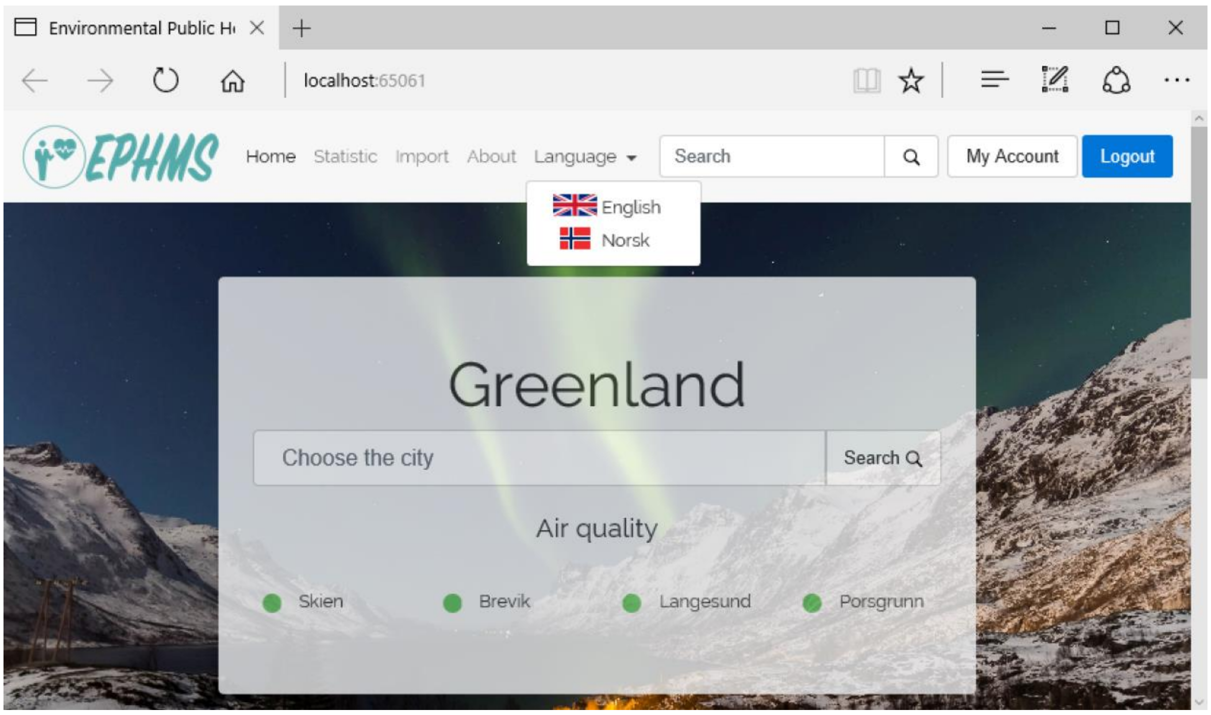
University College of Southeast Norway

www.usn.no

FMH606 Master's Thesis 2017

Industrial IT and Automation

Environmental Public Health Information Management System



Artem Chynchenko

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

University College of Southeast Norway

www.usn.no

Course: FMH606 Master's Thesis, 2017

Title: Environmental Public Health Information Management System

Number of pages: 50

Keywords: ASP.Net Core, SQL, MSSQL, HTML, CSS, JavaScript, MVC

Student:	Artem Chynchenko
Supervisor:	Hans-Petter Halvorsen
External partner:	Tel-Tek, Porsgrunn municipality/Porsgrunn Kommune, Telemark Hospital-Department of Occupational Medicine/Sykehuset Telemark
Availability:	Confidential

Approved for archiving:
(supervisor signature)

(Hans-Petter Halvorsen)

Summary:

The goal of the project is a web application that gives public access to the air quality and environmental measurements from the database.

ASP.Net Core had been selected as a primary programming language to create web application. The assist technologies are MSSQL, HTML, CSS, Telerik Kendo UI and Bootstrap. Visual Studio 2017 was used as programming environment for ASP.Net Core. SQL Server 2014 Management Studio and erwin Data Modeler r9.7 used to work with SQL.

Home, statistic pages to retrieve and import page to seed measurements was implemented. Import use Excel file as data source. Authentication and authorization were developed for web application security. Relevant Login and register page created.

Preface

Despite the experience in programming using C# language, the project was challenging. A new approach and paradigm of programming the web application development required more attention and knowledge enhancement. Instead of using inheritance of the classes “is a” relationship, dependency injection principle “has a” relationship. Razor engine technology with Razor syntax and TagHelpers are powerful features. Model binding for mapping data and processing the HTTP requests. Only with listed feathers development would be possible.

The main objectives and task description could be found in Appendix A.

To understand the report the reader should be familiar with C# 6.0 which is ASP.NET Core uses. But the knowledge of the C# 5.0 should be enough. Knowledge of SQL Server 2008+ is beneficial. LINQ is used for querying the Database. Basic knowledge of LINQ is required.

Also, recommended basic knowledge of the HTML5, CSS and JavaScript. Knowledge of the Bootstrap and jQuery libraries.

The reader should be familiar with Visual Studio 2017 interface, as well as installing NuGet and Bower packages. Knowledge of the Entity Framework and Identity Entity Framework is a plus.

The reader should understand MVC and Repository patterns, along with understanding OOP principals.

I want to express my gratitude to the external partners for such un interesting project and to supervisor Hans-Petter Halvorsen for the suggestion to use ASP.Net Core. Alexander Zhang Gjerseth and Lucille Ang and all people who had helped them for good the report and database this master project is based on.

The project folder along with database creation scripts, required seed scripts and import data are coming as an attached .zip file.

The final version of the project was hosted in the Microsoft Azure service at http://ephms-app.azurewebsites.net as well as the database.

Porsgrunn, 15. May 2017

Artem Chynchenko

Nomenclature

.Net	–	the list of computer programming languages that are used to produce libraries known as CLI languages (1)
ASP	–	Active Server Pages
CLI	–	Common Intermediate Language
CSS	–	Cascading Style Sheets
DB	–	Data Base
HTML	–	Hypertext Markup Language
IIS	–	Internet Information Services
JS	–	JavaScript
JSON	–	JavaScript Object Notation
MSSQL	–	RDBM developed by Microsoft
MVC	–	Model–view–controller
noSQL	–	"non-relational" or "not only SQL"
RDB	–	Relational Data Base
RDBMS	–	relational database management system
SQL	–	Structured Query Language
VDS	–	Virtual Dedicated Server
VPS	–	Virtual Private Server
XML	–	Extensible Markup Language
FURPS+	–	Functionality-Usability-Reliability-Performance-Supportability-Pluss various attributes (2)
FDUCD	–	Fully dressed Use case document
CRUD	–	Create/read/update/delete

RI Action	–	Referential Integrity Action
LINQ	–	Language Integrated Query

List of Figures

Figure 2-1: Simple HTML page.

Figure 2-2: HTML page styled with CSS.

Figure 2-3: HTML page before button “Try it!” is pressed.

Figure 2-4: HTML page after button “Try it!” is pressed.

Figure 2-5: Link tag styled with Bootstrap.

Figure 2-6: Nav tag styled with Bootstrap.

Figure 2-7: Telerik DatePicker element.

Figure 2-8: ASP.Net overview.

Figure 2-9: Middleware communication in ASP.Net Core.

Figure 2-10: ASP.Net Core “Hello World!” run result.

Figure 2-11: “Hello world!” page source code.

Figure 2-12: Model view controller overview.

Figure 4-1: Global representation of the application.

Figure 4-2: “USER” table.

Figure 4-3: “PASSWORD” table.

Figure 4-4: “USERROLE” table.

Figure 4-5: “LOGINATTEMPT” table.

Figure 4-6: “MEASUREMENT” table.

Figure 4-7: “PROPERTY” table.

Figure 4-8: Entity frameworktables.

Figure 4-9: “__EFMigrationsHistory” table.

Figure 4-10: Main structure tables.

Figure 4-11: File and folder structure (without Views folder).

Figure 4-12: View folder structure.

Figure 4-13: Home page.

Figure 4-14: Import page.

Figure 4-15:Import from Excel.

Figure 4-16: Login page.

Figure 4-17: Register page.

Figure 4-18: Statistic page.

List of Tables

Table 3-1: Requirements overview.

Table 3-2: Overview of different hosting environment.

Table 3-3: Overview of the servers.

Contents

1 .. **Introduction** 10
1.1 Report structure 10
2 .. **Theory** 11
2.1 SQL 11
2.1.1 Tables *11*
2.2 HTML 11
2.3 CSS 12
2.4 JavaScript 12
2.5 Bootstrap 13
2.6 Telerik Kendo UI 13
2.7 ASP.NET Core 1.1 14
2.7.1 What is ASP.NET Core 1.1? *14*
2.7.2 Middleware *15*
2.7.3 Hello World! *16*
2.7.4 MVC *17*
2.7.5 Razor *18*
2.7.6 TagHelpers *19*
3 .. **System description** 20
3.1 Customer and technical requirements 20
3.2 Hosting and deployment 20
3.2.1 Hardware configuration *20*
3.2.2 Server configuration/software *21*
3.3 Analysis 22
3.3.1 FURPS+ *22*
4 .. **Results** 24
4.1 Database 24
4.1.1 Naming convention *24*
4.1.2 Redesign and update of the database *24*
4.1.3 Database identity migration *27*
4.1.4 Result *28*
4.2 Web application 29
4.2.1 Create Web application *29*
4.2.2 Configuring NuGet and Bower packages *30*
4.2.3 MVC structure *31*
4.2.4 Entity Framework core *32*
4.2.5 Core Identity *32*
4.2.6 Repository pattern *33*
4.3 Views 34
4.3.1 Home page *35*
4.3.2 Import *35*
4.3.3 Import from Excel *35*
4.3.4 Statistic *36*
4.3.5 Login/Register *36*
5 .. **Discussion** 38
6 .. **Summary** 39

7 .. Future work 40
7.1 Business logic 40
7.2 RESTfull API 40
7.3 AJAX 40
7.4 Globalization and localization 40
Appendix A 43
Appendix B 45
Appendix C 48
Appendix D 49

1 Introduction

One of challenges for modern world is a control over air pollution. The air quality affects not only people health, but the planet itself with global warming. In order take over the control this problem the United Nations Framework Convention on Climate Change agreement was signed between countries, including Norway. Its mission in regulation of the greenhouse gas produced by human activity.

To fulfil the requirements over air pollution from the agreement the external censors Tel-Tek, Porsgrunn municipality/Porsgrunn Kommune, Telemark Hospital-Department of Occupational Medicine/Sykehuset Telemark issued the project for creating the Environmental Public Health Information Management System. The system must collect data about air quality from local stations and make them publicly available.

There are several similar systems that already exist, but most of the them includes only area general information. Environmental Public Health Information Management System should be designed specifically for Porsgrunn Kommune and local industry, but should be flexible enough to add another municipalities.

The goal of this thesis is to continue system development based on the previous projects completed at University Collage of South-East Norway. In the previous work, the focus was on the database design and data import. Simple PHP representation was also created. The list of the objectives can be found in Appendix A. The task involved:

- Overview of available tools and methods of web application development;
- Choice of architecture for the database: SQL, noSQL;
- Development of secure infrastructure for user registration/login;
- Web application logic and infrastructure development;
- Make import functions from excel files.
- Data presentation;
- Hosting of the data;

The tasks that were for the future work:

- RESTfull API development;
- Exporting of the data;

1.1 Report structure

This report consists 7 chapters. Chapter 2 very briefly describes the theory of the methods that was used during development of the web application. Chapter 3 will give the understanding of the hosting environments and capabilities. Chapter 4 and 5 discuss the results that was achieved and recommendations. Chapter 6 will give a short summary about achieved result. Final chapter 7 will give the overview of the future work.

2 Theory

This chapter give an overview of the theory used to during application development.

2.1 SQL

This paragraph will show SQL basics

2.1.1 Tables

The code below will generate a table:

```
CREATE TABLE TAB
(
       Col1  int  NOT NULL  IDENTITY ( 1,1 ) ,
       Col2  varchar()  NOT NULL ,
       PRIMARY KEY CLUSTERED (Col1 ASC)
)
go
```

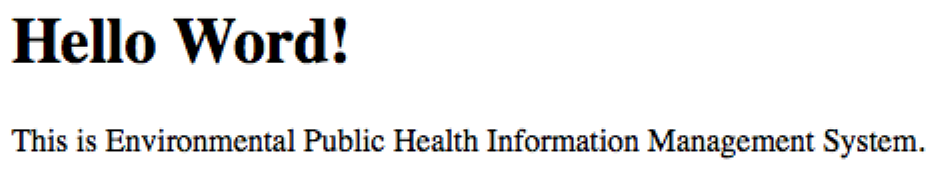
It creates a table with int and varchar column, both are not null. The Col1 has identity that starts from 1, each new value increasing by 1. Col1 used as a primary key is another table.

2.2 HTML

HTML is an industry standard to markup the Web page. It stands for Hyper Text Markup Language. HTML elements are like building blocks represented by tags. `<html>`, `<head>` and `<body>` are required to be in the .html document.(3)
Simple HTML document will look as follows:

```
<!DOCTYPE html>
<html>
<head>
<title>EPHMS</title>
</head>
<body>

<h1>Hello Word!</h1>
<p> This is Environmental Public Health Information Management System. </p>

</body>
</html>
```

If you will open it with the browser you will see the result as on the Figure 2-1



Hello Word!

This is Environmental Public Health Information Management System.

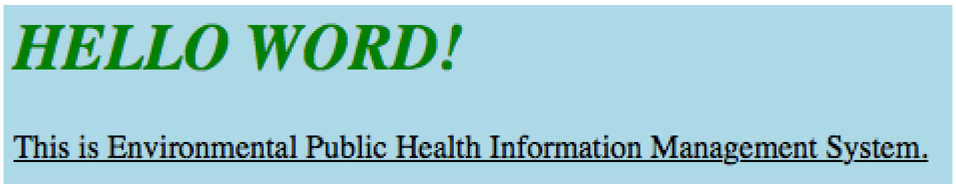
Figure 2-1: Simple HTML page.

