

Web Programming ASP.NET Core

Hans-Petter Halvorsen



<https://www.halvorsen.blog>

Web Programming

ASP.NET Core

Hans-Petter Halvorsen

2021

Preface

This textbook gives an overview of Web and Web programming in general and with focus on ASP.NET and ASP.NET Core. ASP.NET and ASP.NET Core are web development frameworks created by Microsoft.

The only way to learn programming is to do a lot of coding by yourself, and not only small code snippets with a few lines of code. You need to make large Applications. It takes time and may be demanding, but that's the only way! The reward is knowledge that goes deep, and you will gain skills that is highly desired by the industry.

Web Page:



ASP.NET: <https://www.halvorsen.blog/documents/programming/web/aspnet/>

Videos:



ASP.NET Core Web Programming YouTube Playlist:

<https://www.youtube.com/watch?v=lcQsWYgQXK4&list=PLdb-TcK6Aqj34rTHSk6C1jZQgeALWS1qO>

Here you will find videos that introduces the ASP.NET Core topics covered in this textbook.

Other useful YouTube Playlists:



C# YouTube Playlist: <https://www.youtube.com/watch?v=l6Mq79Dai7M&list=PLdb-TcK6Aqj0fji9OdAl4L9ydhiD3KUX8>



Visual Studio YouTube Playlist:
<https://www.youtube.com/watch?v=3NQAWzatqvA&list=PLdb-TcK6Aqj3pVNwegVKUGoHN3mi6IXjk>



SQL Server YouTube Playlist:
<https://www.youtube.com/watch?v=pMGW353gauo&list=PLdb-TcK6Aqj3DCOx-CiG0ddUrUQ86r2Nz>



Database Systems YouTube Playlist: <https://www.youtube.com/watch?v=n75iPNrzN-o&list=PLdb-TcK6Aqi0PedGwO7CUI6WBRyia7EQh>

Information about the author:



Hans-Petter Halvorsen

The author currently works at the University of South-Eastern Norway. The author has been working with Software Engineering and Industrial IT Projects for more than 20 years.

My Web Site:

<https://www.halvorsen.blog/>

You may also scan the QR code below:



My YouTube Channel “Industrial IT and Automation”:

<https://www.youtube.com/IndustrialITandAutomation>

<https://www.halvorsen.blog>

Table of Contents

Preface.....	2
Part 1 : Introduction	12
1 Introduction.....	13
1.1 Applications.....	14
1.1.1 Desktop Applications	14
1.1.2 Web Applications.....	14
1.1.3 Mobile Applications	15
1.2 .NET.....	15
1.3 Web	16
2 ASP.NET.....	17
2.1 ASP.NET Web Forms	18
2.2 ASP.NET Core with Razor	18
Part 2 : Visual Studio and C#	20
3 Visual Studio.....	21
3.1 Visual Studio macOS	22
4 Desktop Applications	25
4.1 Windows Forms App.....	25
4.2 WPF Application	25
4.3 Universal Applications	25
5 C# Programming Language.....	26
5.1 Introduction	26
5.2 Classes.....	27
5.3 Inheritance	27
5.3.1 Example	27
5.4 Polymorphism	29
5.4.1 Example	29
5.5 Interfaces	29
5.5.1 Example	30

5.6	Generics	30
5.7	Additional C# Resources.....	31
5.7.1	Windows Forms Appa.....	31
Part 3 : Web Fundamentals.....		32
6	The Web.....	33
6.1	Web Programming.....	35
6.2	Client-Server	36
6.3	Web Server	37
6.4	Web Browsers	39
6.5	HTML.....	40
6.6	CSS	41
6.7	JavaScript	41
6.8	Server-side Frameworks.....	42
6.8.1	PHP.....	42
6.8.2	ASP.NET.....	43
6.8.3	Django.....	43
6.8.4	JavaScript Server-side Frameworks	44
6.9	Web Data Formats	44
6.9.1	XML.....	44
6.9.2	JSON.....	44
7	HTML.....	46
7.1	HTML in Visual Studio	47
7.2	HTML Tags	47
7.2.1	Title	48
7.2.2	Headers.....	48
7.2.3	Paragraphs	48
7.2.4	Hyperlinks	48
7.2.5	Images.....	48
7.2.6	Tables.....	49

7.2.7	Comments.....	49
7.3	Additional Resources	49
8	CSS.....	50
8.1	External Style Sheets.....	51
8.2	Bootstrap	51
8.3	Font Awesome.....	52
8.4	Additional Resources	52
9	JavaScript.....	53
9.1.1	Additional Resources	54
9.2	jQuery	54
9.3	AngularJS.....	55
9.4	TypeScript (Microsoft)	55
10	Server-side Frameworks	56
10.1	PHP	56
10.2	Django.....	56
10.3	ASP.NET.....	56
Part 4 : ASP.NET Core.....		57
11	Introduction to ASP.NET Core	58
11.1	Resources	59
11.2	Hello World Application	59
11.3	ASP.NET Core with Razor	64
11.3.1	Basic Examples	65
11.4	Query String Data	70
11.5	Form Data.....	71
12	ASP.NET Core Fundamentals.....	74
12.1	Startup Class.....	75
12.2	Web root	75
12.3	appsettings.json	76
12.4	Shared Pages	77
12.4.1	Layout.....	77
12.5	Models	78

12.6	Razor Pages	79
12.6.1	Sending data from the Page Model to the Razor File.....	80
12.7	Additional Resources	82
13	<i>Razor</i>	83
13.1	Razor Syntax.....	83
13.2	Model.....	84
<i>Part 5 Database Communication</i>		86
14	<i>Database Systems</i>	87
14.1	SQL Server	87
14.1.1	SQL Server Management Studio.....	88
14.2	Structured Query Language (SQL).....	91
14.2.1	Tables	92
14.2.2	Views	101
14.2.3	Stored Procedures.....	103
14.2.4	Triggers.....	104
15	<i>ADO.NET</i>	105
16	<i>Data from Database</i>	106
16.1	Demo Application	106
16.1.1	Database.....	107
16.1.2	Visual Studio.....	109
16.2	Where should we put the Connection String?.....	120
16.2.1	appSettings.json	120
17	<i>CRUD Applications</i>.....	123
17.1	Demo Application	123
17.1.1	Create the Visual Studio Project.....	125
17.1.2	Database.....	127
17.1.3	Index (Start Page)	131
17.1.4	Models.....	131
17.1.5	Show Books	137

17.1.6	New Book	139
17.1.7	Edit Book	141
17.1.8	Delete Book	144
Part 6 Additional ASP.NET Core Features		146
18	Session Data	147
18.1	Session State in ASP.NET Core	148
18.2	Example - Share Data between 2 Web Pages	150
18.2.1	Page1	151
18.2.2	Page2	153
18.3	Additional Resources	155
Part 7 Charting		156
19	Charting	157
19.1	Introduction	157
20	Google Charts	159
20.1	Google Charts Implementation	160
20.2	Google Charts Examples	161
20.2.1	Basic Chart Example	162
20.2.2	Database Examples.....	163
20.2.3	Line Chart Example.....	166
20.2.4	Bar Chart Example	168
20.2.5	Column Chart Example	169
20.2.6	Multi-Line Chart Example	170
21	Chart.js	173
Part 8 APIs.....		174
22	Class Libraries	175
22.1	Demo Application	176
22.1.1	Class Library.....	178
22.1.2	BookAdm App.....	180
22.1.3	BookStore App.....	184

22.2	Final System	186
23	Web API.....	188
Part 9 User Login and ASP.NET Core Identity		189
24	User Identity and Login	190
24.1	Password Security.....	190
24.1.1	Encryption and Decrypting	190
24.1.2	Hashing.....	191
24.1.3	Salting	193
24.2	Microsoft.AspNetCore.Identity	194
24.2.1	PasswordHasher<TUser> Class.....	194
24.3	Session State in ASP.NET Core.....	195
24.4	Demo Application	195
24.4.1	Login	197
24.4.2	Create User	197
24.4.3	Update User Information	198
24.4.4	More Features	198
25	ASP.NET Core Identity.....	199
25.1	Introduction	199
25.1.1	Scaffold Identity in ASP.NET Core Projects.....	199
25.2	Demo Application	200
25.2.1	Create Project in Visual Studio with Identity Enabled.....	200
25.2.2	Create Identity Database.....	202
25.2.3	Register New Account and Log In.....	204
25.2.4	2 Factor Authentication.....	207
25.2.5	Start Creating your Application	209
25.2.6	Scaffolding	211
25.3	Additional Resources	215
Part 10 Testing		216
26	Unit Testing	217

Part 11 Deployment	218
27 Web Servers	219
28 Deployment in Visual Studio	222
29 Internet Information Services (IIS)	223
29.1 Installation	223
29.1.1 Windows Features.....	224
29.1.2 .NET Core Hosting Bundle	224
29.2 Demo Application	224
29.2.1 Add Application	225
Part 12 Microsoft Azure	227
30 Introduction to Azure	228
30.1 Azure Web Portal	228
31 Databases in Azure	229
31.1 Create the Database	229
31.1.1 Azure Data Studio.....	229
31.2 Create Tables, etc.	230
31.2.1 SQL Server Management Studio.....	231
31.2.2 Azure Data Studio.....	231
32 Web Applications in Azure	232
32.1 App Service	232
32.2 Default Document	233
Part 13 Resources	234
33 Bootstrap	235
34 Font Awesome	236
Part 14 Applications	237
35 Weather System	238
35.1 ASP.NET Core Web Application	239
35.2 Database	240
35.3 Visual Studio Project	241
35.4 Connection String	242

35.4.1	appSettings.json	242
35.5	Index Page (Start Page)	243
35.6	Weather Overview Page	244
35.7	Charts Page.....	244
35.8	Weather Parameters Page	249
35.9	Weather Information Page.....	249
35.10	About Page.....	250
35.11	Deployment to Azure.....	251
36	<i>Voting System</i>	253
37	<i>Data Management System (DMS).....</i>	257

Part 1 : Introduction

This part introduces the topics covered in this textbook and puts it into a proper context.

