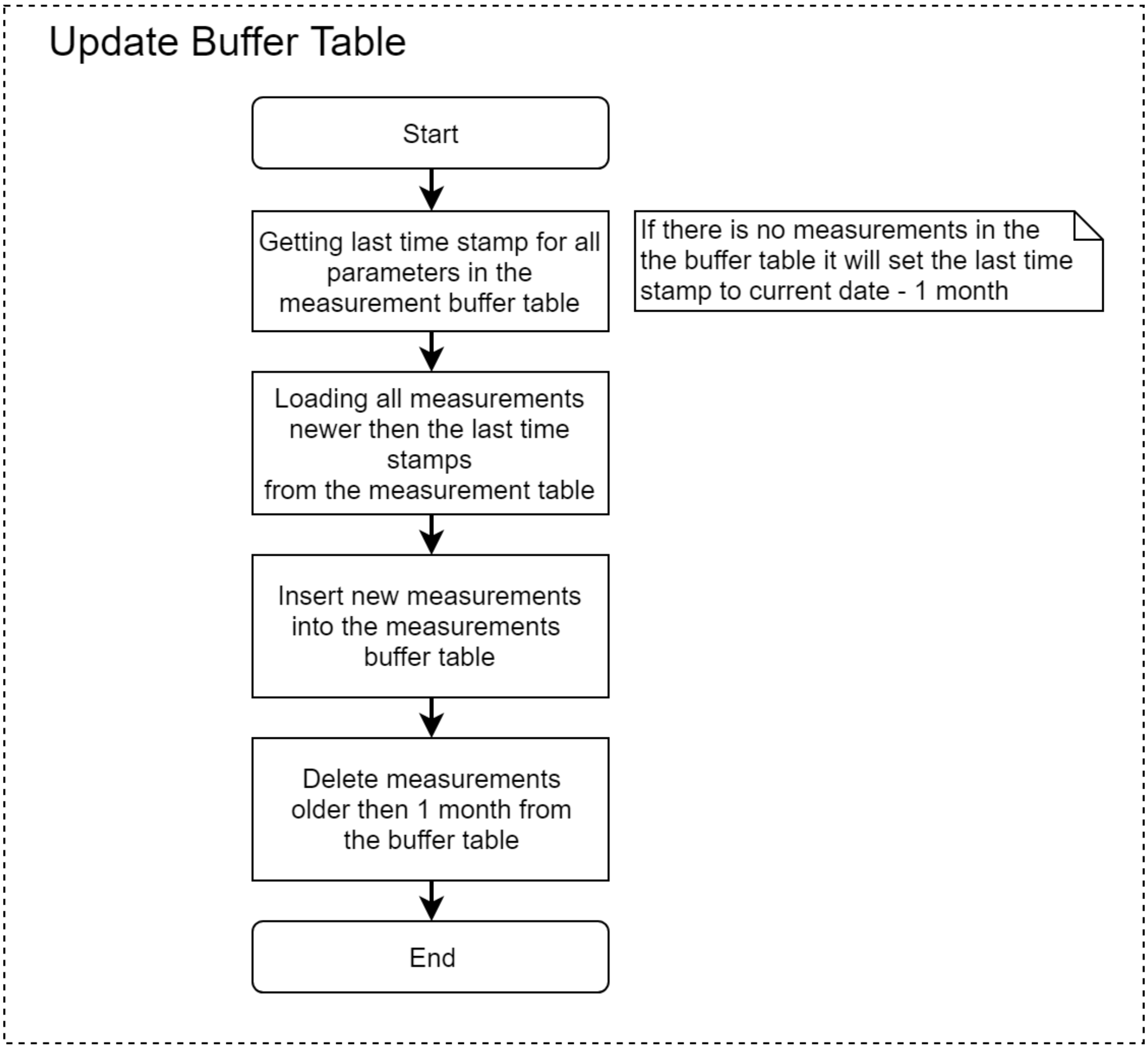

Update Buffer Table
Start
Getting last time stamp for all parameters in the measurement buffer table
If there is no measurements in the the buffer table it will set the last time stamp to current date - 1 month
Loading all measurements newer then the last time stamps from the measurement table
Insert new measurements into the measurements buffer table
Delete measurements older then 1 month from the buffer table
End

Figure 9.9: Updating buffer table

10 Results

This chapter will cover the results from the website, mobile app, services and database.

10.1 Website

Each page on the web page will be illustrated with images presenting the functionality in use. The website contains an ”About” and a ”Get App” page, but will not be illustrated with images. The ”Get App” gives the users a download link and a simple explanation of the app. The ”About” page will tell the user some simple information about the system.

10.1.1 Dashboard

The dashboard presents the most relevant and important information, such as parameters and last measured values. Each source includes a plot showing trending data for the last day. Status is given focus as it might be the most important indication for most of the users. Furthermore, status for each parameter is also shown by giving color to the last value field. It can be clearly seen that each parameter uses the same color, which increases readability of the plot. By clicking on the details link, the user will be forwarded to the trending web page showing the relevant source. Figure 10.1 shows the dashboard.

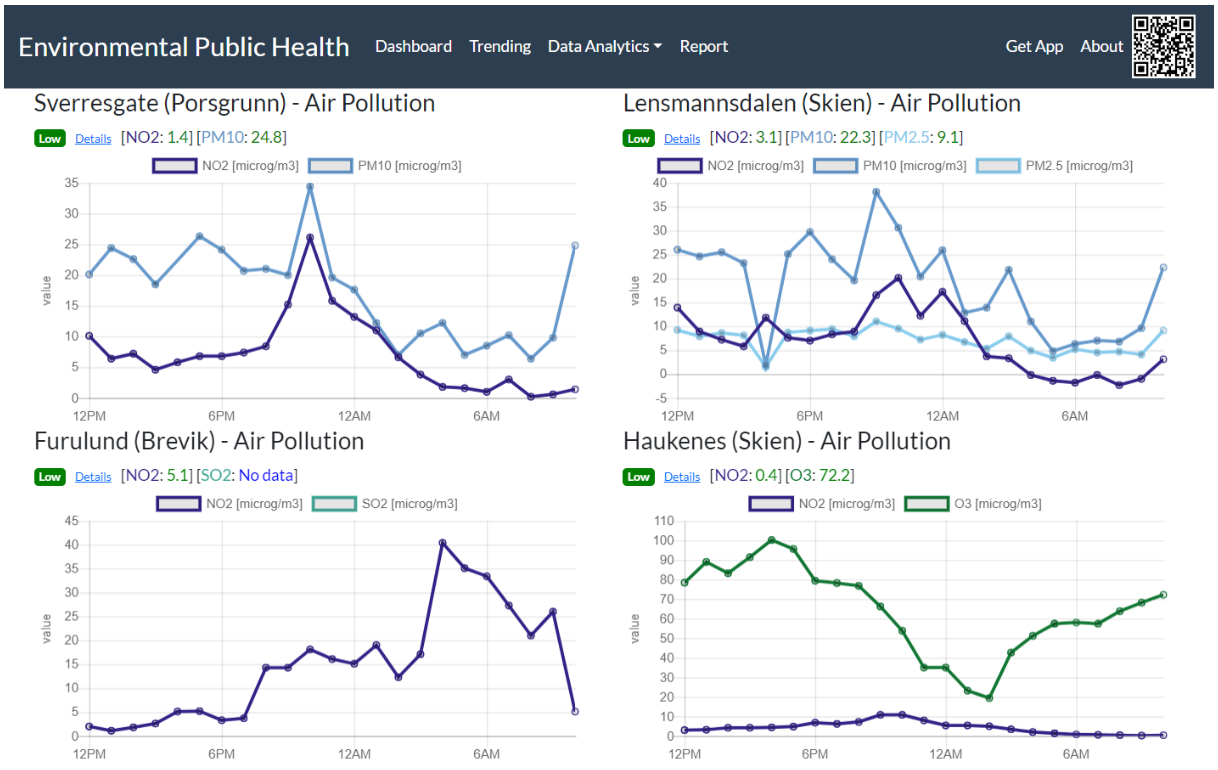
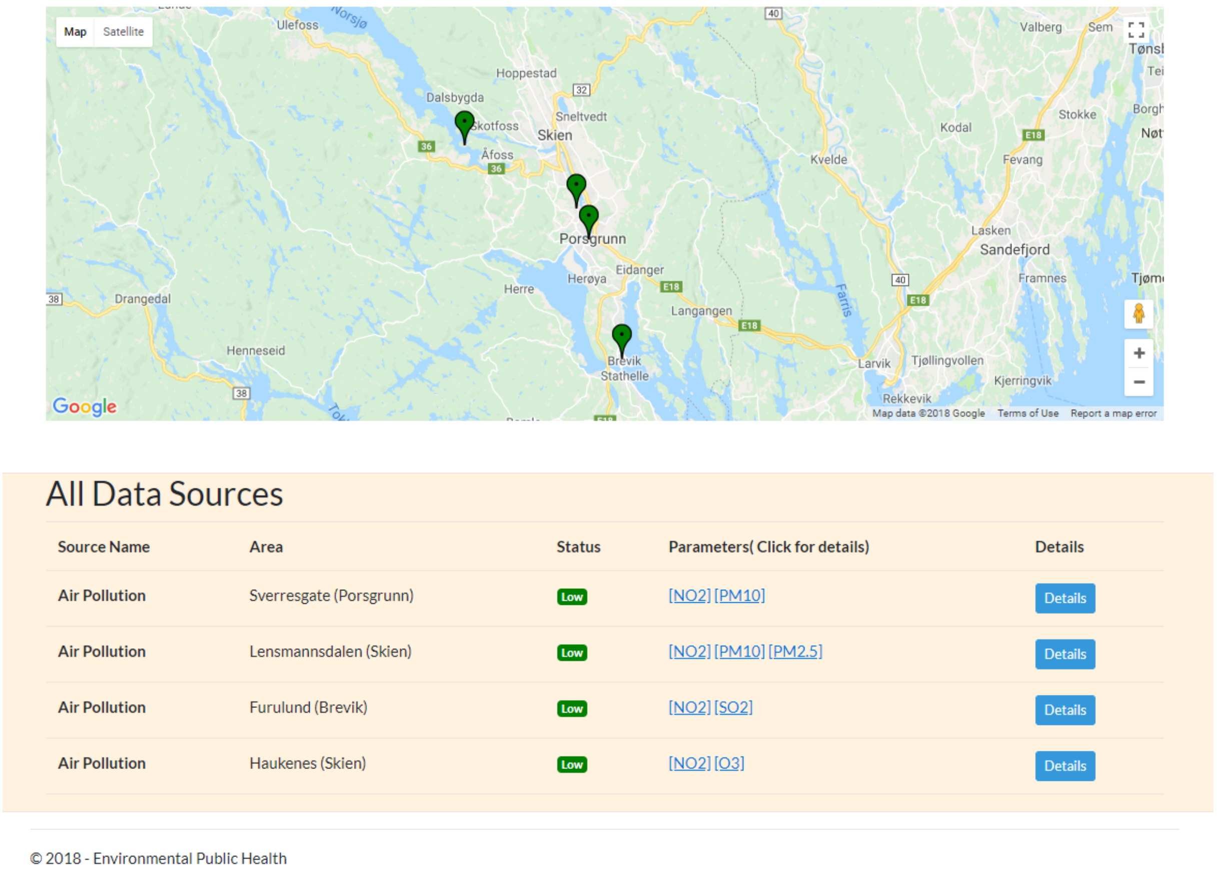

Environmental Public Health
Dashboard
Trending
Data Analytics
Report
Get App
About
Sverresgate (Porsgrunn) - Air Pollution
Low Details [NO2: 1.4] [PM10: 24.8]
NO2 [microg/m3]
PM10 [microg/m3]
Lensmannsdalen (Skien) - Air Pollution
Low Details [NO2: 3.1] [PM10: 22.3] [PM2.5: 9.1]
NO2 [microg/m3]
PM10 [microg/m3]
PM2.5 [microg/m3]
Furulund (Brevik) - Air Pollution
Low Details [NO2: 5.1] [SO2: No data]
NO2 [microg/m3]
SO2 [microg/m3]
Haukenes (Skien) - Air Pollution
Low Details [NO2: 0.4] [O3: 72.2]
NO2 [microg/m3]
O3 [microg/m3]
value
12PM
6PM
12AM
6AM

Figure 10.1: Dashboard

By scrolling down on the dashboard a map and more details about the sources can be seen. Each source is represented on the map by the color indicating the status. The sources on the map are clickable, where a pop up will include the source name, status and a link forwarding to the trending web page. A table is representing more details about all the sources, potentially including the sources which are not displayed on the dashboard. Figure 10.2 shows the map and the table.