2.3 CSS

CSS (Cascading Style Sheets) – helps to style the HTML styles. They declare how the page will look like. After adding next styles to our document:

```
body {
    background-color: lightblue;
}

h1 {
    color: green;
    text-transform: uppercase;
    font-style: italic;
}
p {
    text-decoration: underline;
}
```

we will get Figure 2-2.


HELLO WORD!

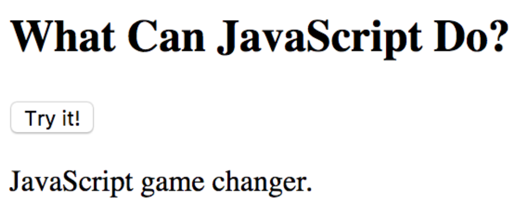
This is Environmental Public Health Information Management System.

Figure 2-2: HTML page styled with CSS.

2.4 JavaScript

JavaScript is a programming language which helps to make client side pages interactive. This JavaScript code is changing the font size in any tag that has an id that equals to “demo” from 16px to 32px when the function `myFunction()` is called. In this case size changes when user is clicking on the button, see Figure 2-3 and Figure 2-4.

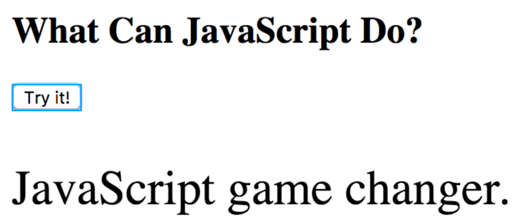
```
<!DOCTYPE html>
<html>
<body>
<h2>What Can JavaScript Do?</h2>
<button type="button" onclick="myFunction()">Try it!</button>
<p id="demo">JavaScript game changer.</p>

<script>
var x = false;
function myFunction() {
        if (x) {
                document.getElementById('demo').style.fontSize='16px';
                x=false;
        } else {
                document.getElementById('demo').style.fontSize='32px';
                x=true;
        }
}
</script>
</body>
</html>
```


What Can JavaScript Do?

Try it!

JavaScript game changer.

Figure 2-3: HTML page before button “Try it!” is pressed.


What Can JavaScript Do?

Try it!

JavaScript game changer.

Figure 2-4: HTML page after button “Try it!” is pressed.

2.5 Bootstrap

Bootstrap is the CSS and JavaScript framework which make easier to create responsive web pages.(4)

The framework consists of components and options for page layout. It has options for different screen resolutions like: Extra small <576px, Small ≥576px, Medium ≥768px, Large ≥992px, Extra-large ≥1200px. Html components comes with prebuild styles. The most useful among them are forms, buttons, links(Figure 2-5), navbars(Figure 2-6), input groups etc.(5)

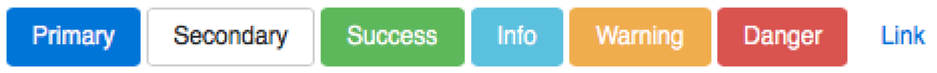


Figure 2-5: Link tag styled with Bootstrap.

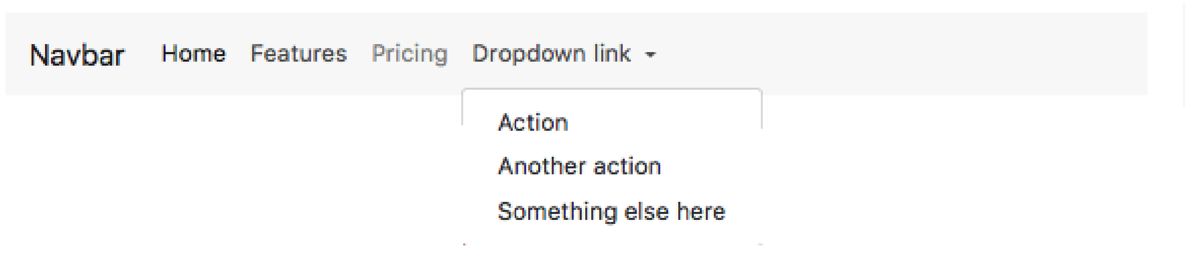


Figure 2-6: Nav tag styled with Bootstrap.

2.6 Telerik Kendo UI

Telerik has UI’s for different environments. Kendo UI for jQuery was used in this project.

As documentation states by imbedding next code to the project, it will generate a DatePicker element:

```
"<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link rel="stylesheet" href="styles/kendo.common.min.css" />
    <link rel="stylesheet" href="styles/kendo.default.min.css" />
    <link rel="stylesheet" href="styles/kendo.default.mobile.min.css" />

    <script src="js/jquery.min.js"></script>
    <script src="js/kendo.all.min.js"></script>
</head>
<body>

        <div id="example">
            <div class="demo-section k-content">

                <h4>Show e-mails from:</h4>
                <input id="datepicker" value="10/10/2011" title="datepicker" style="width:
100%" />
            </div>
        <script>
            $(document).ready(function() {
                // create DatePicker from input HTML element
                $("#datepicker").kendoDatePicker();
            });
        </script>
        </div>

</body>
</html>"
```

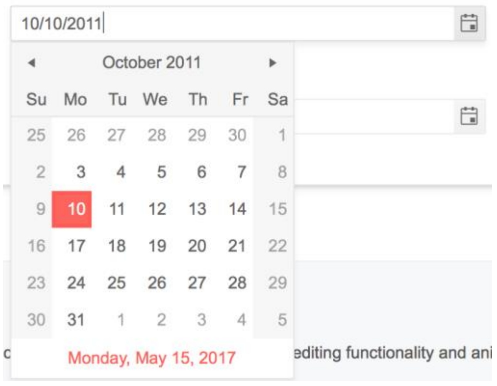


Figure 2-7: Telerik DatePicker element.

2.7 ASP.NET Core 1.1

This paragraph is a brief overview of the ASP.Net Core

2.7.1 What is ASP.NET Core 1.1?

ASP.NET Core 1.0 is a logical continuation of ASP.NET 4.6. Next framework should get the name of ASP.NET 5. But because of the big update, absolutely new approach and libraries it is called ASP.NET Core 1.0 (6) Figure 2-8.

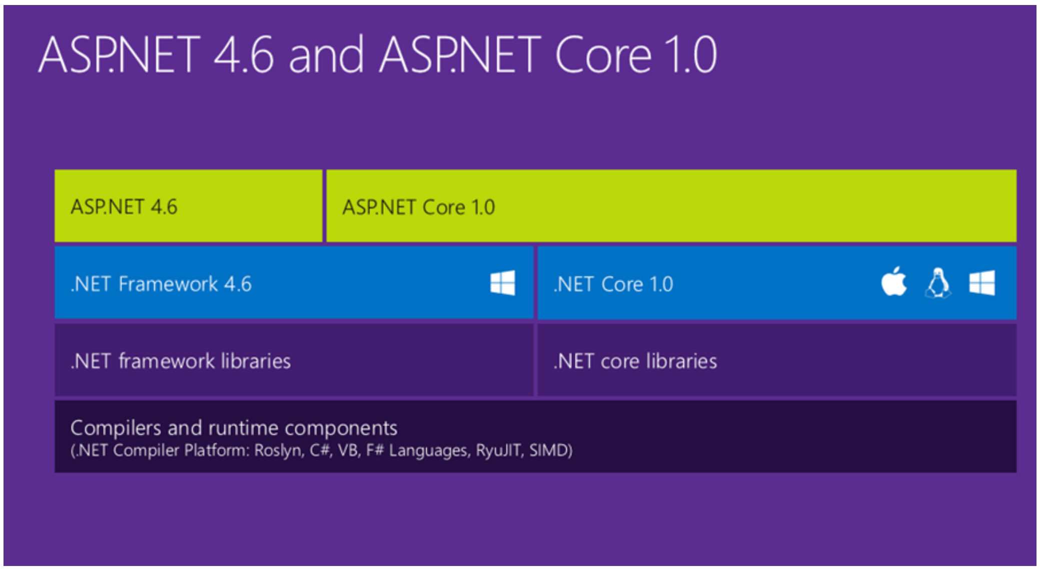

ASP.NET 4.6 and ASP.NET Core 1.0
ASP.NET 4.6
ASP.NET Core 1.0
.NET Framework 4.6
.NET Core 1.0
.NET framework libraries
.NET core libraries
Compilers and runtime components
(.NET Compiler Platform: Roslyn, C#, VB, F# Languages, RyuJIT, SIMD)

Figure 2-8: ASP.Net overview.

ASP.NET Core relatively new, written from scratch, open-source programing language developed by Microsoft. It is intended for building modern and cloud based services and applications like web apps, IoT, etc. It is cross-platform. Mac OS, Linux and Windows could be used for developing and hosting such apps and services (7).

The flexibility, consistency, high performance and lightweight of the framework archived by its modularity. Those modules could be server sided like: ASP.NET Core MVC, Entity Framework, Identity Entity Framework, Razor syntax, etc. As well as, client sided like: Bootstrap, jQuery, AngularJS, Node.js. NuGet and Bower packages serve this purpose.

2.7.2 Middleware

Middleware is a software with its logic to process the request of the user and provide the response if it is required by the request data. This software blocks create “pipeline” shown on the Figure 2-9. Each middleware decides whether to add logic or to pass it further to the next one. The request first goes forward through and then returning though each middleware. The logic could be added on any step.(8)

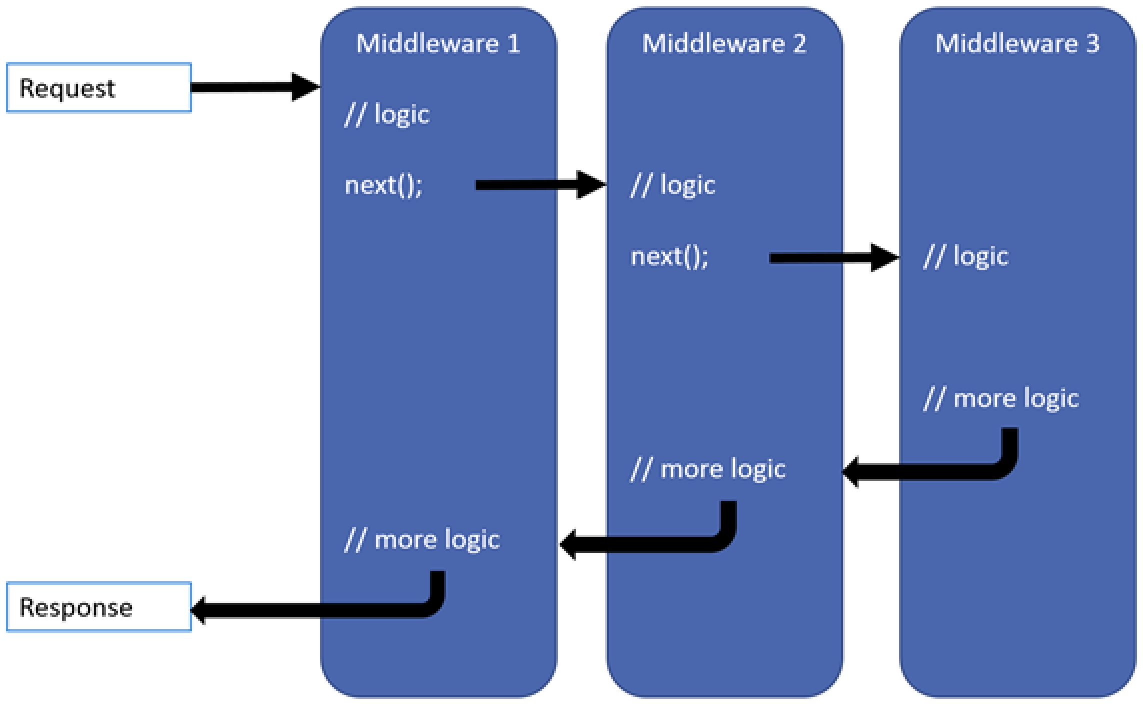


Figure 2-9: Middleware communication in ASP.Net Core.

2.7.3 Hello World!

The ASP.NET Core code below is the “main” class of the web app project. After compiling the code the program will represent the server side to run on server.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace WebApplication1
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the
container.
        public void ConfigureServices(IServiceCollection services)
        {
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory
loggerFactory)
        {
            loggerFactory.AddConsole();

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.Run(async (context) =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        }
    }
}
```

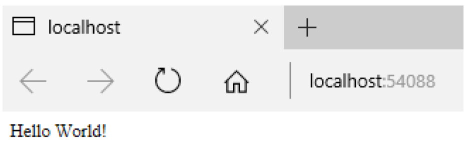
This file is named Startup.cs. Every `using` statement is loading required library to compile the app. It is required, by convention, that the Startup.cs file should contain Startup class inside the app namespace. Startup class is used to configure services inside

`ConfigureServices()` method used to specify additional features and options if some required for the `Configure()` method. `ConfigureServices()` method can be omitted . It is required to call it before the `Configure()` method if `ConfigureServices()` is used.

Configure method represents how the application will react to the HTTP request. It constructs such a called "pipeline" of the middleware which will process user request. (9)

The `Configure()` method "pipeline" is shown on the picture. After `Configure()` method receives the request the middleware starts to process it. (8)

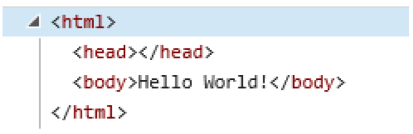
The middleware inside the `Configure()` method can generate the HTML code. In the code example the "Hello World!" text does not contain any HTML formatting. After compiling and starting the application a result of the `app.Run()` middleware will be shown on the Figure 2-10.



Hello World!

Figure 2-10: ASP.Net Core "Hello World!" run result.

The page shows just "Hello World!", but page source code shows minimal html formatting, see Figure 2-11.