1 Introduction

Learning Web Technology is essential today because Internet has become the number one source to information, and many of the traditional software applications have become Web Applications. Web Applications have become more powerful and can fully replace desktop application in most situations.



Figure 1-1: Web Applications – The New Era of Application Development

That's why you need to know basic Web Programming, including HTML, CSS and JavaScript. To create more powerful Web Sites and Web Applications you also need to know about Web Servers, Database Systems and Web Frameworks like PHP, ASP.NET, etc.

It all started with Internet (1960s) and the World Wide Web - WWW (1991). The first Web Browser, Netscape, came in 1994. This was the beginning of a new era, where everything is connected on internet, the so-called Internet of Things (IoT).

This textbook contains of the following:

- Part 1: Introduction
- Part 2: Visual Studio and C#
- Part 3: Web Fundamentals
- Part 4: ASP.NET Core

- Part 5: Database Communication
- Part 6: Additional ASP.NET Core Features
- Part 7: Charting
- Part 8: APIs
- Part 9: User Login and ASP.NET Core Identity
- Part 10: Testing
- Part 11: Deployment
- Part 12: Microsoft Azure
- Part 13: Resources
- Part 14: Applications

1.1 Applications

We can separate applications in different categories:

- Desktop Applications
- Web Applications
- Mobile Applications

These will shortly be described below. In this textbook we will focus on creating web applications using ASP.NET Core.

1.1.1 Desktop Applications

Desktop Applications comes in many flavors:

- Windows WinForms Desktop Applications (Windows Forms App)
- WPF Desktop Applications
- Universal Applications

Windows Forms Apps are the oldest but are still very popular.

1.1.2 Web Applications

Web Applications use a mix of different technologies and different programming languages.

We have:

- Static Web Pages: HTML, CSS, JavaScript
- Dynamic Web Applications: PHP, ASP.NET, Django, etc.

1.1.3 Mobile Applications

So-called Apps has been very popular since the release of the first iPhone in 2007. Today we have Apple products using iOS and we have Android devices.

We have:

- iOS
- Android
- iPadOS
- etc.

The great thing about Web Applications is that they also work on Mobile platforms. Native apps for Android phones and Apple devices typically need to be developed in a strict environment and with specific programming languages and they only work for the platform they are developed for. For Android development you typically will use the Kotlin programming language in combination with Android Studio. While for development for the Apple platform you will need a Mac computer and use the Swift programming language as part of the Xcode development environment.

1.2 .NET

The .NET is a development platform from Microsoft.

Previously we had the following:

- .NET Framework (latest version is .NET Framework 4.x)
- .NET Core (latest version is .NET Core 3.x)

.NET Framework was first introduced in 2002 and works only for the Windows operating system, while .NET Core was introduced in 2016. .NET Core is cross-platform, meaning it will work on Windows, Linux and macOS.

The newly released .NET 5 has merged .NET Framework and .NET Core into a more unified package.

.NET Core and the newly released .NET 5 (that is built on .NET Core) will be the future of .NET. Microsoft is still committed to support the previous .NET Framework for years to come, but for new applications your choice should be .NET 5 either you want to develop desktop applications (“Windows Forms App”) or web applications (“ASP.NET Core Web App”).

You can use many different programming languages with .NET, but the “default” language is C#, which is the language we will use in this textbook.

The .NET 5 (and newer) is aiming to be cross-platform, and it is supported on Windows, macOS, and Linux when possible.

The .NET web site: <https://dotnet.microsoft.com>

1.3 Web

We have 2 types of web pages:

- Static web pages
- Dynamic web pages

Static web pages are pure HTML web pages where the contents is written in manually and it doesn't change unless the user updates the contents.

Dynamic Web Pages typically get contents from a Database and have deeper interaction with the user.

Dynamic Web Pages using e.g., ASP.NET or PHP Executes Code on the Server-side and Generates HTML Code that is sent to the Client (Web Browser). This is known as Server-side code.

You can also create Dynamic content on the Client using JavaScript.

Static Web Pages typically contains Text and Images that is not changing (unless a person changes the page and upload a new version). Static Web Pages are Pure HTML pages

Video:



Web Programming Overview: <https://youtu.be/plRBYKbQSuE>

2 ASP.NET

ASP.NET is an open-source web framework, created by Microsoft, for building web apps and services using the .NET Framework or the .NET Core. We have both ASP.NET and ASP.NET Core. ASP.NET Core is the new approach built on .NET Core.

Figure 2-1 shows the concept of ASP.NET.

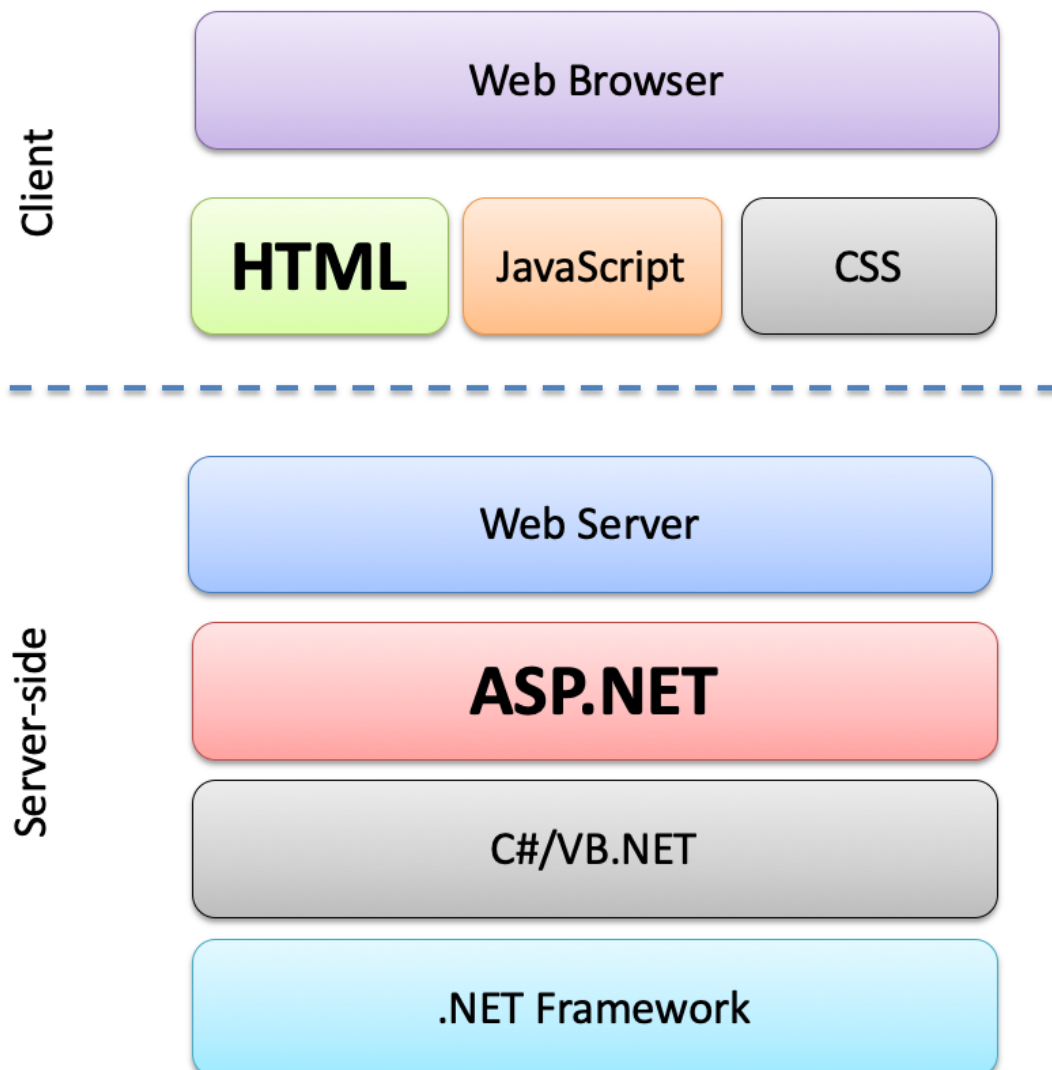


Figure 2-1: ASP.NET

ASP.NET comes in many flavors:

- ASP.NET Web Forms - The same programming model as WinForms. If you already know WinForms, this is an easy access to the world of web programming.

- ASP.NET MVC (Model-View Controller). If you are familiar with the MVC approach, this could be your choice.
- ASP.NET with Razor Pages - This is the latest and recommended way. This has become the "default" approach for ASP.NET today. It mixes the best from all the others combined with PHP like syntax (PHP is probably the most popular Web Framework today)

This textbook will focus on **ASP.NET Core with Razor Pages**.

2.1 ASP.NET Web Forms

The same programming model as WinForms. If you already know WinForms, this is an easy access to the world of web programming.

This textbook will focus on **ASP.NET Core with Razor Pages**.

Still, if you want to get an overview of ASP.NET Web Forms you may take a look at the following:



Introduction to ASP.NET Web Programming using Web Forms:

<https://youtu.be/R7VuJt6TqA8>

2.2 ASP.NET Core with Razor

Again, we have different options we can use:

- Razor with MVC
- Razor Single Page Model (comparable to PHP but using C# syntax instead)
- Razor with Page Model (Code and Layout are separated in different Files)

This textbook will focus on **Razor with Page Model** (Code and Layout are separated in different Files).

An ASP.NET Razor page has the “.cshtml” (e.g., “Index.cshtml”) file extension. This file contains a mix of HTML and Razor syntax. The Razor syntax is actually C# code mixed together with the HTML code.

The Razor parts of the file are executed on the web server before the page is sent to the client (your web browser).