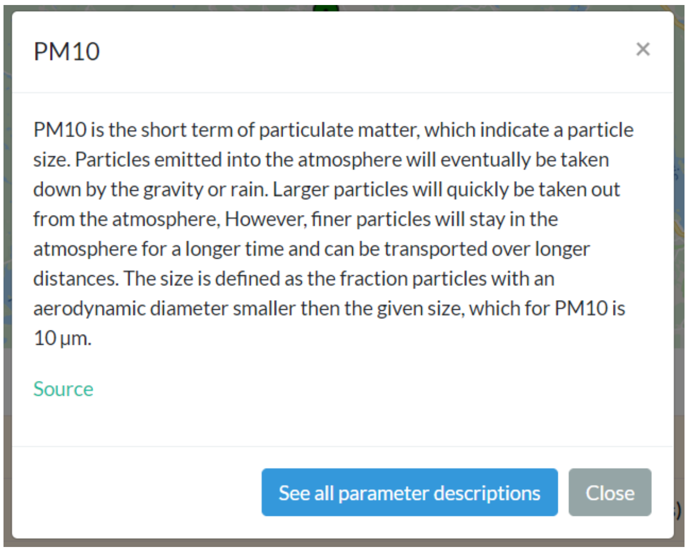
All Data Sources

Source Name	Area	Status	Parameters(Click for details)	Details
Air Pollution	Sverresgate (Porsgrunn)	Low	[NO2] [PM10]	Details
Air Pollution	Lensmannsdalen (Skien)	Low	[NO2] [PM10] [PM2.5]	Details
Air Pollution	Furulund (Brevik)	Low	[NO2] [SO2]	Details
Air Pollution	Haukenes (Skien)	Low	[NO2] [O3]	Details

© 2018 - Environmental Public Health

Figure 10.2: More details on dashboard

By clicking on the parameter name in the table, a pop up containing the information about the parameter will be displayed as shown in figure 10.3. From the pop up it is possible to navigate to a web page containing information about all the parameters within the system, which can be seen in figure 10.4.

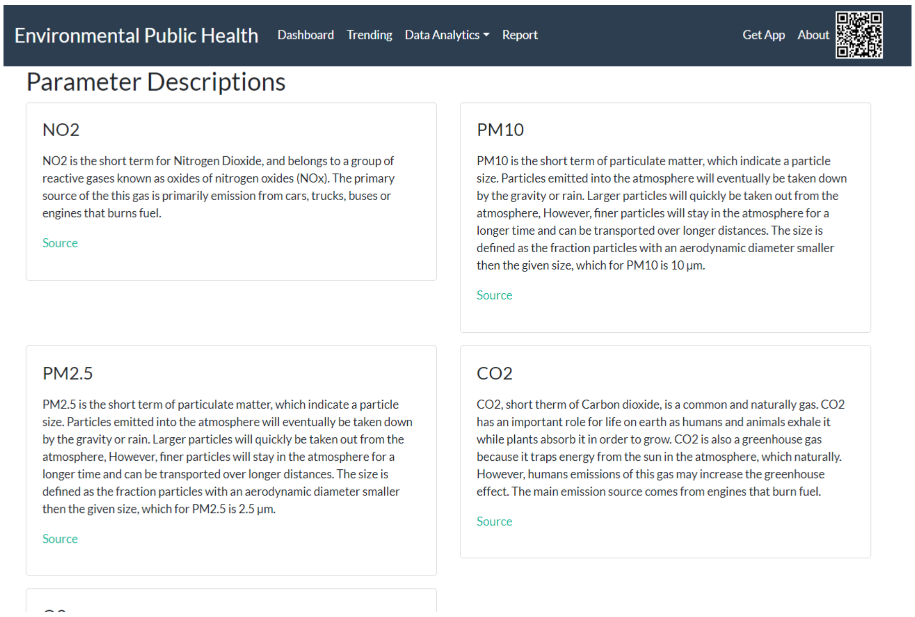


PM10

PM10 is the short term of particulate matter, which indicate a particle size. Particles emitted into the atmosphere will eventually be taken down by the gravity or rain. Larger particles will quickly be taken out from the atmosphere, However, finer particles will stay in the atmosphere for a longer time and can be transported over longer distances. The size is defined as the fraction particles with an aerodynamic diameter smaller then the given size, which for PM10 is 10 µm.

Source

Figure 10.3: Parameter description



Parameter Descriptions

NO2

NO2 is the short term for Nitrogen Dioxide, and belongs to a group of reactive gases known as oxides of nitrogen oxides (NOx). The primary source of the this gas is primarily emission from cars, trucks, buses or engines that burns fuel.

Source

PM10

PM10 is the short term of particulate matter, which indicate a particle size. Particles emitted into the atmosphere will eventually be taken down by the gravity or rain. Larger particles will quickly be taken out from the atmosphere, However, finer particles will stay in the atmosphere for a longer time and can be transported over longer distances. The size is defined as the fraction particles with an aerodynamic diameter smaller then the given size, which for PM10 is 10 µm.

Source

PM2.5

PM2.5 is the short term of particulate matter, which indicate a particle size. Particles emitted into the atmosphere will eventually be taken down by the gravity or rain. Larger particles will quickly be taken out from the atmosphere, However, finer particles will stay in the atmosphere for a longer time and can be transported over longer distances. The size is defined as the fraction particles with an aerodynamic diameter smaller then the given size, which for PM2.5 is 2.5 µm.

Source

CO2

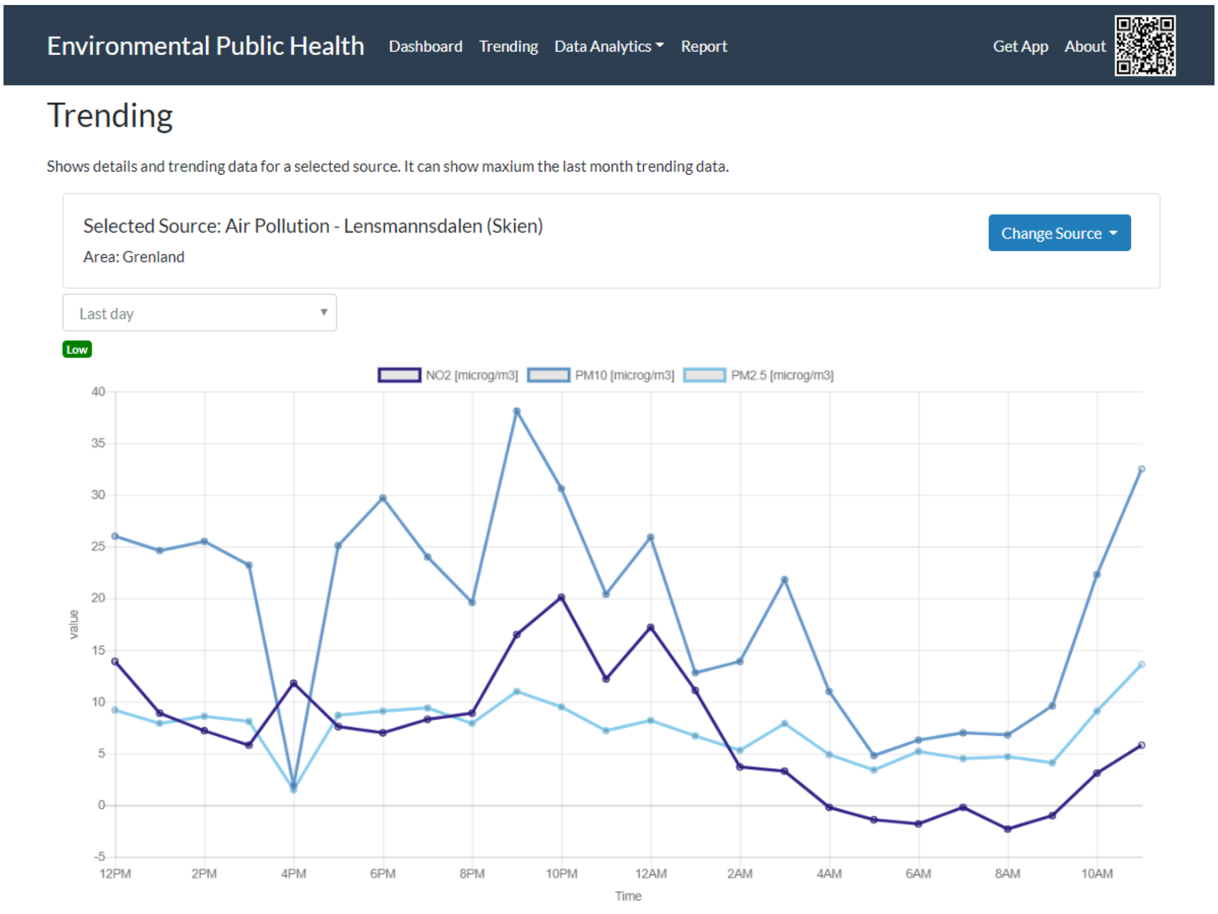
CO2, short therm of Carbon dioxide, is a common and naturally gas. CO2 has an important role for life on earth as humans and animals exhale it while plants absorb it in order to grow. CO2 is also a greenhouse gas because it traps energy from the sun in the atmosphere, which naturally. However, humans emissions of this gas may increase the greenhouse effect. The main emission source comes from engines that burn fuel.

Source

Figure 10.4: All parameter descriptions

10.1.2 Trending

The trending web page gives the user more information about trending data and details. Since only one source is plotted, the plot is made bigger, which makes it easier to read. It can show historical data from the last day, last three days, last week and last month. It is not possible to choose other time periods. All the sources are available through a drop down menu. Figure 10.5 shows the plot and the different options for changing source and time period.