```
<html>
  <head></head>
  <body>Hello World!</body>
</html>
```

Figure 2-11: "Hello world!" page source code.

2.7.4 MVC

MVC stands for Model-View-Controller Figure 2-12. This pattern was developed to maintain separation of concerns. It is divides the application into three pieces. This pattern leads to more clean and understandable code, makes scale of the applications much easier. Update, test and debug are much easier with this pattern. As it is stated in the documentation you can keep the business logic without changes if it is meet the requirements, but change the client design more frequently. (10)

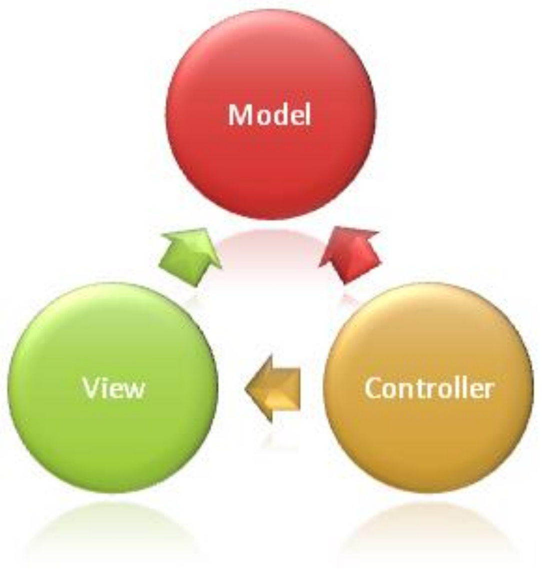


Figure 2-12: Model view controller overview.

Model usually represents all settings and data inside the application. The application uses this business logic as a data storage typically represented by database, XML or JSON files. For the views like website pages it is recommended to use View Model types. They only hold the information that will be described on the page. (10)

View is showing the content to the use. View uses Razor view engine. Razor view engine processes the Razor markup syntax, .Net code, TagHelpers, etc. to generate the HTML page.

Views are ONLY displaying and requesting the information.(10)

Controller is maintaining the interaction between view and model. He decides what data from Model will be shown and which view will display the data.(10)

To use the MVC services must be registered. This is done by adding next line of code to the `ConfigureServices()` method:

```
services.AddMvc();
```

And the middleware must be set up in the `Configure()` method with the code:

```
app.UseMvcWithDefaultRoute();
```

2.7.5 Razor

Razor is a markup syntax design specifically to run-in server base code into regular HTML page. If the user want to see current date and time each time the page is reloaded the programmer can use Razor syntax inside the View to execute the C# code (11), as follows:

```
<p>@DateTime.Now</p>
<p>@DateTime.IsLeapYear(2016)</p>
```

2.7.6 TagHelpers

TagHelpers is a special back-end code helper for the Razor engine. They help the views look more like HTML syntax. (12)

Instead of using:

```
@using (Html. Begin Form("Register”, “Account”, FormMethod.Post, new { @class form-horizontal
}))
{
 @Html.AntiForgeryToken()
…
}
```

The code above is (Razor markup syntax) creates a Form. After the Razor engine will process the it the page will have regular HTML form. That syntax is new for the front-end developers. That is why they can use TagHelpers instead. The code below is identical, but it is more understandable for the programmer(12):

```
<form asp-controller="Account" asp-action="Register" asp-antiforgery="true" method="post"
class="form-horizontal" >
…
</form>
```

This code looks more like regular HTML. `asp-controller`, `asp-action` and `asp-antiforgery` are the TagHelpers.(12)

3 System description

3.1 Customer and technical requirements

The customer and technical requirement listed in the Table 3-1.

Table 3-1: Requirements overview.

Customer requirement:	Technical requirement:
1) Data available online via website; 2) Data should be to open to the public; 3) Statistics page should present data with graphs, tables and/or diagrams; 4) Report generation; 5) Flexible import/export should be implemented the excel sheets or .csv files; 6) System should be safe, secure and reliable; 7) System should store data about air pollution and quality, emissions from industry, cars gases emissions which are collected with different systems (weather stations, etc.); 8) REST API should be implemented.	– Scrum method used for development with 2 period; – Front-end should use HTML + CSS + JavaScript + jQuery; – JS, CSS libraries used in the project: o Bootstrap; o jQuery; o Flowtype.js – Back-end with ASP.Net Core as executable on the Web Server; – ASP.Net Core MVC used for low coupling of the component; – Hosting and deployment: Azure, USN Server or Dedicated hosting server (customer choice); – Import/export of the data; – Data stored in the SQL DB; – MS SQL Server used for the database; – Safety system using User/User Roles functionality.

3.2 Hosting and deployment

3.2.1 Hardware configuration

In this paragraph will be discussed what hardware the customer can use for hosting the web application.

There are three options available to host an application, see Table 3-2:

- Own server;
- Dedicated server / VPS, VDS;
- Microsoft Azure.

Own server / Dedicated server is a kind of hosting in which the whole client is provided as a separate physical machine (as opposed to virtual hosting).

VDS/VPS is a server which shares resources of a real physical server with another VDS/VPS. The user can receive maximum rights root for Unix and Administrator for Windows. VDS / VPS emulate the work of a real server. The user can install custom OS and software that is required.

Microsoft Azure has specialized services like “WebApp” and “SQL Database” or “Web App + SQL” for running ASP.NET Core app.

Table 3-2: Overview of different hosting environment.

	Own server	Dedicated server	VPS, VDS	Microsoft Azure
Full control (OS, software, rights)	+	+	+	-
Limitations (number of websites, DB, domains etc.)	-	-	-	-
Independence (resources, etc.)	+	+	+	+
Necessity of administration	+	+	+	-
Price	Server price + resources for maintenance price	Monthly or yearly fees	Monthly or yearly fees	Monthly or yearly fees/ Pay as you go

3.2.2 Server configuration/software

ASP.NET Core application are cross-platform, it can be installed to any operating system as Windows, MacOS or any Unix system(Linux).(13)

Documentations only have recommendations which server to use see Table 3-3

Table 3-3: Overview of the servers.

Windows:	IIS, Windows Service

MacOS:	Apache HTTP Server*, nGinx*
Unix:	

* – possible to use with Windows but recommended.

3.3 Analysis

In this chapter, the analysis of the project will be covered.

3.3.1 FURPS+

Functionality – describes the use cases and functions of the application(2):

- Application should be able to perform CRUD data requests in the Models for instances of:
  - Country;
  - Area;
  - City;
  - Emission source;
  - Measurement;
  - Measurement groups;
  - Measurement type;
  - Measurement unit;
  - Measurement unit groups;
  - Measurement station;

within the Controllers independently from a source: user request (hand input, excel sheet, WebAPI);

- Generate the Views;
- Application should handle register/login/logout requests;
- Use HTTP protocol to interact with application;

Usability – describes the interaction of the user with the application:

- Client side should use HTML + CSS + JavaScript + jQuery;
- Change filter setting for the measurement request;
- Data displays with raw value in table/chart;

Reliability:

- System should work 24/7.
- Tier Level III-IV.

Performance – describes what amount of the resources system will consume:

- Database storage minimum 20 gigabytes;

- Ram capacity minimum 256mb;
- Application storage minimum 1gigabyte;
- CPU minimum performance dual core 2.7GHz.

Supportability:

- OS: Windows, Usix, MacOS;
-

+:

- Application languages: English, Norwegian;

4 Results

This chapter is will show the process of creation of the application. The application conditionally divided into blocks that you can observe on Figure 4-1.

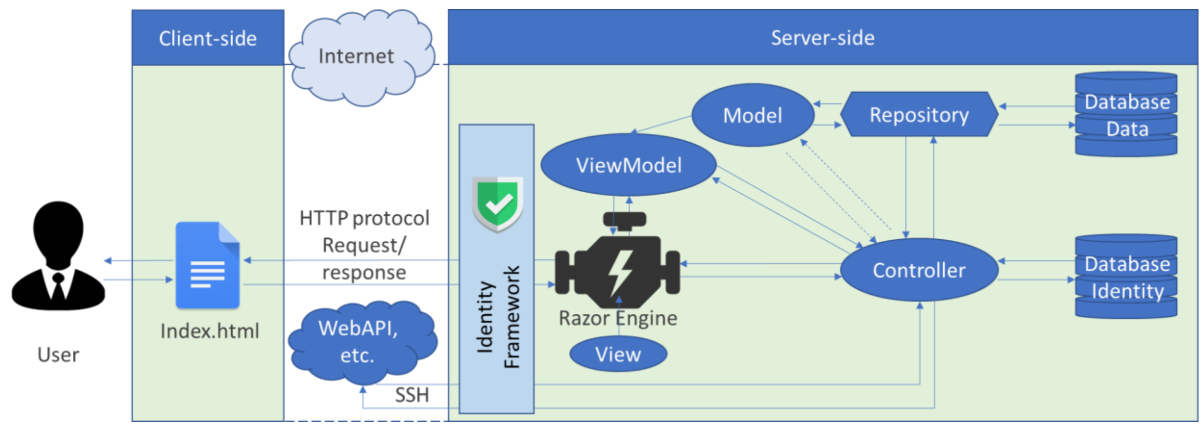


Figure 4-1: Global representation of the application.

4.1 Database

The database from paragraph 2.1 has been updated and modified not only using SQL modeling tool Erwin described in paragraph 2.1.

4.1.1 Naming convention

The renaming of the tables in the database was performed according to the naming conventions rulers:

- Tables name should use uppercase.
- Column name should include table name + column name without spaces, capitalizing each word.

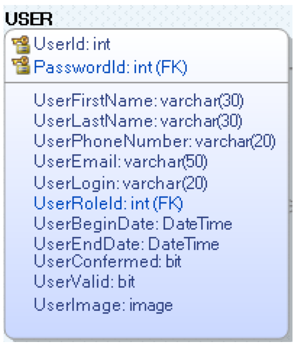
4.1.2 Redesign and update of the database

“USER” table was redesigned to increase the security of the system. The password column has been separated and redesigned to the new table “PASSWORD” with the same purpose. “USERROLE” and “LOGINGATTEMPTS” has been added to manage login, logout and registration of the user. Figure 4-2

The “USER” table has been populated with:

- UserFirstName,
- UserLastName,
- UserLogin,
- UserBeginDate,
- UserEndDate,
- UserConfermed,
- UserPhoneNumber,

- UserValid,
- UserRoleId,
- PasswordId,
- UserImage columns.



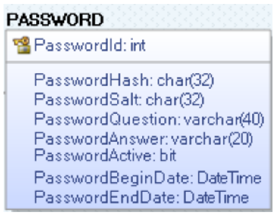
```
USER
UserId: int
PasswordId: int (FK)

UserFirstName: varchar(30)
UserLastName: varchar(30)
UserPhoneNumber: varchar(20)
UserEmail: varchar(50)
UserLogin: varchar(20)
UserRoleId: int (FK)
UserBeginDate: DateTime
UserEndDate: DateTime
UserConfermed: bit
UserValid: bit
UserImage: image
```

Figure 4-2: “USER” table.

Separate table should be used to save the passwords hashes and salt of the users. Ideally hashes and salts should be located on a separate database server. Both servers with hash and salt should be cut-off from the world (and from each other) and only accessible from the server that runs your Web application.(14)

That is why “PASSWORD” (Figure 4-3) table was designed as a separate table with next column properties added:

- PasswordSalt – random salt for each user;
- PasswordHash – hashed user password with MD5, SHA256or SHA512 algorithm using PasswordSalt column property;(15)
- PasswordQuestion – security question;
- PasswordAnswer – security answer;
- PasswordActive – password status variable;
- PasswordBeginDate – date and time when password created;
- PasswordEndDate – date and time when password changed, forgot, etc.