The Razor page may also have a C# code file linked to it, this file has the extension “.cshtml.cs” (e.g., “Index.cshtml.cs”). The “.cshtml.cs” file is called the Page Model.

In Razor with Page Model each Razor page is a pair of files:

- A “.cshtml” file that contains HTML markup with C# code using Razor syntax.
- A “.cshtml.cs” (“code behind”) file that contains C# code that handles page events.

In this textbook we will go through ASP.NET Core in detail. In Part 3: Web Fundamentals we go through the foundations for creating web applications in general, while in Part 4: ASP.NET Core we start creating ASP.NET Core Web Applications. If you cannot wait to start with ASP.NET Core, you take sneak peek at the introduction videos below.

Videos:

Below you find some short introduction videos to ASP.NET Core:



ASP.NET Core - Hello World: <https://youtu.be/lcQsWYgQXK4>



ASP.NET Core – Introduction: <https://youtu.be/zkOtiBcwo8s>

Part 2 : Visual Studio and C#

This part gives an overview of Visual Studio and basic C# programming, which will be the foundation for learning and development of ASP.NET Core Web Applications. If you already are familiar with using Visual Studio and C# for development of Windows Forms Apps, you can skip this part.

3 Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services, etc.

Home page of Visual Studio: <http://www.microsoft.com/visualstudio>

Figure 3-1 shows Visual Studio.

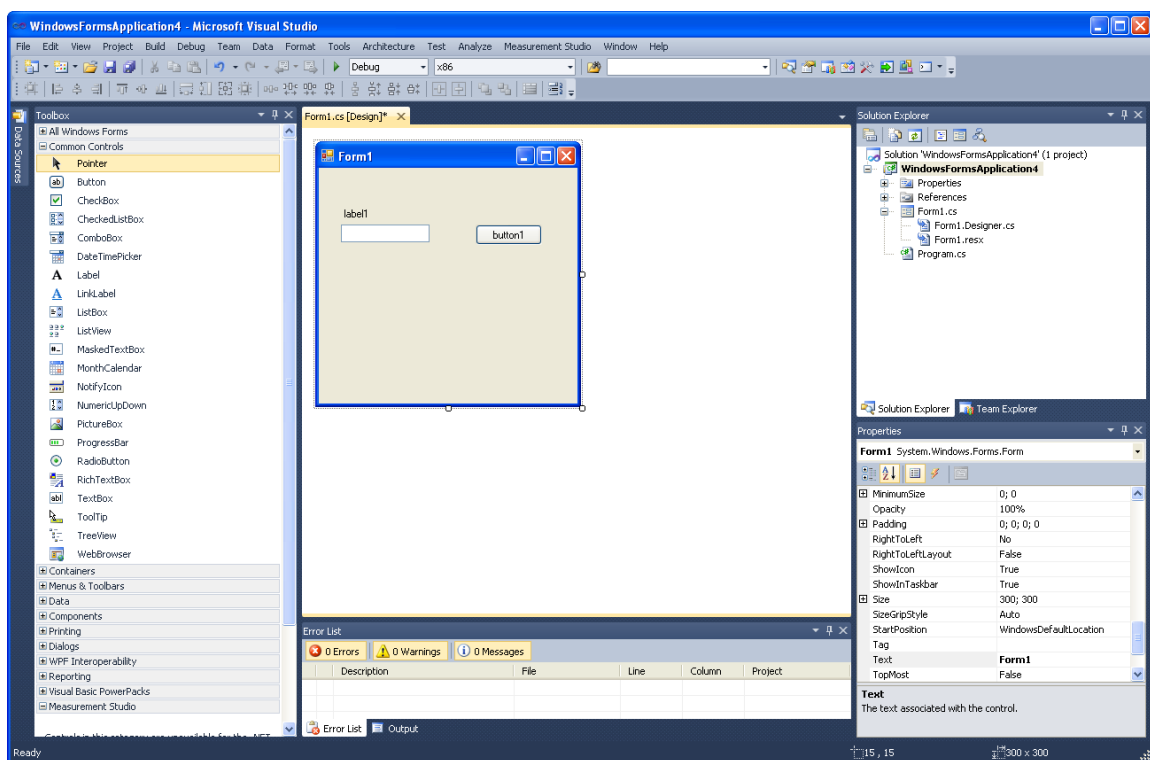


Figure 3-1: Visual Studio

New projects are created from the “New Project” window (Figure 3-2).

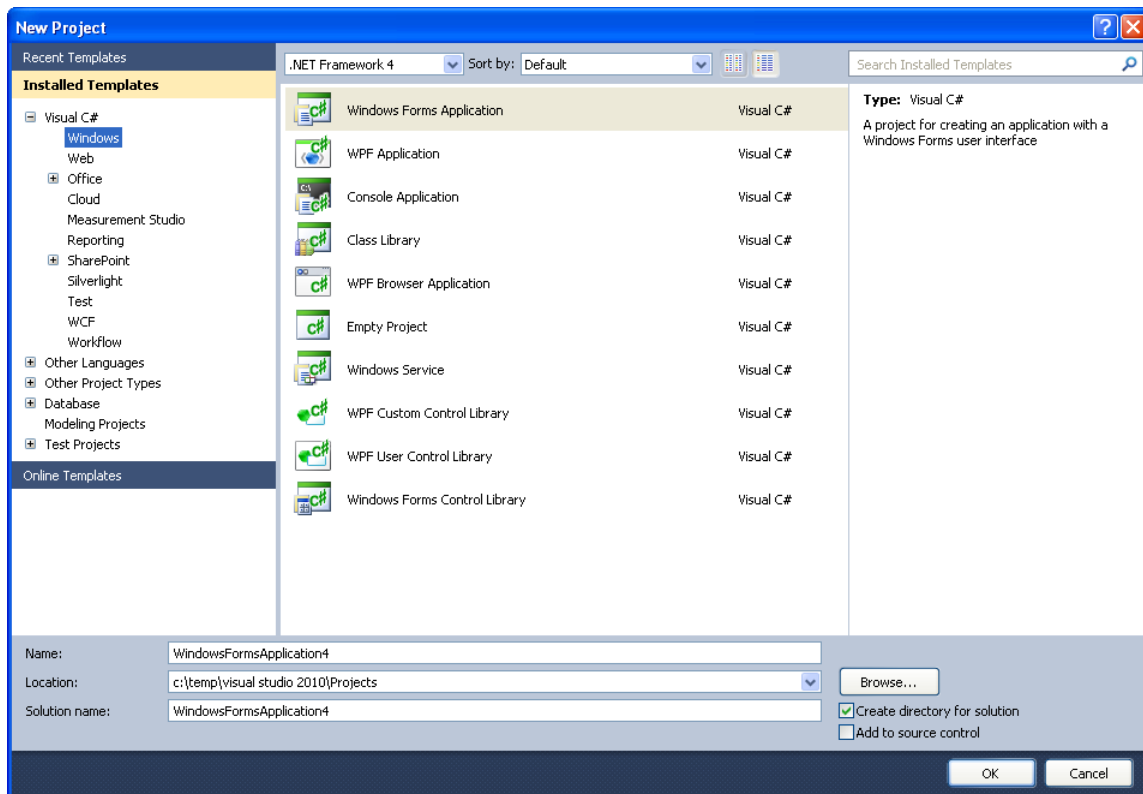


Figure 3-2: Visual Studio – New Project

3.1 Visual Studio macOS

Visual Studio for macOS supports .NET Core and Web programming (both ASP.NET Core and ordinary ASP.NET). It does not support ordinary Windows Desktop Programming, but you can create macOS desktop applications, iOS applications, tvOS applications, etc.

Figure 3-3 shows the New Project window in Visual Studio for macOS.

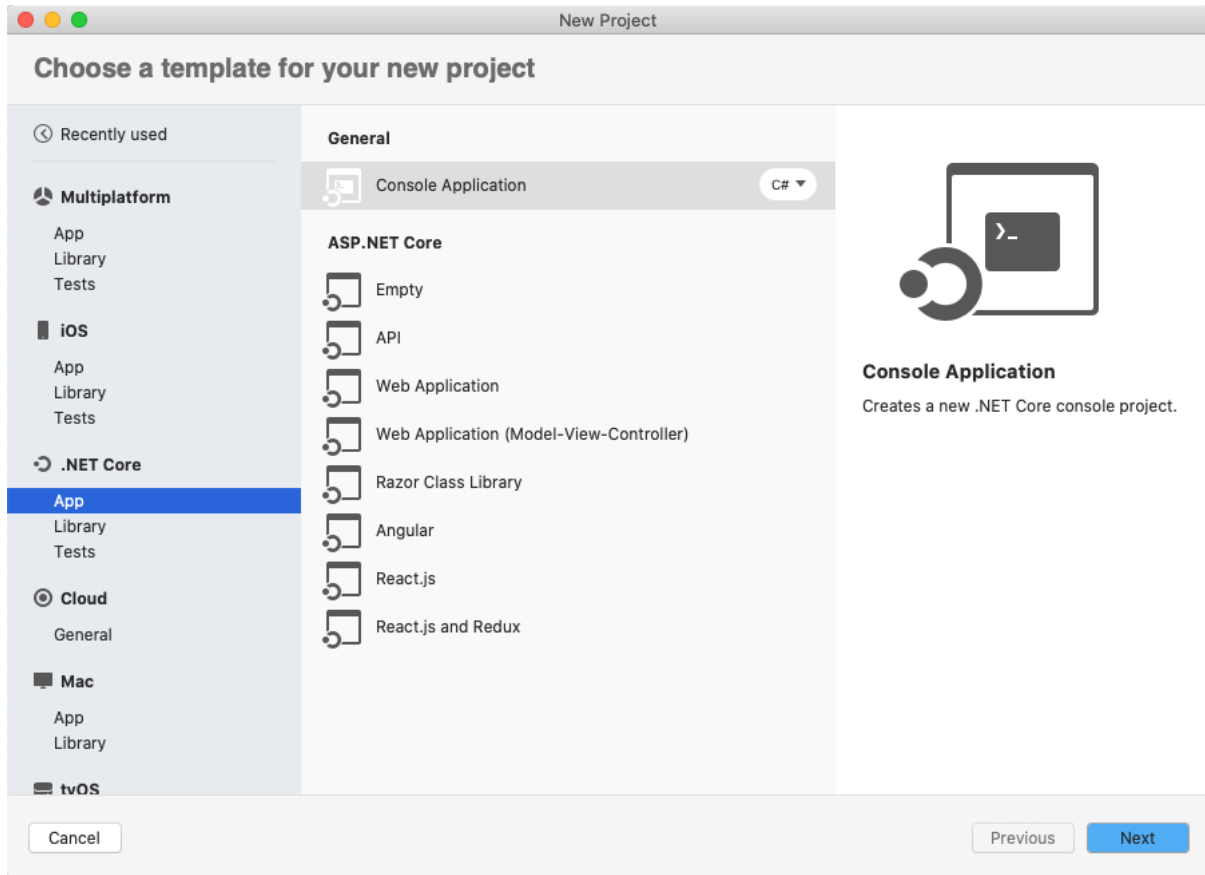


Figure 3-3: Visual Studio macOS – New Project

If you want to make Web Applications and you have a Mac, the Visual Studio for macOS has become an excellent choice.