Trending

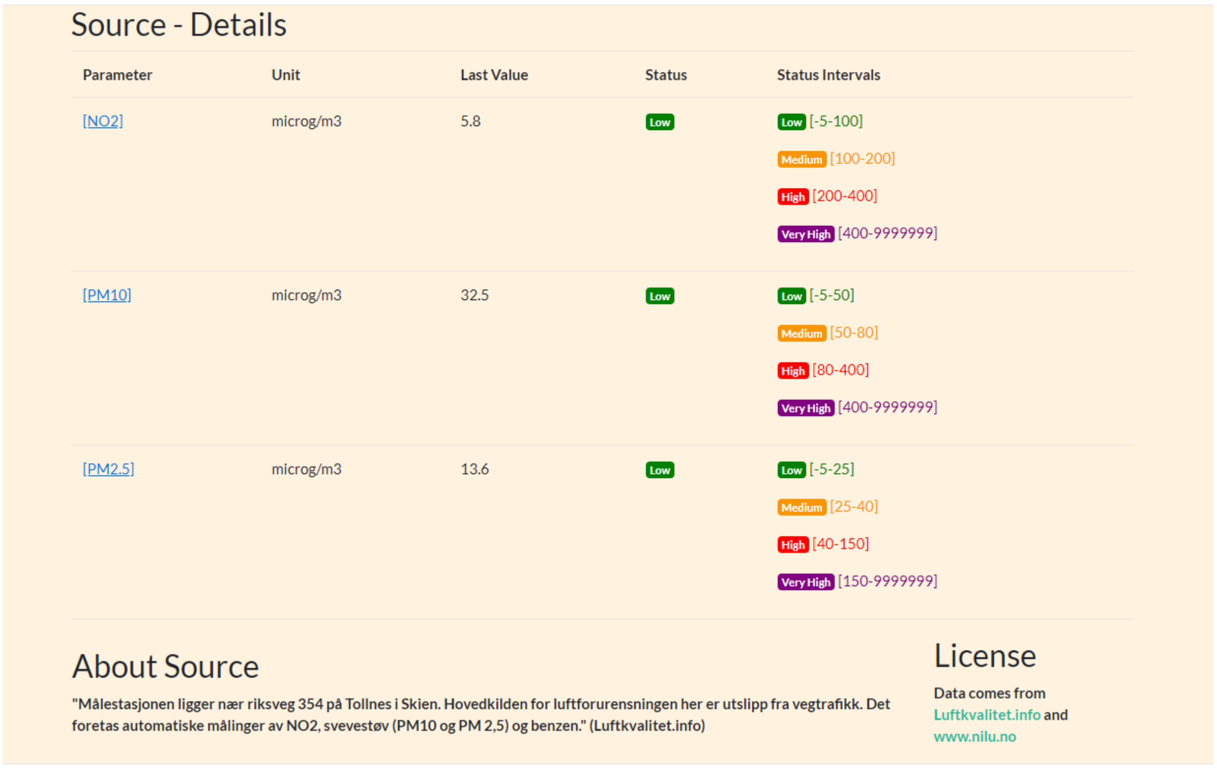
Shows details and trending data for a selected source. It can show maxium the last month trending data.

Selected Source: Air Pollution - Lensmannsdalen (Skien)

Area: Grenland

Figure 10.5: Trending

Furthermore, the trending page contains a table with details about the associated parameters like unit, last value, status and the different status intervals. The status intervals are indicated by values and associated colors. License information and about source are also displayed. The details can be seen in figure 10.6.



Source - Details

Parameter	Unit	Last Value	Status	Status Intervals
[NO2]	microg/m3	5.8	Low	Low [-5-100] Medium [100-200] High [200-400] Very High [400-9999999]
[PM10]	microg/m3	32.5	Low	Low [-5-50] Medium [50-80] High [80-400] Very High [400-9999999]
[PM2.5]	microg/m3	13.6	Low	Low [-5-25] Medium [25-40] High [40-150] Very High [150-9999999]

About Source

"Målestasjonen ligger nær riksveg 354 på Tollnes i Skien. Hovedkilden for luftforurensningen her er utslipp fra vegtrafikk. Det foretas automatiske målinger av NO2, svevestøv (PM10 og PM 2,5) og benzen." (Luftkvalitet.info)

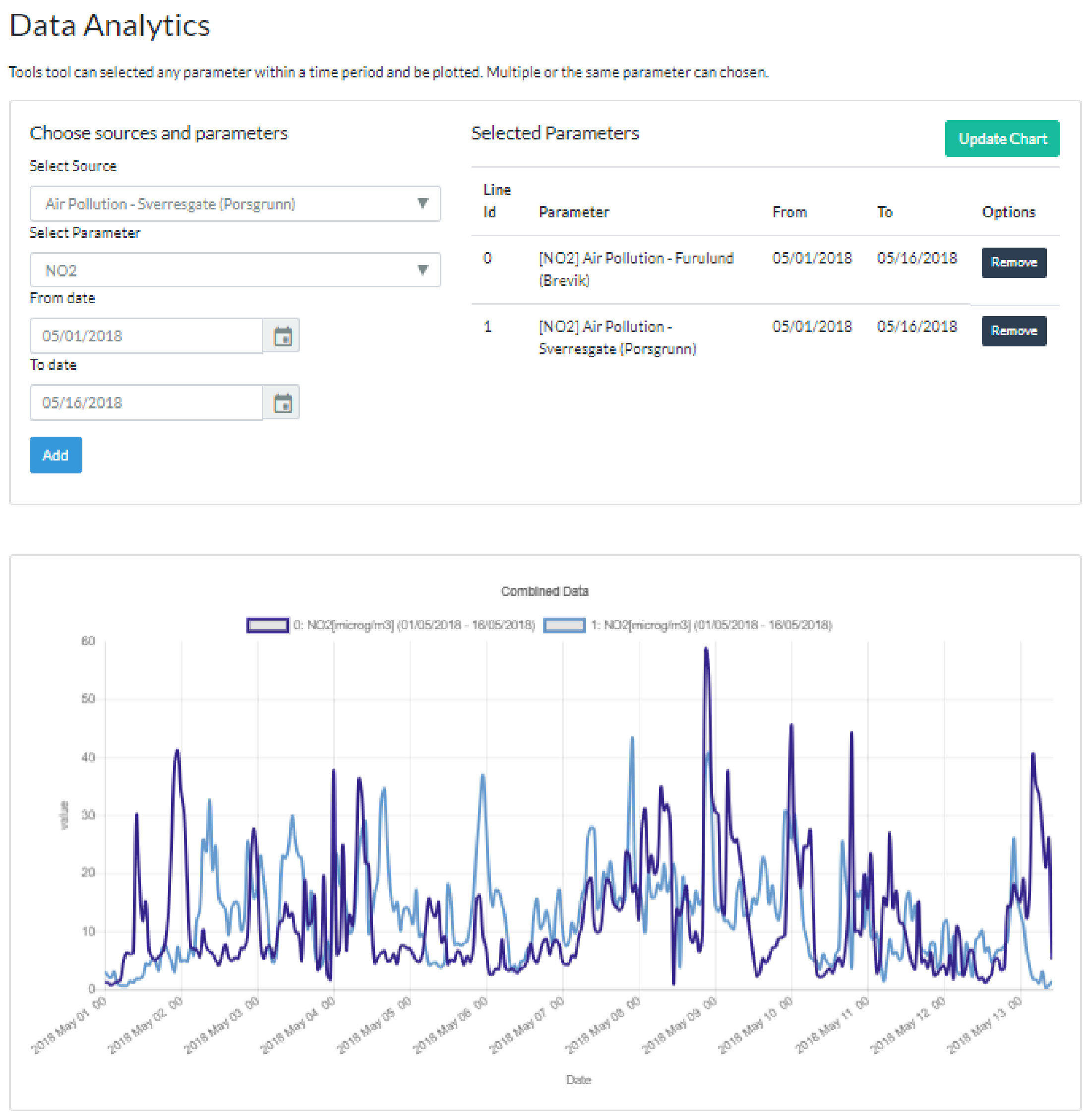
License

Data comes from Luftkvalitet.info and www.nilu.no

Figure 10.6: Trending - more details

10.1.3 Data Analytics

The analytic functionality allows the user to combine any data from any source together on the same plot, which happens without the page refreshing. This allows the user to investigate and compare different parameters against each other. If plotting data from a large time period it becomes difficult to read the plot. There is no method for scaling the data. An image from the web page can be seen in figure 10.7.



Data Analytics

Tools tool can selected any parameter within a time period and be plotted. Multiple or the same parameter can chosen.

Choose sources and parameters

Select Source: Air Pollution - Sverresgate (Porsgrunn)

Select Parameter: NO2

From date: 05/01/2018

To date: 05/16/2018

Selected Parameters

Line Id	Parameter	From	To	Options
0	[NO2] Air Pollution - Furulund (Brevik)	05/01/2018	05/16/2018	Remove
1	[NO2] Air Pollution - Sverresgate (Porsgrunn)	05/01/2018	05/16/2018	Remove

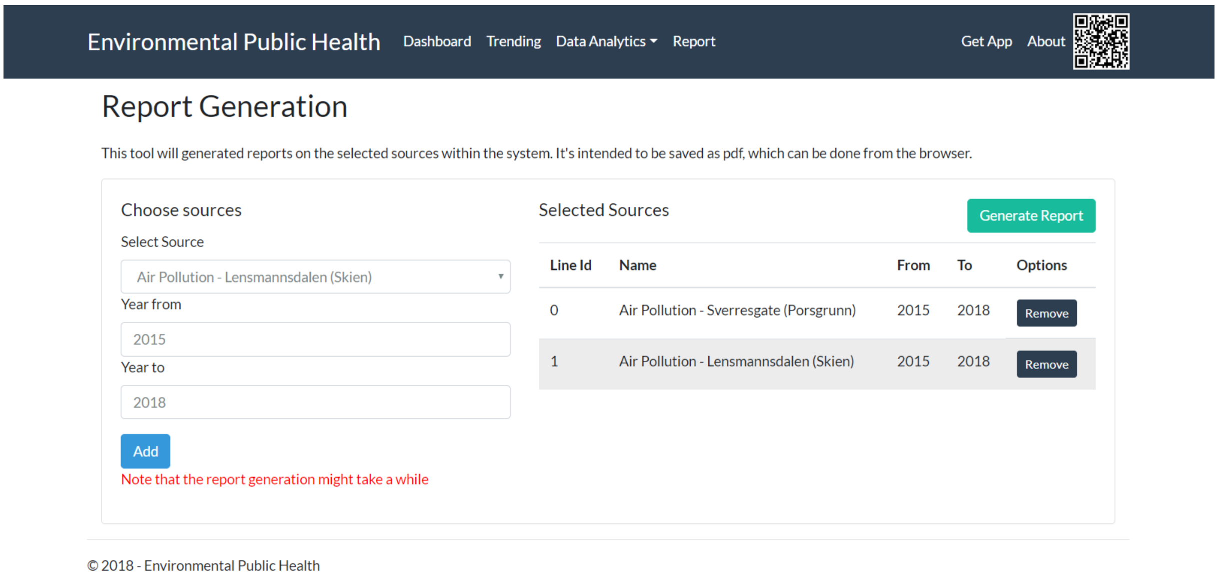
Figure 10.7: Data analytic

10.1.4 Data Analytics - Compare Periods

The data analytic functionalities given in section 10.1.3 will not be able to lay over different periods in the plot. Therefore, a second data analytic page was made dedicated for comparing weeks, months and years. For example, when selecting weeks, it will scale all the time stamps to fit into one week. It is possible to plot different weeks from the same parameter side by side. The web page can be seen in appendix D.