```
PASSWORD
PasswordId: int

PasswordHash: char(32)
PasswordSalt: char(32)
PasswordQuestion: varchar(40)
PasswordAnswer: varchar(20)
PasswordActive: bit
PasswordBeginDate: DateTime
PasswordEndDate: DateTime
```

Figure 4-3: “PASSWORD” table.

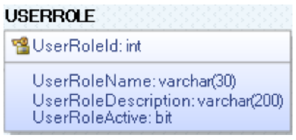
“USERROLE” (Figure 4-4) table determine the access level or permissions of a person authorized or invited by an Administrator, to use the website. (16)

For example:

- Administrator nothing is off limits (service engineers);
- Editor has access to all posts, import/export page;
- Author can write, edit, and publish their posts (for example some reports, news or studies.

"USERROLE" (Figure 4-4) table structure:

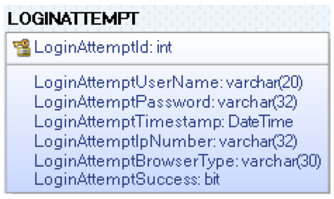
- Overruled – value to identify the user role;
- UserRoleName – the name of the role;
- UserRoleDescription – variable that saves the description for UserRoleName;
- UserRoleActive – shows if the UserRole is active.



USERROLE
UserRoleId: int
UserRoleName: varchar(30)
UserRoleDescription: varchar(200)
UserRoleActive: bit

Figure 4-4: "USERROLE" table.

"LOGINATTEMPT" table (Figure 4-5) indicates when, from what IP and browser, what password and what user name was used to login and was the attempt successful.

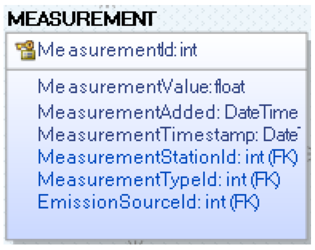


LOGINATTEMPT
LoginAttemptId: int
LoginAttemptUserName: varchar(20)
LoginAttemptPassword: varchar(32)
LoginAttemptTimestamp: DateTime
LoginAttemptIpNumber: varchar(32)
LoginAttemptBrowserType: varchar(30)
LoginAttemptSuccess: bit

Figure 4-5: "LOGINATTEMPT" table.

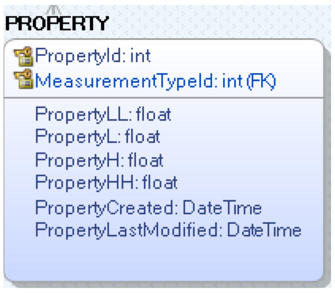
The "Timespamp" column in the "MEASUREMENT" table was updated to two columns. First is MeasurementTimestamp of the actual measurement time and MeasurementAdded to watch for the time of the data import see Figure 4-6. The LimitValue column was moved to the separate table "PROPERTY" (Figure 4-7) to see what exactly limit is this HIGH or LOW. This table can not only identify the range but also the sensor crash.

"ANNUALDATA" table has been dropped.



MEASUREMENT
MeasurementId: int
MeasurementValue: float
MeasurementAdded: DateTime
MeasurementTimestamp: Date
MeasurementStationId: int (FK)
MeasurementTypeId: int (FK)
EmissionSourceId: int (FK)

Figure 4-6: "MEASUREMENT" table.



PROPERTY
PropertyId: int
MeasurementTypeId: int (FK)
PropertyLL: float
PropertyL: float
PropertyH: float
PropertyHH: float
PropertyCreated: DateTime
PropertyLastModified: DateTime

Figure 4-7: "PROPERTY" table.

The primary key columns in all the tables except “MEASUREMENT” where set to the value of int with the Identity(1,1), which means that the value for this columns will automatically generated starting from number 1 and will be increasing by 1 with each new created row.

The MeasurementId column in “MEASUREMENT” table is nvarchar(36). SQL server uses int as Int32 which value range lies from -2,147,483,648 to 2,147,483,647. When the system will have 2,147,483,647 measurements, next value will start from -2,147,483,648 and it get to 0 the database data will be full and “crashed”. The MeasurementId column should be a unique generated value that will never repeat itself. `Guid.NewGuid().ToString()` generates a unique value from:

- The MAC address of the Server that runs the application;
- Timestamp at the request processing;
- Extra "emergency uniquifier bits";
- Identifier for the algorithm.

The value consists from 32 letters and/or number separated by 4 hyphens that gives the overall length of 36 characters. The generation of the MeasurementId identificatory using GUID method should be handled by the application logic.(17-19)

RI action 12 relationship between tables have been modified to maintain the identity of the database. The Delete Rule and Update Rule for Parent or Child has been set to “Cascade”. That means when for example the user deletes the entry from “MEASUREMENTTYPE” table the data from “MEASUREMENT” table should be deleted and handled by the database. All the alarms and warning on the consequences should be handled by the application logic.

4.1.3 Database identity migration

When the database was for started implementing the application, the first thing in the list was to implement User login/logout/register logic. After a question research about password hashing I found next from Defuse Security at:

“DO NOT WRITE YOUR OWN CRYPTO! The problem of storing passwords has already been solved.”

(Defuse Security (September 26, 2016) Salted Password Hashing - Doing it Right. (15))

That gave me the idea that I should not implement my own security either. It turns out that ASP.Net Core has built in security feathers which implementation will be shown in the paragraph 4.2.5. The bottom line of the Core Identity Framework is that it adds all the necessary tables to roll the security. The operation includes migration and extensions of the database with next tables (Figure 4-8):

- AspNetUser;
- AspNetUserClaims;
- AspNetRoles;
- AspNetRoleClaims;
- AspNetUserRoles;
- AspNetUserLogins;

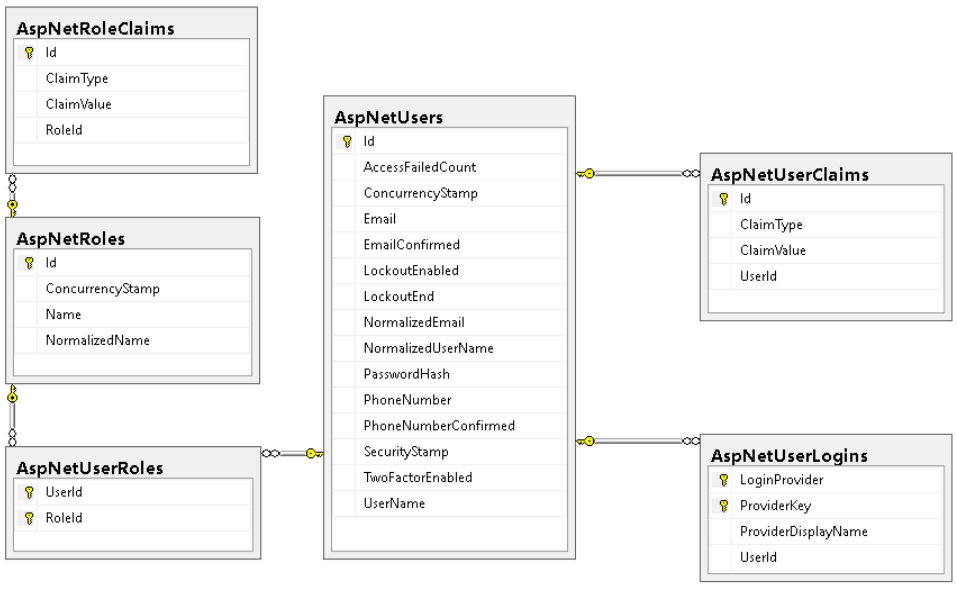


Figure 4-8: Entity frameworktables.

That means that all our tables for web application security can be dropped.

In addition, the table "__EFMigrationsHistory" (Figure 4-9) has been created to save and track the history of other migrations.

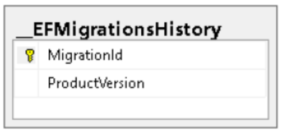


Figure 4-9: "__EFMigrationsHistory" table.

The migration will happen even if during we are going to change the development database to production version. There is no need to manually migrate or create Entity framework tables. These lines of code from the project should perform migration even if tables were deleted:

```
if (Database.GetPendingMigrations().Any())
            {
                Database.Migrate();
            }
```

4.1.4 Result

In the result database will consist from tables shown on the Figure 4-8, Figure 4-9 and Figure 4-10.

Seed data that was used for Entity framework tables are in Appendix C.

Data used to seed the first entities of the database are in the Appendix D. First version of the script based on the Master project report (20).

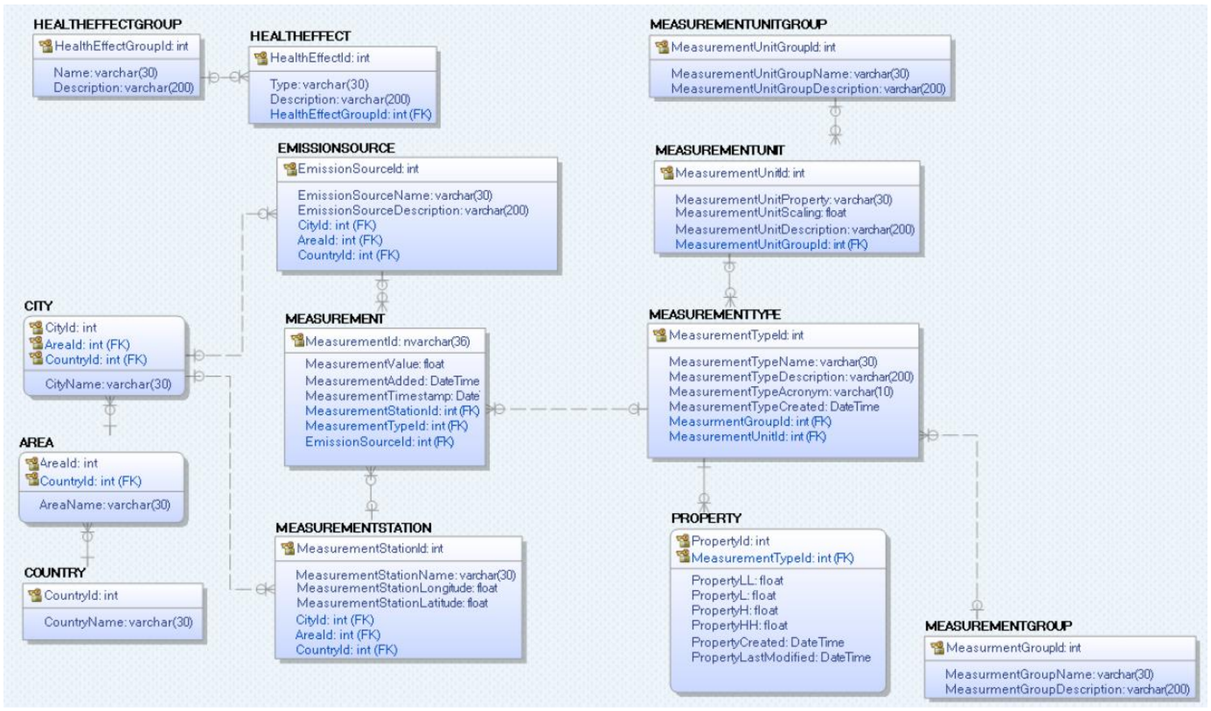

HEALTHEFFECTGROUP
HealthEffectGroupId: int
Name: varchar(30)
Description: varchar(200)
HEALTHEFFECT
HealthEffectId: int
Type: varchar(30)
Description: varchar(200)
HealthEffectGroupId: int (FK)
MEASUREMENTUNITGROUP
MeasurementUnitGroupId: int
MeasurementUnitGroupName: varchar(30)
MeasurementUnitGroupDescription: varchar(200)
EMISSIONSOURCE
EmissionSourceId: int
EmissionSourceName: varchar(30)
EmissionSourceDescription: varchar(200)
CityId: int (FK)
AreaId: int (FK)
CountryId: int (FK)
MEASUREMENTUNIT
MeasurementUnitId: int
MeasurementUnitProperty: varchar(30)
MeasurementUnitScaling: float
MeasurementUnitDescription: varchar(200)
MeasurementUnitGroupId: int (FK)
CITY
CityId: int
AreaId: int (FK)
CountryId: int (FK)
CityName: varchar(30)
MEASUREMENT
MeasurementId: nvarchar(36)
MeasurementValue: float
MeasurementAdded: DateTime
MeasurementTimestamp: Date
MeasurementStationId: int (FK)
MeasurementTypeId: int (FK)
EmissionSourceId: int (FK)
MEASUREMENTTYPE
MeasurementTypeId: int
MeasurementTypeName: varchar(30)
MeasurementTypeDescription: varchar(200)
MeasurementTypeAcronym: varchar(10)
MeasurementTypeCreated: DateTime
MeasurmentGroupId: int (FK)
MeasurementUnitId: int (FK)
AREA
AreaId: int
CountryId: int (FK)
AreaName: varchar(30)
MEASUREMENTSTATION
MeasurementStationId: int
MeasurementStationName: varchar(30)
MeasurementStationLongitude: float
MeasurementStationLatitude: float
CityId: int (FK)
AreaId: int (FK)
CountryId: int (FK)
PROPERTY
PropertyId: int
MeasurementTypeId: int (FK)
PropertyLL: float
PropertyL: float
PropertyH: float
PropertyHH: float
PropertyCreated: DateTime
PropertyLastModified: DateTime
COUNTRY
CountryId: int
CountryName: varchar(30)
MEASUREMENTGROUP
MeasurmentGroupId: int
MeasurmentGroupName: varchar(30)
MeasurmentGroupDescription: varchar(200)

Figure 4-10: Main structure tables.

You can find the generation script in the Appendix B.

All these scripts were applied to the Azure database. During the web application creation, the database was hosted in the cloud.

4.2 Web application

This chapter will walk the user through the codding part of the project.

4.2.1 Create Web application

First, the project was created in Visual Studio 2017 using the Telerik UI ASP.NET Core. This is a separate product for the Visual Studio. This solution adds advanced UI toolset for your project. This free trial for 30days can be downloaded from http://www.telerik.com/aspnet-core-ui.

This project template looks almost the same as described in paragraph 2.7.3. It is only adds "Telerik.UI.for.AspNet.Core.Trial" NuGet package to the project dependencies, sets up the Kendo UI middleware and register Kendo servises in the Startup.cs `Configure()` and `ConfigureServices()` methods.

Telerik ASP.NET Core project used the ASP.NET Core 1.0. The new version of ASP.NET Core 1.1 is already available. The update to the ASP.NET Core 1.1 has been performed by changing next lines in the EPHMS.csproj file:

```
<PropertyGroup>
    <TargetFramework>netcoreapp1.1</TargetFramework>
    <PreserveCompilationContext>true</PreserveCompilationContext>
    <AssemblyName>EPHMS</AssemblyName>
    <OutputType>Exe</OutputType>
    <PackageTargetFallback Condition=" '$(TargetFramework)' == 'netcoreapp1.1'
">$(PackageTargetFallback);dotnet5.6;portable-net45+win8</PackageTargetFallback>
  </PropertyGroup>
```

And executing next commands in the Package console manager(21):