You can create:

.NET Framework:

- ASP.NET Web Forms
- ASP.NET MVC

.NET Core:

- ASP.NET Core Web Application (Razor) – This is the default and recommended option.
- ASP.NET Core Web Application (MVC)
- Angular Web Applications
- React.js Web Applications
- Web API
- etc.

This means you have all the necessary tools available to create great web applications using C# on your Mac.

4 Desktop Applications

In the latest version of Visual Studio 2019 we can develop WinForms Desktop Applications and WPF Desktop Applications for both .NET Framework (4.x) and .NET Core (3.x).

.NET WinForms vs. .NET Core WinForms: Note that .NET Core is cross-platform, but .NET Core WinForms Desktop Application will (of course) only work on Windows.

4.1 Windows Forms App

This is the default approach. This is the number one method for creating Windows desktop applications with Visual Studio.

4.2 WPF Application

This is a newer approach for creating desktop applications in Visual Studio. It has a deeper separation of the GUI and the code.

WPF – Windows Presentation Foundation.

4.3 Universal Applications

This is an attempt to make universal application that works on any kind of devices from desktops to mobile phones.

They call it Universal Windows Platform applications, or UWP.

5 C# Programming Language

5.1 Introduction

C# is pronounced “see sharp”. C# is an object-oriented programming language and part of the .NET family from Microsoft. C# is very similar to C++ and Java. C# is developed by Microsoft and originally it worked only on the Windows platform. Now .NET has become an open-source project, and the new .NET Core is also cross platform meaning it works on Windows, macOS and Linux.

Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you've identified an object, you generalize it as a class of objects and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called an “object” or an “instance of a class”. The object or class instance is what you run in the computer. Its methods provide computer instructions and the class object characteristics provide relevant data. You communicate with objects - and they communicate with each other.

Important features with OOP are:

- Classes and Objects
- Inheritance
- Polymorphism
- Encapsulation

Simula was the first object-oriented programming language. Simula was developed in the 1960s by Kristen Nygaard from Norway.

Java, Python, C++, Visual Basic .NET and C# are popular OOP languages today.



C# Documentation: <https://docs.microsoft.com/en-us/dotnet/csharp/>

5.2 Classes

Classes are the fundamental building blocks in C#.

5.3 Inheritance

Inheritance is a feature of object-oriented programming languages that allows you to define a base class that provides specific functionality (data and behavior) and to define derived classes that either inherit or override that functionality.

5.3.1 Example

Assume you have different types of sensors sharing some common features, then you can, e.g., have a Base Class called `Sensor()` and then other derived Classes like `TemperatureSensor()`, etc. that either inherit or override the functionality of the base class.

If you have different types of Temperature Sensor, you can make derived classes like `Thermistor()` that inherit/ override functionality of `TemperatureSensor()`. See Figure 5-1.

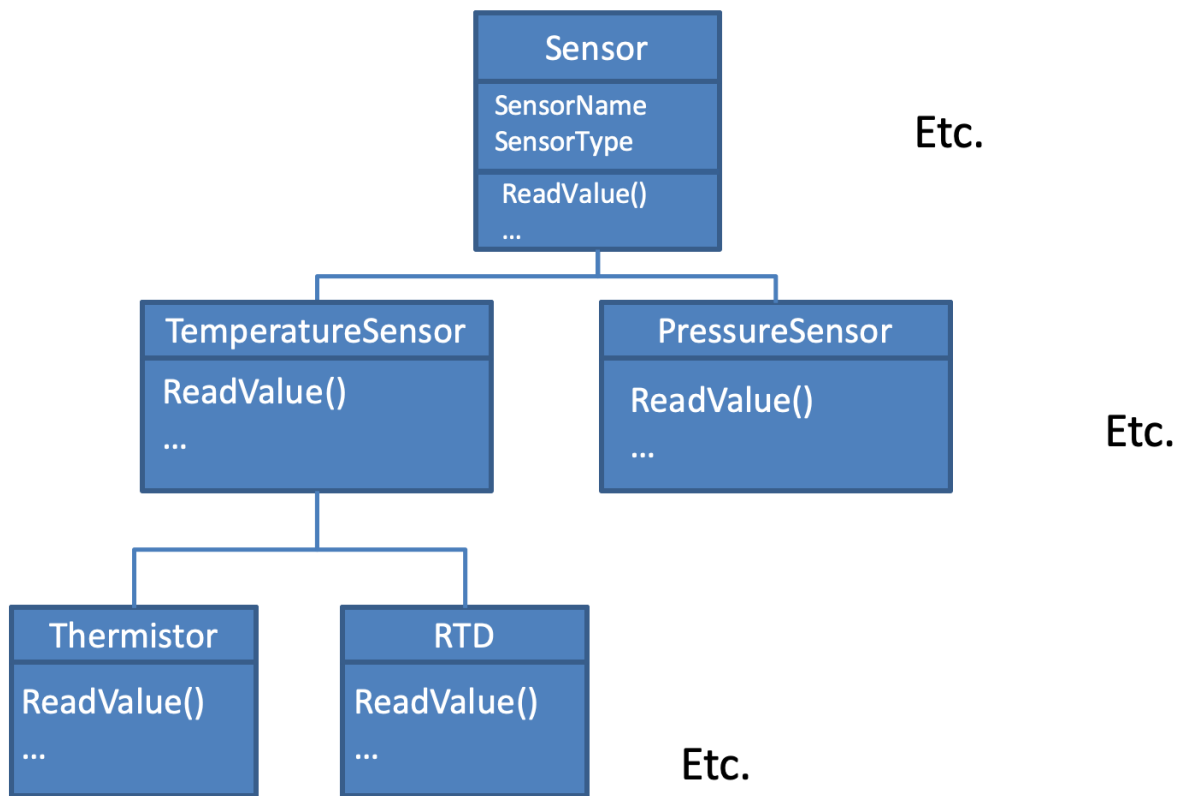


Figure 5-1: Inheritance Class Diagram - Example

Figure 5-2 shows a code example.

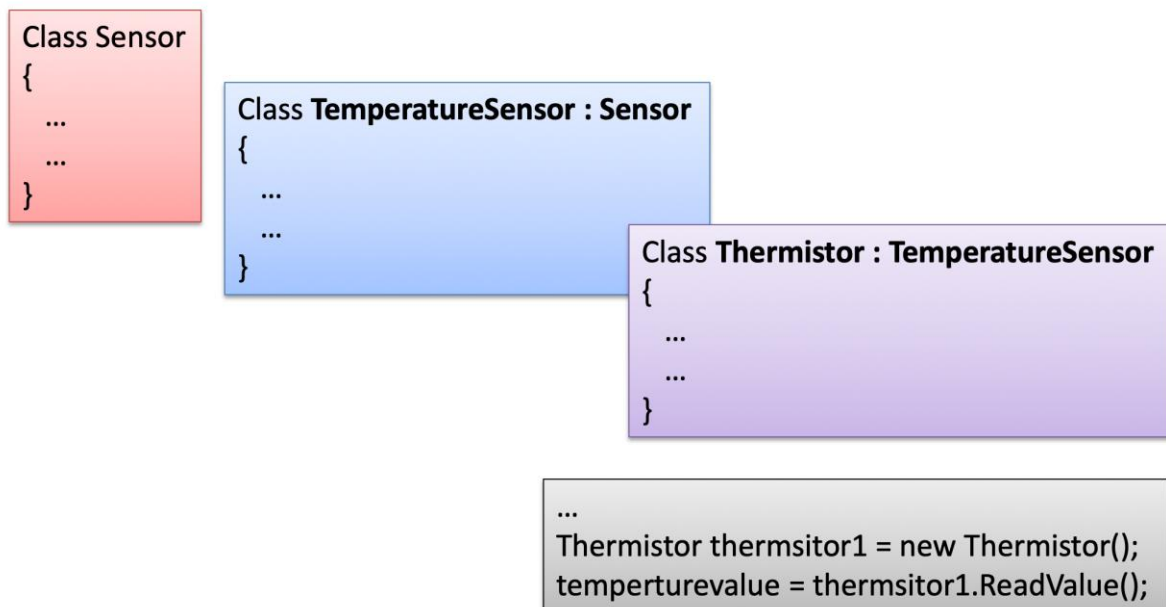


Figure 5-2: Inheritance – Code Example

5.4 Polymorphism

Polymorphism is an object-oriented feature that is part of all object-oriented programming languages.