10.1.5 Report Generation

The report generation page allows the user to generate reports based on selected sources and year span. Instead of plotting each measurement, all measurements within one month will be combined into one measurement were the average value is used. Max, min and average value for each parameter will be calculated both for yearly and the whole time span. The setup page for report generation can be seen in figure 10.8. An example report was generated and can be found in appendix E.



Report Generation

This tool will generated reports on the selected sources within the system. It's intended to be saved as pdf, which can be done from the browser.

Choose sources

Select Source: Air Pollution - Lensmannsdalen (Skien)

Year from: 2015

Year to: 2018

Note that the report generation might take a while

Selected Sources

Line Id	Name	From	To	Options
0	Air Pollution - Sverresgate (Porsgrunn)	2015	2018	Remove
1	Air Pollution - Lensmannsdalen (Skien)	2015	2018	Remove

© 2018 - Environmental Public Health

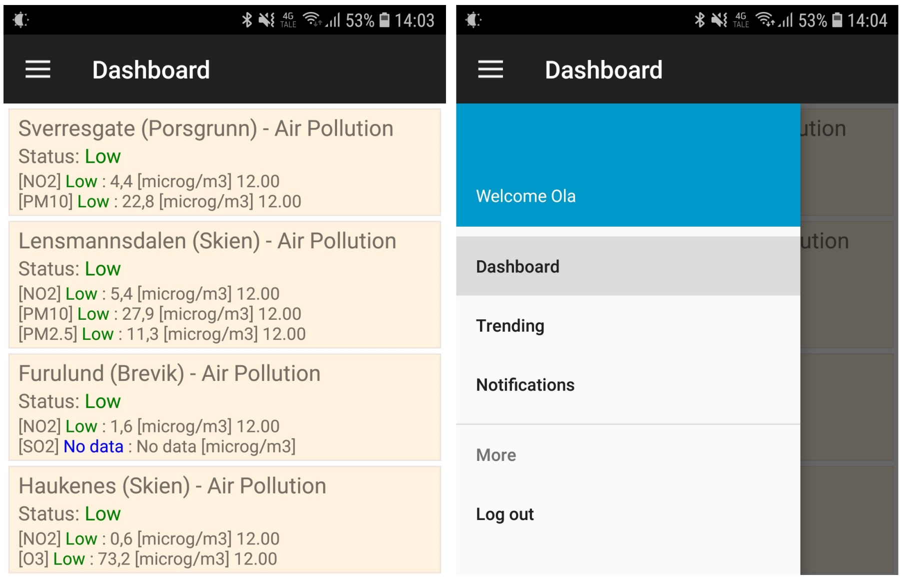
Figure 10.8: Report generation - setup

10.2 Mobile App

The results from the mobile app will be illustrated with images from the different pages. All the functionalities are tested and work as intended.

10.2.1 Dashboard

The dashboard page and the navigation menu can be seen in figure 10.9. Each source is presented in a list, where a tap on the element containing the source will redirect the trending page. The status for the sources is indicated with text and color. In addition, status for each parameter is also shown. Since the dashboard does not feature plotting, time stamp for the last values are included.


53% 14:03

Dashboard

Sverresgate (Porsgrunn) - Air Pollution
Status: Low
[NO2] Low : 4,4 [microg/m3] 12.00
[PM10] Low : 22,8 [microg/m3] 12.00

Lensmannsdalen (Skien) - Air Pollution
Status: Low
[NO2] Low : 5,4 [microg/m3] 12.00
[PM10] Low : 27,9 [microg/m3] 12.00
[PM2.5] Low : 11,3 [microg/m3] 12.00

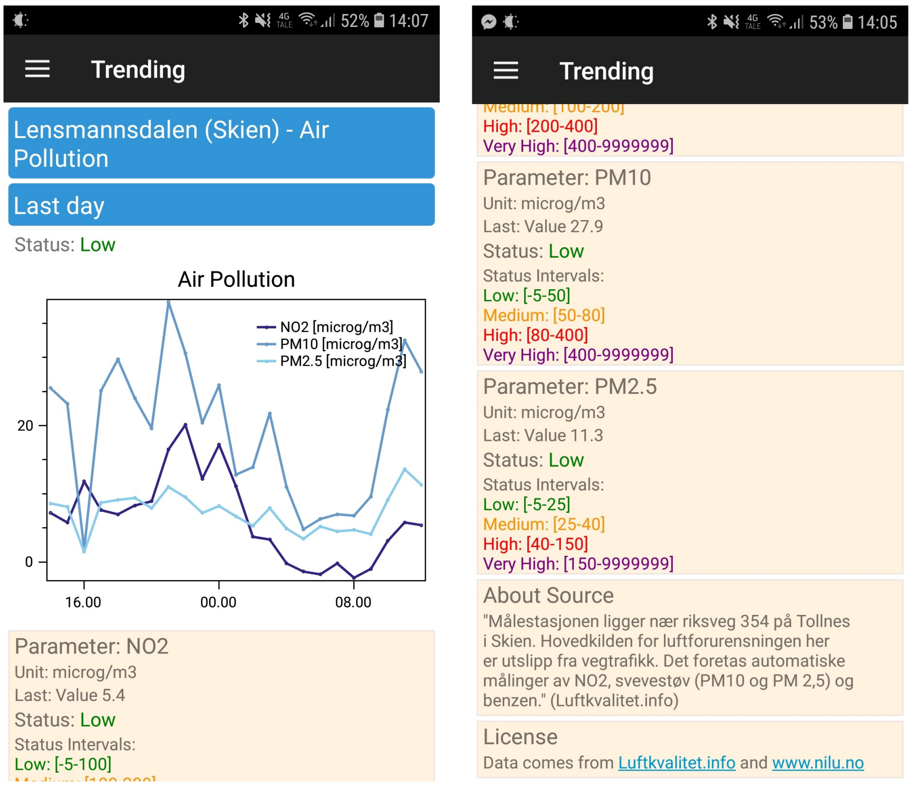
Furulund (Brevik) - Air Pollution
Status: Low
[NO2] Low : 1,6 [microg/m3] 12.00
[SO2] No data : No data [microg/m3]

Haukenes (Skien) - Air Pollution
Status: Low
[NO2] Low : 0,6 [microg/m3] 12.00
[O3] Low : 73,2 [microg/m3] 12.00

Figure 10.9: Dashboard and navigation

10.2.2 Trending

The trending page contains the same details as the website does. The difference is mainly the screen size. Two drop down menus are used to change source and select last time period for the historical data. The plot supports scrolling, but cause some problem as the plot itself is inside a list which also is scrollable. By scrolling down more details about the source can be seen like status intervals, unit, about source and license data. The trending page is shown in figure 10.10.



Status: Low

Parameter: NO2
Unit: microg/m3
Last: Value 5.4
Status: Low
Status Intervals:
Low: [-5-100]

Medium: [100-200]
High: [200-400]
Very High: [400-9999999]

Parameter: PM10
Unit: microg/m3
Last: Value 27.9
Status: Low
Status Intervals:
Low: [-5-50]
Medium: [50-80]
High: [80-400]
Very High: [400-9999999]

Parameter: PM2.5
Unit: microg/m3
Last: Value 11.3
Status: Low
Status Intervals:
Low: [-5-25]
Medium: [25-40]
High: [40-150]
Very High: [150-9999999]

About Source
"Målestasjonen ligger nær riksveg 354 på Tollnes i Skien. Hovedkilden til luftforurensningen her er utslipp fra vegtrafikk. Det foretas automatiske målinger av NO2, svevestøv (PM10 og PM 2,5) og benzen." (Luftkvalitet.info)

License
Data comes from Luftkvalitet.info and www.nilu.no

Figure 10.10: Trending

10.2.3 Log In / Register new user

The log in and register pages works as intended. If a user is not logged in he/she will be redirected to the log in page. Furthermore, if the user has no associated account it is possible to navigate to the register new user page, were the user can enter valid email, user name and password. The user will either get a message indicating success or fault when signing up. If the server detects invalid password or email, or that the email or user name is already taken, it will return an error message indicating the error. The log in and register pages can be seen in figure 10.11.

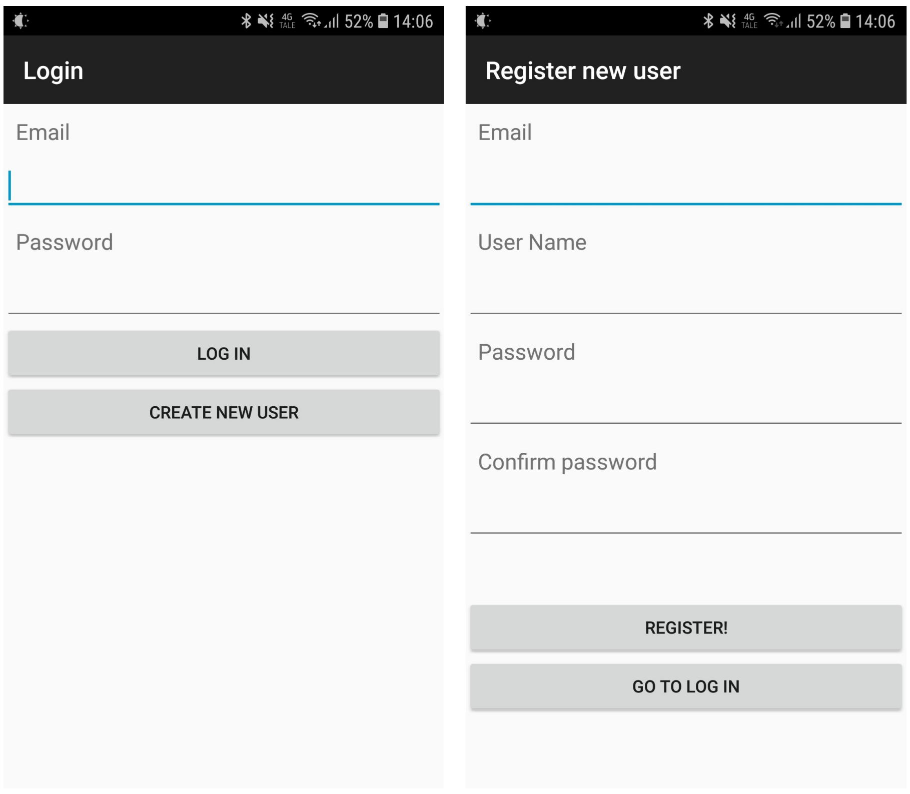
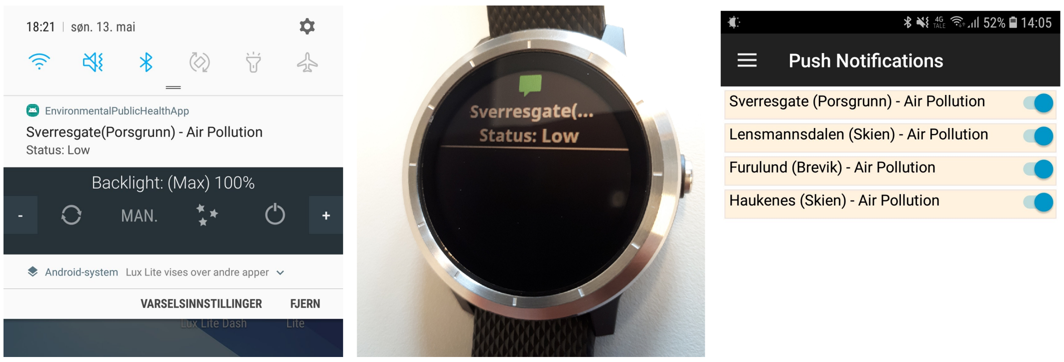


Figure 10.11: Log in/register

10.2.4 Push Notifications

The push notifications work as intended, but the functionalities for disabling push within a time period was not implemented on the app. The app will receive one push notification for each source that changes status. If system somehow stops receiving measurements for a source, the status will be indicated as ”No Data” after five hours. This status change will generate a push notification indicating no data. If the source starts receiving data again, this also will trigger a push notification. Note that this will only happen if all the parameters within the source are indicated as ”No Data”. Figure 10.12 shows the configuration and when the app receives a push notification. The figure also shows a smart watch receiving push notification from an Android phone, which is a fine feature for mobile apps.