```
dotnet restore
dotnet build
```

4.2.2 Configuring NuGet and Bower packages

In this paragraph, all required NuGet and Bower packages will be listed. Some packaged have required dependencies that install automatically with the main package, so they will not be listed. You can find dependencies list in the NuGet package manager of at https://www.nuget.org .

It is not necessary to add all the packages at once. They can be installed as needed. This shows how flexible and configurable the ASP.NET Core was designed.

NuGet packages required to work with SQL Server, to able to perform CRUD requests and design and publish table to the database if required(22):

- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools
- Microsoft.EntityFrameworkCore.SqlServer.Design

NuGet packages required to work with MVC(23):

- Microsoft.AspNetCore.Mvc

NuGet packages required to work with Web API(23):

- Microsoft.EntityFrameworkCore.InMemory
- Microsoft.AspNetCore.Authentication.jwtBearer

NuGet packages required to work with JSON, XML, connection string and other options(24):

- Microsoft.Extensions.Options.ConfigurationExtensions
- Microsoft.AspNetCore.Mvc.Formatters.Json
- Microsoft.AspNetCore.Mvc.Formatters.Xml

NuGet packages required to work with Razor engine TagHelpers(12):

- Microsoft.AspNetCore.Razor.Tools

NuGet packages required to work with CSS, JavaScript and other static file in the wwwroot folder(25):

- Microsoft.AspNetCore.StaticFiles

NuGet packages required to work with Identity Core framework(26):

- Microsoft.AspNetCore.Identity.EntityFrameworkCore

NuGet packages required to work with Localization of the web application(27):

- Localization.AspNetCore.TagHelpers

Bower packages required to work with Bootstrap(28):

- "bootstrap": "v4.0.0-alpha.6",
- "tether": "1.4.0",
- "jquery": "3.2.1"

Bower packages to enable responsive text and charts:

- "Flowtype.js": "1.1.0",
- "chartist": "v0.11.0"

4.2.3 MVC structure

In this chapter, the MVC framework will be added to the project and the folder structure will be created.

The folder stricture for the MVC files will be created now. This will make the project look clean with understandable structure.

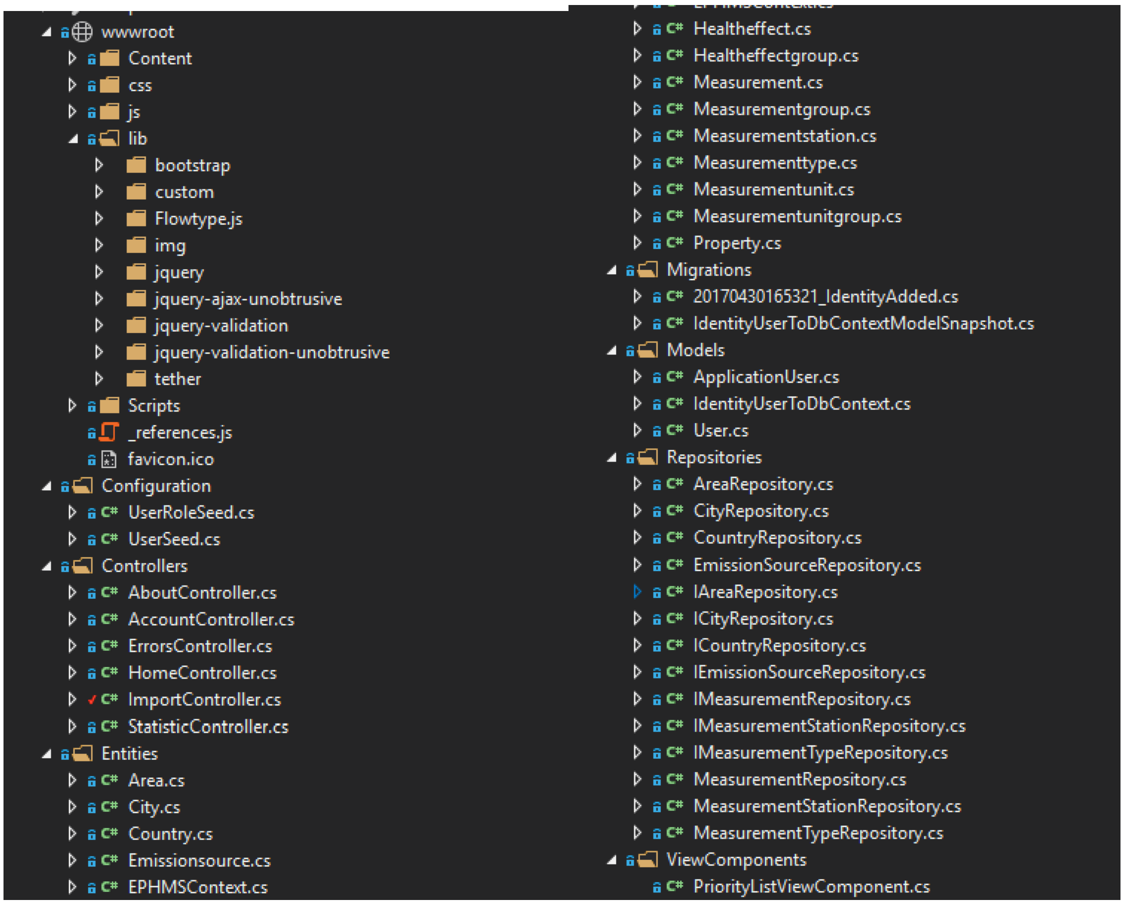
MCV stand for Model, View and Controller, so three folders must be created: Models, Views and Controllers. In Addition to them the Entities folder created. This folder will also keep models that will be “scaffolded” from the database, see paragraph 0.

Also, Configuration folder created. It will contain Seed method that must populate database with required information.

Controllers according to the requirement must be created. Controller name must consist from “Controller name” plus “Controller” word.

Every controller has action/actions inside. Any action can return a view, redirect to another action, etc. For those actions that are returning the view, those views must be created.

The overall project structure shown on the Figure 4-11 and Figure 4-12.



```
wwwroot
    Content
    css
    js
    lib
        bootstrap
        custom
        Flowtype.js
        img
        jquery
        jquery-ajax-unobtrusive
        jquery-validation
        jquery-validation-unobtrusive
        tether
    Scripts
    _references.js
    favicon.ico
Configuration
    UserRoleSeed.cs
    UserSeed.cs
Controllers
    AboutController.cs
    AccountController.cs
    ErrorsController.cs
    HomeController.cs
    ImportController.cs
    StatisticController.cs
Entities
    Area.cs
    City.cs
    Country.cs
    Emissionsource.cs
    EPHMSContext.cs
    Healtheffect.cs
    Healtheffectgroup.cs
    Measurement.cs
    Measurementgroup.cs
    Measurementstation.cs
    Measurementtype.cs
    Measurementunit.cs
    Measurementunitgroup.cs
    Property.cs
Migrations
    20170430165321_IdentityAdded.cs
    IdentityUserToDbContextModelSnapshot.cs
Models
    ApplicationUser.cs
    IdentityUserToDbContext.cs
    User.cs
Repositories
    AreaRepository.cs
    CityRepository.cs
    CountryRepository.cs
    EmissionSourceRepository.cs
    IAreaRepository.cs
    ICityRepository.cs
    ICountryRepository.cs
    IEmissionSourceRepository.cs
    IMeasurementRepository.cs
    IMeasurementStationRepository.cs
    IMeasurementTypeRepository.cs
    MeasurementRepository.cs
    MeasurementStationRepository.cs
    MeasurementTypeRepository.cs
ViewComponents
    PriorityListViewComponent.cs
```

Figure 4-11: File and folder structure (without Views folder).

4.2.4 Entity Framework core

To use the table as classes in ASP.Net Core special command – “scaffolded” was used. It takes connection string as a parameter, file name of the tables context and output folder. User can also specify tables he wants to scaffolded.

```
Scaffold-DbContext "Data Source=ephms-db-server.database.windows.net;Initial Catalog=EPHMS-
db;User ID=adminArtem;Password=p0int->tjsMk8%" Microsoft.EntityFrameworkCore.SqlServer -
OutputDir Entities –context EPHMSContext -tables (“AREA” , “CITY”, “COUNTRY” ,
“EMISSIONSOURCE‟ , “HEALTHEFFECT‟ , “HEALTHEFFECTGROUP‟ , “MEASUREMENT‟ , “MEASUREMENTGROUP‟ ,
“MEASUREMENTSTATION‟ , “MEASUREMENTTYPE‟ , “MEASUREMENTUNIT‟ , “MEASUREMENTUNITGROUP‟ ,
“PROPERTY‟ )
```

4.2.5 Core Identity

To create the secure registration and login the Identity Entity framework was used. To use it we need to create some class that inherits from Identity user, like this:

```
    public class ApplicationUser : IdentityUser
{
}
```

Then create the context for the Identity as follows:

```
namespace EPHMS.Models
{
    public class IdentityUserToDbContext : IdentityDbContext<ApplicationUser>
    {
        public IdentityUserToDbContext(DbContextOptions<IdentityUserToDbContext> options) :
base(options)
        {
            if (Database.GetPendingMigrations().Any())
            {
                Database.Migrate();
            }
        }
    }
}
```

In the startup class services must be added and middleware registered with next lines:

```
services.AddDbContext<IdentityUserToDbContext>(options => options.UseSqlServer(@"Data
Source=tcp:ephms-db-server.database.windows.net,1433;Initial Catalog=EPHMS-db;User
ID=adminArtem@ephms-db-server;Password=PASSWORD"));
```

And

```
app.UseIdentity();
```

The seed methods can then be called before closing curly bracket of the Configuration method.

```
new UserRoleSeed(app.ApplicationServices.GetService<RoleManager<IdentityRole>>()).Seed();
new UserSeed(app.ApplicationServices.GetService<UserManager<ApplicationUser>>()).Seed();
```

4.2.6 Repository pattern

To maintain the single responsibility principle the repositories was created, see Figure 4-11. Before using the repository pattern controllers where highly dependent on the data base context. If the customer will decide not to use specific tables from the database and use web API instead, if the API can fulfil the repository interface there is need to change the controller logic. Separate methods were used to read required data from the database. Then the Interfaces were extracted from the methods.

Example of the CityRepository:

```
namespace EPHMS.Repositories
{
    public class CityRepository : ICityRepository
    {
        private EPHMSContext cityDb = new EPHMSContext();
        public City CreateCity(City newCity)
        {
            cityDb.City.Add(newCity);
            cityDb.SaveChanges();
            return newCity;
        }
        public City GetCity(int cityId)
        {
            return cityDb.City.FirstOrDefault(c => c.CityId == cityId);
        }
        public List<City> GetAllCities()
        {
            return cityDb.City.ToList();
        }
        public City UpdateCity(City updatedCity)
        {
            var originalCity = cityDb.City.FirstOrDefault(c => c.CityId ==
updatedCity.CityId);
            originalCity.CityName = updatedCity.CityName;
            originalCity.AreaId = updatedCity.AreaId;
            originalCity.CountryId = updatedCity.CountryId;
            cityDb.SaveChanges();
            return originalCity;
        }
        public bool DeleteCity(int cityId)
        {
            City deleteCity = cityDb.City.FirstOrDefault(c => c.CityId == cityId);
            if (deleteCity != null)
            {
                cityDb.City.Remove(deleteCity);
                cityDb.SaveChanges();
            }
            return (cityDb.City.FirstOrDefault(c => c.CityId == cityId) == null);
        }
    }
}
```

And ICityRepository interface:

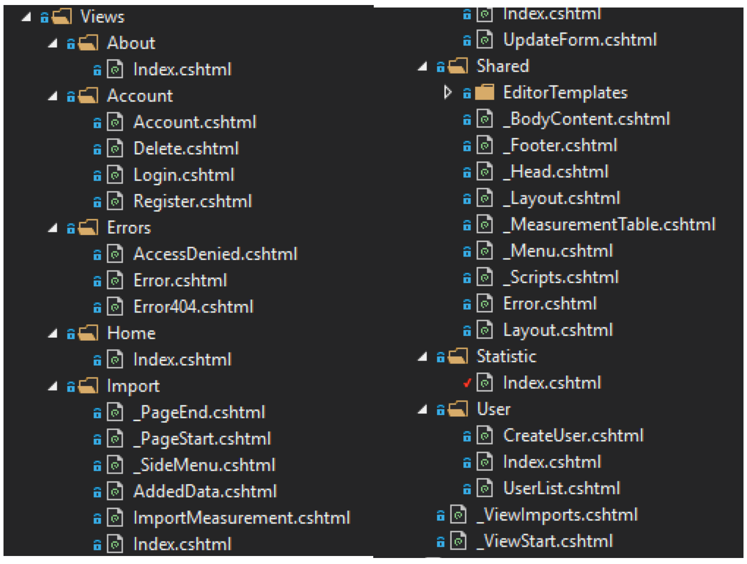
```
namespace EPHMS.Repositories
{
    public interface ICityRepository
    {
        City CreateCity(City newCity);
        bool DeleteCity(int cityId);
        List<City> GetAllCities();
        City GetCity(int cityId);
        City UpdateCity(City updatedCity);
    }
}
```

4.3 Views

Front-end representation will be covered in this paragraph.

The view in the ASP.Net Core are not just static files. They have complex structure Figure 4-12: View folder structure. that gives a lot of flexibility and allow to reuse code for example navigation bar – _Menu.chhtlm from shared folded used in all pages. HTML TagHelper - `@Html.Partial()` renders it inside the _Layout file before redirecting the request to the user.

As a result this structure generates required HTML pages according to the HTTP request.