5.4.1 Example

```

Class TemperatureSensor
{
    public virtual double y Scaling(double x)
    {
        y = 5*x;
    }
}
...
Class Thermistor : TemperatureSensor
{
    public override double y Scaling(double x)
    {
        y = 2*x + 2;
    }
}
...
Class Tmp36 : TemperatureSensor
{
    public override double y Scaling(double x)
    {
        y = x*(9/5) + 45;
    }
}

```

The override modifier allows a method to override the virtual method of its base class at run-time.

```

...
N = //Get number of sensors stored in the Database
TemperatureSensor[] tempsensor = new TemperatureSensor[N]

//Here you should use a for loop instead
tempsensor[0] = new TemperatureSensor()
tempsensor[1] = new Thermistor()
tempsensor[2] = new Tmp36()

foreach (Temperature sensor in tempsensor)
{
    x = //Get Value from DAQ device
    y = sensor.Scoring(x); //Scale the value
    SaveData(x); //Save Data to the Database
}

```

Here, depending of the Sensor, different Scaling formula will be used

Figure 5-3: Polymorphism Example

5.5 Interfaces

Interfaces are used along with classes to define what is known as a **contract**. A contract is an agreement on what the class will provide to an application.

An interface declares the properties and methods. It is up to the class to define exactly what the method will do.

An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies).

It is considered good practice to start with the letter "I" at the beginning of an interface, as it makes it easier for yourself and others to remember that it is an interface and not a class.

By default, members of an interface are abstract and public.

Note: Interfaces can contain properties and methods, but not fields

5.5.1 Example

Assume you make this system as an open platform meaning other developers can use it to add logging functionality from other sensors.

The system will not work if they don't implement a Name for the Sensor and a ReadValue() method.

To make sure that they follow this, you should implement Interfaces

Interface Example:

```
public interface ISensorType
{
    ...
    ...
    public double ReadValue (int id);
}
```

A Class that Implements the Interface

```
public class Rain : ISensorType
{
    ...
    public double ReadValue (int id)
    {
        double value;
        value = ReadDaq(id);
        return value*(9/5) +32;
    }
}
```

Figure 5-4: Interfaces Example

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class. To implement an interface, use the : symbol (just like with inheritance). The body of the interface method is provided by the "implement" class. Note that you do not have to use the override keyword when implementing an interface.

5.6 Generics

Generics allow you to define the specification of the data type of programming elements in a class or a method, until it is actually used in the program. In other words, generics allow you to write a class or method that can work with any data type.

You write the specifications for the class or the method, with substitute parameters for data types. When the compiler encounters a constructor for the class or a function call for the method, it generates code to handle the specific data type.

5.7 Additional C# Resources

This textbook is not a textbook for learning the basics within the C# programming language. A proper introduction to the C# Programming language is given here:



Visual studio and C#: <https://www.halvorsen.blog/documents/programming/csharp>



W3Schools: <https://www.w3schools.com/cs/index.php>

5.7.1 Windows Forms Appa

To get an overview of Windows Forms Apps you may want to take a closer look at the videos below.

Videos:

Below you find some videos regarding Visual Studio and C# programming creating Windows Forms Apps:



Simulation and Control with C# and WinForms: <https://youtu.be/l6Mg79Dai7M>



SQL Server with C# Windows Forms App: <https://youtu.be/rXugzELsQl0>



Datalogging using SQL Server with C#: <https://youtu.be/SkVcfQvRFDI>

Part 3 : Web Fundamentals

This part gives an overview of the fundamentals in web and web programming. ASP.NET Core is a framework for creating Web Applications, this means you need to have basic knowledge about the web and web programming in general before you can start creating ASP.NET Core Web Applications.

If you already are familiar with the fundamentals of the web, familiar with creating HTML web pages and have used, e.g., PHP for creating Web Applications, you can skip this part.

6 The Web

Learning Web Technology is essential today because Internet has become the number one source to information, and many of the traditional software applications have become Web Applications. Web Applications have become more powerful and can fully replace desktop application in most situations.

It all started with Internet (1960s) and the World Wide Web - WWW (1991). The first Web Browser, Netscape, came in 1994. This was the beginning of a new era, where everything is connected on internet, the so-called Internet of Things (IoT).

That's why you need to know basic Web Programming, including HTML, CSS and JavaScript. To create more powerful Web Sites and Web Applications you also need to know about Web Servers, Database Systems and Web Frameworks like PHP, ASP.NET, etc.

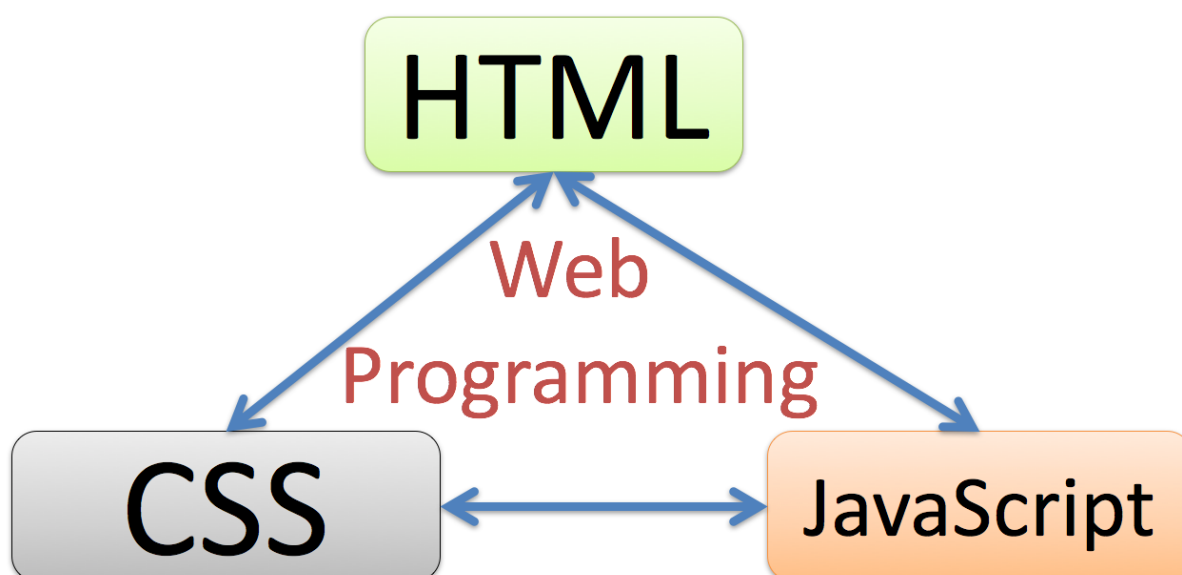


Figure 6-1: Web Programming Fundamentals

Use HTML to define the content of web pages, CSS is used to specify the layout of web pages, while JavaScript is used to program the behavior of web pages.

For creating more dynamic web pages, we typically also use a web framework like PHP or ASP.NET, etc. With these frameworks you can communicate with a database for storing or retrieving data.

Web is the Present and the Future

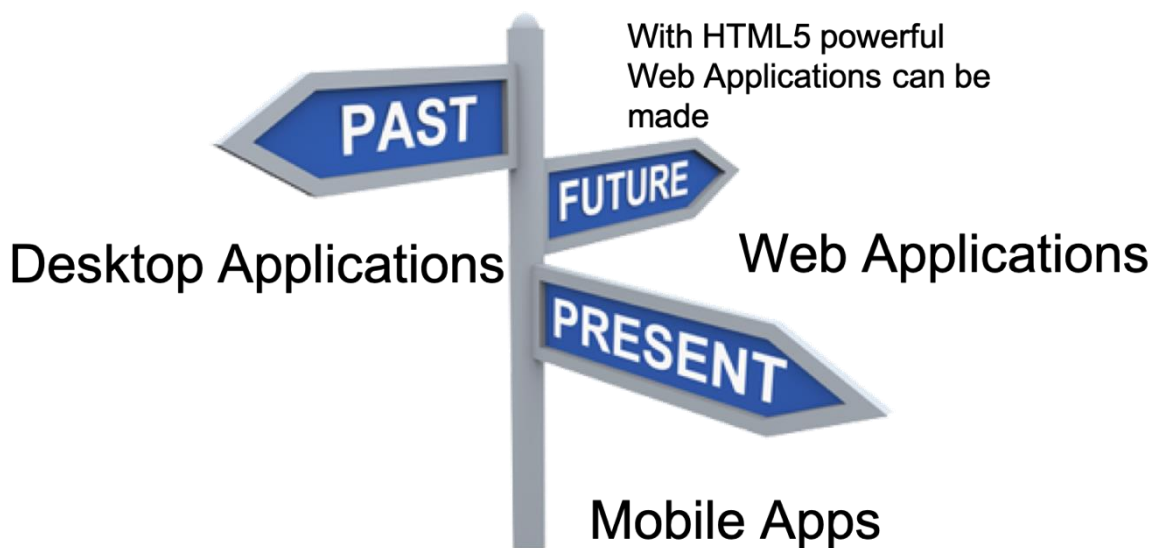


Figure 6-2: The Future of Programming

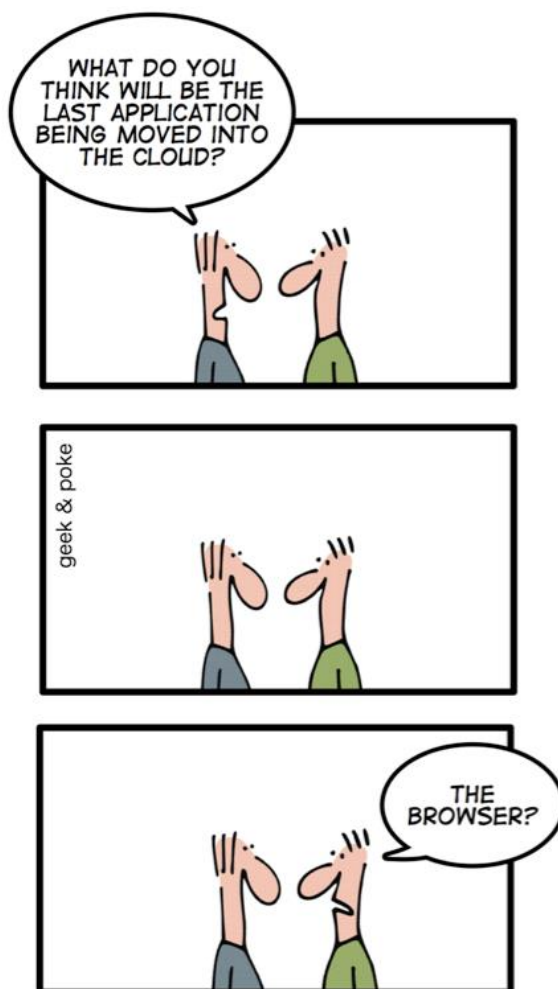


Figure 6-3: The new Era of Programming [<http://geek-and-poke.com>]