18:21 | søn. 13. mai

EnvironmentalPublicHealthApp
Sverresgate(Porsgrunn) - Air Pollution
Status: Low

Backlight: (Max) 100%

Android-system Lux Lite vises over andre apper

VARSELSINNSTILLINGER FJERN

Sverresgate(...
Status: Low

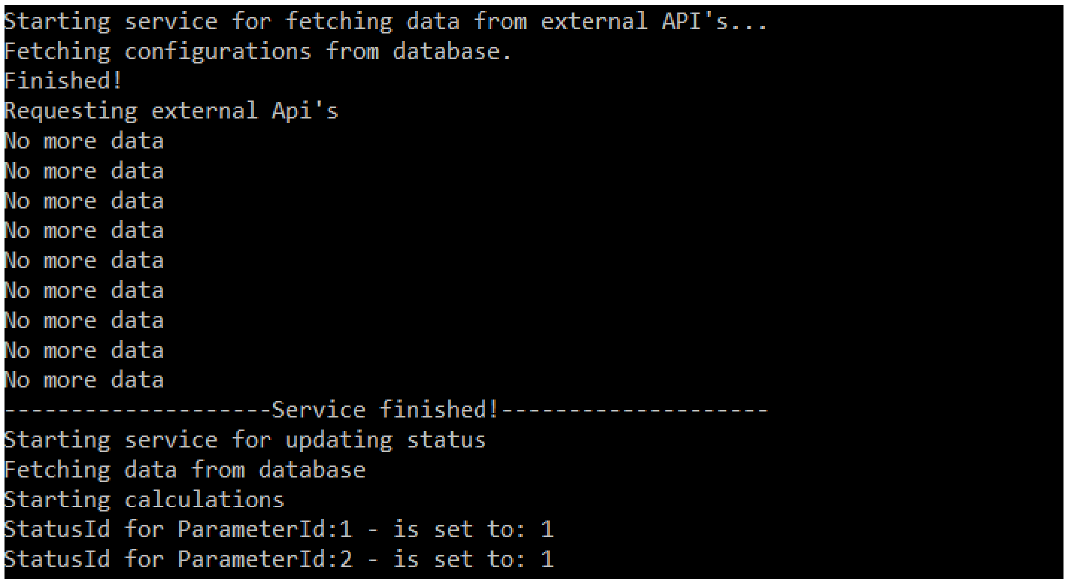
52% 14:05
Push Notifications

Sverresgate (Porsgrunn) - Air Pollution
Lensmannsdalen (Skien) - Air Pollution
Furulund (Brevik) - Air Pollution
Haukenes (Skien) - Air Pollution

Figure 10.12: Push notifications

10.3 Services

The services are successfully running at Azure, and no failure has been detected till this point while running there. The measurements within the system has been cross checked against luftkvalitet.info many times, and no mismatch was detected[2]. Figure 10.13 shows the console application when the services are running locally. After gathering data from 2015 to 2018 for the sources available in Grenland area, the database storage usage was 24 MB. Moreover, the implementation of buffer table improved the loading time of the dashboard.



```
Starting service for fetching data from external API's...
Fetching configurations from database.
Finished!
Requesting external Api's
No more data
No more data
No more data
No more data
No more data
No more data
No more data
No more data
No more data
--------------------Service finished!--------------------
Starting service for updating status
Fetching data from database
Starting calculations
StatusId for ParameterId:1 - is set to: 1
StatusId for ParameterId:2 - is set to: 1
```

Figure 10.13: Services running locally

11 Discussion

Track changes is off As mentioned in chapter 5, RAP was chosen as software development method, which gave the advantage of continuous improvements based on feedback during development.

11.1 Website

The functionality for logging in and registering user are implemented, but hidden as no functionality is linked to any user. As mentioned in section 3.2, it was planned that the users should be able to configure some of the functionality, but due to the work load and given time this was not implemented. Moreover, administrative user interface functionality intended to allow creating/editing source, parameter, API configurations etc was not implemented. However, the database design allows this to be implemented later on.

The dashboard web page gives the users a quick overview of the statuses and trending for the last day. The fields indicating the status can easily be seen for each source. Which sources that should be presented on the dashboard is configurable in the database, but changes apply to all users, which was planned to be a personal configuration for logged in users. Moreover, the map gives users quick information about the location of the sources, which is indicated with colors in order to give a geographical overview about the statuses. By clicking on the detail link on the dashboard, map and details table, the users will be redirected to the trending site where the selected source is preloaded.

The trending web page shows trending data for the selected source, and corresponding details about the source and associated parameters. Furthermore, the status intervals are displayed as text for each parameter in the parameters table. This could have been solved differently, for example by plotting them in the plot. Each parameter may have many limits, so this may lead to many lines in the plot. Logic could have been added to only draw the closest status limits. This could have been added as an option that could been activated when needed.

The analytic functionalities allow users to compare parameters against each others. Due to readability when reading plots with large data sets, it was planned to implement scaling logic as an option, but given the time it was not prioritized. Scaling could be done in both the database and in back-end code. The data could have been scaled based on days, weeks or month. This could have been added as an option where the user can select the desired scaling option. Moreover, it was also planned to implement an option for logged in users to save any setup for choosing parameters, but this was not implemented. This would have been useful for users frequently investigating the same parameters.

As planned, report generation was implemented to be used as a basis for decision making. It was originally planned to generate reports as pdf files, but due to time they are now generated as HTML, which the user can print as pdf since most browsers support this. The reports will be generated by scaling the data sets based on months. However, more options for scaling should have been implemented. Both the analytic and report generation functionalities were planned to be restricted to logged in users in case of shortage in server resources, but are instead made available to the public. The database and solution design allow for restriction in usage so it is possible to implement this later on.

Figure 11.1 shows a plot extracted from the generated report given in appendix E. The plot clearly shows trends where the NO2 levels are lowest at summer time and highest during the winter. This correspond with the known facts that NO2 actually is higher during the winter [52]. The detection of this trending indicates the solution could be used by government or companies as a base for taking measures for reducing emissions.

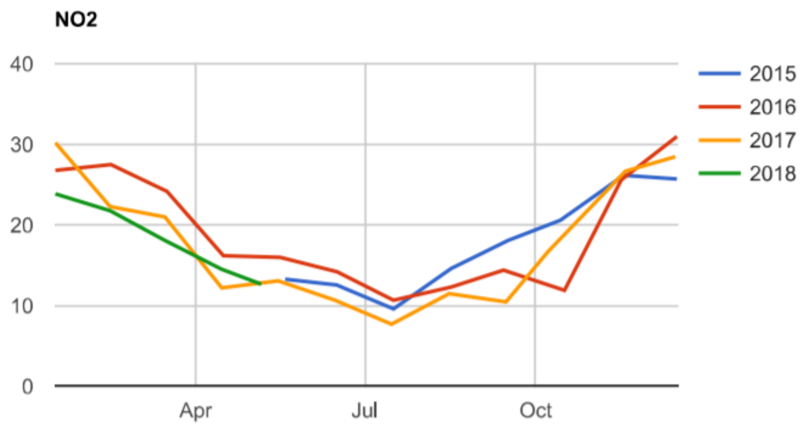


Figure 11.1: Report generation - detecting trending

11.2 Mobile App

The mobile app works as intended and provides some of the functionalities as planned in section 3.3. The design was not prioritized. Disabling push notifications within a time period is supported in the database, but was not implemented in the mobile app. However, the Android operating system itself offers some functionalities for disabling push notifications from the installed apps. Moreover, if subscribing on multiple sources, each source will generate one push notification. It might be a better solution to combine all the sources into one notification. However, newer versions of Android will group push notifications automatically.

Implementing the mobile app only in Android was necessary due to the available hardware and time, but the drawback is any potential limitation in usage and feedback as many use iPhone. However, it is a normal approach to develop an app to only one platform first, and then later on add support for other platforms.

There is also a bug in the trending page while scrolling over the plot. This is because both of them are scrollable. Fixing this bug was not in focus due to time. However, a possible solution might be to override the method that performs the scrolling functionality for the list. If detecting scrolling over the plot, the override method must disable scrolling for the list. If solving the problem as explained, it is important that the plot does not fill the whole screen as it would be impossible to scroll down the list due to the override method.

11.3 Services and Database

The service for collecting data through external APIs works when it is querying data from NILU. However, it might not work against other APIs depending on how the data is returned. The database holding the configuration only allows a specific JSON hierarchy structure to be returned in order to convert the string to an iterable object. To overcome the problem, the database must be able to support dynamically added hierarchy structure. This was not implemented due time limitations. See section 9.2.1 for more information about how the service decodes the JSON strings. Furthermore, the service only supports JSON formatted string, but not XML. Yr, for instance, return data as XML. The logic for the service does not need to support XML, as it is possible to convert received XML to JSON. The database as a column indicating the format, but conversion was not implemented in the service as it would be best to prioritize dynamic hierarchy structure first.

The database queries during report generation seems to be slow. The query might need some optimization. If choosing a source and a time span, for instance 2015 to 2018, each year for each parameter will be one query, were the database must search for relevant data in the table containing the measurements. The database is efficient, so one query itself might not take long time, but combined the delay would be noticeable long. A solution could be to expand the database with tables containing prescaled measurement, which could be calculated by an service at fixed intervals. This would would not optimize the queries itself, but will give a massive performance improvement as the data already is scaled and the tables contain fewer rows.

12 Further Work

Suggestion for further work will be covered in this chapter. Minor suggestion will not be emphasized or suggested.

12.1 Website

Options for better configurations for logged in users is a good suggestion for future work. The users should be able to choose which source they want be displayed on the dashboard. The ability to save setup for the analytic and report generation would make the website more user friendly.

Better scaling options for the analytic and report generation web pages would increase the usefulness of these features as it might be easier to read to plots and detect trends.