```
Views
  About
    Index.cshtml
  Account
    Account.cshtml
    Delete.cshtml
    Login.cshtml
    Register.cshtml
  Errors
    AccessDenied.cshtml
    Error.cshtml
    Error404.cshtml
  Home
    Index.cshtml
  Import
    _PageEnd.cshtml
    _PageStart.cshtml
    _SideMenu.cshtml
    AddedData.cshtml
    ImportMeasurement.cshtml
    Index.cshtml
    Index.cshtml
    UpdateForm.cshtml
  Shared
    EditorTemplates
    _BodyContent.cshtml
    _Footer.cshtml
    _Head.cshtml
    _Layout.cshtml
    _MeasurementTable.cshtml
    _Menu.cshtml
    _Scripts.cshtml
    Error.cshtml
    Layout.cshtml
  Statistic
    Index.cshtml
  User
    CreateUser.cshtml
    Index.cshtml
    UserList.cshtml
  _ViewImports.cshtml
  _ViewStart.cshtml
```

Figure 4-12: View folder structure.

4.3.1 Home page

The running home page will look as described in Figure 4-13

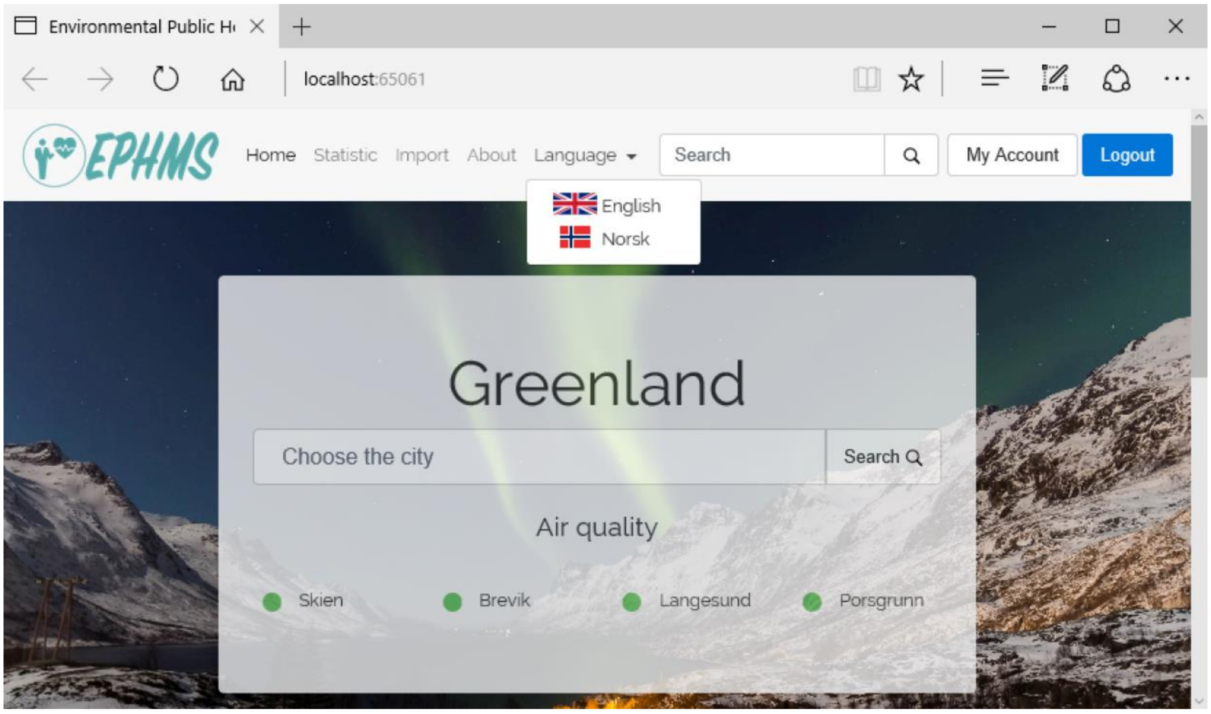
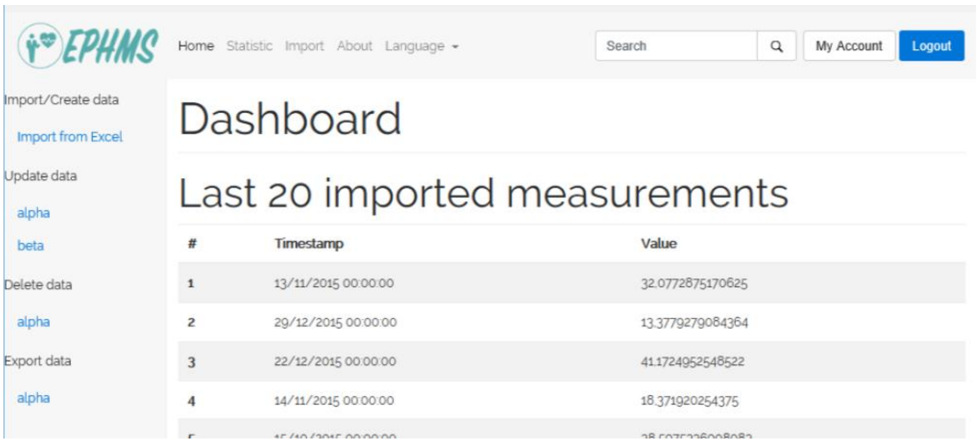


Figure 4-13: Home page.

4.3.2 Import

The running Import page will look as described in Figure 4-14



Dashboard

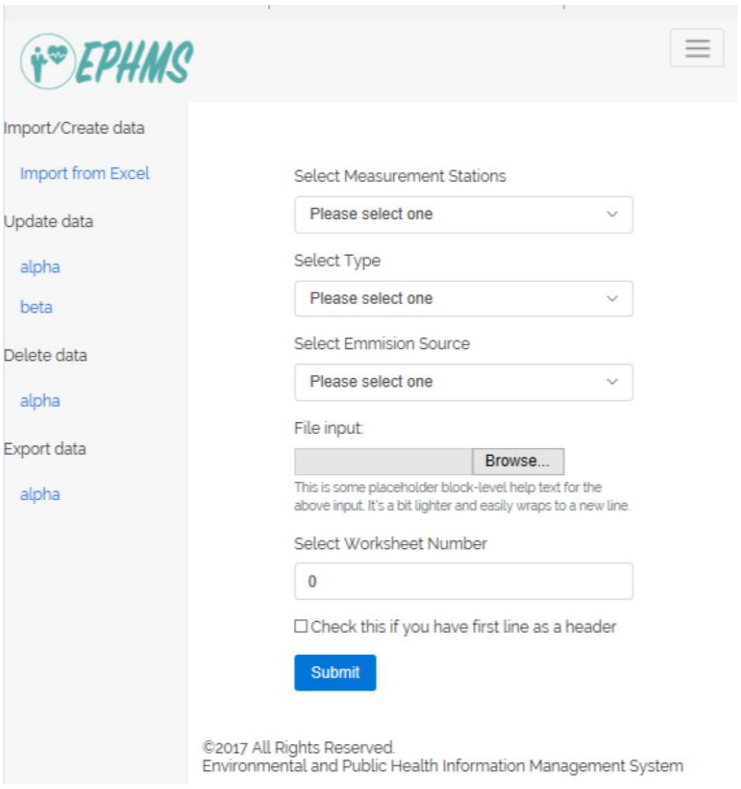
Last 20 imported measurements

#	Timestamp	Value
1	13/11/2015 00:00:00	32.0772875170625
2	29/12/2015 00:00:00	13.3779279084364
3	22/12/2015 00:00:00	41.1724952548522
4	14/11/2015 00:00:00	18.371920254375

Figure 4-14: Import page.

4.3.3 Import from Excel

The running Statistic page will look as described in Figure 4-15



Select Measurement Stations

Please select one

Select Type

Please select one

Select Emmision Source

Please select one

File input

Browse...

This is some placeholder block-level help text for the above input. It's a bit lighter and easily wraps to a new line.

Select Worksheet Number

0

☐ Check this if you have first line as a header

Submit

©2017 All Rights Reserved.
Environmental and Public Health Information Management System

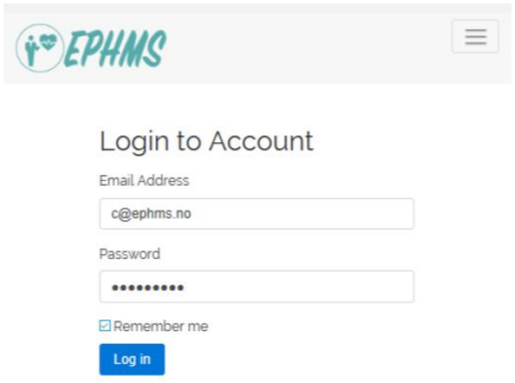
Figure 4-15:Import from Excel.

4.3.4 Statistic

The running Statistic page will look as described in Figure 4-18

4.3.5 Login/Register

The running login and registration pages will look as described in Figure 4-16 and Figure 4-17



Login to Account

Email Address

c@ephms.no

Password

•••••••••

☑ Remember me

Log in

Figure 4-16: Login page.

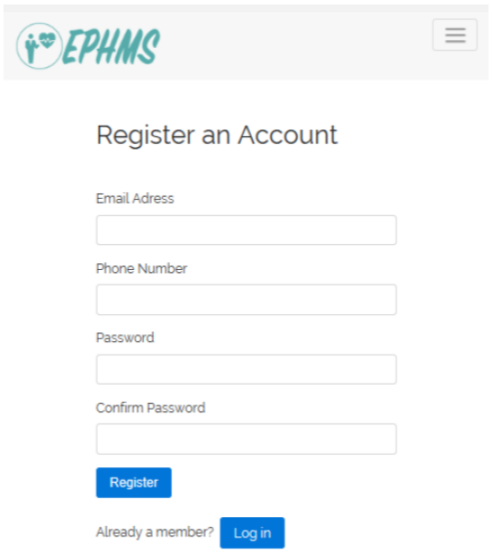
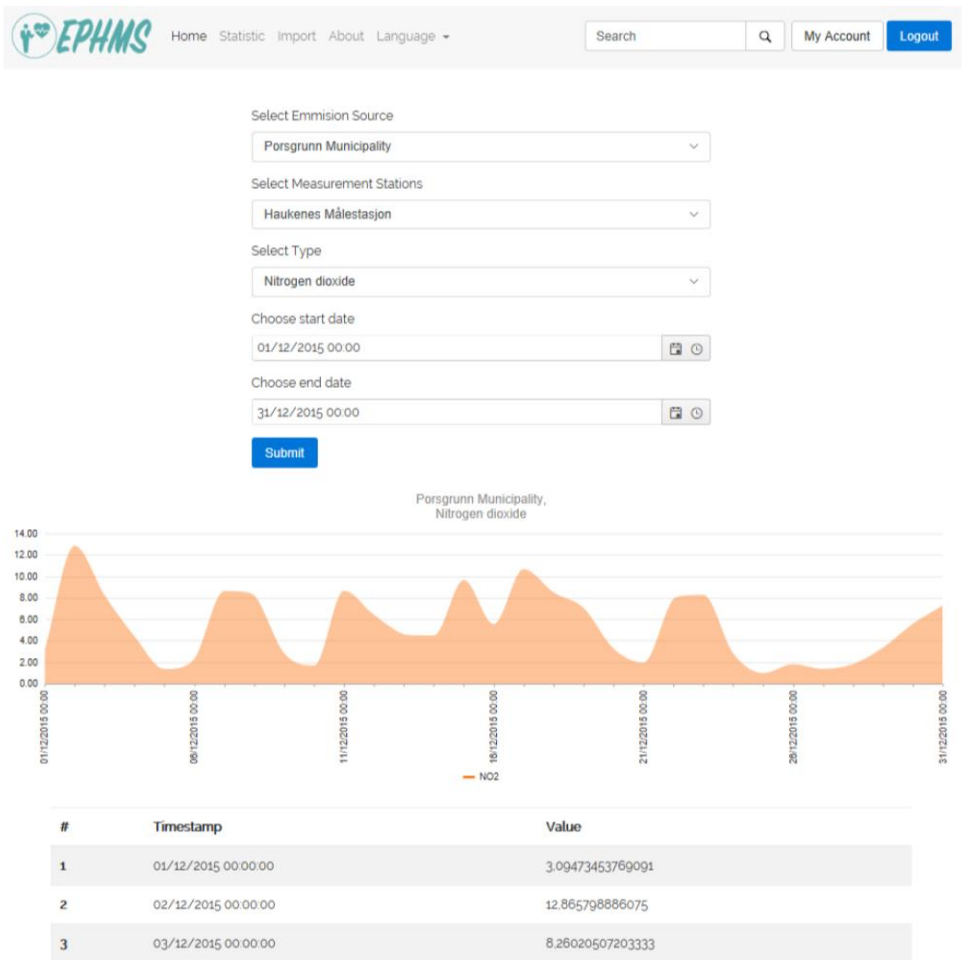


Figure 4-17: Register page.



Select Emmision Source: Porsgrunn Municipality

Select Measurement Stations: Haukenes Målestasjon

Select Type: Nitrogen dioxide

Choose start date: 01/12/2015 00:00

Choose end date: 31/12/2015 00:00

#	Timestamp	Value
1	01/12/2015 00:00:00	3.09473453769091
2	02/12/2015 00:00:00	12.865798886075
3	03/12/2015 00:00:00	8.26020507203333

Figure 4-18: Statistic page.

5 Discussion

ASP.Net Core is new, written from scratch as a logical continuation of the ASP.Net 4.6 framework. It can use C# as primary language for programing. All that gives it very powerful feathers. For example, it was decided to use LINQ, instead of creating View, Store procedures, etc. in the database. Because with the help of Entity framework we can have our tables as classes directly inside application. Another plus of the ASP.Net core is that it is module based framework. It makes the application very lightweight. But, since it is modular setting up all the required project modules and libraries takes time. So, it is recommended to use the project as template for future development. Another powerful feather of ASP.Net Core called Razor engine was used for creating views. It allows to write C# code inside the page.
Telerik Kendo UI, Bootstrap and other CSS and JavaScript libraries user for creating the representation and views of the application. They help to create nice looking and dynamic pages.

Visual Studio 2017, SQL Server 2014 Management Studio and erwin Data Modeler r9.7 was used during development. Those tools suites very well for creating such application. The HTML pages’ representation was created before using the ASP.Net Core and Visual Studio 2017. Open-source editor called Brackets was used. The advantages of this editor are that it has quick access to the documentation, live preview and it works with HTML and JavaScript (29).

Objectives of their implementation discussed in summary.

Future work discussed in the chapter 7.

6 Summary

Among many available back-end languages PHP, ASP.Net 4.6, Python, Java, Ruby, Perl for web development, light and open-source ASP.Net Core 1.1 was chosen. It was decided to use SQL as a primary source of data since it suits best for stretchered data like measurements.

CSS and JavaScript libraries helped to speed up the development process. During web development, next pages were created:

- Home;
- Login;
- Register;
- Statistics;
- Import;
- Import from excel.

Login and register pages use ASP.Net Core Security feathers from. Special seed functions were developed to populate Entity framework tables in the database.

The functionality of statistics pages can show the data from different emission source, measurement stations and types. Also, it followed by graphical representation and table of measurements.

Import page shows last imported data and it is available only for users that login with the “Admin” privilege. Import page also has a function of the hub for other import/export/update/delete pages that will be implemented in future. The Import from excel function is available from this page. Import from excel page has a for with required parameters and file download form. The form has evaluation of broken data with the value of -1 for the measurements.

The cascade properties were added to the database to maintain database identity. Repositories created in the web application to maintain the single responsibility pattern. The Interfaces has been extracted from repositories, so using of web APIs is not affecting the controller’s logic. Dependency injection technic was used to archive loose coupling.

7 Future work

7.1 Business logic

The main objective for the future work is increasing of the business logic in the application. That will add more functionality and will make the application more flexible. Bigger functionality will require more testing (manual and automatic) of the software, as well as optimization acquirer more computational power with less resources.

7.2 RESTfull API

RESTfull API must be implemented for automatic data collection in addition to manual import from excel. RESTfull services will help to receive data from sources not only at the same time.

7.3 AJAX

AJAX capabilities should be implemented. This help to make the website more dynamic and intuitive, as well as will allow to reload data without reloading the page.

7.4 Globalization and localization

Second language implementation. The required libraries are already installed in the project.

References

1. List of CLI languages [Internet]. 2017 [cited 18.04.2017]. Available from: https://en.wikipedia.org/wiki/List_of_CLI_languages.

2. FURPS [Internet]. 2017 [cited 21.04.2017]. Available from: https://en.wikipedia.org/wiki/FURPS.

3. HTML Introduction [Internet]. [cited 22.04.2017]. Available from: https://www.w3schools.com/html/html_intro.asp

4. Getting started (Bootstrap) [Internet]. [cited 15.04.2017]. Available from: https://v4-alpha.getbootstrap.com/getting-started/introduction/.

5. Components (Bootstrap) [Internet]. [cited 12.04.2017]. Available from: https://v4-alpha.getbootstrap.com/components/navbar/.

6. ASP.NET 5 is dead - Introducing ASP.NET Core 1.0 and .NET Core 1.0 [Internet]. 2016 [cited 20.04.2017]. Available from: http://www.hanselman.com/blog/ASPNET5IsDeadIntroducingASPNETCore10AndNETCore10.aspx.

7. Introduction to ASP.NET Core [Internet]. 2016 [cited 12.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/.

8. ASP.NET Core Middleware Fundamentals [Internet]. 2017 [cited 18.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware.

9. Application Startup in ASP.NET Core [Internet]. 2017 [cited 20.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup.

10. Overview of ASP.NET Core MVC [Internet]. 2016 [cited 22.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview.

11. Razor syntax [Internet]. 2017 [cited 30.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor.

12. Introduction to Tag Helpers in ASP.NET Core [Internet]. 2016 [cited 28.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro

13. Host an ASP.NET Core app in a Windows Service [Internet]. 2017 [cited 01.05.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/publishing/.

14. Store users and pass in single table or separate table [Internet]2012. Available from: http://stackoverflow.com/questions/13322841/store-users-and-pass-in-single-table-or-separate-table.

15. Salted Password Hashing - Doing it Right [Internet]. 2016 [cited 1.05.2017]. Available from: https://crackstation.net/hashing-security.htm.

16. User Roles (WordPress) [Internet]. [cited 22.04.2017]. Available from: [https://en.support.wordpress.com/user-roles/.

17. SQL Server Data Type Mappings [Internet]. [cited 02.05.2017]. Available from: https://msdn.microsoft.com/en-us/library/cc716729(v=vs.110).aspx.

18. Chen R. GUIDs are globally unique, but substrings of GUIDs aren’t [Internet]2008. Available from: https://blogs.msdn.microsoft.com/oldnewthing/20080627-00/?p=21823/.

19. Guid.ToString Method (String) [Internet]. [cited 18.04.2017]. Available from: https://msdn.microsoft.com/en-us/library/97af8hh4(v=vs.110).aspx.

20. Alexander Zhang Gjerseth, Ang L. Development of a Database System for Environmental and Public Health Information. 2016.

21. .NET Core command-line interface (CLI) tools [Internet]. 2017 [cited 07.05.2017]. Available from: https://docs.microsoft.com/en-us/dotnet/articles/core/tools/.

22. ASP.NET Core - Existing Database [Internet]. 2016 [cited 08.05.2017]. Available from: https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db.

23. Building Your First Web API with ASP.NET Core MVC and Visual Studio [Internet]. 2017 [cited 04.05.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api.

24. Introduction to formatting response data in ASP.NET Core MVC [Internet]. 2016 [cited 5.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/mvc/models/formatting.

25. Introduction to working with static files in ASP.NET Core [Internet]. 2017 [cited 13.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/static-files.

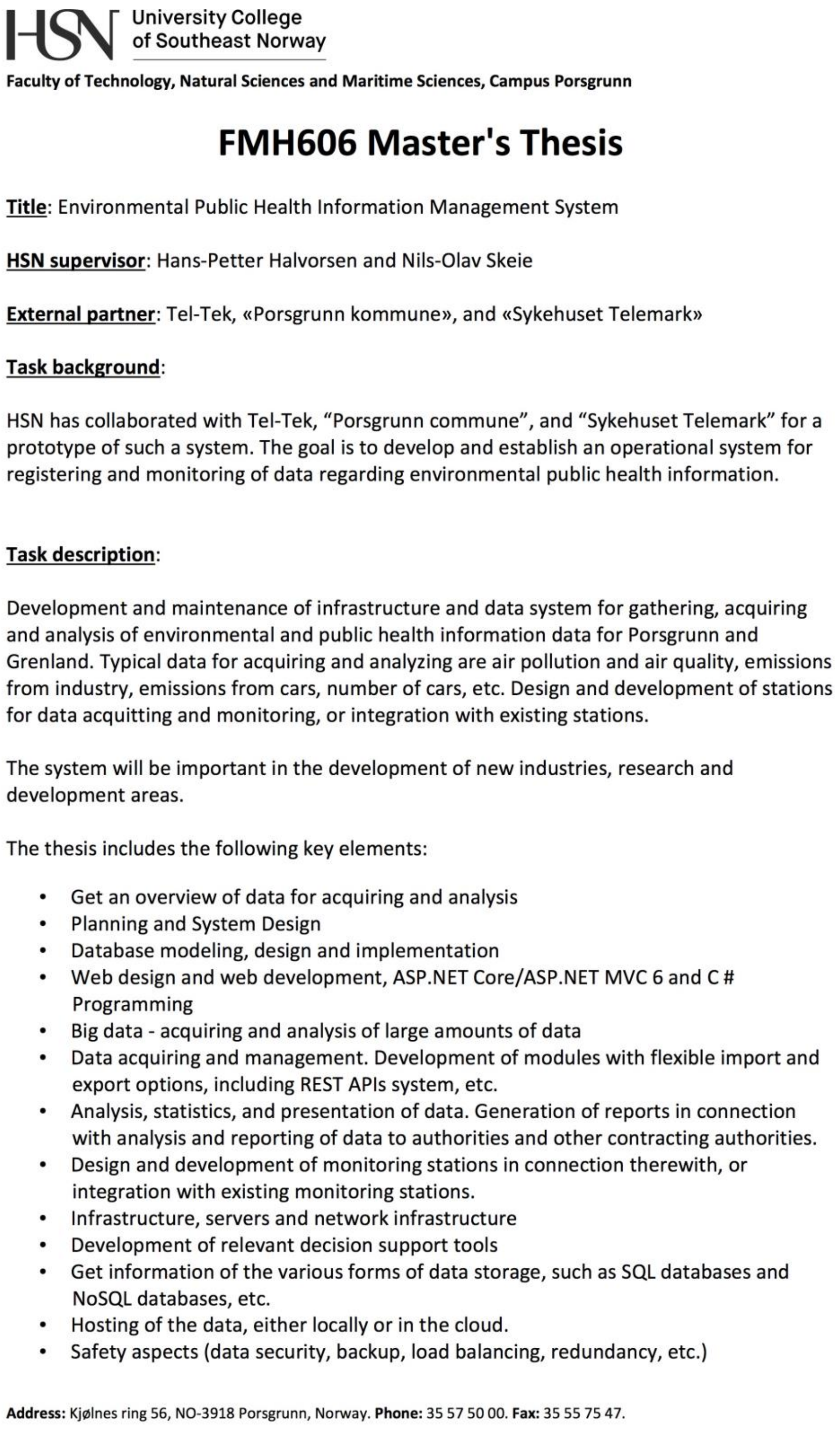
26. Introduction to Identity [Internet]. 2016 [cited 16.04.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity.

27. Globalization and localization [Internet]. 2017 [cited 02.05.2017]. Available from: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization.

28. Getting Started with ASP.NET Core and Bootstrap 4 [Internet]. 2016 [cited 21.04.2017]. Available from: https://www.packtpub.com/books/content/getting-started-aspnet-core-and-bootstrap-4.

29. Brackets (text editor) [Internet]. 2017 [cited 28.04.2017]. Available from: https://en.wikipedia.org/wiki/Brackets_(text_editor).

Appendix A



University College of Southeast Norway

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

FMH606 Master's Thesis

Title: Environmental Public Health Information Management System

HSN supervisor: Hans-Petter Halvorsen and Nils-Olav Skeie

External partner: Tel-Tek, «Porsgrunn kommune», and «Sykehuset Telemark»

Task background:

HSN has collaborated with Tel-Tek, "Porsgrunn commune", and "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.

Task description:

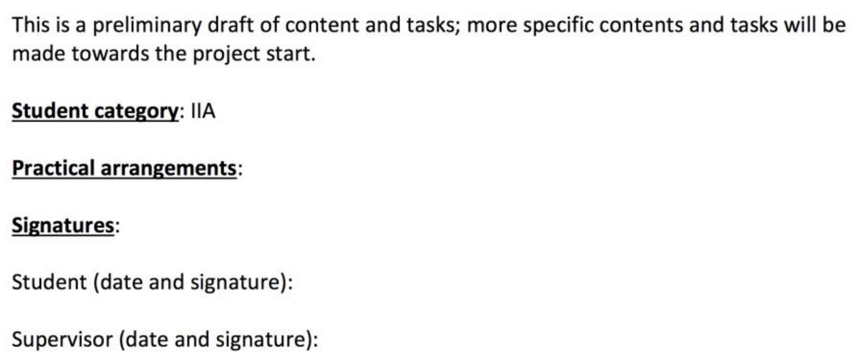
Development and maintenance of infrastructure and data system for gathering, acquiring and analysis of environmental and public health information data for Porsgrunn and Grenland. Typical data for acquiring and analyzing are air pollution and air quality, emissions from industry, emissions from cars, number of cars, etc. Design and development of stations for data acquitting and monitoring, or integration with existing 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, ASP.NET Core/ASP.NET MVC 6 and C # Programming
- Big data - acquiring and analysis of large amounts of data
- Data acquiring and management. Development of modules with flexible import and export options, including REST APIs system, etc.
- Analysis, statistics, and presentation of data. Generation of reports in connection with analysis and reporting of data to authorities and other contracting authorities.
- Design and development of monitoring stations in connection therewith, or integration with existing monitoring stations.
- Infrastructure, servers and network infrastructure
- Development of relevant decision support tools
- Get information of the various forms of data storage, such as SQL databases and NoSQL databases, etc.
- Hosting of the data, either locally or in the cloud.
- Safety aspects (data security, backup, load balancing, redundancy, etc.)

Address: Kjølnes ring 56, NO-3918 Porsgrunn, Norway. **Phone:** 35 57 50 00. **Fax:** 35 55 75 47.



This is a preliminary draft of content and tasks; more specific contents and tasks will be made towards the project start.

Student category: IIA

Practical arrangements:

Signatures:

Student (date and signature):

Supervisor (date and signature):

Appendix B