History of the Web:

- Internet (1960s)
- World Wide Web - WWW (1991)
- First Web Browser - Netscape, 1994
- Google, 1998
- Facebook, 2004
- Smartphones, 2007
- Tablets, 2010

6.1 Web Programming

We have different Web Development Environments:

Microsoft:

- Visual Studio (Windows, a scaled version is in beta for MacOS)
- Visual Studio Code (Cross-platform, open-source)

Others:

- WebStorm (JavaScript IDE, client-side development and server-side development with Node.js, etc.)
- Eclipse
- Atom (free and open-source text and source code editor for macOS, Linux, and Windows)
- Sublime
- Etc.

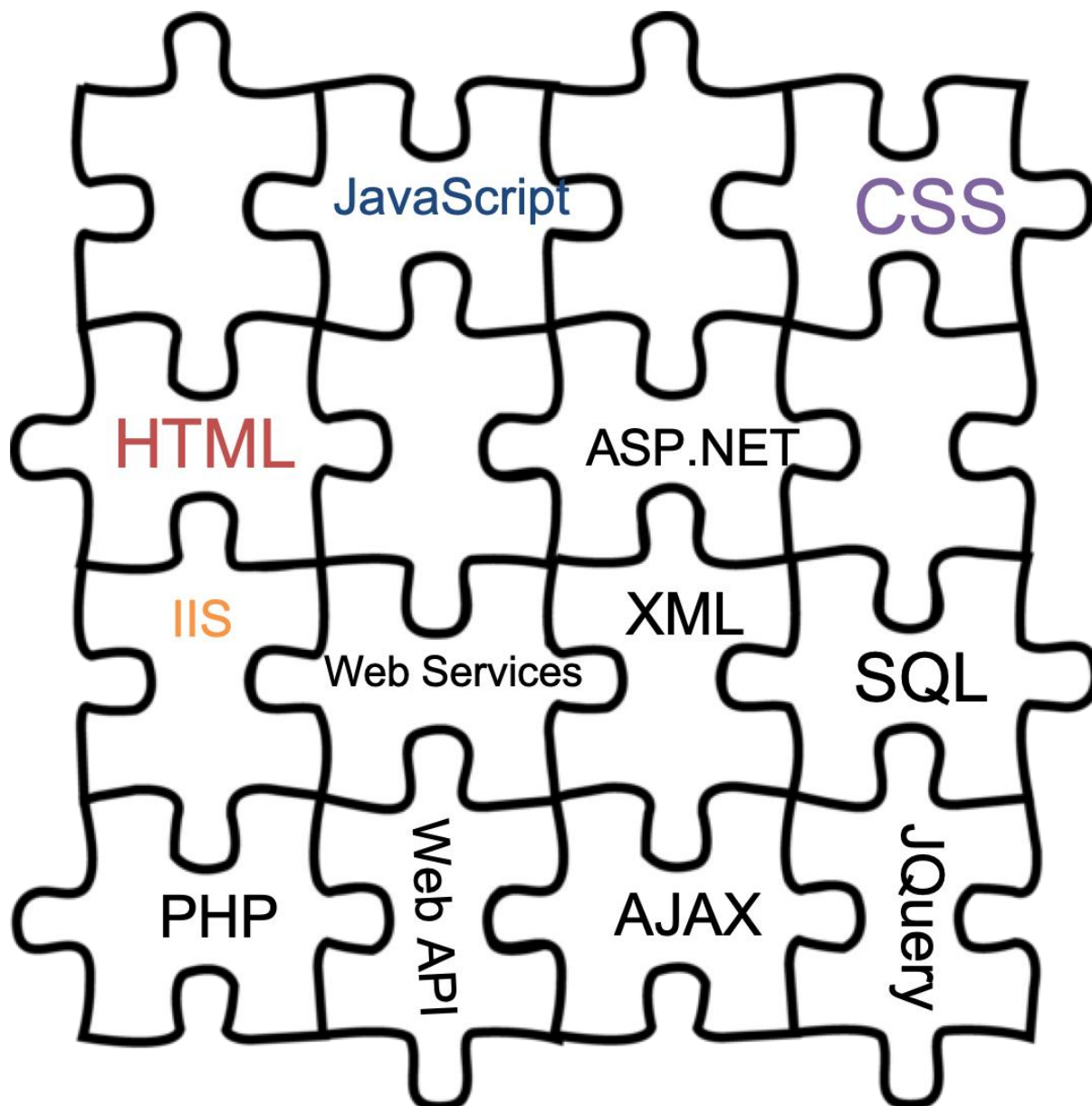


Figure 6-4: The Puzzle of Web Programming



Web Programming Overview: <https://youtu.be/plRBYKbQSuE>

6.2 Client-Server

The basic principle of web is that it is a server and a client. The server hosts the web pages and you request a web page from the client using a web browser.

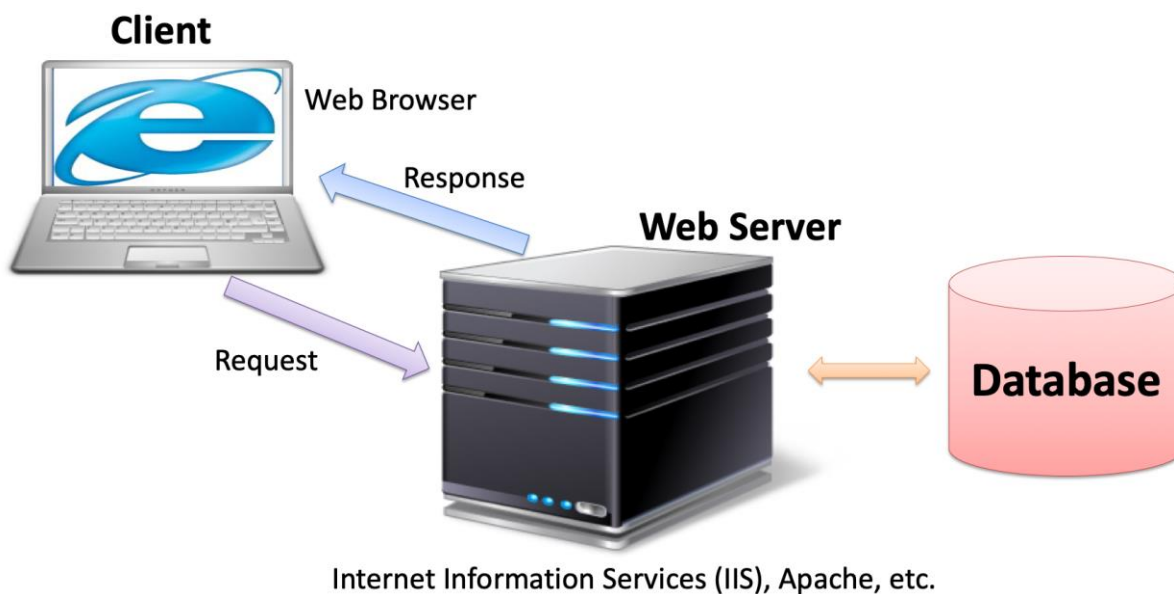


Figure 6-5: The Client-Server Nature of Web

6.3 Web Server

The term web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet.

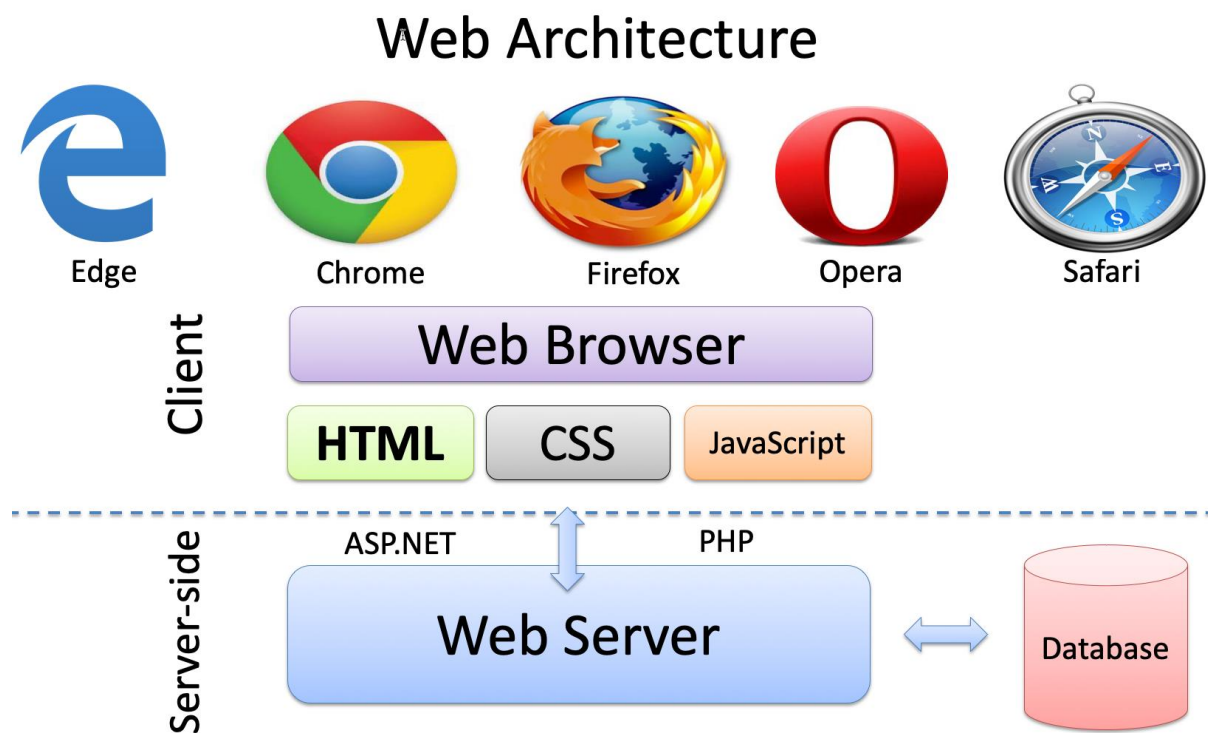


Figure 6-6: Web Architecture

The most common use of web servers is to host websites, but there are other uses such as gaming, data storage or running enterprise applications.