12.2 Mobile App

Expanding the mobile app to work with iOS will increase the potentially usage.

12.3 Services and Database

The service for gathering data through external APIs should allow dynamically added hierarchy structure for the JSON data. Implementation of XML to JSON conversion would also be beneficial. These two suggestion would allow the service to target a wider range of APIs.

Expand the database to include tables storing statistics about the measurements can be useful. For example, a table could be added to allow monthly storage containing min, max and average values. As the database design is now, statistics must be calculated for each request. Moreover, the database could be expanded with tables for storing scaled data, which could be updated at fixed intervals. This would lead to a massive improvement in performance of loading scaled data as the data is already scaled.

13 Conclusion

Increased focus on environmental health by government, companies and the public laid the basis for this project. Different services providing environmental data already exist today, but an important aim of this thesis is to show that such kind of data can be collected and provided from one single service. People interested in their local environment, or people who could have serious health reactions to bad air quality, asthmatic people etc., can simply install an app and get push notifications if e.g. the pollution levels in their area reach certain limits. This eliminates the need of actively checking different web pages and services to get a good overview over the current status of pollution levels, which increases user convenience, and potentially health safety.

From a technical point of view, the website was successfully developed and deployed to Microsoft Azure. The mobile app was made available on the website for downloading. Furthermore, the app has been tested and seems to work as intended. Subscribing/unsubscribing on push notifications seem to work.

The service which collects data from the NILU API seems to work as intended, but some downtime have been registered at the external API.

This project has shown that it is possible to create and run a single service for collecting environmental data and pushing related notifications to users, and also to make it run relatively stable. If there were to be more interest from the government, companies or the public in making environmental data more available, it can certainly be done, and probably expanded upon further in terms of finding new ways to make use of these data.

Bibliography

[1] NILU, *Nilu*, Accessed 10.05.2018. [Online]. Available: `https://www.nilu.no/`.

[2] ——, *Luftkvalitet.info*, Accessed 10.05.2018. [Online]. Available: `http://www.luftkvalitet.info/home.aspx`.

[3] Miljødirektoratet, *Norske utslipp*, Accessed 10.05.2018. [Online]. Available: `http://www.norskeutslipp.no/`.

[4] Nilu, *Nilu api docs*, Accessed 08.05.2018. [Online]. Available: `https://api.nilu.no/docs/`.

[5] EPA, *Basic information about no2*, Accessed 08.05.2018. [Online]. Available: `https://www.epa.gov/no2-pollution/basic-information-about-no2`.

[6] irceline, *What is pm10 and pm2.5?*, Accessed 08.05.2018. [Online]. Available: `http://www.irceline.be/en/documentation/faq/what-is-pm10-and-pm2.5`.

[7] greenfacts.org, *What is ozone (o3)*, Accessed 08.05.2018. [Online]. Available: `https://www.greenfacts.org/en/ozone-o3/l-3/1-presentation.htm`.

[8] globalccsinstitute.com, *What is carbon dioxide (co2)?*, Accessed 08.05.2018. [Online]. Available: `https://hub.globalccsinstitute.com/publications/what-happens-when-co2-stored-underground-qa-ieaghg-weyburn-midale-co2-monitoring-and-storage-project/1-what-carbon-dioxide-co2`.

[9] Microsoft, *Choosing between asp.net and asp.net core*, Accessed 21.04.2018. [Online]. Available: `https://docs.microsoft.com/en-us/aspnet/core/choose-aspnet-framework?view=aspnetcore-2.1`.

[10] T. P. Group, *Php*, Accessed 21.04.2018. [Online]. Available: `http://php.net/`.

[11] Microsoft, *Php open source*, Accessed 21.04.2018. [Online]. Available: `http://php.net/license/index.php`.

[12] Google, *Android studio*, Accessed 21.04.2018. [Online]. Available: `https://developer.android.com/studio/intro/index.html`.

[13] Apple, *Xcode*, Accessed 21.04.2018. [Online]. Available: `https://developer.apple.com/xcode/`.

[14] X. Inc, *Xamarin*, Accessed 24.04.2018. [Online]. Available: `https://www.xamarin.com/`.

[15] Microsoft, *Microsoft sql server*, Accessed 21.04.2018. [Online]. Available: https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation?view=sql-server-2017.

[16] O. Corporation, *Mysql*, Accessed 21.04.2018. [Online]. Available: https://dev.mysql.com/downloads/.

[17] opensource.org, *The mit license*, Accessed 08.05.2018. [Online]. Available: https://opensource.org/licenses/MIT.

[18] Chart.js, *Chart.js*, Accessed 08.05.2018. [Online]. Available: https://www.chartjs.org/.

[19] Google, *Google chart*, Accessed 08.05.2018. [Online]. Available: https://developers.google.com/chart/.

[20] Rotativa, *Rotativa.aspnetcore*, Accessed 08.05.2018. [Online]. Available: https://github.com/webgio/Rotativa.AspNetCore.

[21] V. Software, *Http api to convert html in pdf files*, Accessed 08.05.2018. [Online]. Available: https://rotativahq.com.

[22] Google, *Firebase cloud messaging*, Accessed 08.05.2018. [Online]. Available: https://firebase.google.com/products/cloud-messaging/.

[23] Apple, *Apns overview*, Accessed 08.05.2018. [Online]. Available: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html.

[24] Microsoft, *Sending push notifications from azure mobile apps*, Accessed 08.05.2018. [Online]. Available: https://developer.xamarin.com/guides/xamarin-forms/cloud-services/push-notifications/azure/.

[25] apache.org, *Apache license*, Accessed 08.05.2018. [Online]. Available: http://www.apache.org/licenses/LICENSE-2.0.

[26] J. Dick, *Pushsharp v4.0*, Accessed 08.05.2018. [Online]. Available: https://github.com/Redth/PushSharp.

[27] Microsoft, *Host asp.net core on windows with iis*, Accessed 08.05.2018. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/?tabs=aspnetcore2x&view=aspnetcore-2.1.

[28] ——, *Host asp.net core on linux with nginx*, Accessed 08.05.2018. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?tabs=aspnetcore2x&view=aspnetcore-2.1.

[29] ——, *Microsoft azure*, Accessed 08.05.2018. [Online]. Available: https://azure.microsoft.com/nb-no/.

[30] ——, *Pay-as-you-go*, Accessed 08.05.2018. [Online]. Available: https://azure.microsoft.com/nb-no/offers/ms-azr-0003p/.

[31] ——, *Asp.net mvc overview*, Accessed 08.05.2018. [Online]. Available: `https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx`.

[32] N.-O. Skeie, *Course sce2006 industrial information technology (iit)*, Lecture notes, Porsgrunn, 2nd Jan. 2017.

[33] Microsoft, *An introduction to nuget*, Accessed 13.05.2018. [Online]. Available: `https://docs.microsoft.com/en-us/nuget/what-is-nuget`.

[34] Twitter, *Bootstrap*, Accessed 08.05.2018. [Online]. Available: `https://getbootstrap.com/`.

[35] X. Inc, *Xamarin*, Accessed 24.04.2018. [Online]. Available: `https://docs.microsoft.com/en-us/xamarin/android`.

[36] Microsoft, *What is .net?*, Accessed 25.04.2018. [Online]. Available: `https://www.microsoft.com/net/learn/what-is-dotnet`.

[37] Google, *Google firebase*, Accessed 08.05.2018. [Online]. Available: `https://firebase.google.com/`.

[38] erwin, *Erwin data modeler*, Accessed 08.05.2018. [Online]. Available: `https://erwin.com/products/data-modeler/`.

[39] Microsoft, *Entity framework documentation*, Accessed 08.05.2018. [Online]. Available: `https://docs.microsoft.com/en-us/ef/`.

[40] ——, *Asp.net mvc core 2.0 middleware*, Accessed 25.04.2018. [Online]. Available: `https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-2.1&tabs=aspnetcore2x`.

[41] ——, *Jwt validation and authorization in asp.net core*, Accessed 25.04.2018. [Online]. Available: `https://blogs.msdn.microsoft.com/webdev/2017/04/06/jwt-validation-and-authorization-in-asp-net-core/`.

[42] J. Westlin, *Apply authorization by default in asp.net core*, Accessed 26.04.2018. [Online]. Available: `https://joonasw.net/view/apply-authz-by-default`.

[43] jquery.com, *Jquery.ajax()*, Accessed 13.05.2018. [Online]. Available: `http://api.jquery.com/jquery.ajax/`.

[44] gijgo.com, *Bootstrap datepicker*, Accessed 13.05.2018. [Online]. Available: `http://gijgo.com/datepicker/example/bootstrap`.

[45] jqueryui.com, *Datepicker*, Accessed 13.05.2018. [Online]. Available: `https://jqueryui.com/datepicker/`.

[46] dnviti, *Easy week picker widget for jquery ui - weekpicker.js*, Accessed 13.05.2018. [Online]. Available: `https://www.jqueryscript.net/time-clock/Easy-Week-Picker-Widget-For-jQuery-UI-weekPicker-js.html`.

[47] X. Inc, *V7 appcompat android support library*, Accessed 08.05.2018. [Online]. Available: `https://components.xamarin.com/view/xamandroidsupportv7appcompat`.

[48] oxyplot.org, *Oxyplot*, Accessed 08.05.2018. [Online]. Available: `http://www.oxyplot.org/`.

[49] Microsoft, *Azure functions support for .net core*, Accessed 07.05.2018. [Online]. Available: `https://azure.microsoft.com/en-us/roadmap/azure-functions-support-for-net-core/`.

[50] ——, *Cron expressions*, Accessed 07.05.2018. [Online]. Available: `https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer#cron-expressions`.

[51] www.w3schools.com, *Json - introduction*, Accessed 08.05.2018. [Online]. Available: `https://www.w3schools.com/js/js_json_intro.asp`.

[52] MILJØDIREKTORATET, *Kulde og tørt vær kan gi helseskadelig luft*, Accessed 14.05.2018. [Online]. Available: `http://www.miljodirektoratet.no/no/Tema/Luft/Lokal_luftforurensning/Kulde-og-tort-var-kan-gi-helseskadelig-luft/`.

Appendix A

Project Task Description

Høgskolen i Sørøst-Norge

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Environmental Public Health Information Management System

Supervisor: Hans-Petter Halvorsen, Nils-Olav Skeie

External Partners: Tel-Tek, Porsgrunn kommune, Sykehuset Telemark

Task Description:

Development and maintenance of infrastructure and data system for gathering, acquiring and analysis of environmental and public health information data for Porsgrunn and Grenland. Typical data for acquiring and analyzing are air pollution and air quality, emissions from industry, emissions from cars, number of cars, etc. Integrate the system with existing monitoring stations. The system will be important in the development of new industries, research and development areas.

The thesis includes the following key elements:

- Get an overview of data for acquiring and analysis
- Planning and System Design
- Database modeling, design and implementation
- Web Design and Web Development
- Mobile Development
- API Development
- Monitoring, Analysis, Statistics, and Presentation of Data. Generation of Reports in connection with analysis and reporting of data to authorities and other contracting authorities.
- Integration with existing monitoring stations.
- Deployment and Hosting

Task Background: USN has collaborated with Tel-Tek, Porsgrunn kommune, Sykehuset Telemark for a prototype of such a system. The goal is to develop and establish an operational system for registering and monitoring of data regarding environmental public health information.