```
DROP TABLE MEASUREMENT
go
DROP TABLE EMISSIONSOURCE
go
DROP TABLE MEASUREMENTSTATION
go
DROP TABLE CITY
go
DROP TABLE AREA
go
DROP TABLE COUNTRY
go
DROP TABLE HEALTHEFFECT
go
DROP TABLE HEALTHEFFECTGROUP
go
DROP TABLE PROPERTY
go
DROP TABLE MEASUREMENTTYPE
go
DROP TABLE MEASUREMENTUNIT
go
DROP TABLE MEASUREMENTUNITGROUP
go
DROP TABLE MEASUREMENTGROUP
go
CREATE TABLE MEASUREMENTGROUP
(
       MeasurementGroupId    int  NOT NULL  IDENTITY ( 1,1 ) ,
       MeasurementGroupName  varchar(30)  NOT NULL ,
       MeasurementGroupDescription varchar(200)  NULL ,
       PRIMARY KEY  CLUSTERED (MeasurementGroupId ASC)
)
go
CREATE TABLE MEASUREMENTUNITGROUP
(
       MeasurementUnitGroupId int  NOT NULL  IDENTITY ( 1,1 ) ,
       MeasurementUnitGroupName varchar(30)  NOT NULL ,
       MeasurementUnitGroupDescription varchar(200)  NULL ,
       PRIMARY KEY  CLUSTERED (MeasurementUnitGroupId ASC)
)
go
CREATE TABLE MEASUREMENTUNIT
(
       MeasurementUnitId    int  NOT NULL  IDENTITY ( 1,1 ) ,
       MeasurementUnitProperty varchar(30)  NOT NULL ,
       MeasurementUnitScaling float  NOT NULL ,
       MeasurementUnitDescription varchar(200)  NULL ,
       MeasurementUnitGroupId int  NOT NULL ,
       PRIMARY KEY  CLUSTERED (MeasurementUnitId ASC),
        FOREIGN KEY (MeasurementUnitGroupId) REFERENCES
MEASUREMENTUNITGROUP(MeasurementUnitGroupId)
              ON DELETE CASCADE
              ON UPDATE CASCADE
)
go
CREATE TABLE MEASUREMENTTYPE
(
       MeasurementTypeId    int  NOT NULL  IDENTITY ( 1,1 ) ,
       MeasurementTypeName  varchar(30)  NOT NULL ,
       MeasurementTypeDescription varchar(200)  NULL ,
       MeasurementTypeAcronym varchar(10)  NOT NULL ,
       MeasurementGroupId    int  NOT NULL ,
       MeasurementUnitId    int  NOT NULL ,
       MeasurementTypeCreated DateTime  NOT NULL ,
       PRIMARY KEY  CLUSTERED (MeasurementTypeId ASC),
        FOREIGN KEY (MeasurementGroupId) REFERENCES MEASUREMENTGROUP(MeasurementGroupId)
              ON DELETE CASCADE
              ON UPDATE CASCADE,
        FOREIGN KEY (MeasurementUnitId) REFERENCES MEASUREMENTUNIT(MeasurementUnitId)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION
)
```