Here are some popular Web Servers:

- Internet Information Services (IIS) - Microsoft Windows
- Apache - Open Source, Cross-platform: UNIX, Linux, OS X, Windows, ...
- Nginx - (pronounced "engine x") - Has become very popular lately
- LiteSpeed
- GWS (Google Web Server)

The Web Browser creates the visual web page you see in the browser based on the

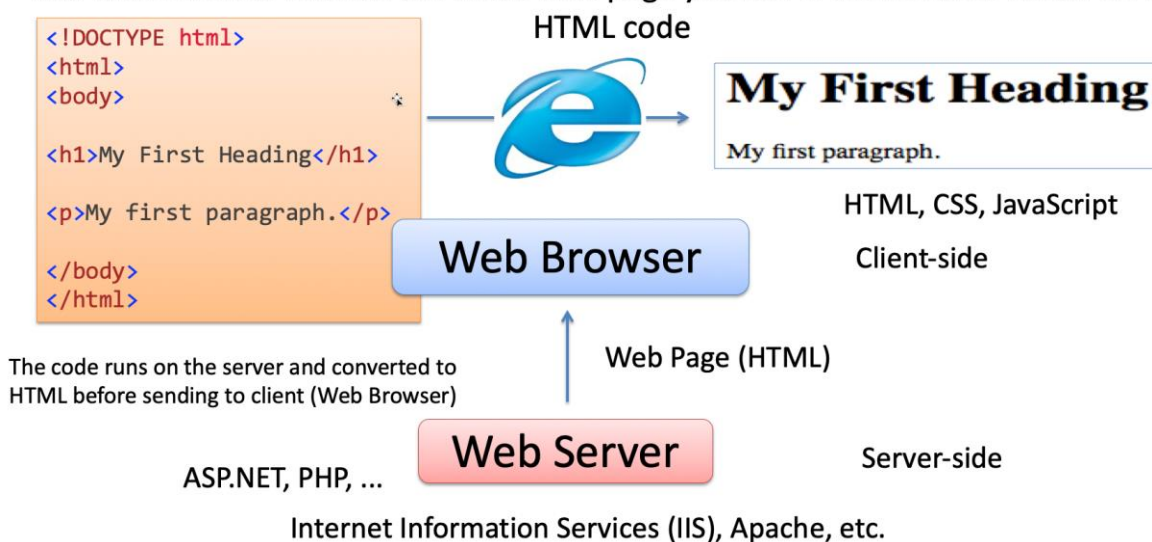


Figure 6-7: The Structure of a Basic Web Page

6.4 Web Browsers

The web browser is the desktop application where we see and interact with the web pages or the web applications.

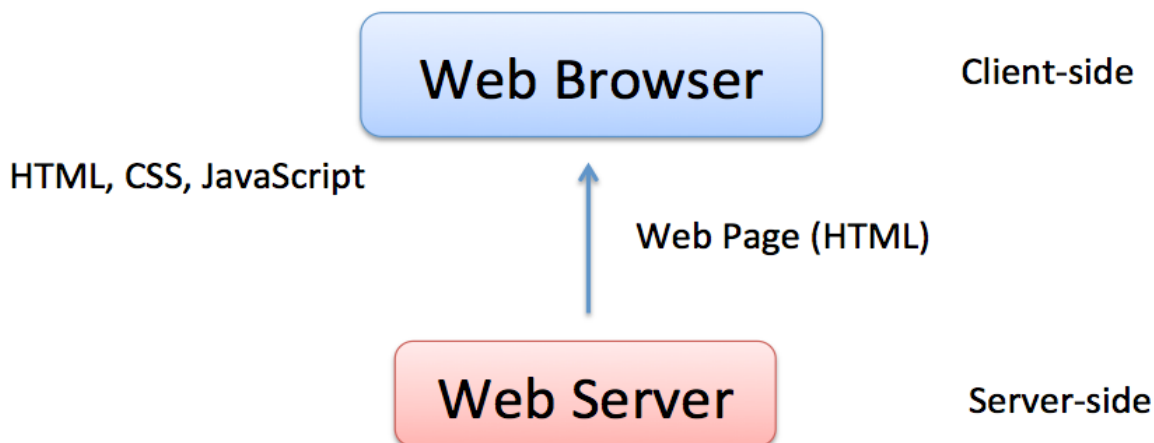


Figure 6-8: Web Browser – Basic Principles

Figure 6-9 shows some of the popular web browsers on the market today.

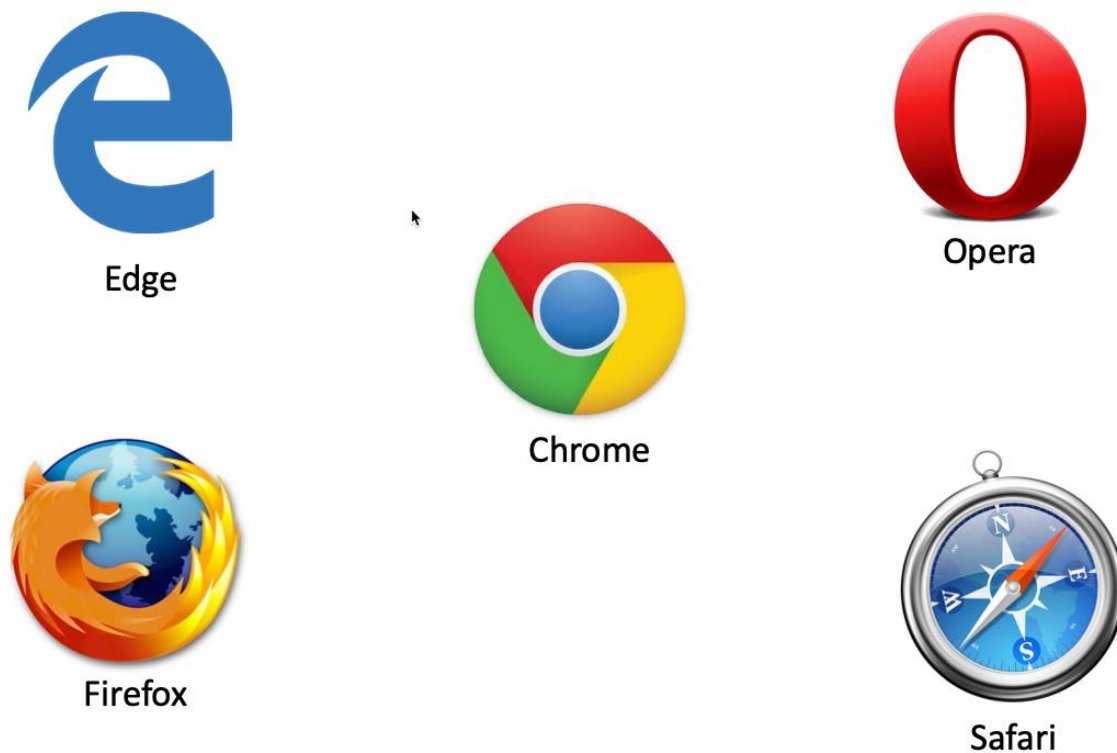


Figure 6-9: Popular Web Browsers

6.5 HTML

HTML, or HyperText Markup Language is the visual appearance of a web site. You could say it is the language spoken by web browsers that makes it possible to see web contents in your web browser. All web browser understands HTML. HTML 5 is the latest. HTML is maintained by W3C - World Wide Web Consortium.

HTML5 is supported in all modern Browsers

WYSIWYG HTML Editors:

- **Adobe Dreamweaver** (Monthly Payment)
- Kompozer (Free)
- Bluegriffon (Free)
- Sparkle (\$80), etc...

WYSIWYG – What You See Is What You Get. You don't need to know HTML syntax - It's just like using MS Word.

HTML Editors (not WYSIWYG)

- **Visual Studio** (ASP.NET)
- **Visual Studio Code**
- CoffeeCup (\$69, Free Trial)
- Coda (\$99)
- NotePad (-> any textbased editor)

Only possible to change the HTML source code and then select “Preview” in order to see how it looks like in a Web Browser. You need to know HTML syntax.

Create a Web Site with Visual Studio:

- Visual Studio is not well suited for creating Static HTML Web Pages.
- Visual Studio has poor WYSIWYG Editing possibilities
- Visual Studio is more optimized for creating Dynamic Web Pages and creating ASP.NET Web Pages in special
- For HTML pages Visual Studio Code may be a better choice.