Student Category: IIA students

Practical Arrangements:

Signatures:
Supervisor (date and signature):
Students (date and signature):

Appendix B

Design Sketches

Figure B.1 shows the dashboard web page. Figure B.2 show the trending web page sketch. Figure B.3 shows how the app might look like.

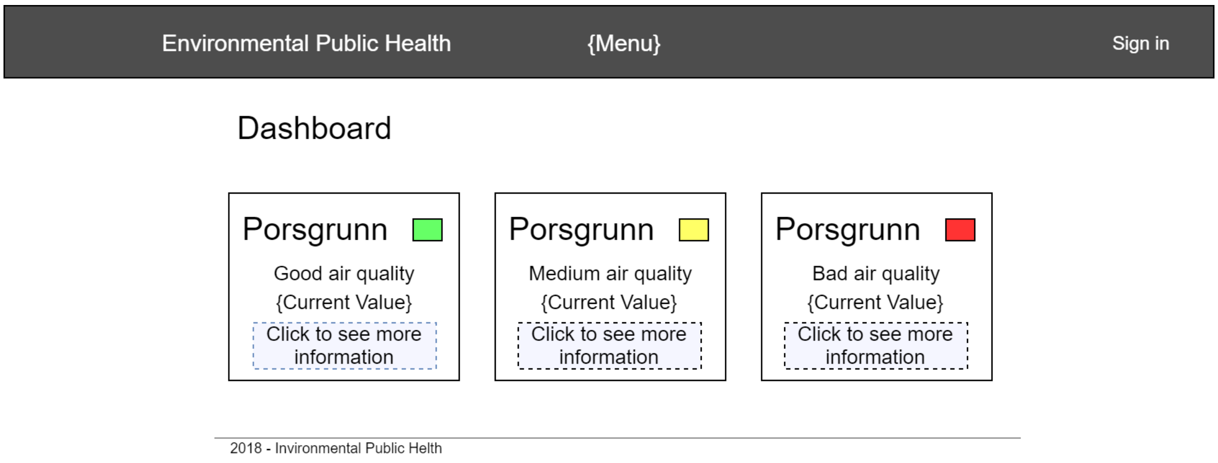

Environmental Public Health
{Menu}
Sign in
Dashboard
Porsgrunn
Good air quality
{Current Value}
Click to see more information
Porsgrunn
Medium air quality
{Current Value}
Click to see more information
Porsgrunn
Bad air quality
{Current Value}
Click to see more information
2018 - Invironmental Public Helth

Figure B.1: Website Dashboard

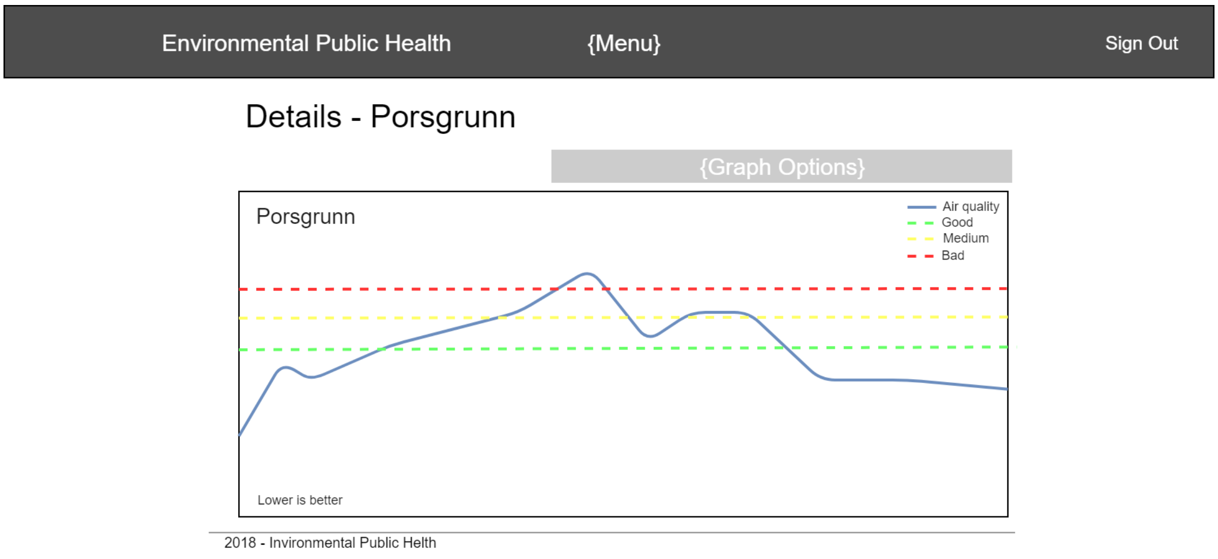


Figure B.2: Website Graph

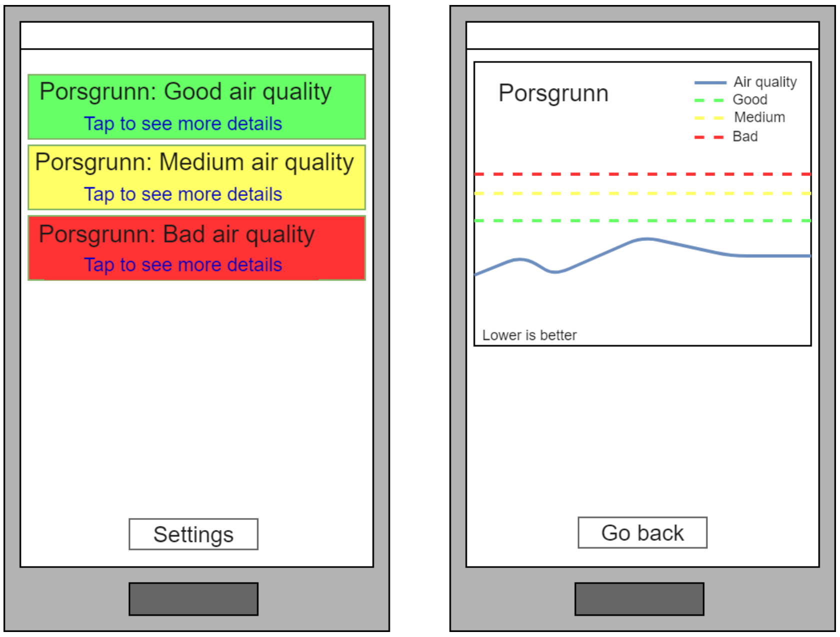


Figure B.3: Mobile-App sketches

Appendix C

Use Cases

The use cases for the website and mobile app can be seen in figure C.1. Figure C.2 show the use cases for the services.