```
go
CREATE TABLE PROPERTY
(
       PropertyId           int  NOT NULL  IDENTITY ( 1,1 ) ,
       PropertyLL           float  NULL ,
       PropertyL            float  NULL ,
       PropertyCreated      DateTime  NOT NULL ,
       MeasurementTypeId    int  NOT NULL ,
       PropertyH            float  NULL ,
       PropertyHH           float  NULL ,
       PropertyLastModified DateTime  NOT NULL ,
       PRIMARY KEY  CLUSTERED (PropertyId ASC,MeasurementTypeId ASC),
        FOREIGN KEY (MeasurementTypeId) REFERENCES MEASUREMENTTYPE(MeasurementTypeId)
              ON DELETE CASCADE
              ON UPDATE CASCADE
)
go
CREATE TABLE HEALTHEFFECTGROUP
(
       HealthEffectGroupId  int  NOT NULL  IDENTITY ,
       Name                 varchar(30)  NULL ,
       Description          varchar(200)  NULL ,
       PRIMARY KEY  CLUSTERED (HealthEffectGroupId ASC)
)
go
CREATE TABLE HEALTHEFFECT
(
       HealthEffectId       int  NOT NULL ,
       Type                 varchar(30)  NULL ,
       Description          varchar(200)  NULL ,
       HealthEffectGroupId  int  NOT NULL  IDENTITY ,
       PRIMARY KEY  CLUSTERED (HealthEffectId ASC),
        FOREIGN KEY (HealthEffectGroupId) REFERENCES HEALTHEFFECTGROUP(HealthEffectGroupId)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION
)
go
CREATE TABLE COUNTRY
(
       CountryId            int  NOT NULL  IDENTITY ( 1,1 ) ,
       CountryName          varchar(30)  NOT NULL ,
       PRIMARY KEY  CLUSTERED (CountryId ASC)
)
go
CREATE TABLE AREA
(
       AreaId               int  NOT NULL  IDENTITY ( 1,1 ) ,
       AreaName             varchar(30)  NOT NULL ,
       CountryId            int  NOT NULL ,
       PRIMARY KEY  CLUSTERED (AreaId ASC,CountryId ASC),
        FOREIGN KEY (CountryId) REFERENCES COUNTRY(CountryId)
              ON DELETE CASCADE
              ON UPDATE CASCADE
)
go
CREATE TABLE CITY
(
       CityId               int  NOT NULL  IDENTITY ( 1,1 ) ,
       CityName             varchar(30)  NOT NULL ,
       AreaId               int  NOT NULL ,
       CountryId            int  NOT NULL ,
       PRIMARY KEY  CLUSTERED (CityId ASC,AreaId ASC,CountryId ASC),
        FOREIGN KEY (AreaId,CountryId) REFERENCES AREA(AreaId,CountryId)
              ON DELETE CASCADE
              ON UPDATE CASCADE
)
go
CREATE TABLE MEASUREMENTSTATION
(
       MeasurementStationId int  NOT NULL  IDENTITY ( 1,1 ) ,
       MeasurementStationName varchar(30)  NOT NULL ,
       MeasurementStationLongitude float  NOT NULL ,
       MeasurementStationLatitude float  NOT NULL ,
       CityId               int  NOT NULL ,
       AreaId               int  NOT NULL ,
       CountryId            int  NOT NULL ,
       PRIMARY KEY  CLUSTERED (MeasurementStationId ASC),
        FOREIGN KEY (CityId,AreaId,CountryId) REFERENCES CITY(CityId,AreaId,CountryId)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION
)
go
CREATE TABLE EMISSIONSOURCE
(
       EmissionSourceId      int  NOT NULL  IDENTITY ( 1,1 ) ,
       EmissionSourceName    varchar(30)  NOT NULL ,
       EmissionSourceDescription varchar(200)  NULL ,
       CityId                int  NOT NULL ,
       AreaId                int  NOT NULL ,
       CountryId             int  NOT NULL ,
       PRIMARY KEY  CLUSTERED (EmissionSourceId ASC),
        FOREIGN KEY (CityId,AreaId,CountryId) REFERENCES CITY(CityId,AreaId,CountryId)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION
)
go
CREATE TABLE MEASUREMENT
(
       MeasurementId        nvarchar(36)  NOT NULL ,
       MeasurementValue     float  NOT NULL ,
       MeasurementTimestamp DateTime  NOT NULL ,
       MeasurementStationId int  NOT NULL ,
       MeasurementTypeId    int  NOT NULL ,
       EmissionSourceId     int  NOT NULL ,
       MeasurementAdded     DateTime  NOT NULL ,
       PRIMARY KEY  CLUSTERED (MeasurementId ASC),
        FOREIGN KEY (MeasurementStationId) REFERENCES
MEASUREMENTSTATION(MeasurementStationId)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION,
        FOREIGN KEY (MeasurementTypeId) REFERENCES MEASUREMENTTYPE(MeasurementTypeId)
              ON DELETE CASCADE
              ON UPDATE CASCADE,
        FOREIGN KEY (EmissionSourceId) REFERENCES EMISSIONSOURCE(EmissionSourceId)
              ON DELETE NO ACTION
              ON UPDATE NO ACTION
)
Go
```

Appendix C

User Role Seed:

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using System.Threading.Tasks;

namespace EPHMS.Configuration
{
    public class UserRoleSeed
    {
        private readonly RoleManager<IdentityRole> _roleManager;
        public UserRoleSeed(RoleManager<IdentityRole> roleManager)
        {
            _roleManager = roleManager;
        }

        public async void Seed()
        {
            await AddUserRoleAsync("User");
            await AddUserRoleAsync("Admin");
            await AddUserRoleAsync("Publisher");
        }

        public async Task AddUserRoleAsync(string userRole)
        {
            if (await _roleManager.FindByNameAsync(userRole) == null)
            {
                await _roleManager.CreateAsync(new IdentityRole { Name = userRole });
            }
        }
    }
}
```

User seed:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using EPHMS.Models;

namespace EPHMS.Configuration
{
    public class UserSeed
    {
        private readonly UserManager<ApplicationUser> _userManager;

        public UserSeed(UserManager<ApplicationUser> userManager)
        {
            _userManager = userManager;
        }

        public async void Seed()
        {
            await AddUserAsync("a@ephms.no", "P@ssword1", "User");
            await AddUserAsync("b@ephms.no", "P@ssword1", "Admin");
            await AddUserAsync("c@ephms.no", "P@ssword1", "Admin");
        }

        public async Task AddUserAsync(string email, string pass, string role)
        {
            var newUser = new ApplicationUser { Email = email, UserName = email };
            if (await _userManager.FindByEmailAsync(email) == null)
            {
                var result = await _userManager.CreateAsync(newUser, pass);
                if (result.Succeeded)
                    result = await _userManager.AddToRoleAsync(newUser, role);
            }
        }
    }
}
```

Appendix D

```
USE EPHMS-db --EMISSIONSOURCE AND ANNUALDATA DOESNT GENERATE IDENTITY.
GO
--DELETING PREVIOUS INFORMATION
DELETE FROM EMISSIONSOURCE
GO
DBCC CHECKIDENT ('EMISSIONSOURCE',RESEED, 0)
GO
DELETE FROM MeasurementSTATION
GO
DBCC CHECKIDENT ('MeasurementSTATION',RESEED, 0)
GO
DELETE FROM CITY
GO
DBCC CHECKIDENT ('CITY', RESEED, 0)
GO
DELETE FROM AREA
GO
DBCC CHECKIDENT ('AREA',RESEED, 0)
GO
DELETE FROM COUNTRY
GO
DBCC CHECKIDENT ('COUNTRY',RESEED, 0)
GO
DELETE FROM MEASUREMENTTYPE
GO
DBCC CHECKIDENT ('MEASUREMENTTYPE',RESEED, 0)
GO
DELETE FROM MeasurementGROUP
GO
DBCC CHECKIDENT ('MeasurementGROUP',RESEED, 0)
GO
DELETE FROM MEASUREMENTUNIT
GO
DBCC CHECKIDENT ('MEASUREMENTUNIT',RESEED, 0)
GO
DELETE FROM MEASUREMENTUNITGROUP
GO
DBCC CHECKIDENT ('MEASUREMENTUNITGROUP', RESEED, 0)
GO
DBCC CHECKIDENT ('PROPERTY', RESEED, 0)
GO

--COUNTRY TABLE INFO
INSERT INTO COUNTRY(CountryName) VALUES ('Norway')
--AREA TABLE INFO
INSERT INTO AREA(AreaName,CountryId) VALUES('Grenland',1)
--CITY TABLE INFO
INSERT INTO CITY(CityName,AreaId,CountryId) VALUES ('Porsgrunn',1,1)
INSERT INTO CITY(CityName,AreaId,CountryId) VALUES ('Skien',1,1)
--MEASUREMENTSTATION TABLE INFO
INSERT INTO
MEASUREMENTSTATION(MeasurementStationName,MeasurementStationLongitude,MeasurementStationLatitu
de,CityId,AreaId,CountryId) VALUES ('Sverresgate Målestasjon',9.652195,59.138162,1,1,1)
INSERT INTO
MEASUREMENTSTATION(MeasurementStationName,MeasurementStationLongitude,MeasurementStationLatitu
de,CityId,AreaId,CountryId) VALUES ('Øyekast Målestasjon',9.642207,59.129194,1,1,1)
INSERT INTO
MEASUREMENTSTATION(MeasurementStationName,MeasurementStationLongitude,MeasurementStationLatitu
de,CityId,AreaId,CountryId) VALUES ('Lensmannsdalen Målestasjon',9.635742,59.159296,2,1,1)
INSERT INTO
MEASUREMENTSTATION(MeasurementStationName,MeasurementStationLongitude,MeasurementStationLatitu
de,CityId,AreaId,CountryId) VALUES ('Haukenes Målestasjon',9.48708966,59.20243518,2,1,1)
INSERT INTO
MEASUREMENTSTATION(MeasurementStationName,MeasurementStationLongitude,MeasurementStationLatitu
de,CityId,AreaId,CountryId) VALUES ('Furulund Målestasjon',9.69563115,59.05730396,1,1,1)
--MEASUREMENTGROUP TABLE INFO
INSERT INTO MEASUREMENTGROUP(MeasurementGroupName,MeasurementGroupDescription) VALUES ('Air
Quality','Contains types of data related to Air Quality.')
INSERT INTO MEASUREMENTGROUP(MeasurementGroupName,MeasurementGroupDescription) VALUES ('Global
Pollution','Global pollutant problem.')
INSERT INTO MEASUREMENTGROUP(MeasurementGroupName,MeasurementGroupDescription) VALUES ('Old
Pollution','The pollutants that was used long time ago.')
--MEASUREMENTUNITGROUP TABLE INFO
```

```
INSERT INTO MEASUREMENTUNITGROUP(MeasurementUnitGroupName,MeasurementUnitGroupDescription)
VALUES ('Density','Mass divided by volume')
INSERT INTO MEASUREMENTUNITGROUP(MeasurementUnitGroupName,MeasurementUnitGroupDescription)
VALUES ('Mass','Mass of substance')
--MEASUREMENTUNIT TABLE INFO
INSERT INTO
MEASUREMENTUNIT(MeasurementUnitProperty,MeasurementUnitDescription,MeasurementUnitScaling,Meas
urementUnitGroupId) VALUES ('ug/m^3','Micro gram per cubic meter',1,1)
--MEASUREMENTTYPE TABLE INFO
INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Nitrogen dioxide','An
intermediate in the industrial synthesis of nitric acid, millions of tons of which are
produced each year.','NO2',1,1, GETDATE())
INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('NOx','Combination of NO and NO2
that is formed in a high temperature.','NOx',1,1,GETDATE())
INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Sulfur Dioxide','Formed by
combustion by sulphurous materials mainly oil and coal.','SO2',1,1,GETDATE())
INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Particulate Matter 10','Dust
particles smaller than 10µm.','PM10',1,1,GETDATE())
INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Particulate Matter 2.5','Dust
particles smaller than 2.5µm.','PM2.5',1,1,GETDATE())
INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Ozone','A reactive gas that
exist both near the ground and in the stratosphere.','O3',1,1,GETDATE())
--INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Carbon Dioxide','The most
important pollutant into global warming. This gas is a man-made','CO2',2,2,GETDATE())
--INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Nitrous Oxide','The third
important natural climate pollutant in Norway.','N2O',2,2,GETDATE())
--INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Sulfur Hexafluoride','A man-made
chemical, non- reactive, colorless, non-flammable and one of the heaviest of all
pollutants.','SF6',2,2,GETDATE())
--INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Dioxins in the fjord','Complex
chemical element that contains chlorine and heavy organic chemicals.','',3,3,GETDATE())
--INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Mercury','The most hazardous
pollutant and constitute a threat for the environment and human health.','Hg',3,3,GETDATE())
--INSERT INTO
MEASUREMENTTYPE(MeasurementTypeName,MeasurementTypeDescription,MeasurementTypeAcronym,Measurem
entGroupId,MeasurementUnitId,MeasurementTypeCreated) VALUES ('Polychlorinated Biphenyl','A
synthetically produced substance with two aromatic rings that can have from 1-10 chlorine
atoms connected to them.','PCB',3,3,GETDATE())
--EMISSIONSOURCE TABLE INFO
INSERT INTO
EMISSIONSOURCE(EmissionSourceName,EmissionSourceDescription,CityId,AreaId,CountryId) VALUES
('Porsgrunn Municipality','Measures the local quality in air',1,1,1)
--PROPERTY TABLE INFO
INSERT INTO
PROPERTY(MeasurementTypeId,PropertyCreated,PropertyLastModified,PropertyLL,PropertyL,PropertyH
,PropertyHH) VALUES (4,GETDATE(),GETDATE(),0,0,50,50)
```