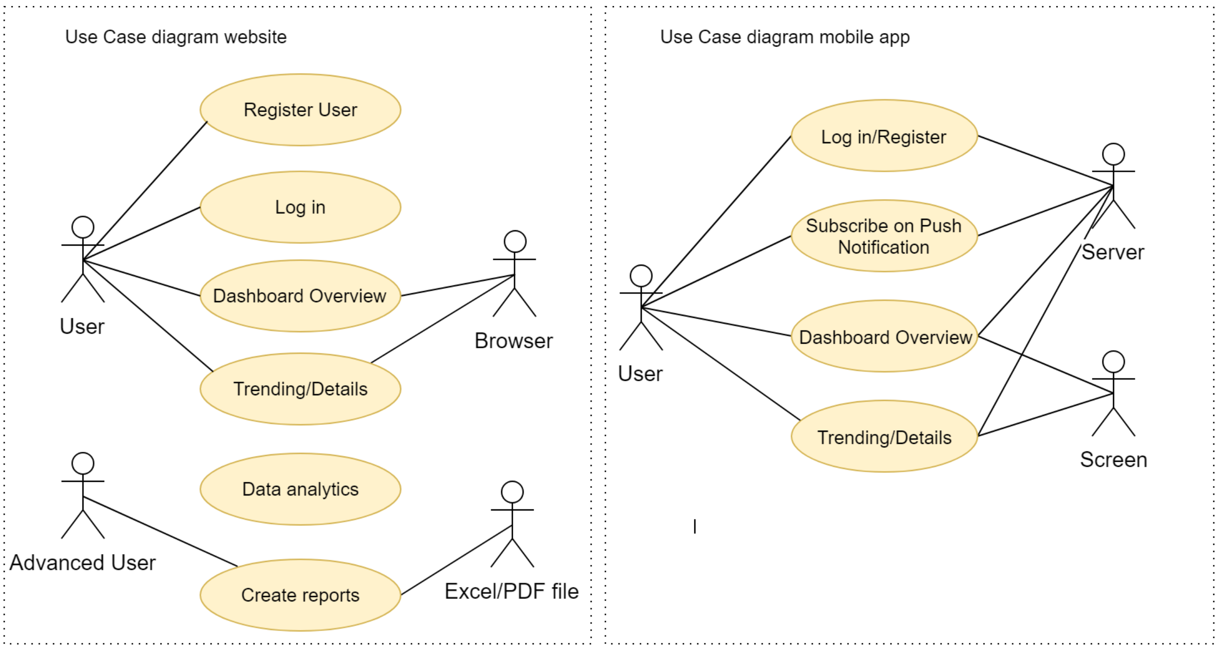


Figure C.1: Use case diagram from web site and mobile app

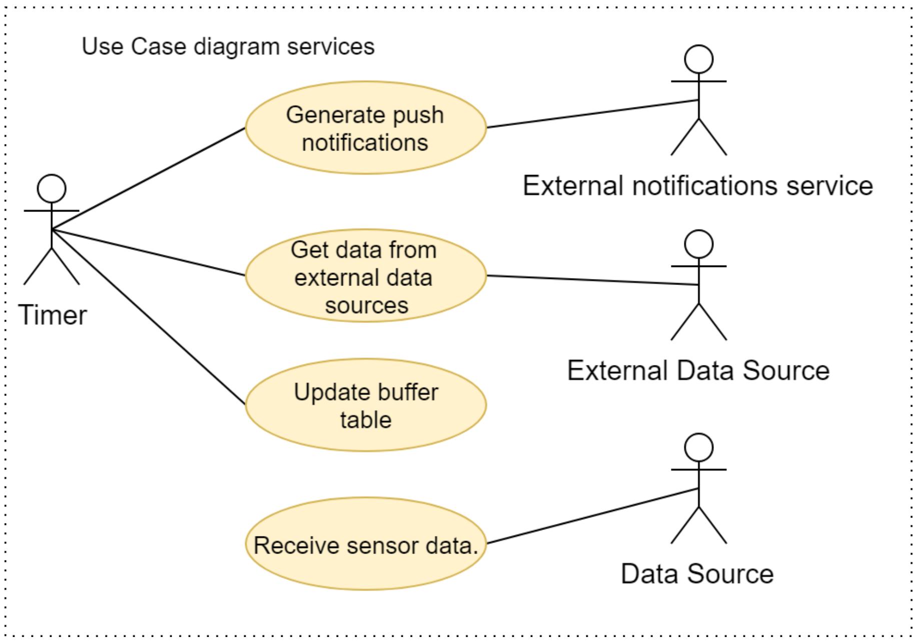
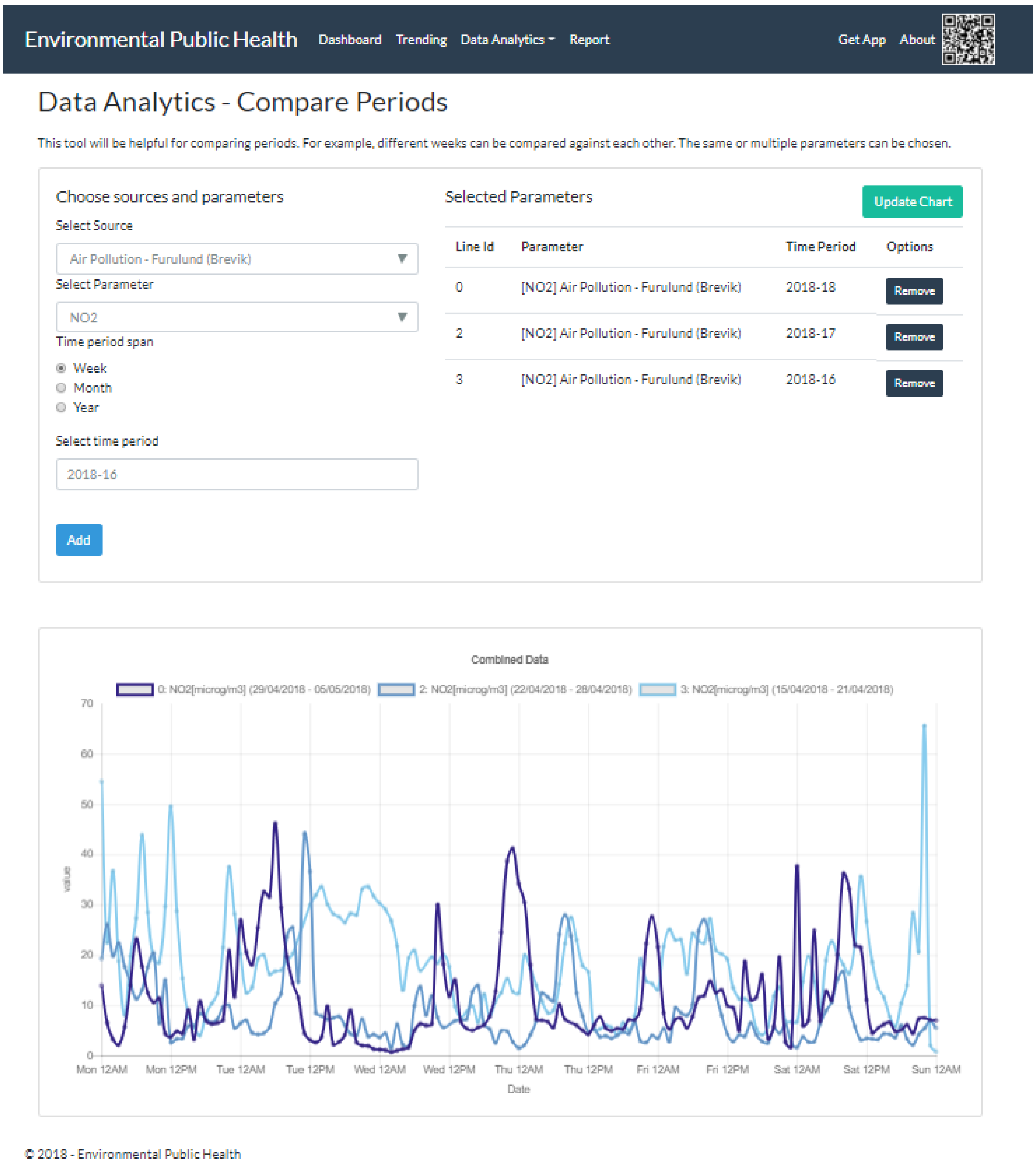

Use Case diagram services
Generate push notifications
Get data from external data sources
Update buffer table
Receive sensor data.
Timer
External notifications service
External Data Source
Data Source

Figure C.2: Mobile-App Use Cases

Appendix D

Data Analytics - Compare Periods



Data Analytics - Compare Periods

This tool will be helpful for comparing periods. For example, different weeks can be compared against each other. The same or multiple parameters can be chosen.

Choose sources and parameters

Select Source: Air Pollution - Furulund (Brevik)

Select Parameter: NO2

Time period span: [✓ Week] Month Year

Select time period: 2018-16

Selected Parameters

Line Id	Parameter	Time Period	Options
0	[NO2] Air Pollution - Furulund (Brevik)	2018-18	Remove
2	[NO2] Air Pollution - Furulund (Brevik)	2018-17	Remove
3	[NO2] Air Pollution - Furulund (Brevik)	2018-16	Remove

© 2018 - Environmental Public Health

Figure D.1: Data analytic - compare periods

Appendix E

Generated Report by the System

Environmental Public Health Report Document

Generated: **05/13/2018**

About the report:

Contain statistic and history data. All measurements within one month is combined, were the average value is used for plotting.

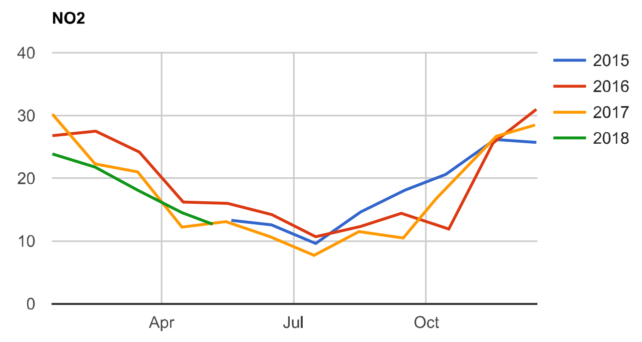
Sources included in this report:

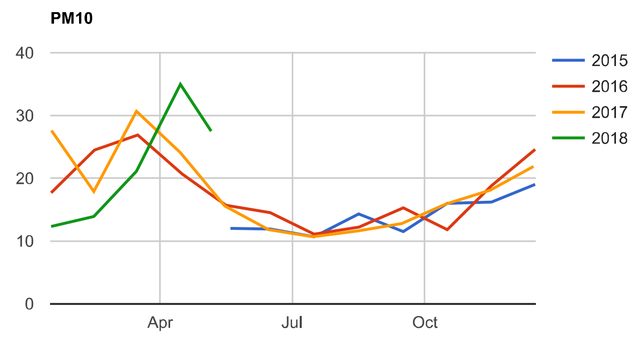
Source name	Area	Parameters	Period
Air Pollution	Sverresgate (Porsgrunn) - Grenland	[NO2] [PM10]	2015 - 2018
Air Pollution	Lensmannsdalen (Skien) - Grenland	[NO2] [PM10] [PM2.5]	2015 - 2018

Source name: Sverresgate (Porsgrunn) - Air Pollution

Statistics about the source parameters:

Parameter Name	Unit	Average	Min	Max	Status intervals
NO2	microg/m3	18.2	0.2	96.1	Low [-5-100] Medium [100-200] High [200-400] Very High [400-9999999]
PM10	microg/m3	22	-3.4	1192.3	Low [-5-50] Medium [50-80] High [80-400] Very High [400-9999999]





Yearly statistics about the source parameters:

Parameter name: **NO2**

Average	Min	Max	Year
25.7	-1.3	86.9	2015
31	-0.3	107.8	2016
28.5	-0.6	113.7	2017
12.7	0.2	96.1	2018

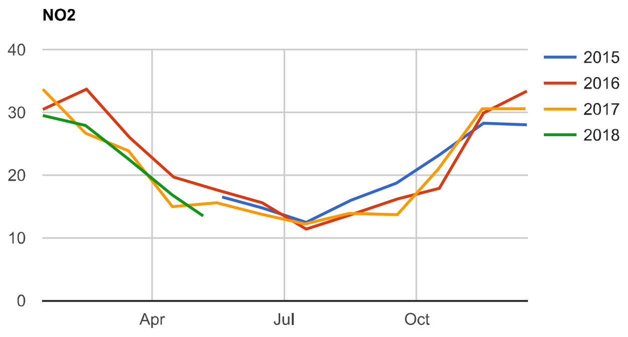
Parameter name: **PM10**

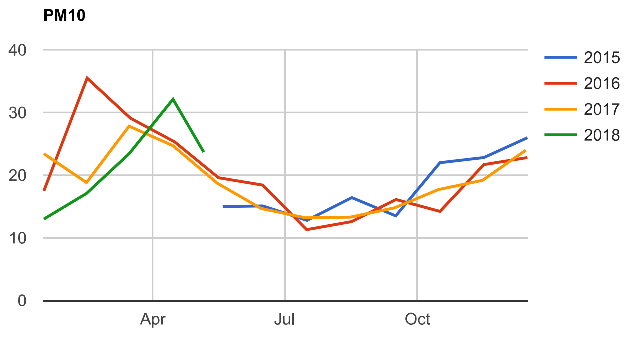
Average	Min	Max	Year
19	-4.6	178.5	2015
24.6	-4.8	1029.9	2016
21.9	-4.5	423.1	2017
27.5	-3.4	1192.3	2018

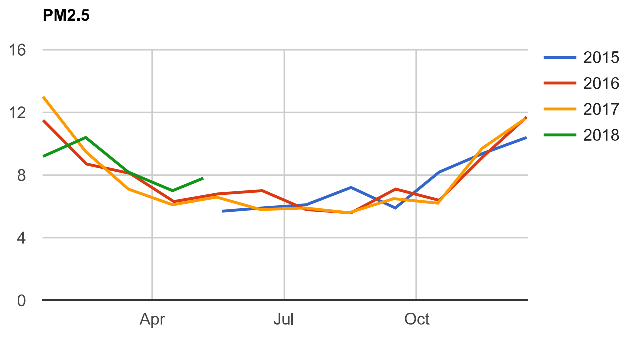
Source name: Lensmannsdalen (Skien) - Air Pollution

Statistics about the source parameters:

Parameter Name	Unit	Average	Min	Max	Status intervals
NO2	microg/m3	22.1	-2.3	103.5	Low [-5-100] Medium [100-200] High [200-400] Very High [400-9999999]
PM10	microg/m3	21.9	0.1	358.7	Low [-5-50] Medium [50-80] High [80-400] Very High [400-9999999]
PM2.5	microg/m3	8.5	-1.7	123.3	Low [-5-25] Medium [25-40] High [40-150] Very High [150-9999999]







Yearly statistics about the source parameters:

Parameter name: **NO2**

Average	Min	Max	Year
28	-0.2	263.8	2015
33.4	0.1	101.1	2016
30.6	-1	104.6	2017
13.5	-2.3	103.5	2018

Parameter name: **PM10**

Average	Min	Max	Year
26	-2.6	214.8	2015
22.8	-3.6	520.9	2016
24	-3.1	360.1	2017
23.7	0.1	358.7	2018

Parameter name: **PM2.5**

Average	Min	Max	Year
10.4	-3.3	50.8	2015
11.7	-3.6	114	2016
11.6	-2.7	194.6	2017
7.8	-1.7	123.3	2018