Video:



HTML: <https://youtu.be/DUEHx7I5a3Y>

6.6 CSS

CSS – Cascading Style Sheets

Styles define **how to display** HTML elements

CSS is used to control the style and layout of multiple web pages all at once

6.7 JavaScript

JavaScript is the de facto client-side programming language. Typically you want to use a JavaScript Framework.

Here are some popular JavaScript Frameworks:

- AngularJS, Angular2 (JavaScript Framework, Google)
- Bootstrap (JavaScript/HTML, CSS Framework), Open-source framework
- JQuery
- TypeScript (Microsoft)

6.8 Server-side Frameworks

Server-side Web Frameworks:

- ASP.NET (Programming Language: C#, IDE: Visual Studio)
- PHP
- Python Django (Programming Language: Python)
- Ruby on Rails (Programming Language: Ruby)
- Node.js (Programming Language: JavaScript)

6.8.1 PHP

PHP is a popular general-purpose scripting language that is especially suited to web development.



PHP Web Page: <https://www.php.net>

PHP is a server-side scripting language for web development. It is used to make dynamic and interactive web pages. PHP is an old and well-known technology, but it is still very popular and easy to learn. PHP is open source (free) and cross-platform. Especially, the combination of PHP and MySQL is a powerful combination used to create rich, dynamic web pages.

PHP is a server-side scripting language for web development. It is used to make dynamic and interactive Web Pages

Old and well-known Technology

Very Popular and easy to learn

Open Source/Free and Cross-platform

PHP + MySQL is a powerful combination

PHP files can contain text, HTML, CSS, JavaScript, and PHP code

PHP code are executed on the server, and the result is returned to the browser as plain HTML

PHP files have extension ".php"

LAMP is popular when it comes to Web Programming. LAMP consists of the following components:

- Linux Operating System
- Apache Web Server (Apache HTTP Server)
- MySQL Database System
- PHP Programming Language d

All of these 4 components are open source, which is one of the reasons for its popularity.

PHP example:

```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Additional Training:



PHP Tutorial: <http://www.w3schools.com/php/>

6.8.2 ASP.NET

ASP.NET is an open-source web framework, created by Microsoft, for building web apps and services using the .NET Framework or the .NET Core. We have both ASP.NET and ASP.NET Core. ASP.NET Core is the new approach built on .NET Core.

6.8.3 Django

Django is based on the Python programming language.

Do you want to learn more about Python?



Python: <https://www.halvorsen.blog/documents/programming/python/>

6.8.4 JavaScript Server-side Frameworks

Typically, you use JavaScript on the client, but several server-side JavaScript frameworks do exist.

Probably, **Node.js** is the most popular server-side JavaScript framework today.

6.9 Web Data Formats

Important data formats on the web are:

- XML
- JSON

Today, JSON has become the dominating standard for data exchange.

6.9.1 XML

XML stands for eXtensible Markup Language.

- XML was designed to store and transport data.
- XML was designed to be both human- and machine-readable.
- XML is often used for distributing data over the Internet.

XML was the dominating format for storing and exchanging data between a browser and a server, but today JSON has taken over.

Additional Resources:



XML Tutorial: <https://www.w3schools.com/xml/>

6.9.2 JSON

JSON: JavaScript **O**bject **N**otation.

JSON is a syntax for storing and exchanging data. When exchanging data between a browser and a server, the data can only be text. JSON is text.

Example:

```
{name: "John", age: 31, city: "New York"};
```

Advantages with JSON:

- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent
- JSON uses JavaScript syntax, but the JSON format is text only.
- Text can be read and used as a data format by any programming language.

Additional Resources:



Introduction to JSON: https://www.w3schools.com/js/js_json_intro.asp

7 HTML

HTML, or HyperText Markup Language is the visual appearance of a web site. You could say it is the language spoken by web browsers that makes it possible to see web contents in your web browser. All web browser understands HTML. HTML 5 is the latest. HTML is maintained by W3C - World Wide Web Consortium.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>
<body>
Content of the document.....
</body>
</html>
```

The history of HTML:

Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1

Figure 7-1: The History of HTML

Video:



HTML: <https://youtu.be/DUEHx7I5a3Y>

7.1 HTML in Visual Studio

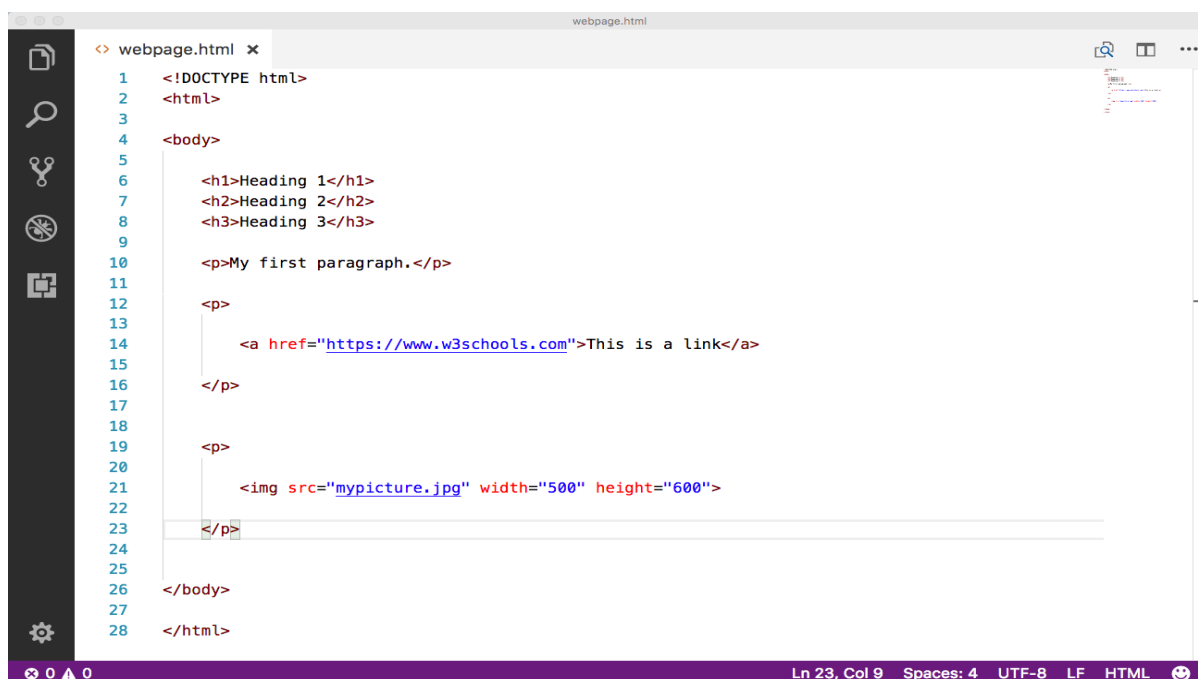
Visual Studio is not well suited for creating Static HTML Web Pages.

Visual Studio has poor WYSIWYG Editing possibilities

We can use Visual Studio because we already use it in our Project – and basic HTML syntax is something you should know about.

Visual Studio is more optimized for creating Dynamic Web Pages and creating ASP.NET Web Pages in special

For pure HTML pages, I recommend that you use **Visual Studio Code** instead



```
webpage.html
1  <!DOCTYPE html>
2  <html>
3
4  <body>
5
6      <h1>Heading 1</h1>
7      <h2>Heading 2</h2>
8      <h3>Heading 3</h3>
9
10     <p>My first paragraph.</p>
11
12     <p>
13
14         <a href="https://www.w3schools.com">This is a link</a>
15
16     </p>
17
18     <p>
19
20         
21
22     </p>
23
24
25
26 </body>
27
28 </html>
```

Ln 23, Col 9 Spaces: 4 UTF-8 LF HTML

Figure 7-2: Visual studio Code

7.2 HTML Tags

HTML consists of HTML tags that is used to format and present contents on the web page. Here, some of the most used HTML tags will be presented.

7.2.1 Title

Title:

```
<title>This is my Title</title>
```

7.2.2 Headers

Headers 1:

```
<h1>This is my Header</h1>
```

Header 2:

```
<h2>This is my Header</h2>
```

Header 3:

```
<h3>This is my Header</h3>
```

7.2.3 Paragraphs

Paragraphs:

```
<p>My first paragraph.</p>
```

7.2.4 Hyperlinks

Hyperlinks:

```
<!DOCTYPE html>
<html>
<body>
<h1>This is a heading</h1>
<a href="http://www.google.com">This is a link to Google</a>
</body>
</html>
```

7.2.5 Images

Images:

```
<!DOCTYPE html>
<html>
<body>
<h1>This is a heading</h1>

</body>
</html>
```

7.2.6 Tables

Tables:

```
<table width="200" border="1">
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
</table>
```

7.2.7 Comments

Comment:

```
<!-- Write your comments here -->
```

7.3 Additional Resources



HTML Tutorial: <https://www.w3schools.com/html/>

8 CSS

CSS is a stylesheet language that describes the presentation of an HTML page.

Use HTML to define the content of web pages, CSS is used to specify the layout of web pages, while JavaScript is used to program the behavior of web pages.

CSS – Cascading Style Sheets

Styles define **how to display** HTML elements

CSS is used to control the style and layout of multiple web pages all at once

Basic Example:

```
body {
  background-color: #d0e4fe;
}
h1 {
  color: orange;
  text-align: center;
}
p {
  font-family: "Times New Roman";
  font-size: 20px;
}
```

There are three ways of inserting a style sheet:

- **External style sheet (Recommended!!)**
 - An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing just one file.
 - An external style sheet can be written in any text editor. The file should not contain any html tags.
 - The style sheet file must be saved with a .css extension
- **Internal style sheet**
 - An internal style sheet should be used when a single document has a unique style.
 - You define internal styles in the head section of an HTML page, inside the <style> tag

- **Inline style**
 - An inline style loses many of the advantages of a style sheet (by mixing content with presentation). Use this method sparingly!

8.1 External Style Sheets

stylesheet.css:

```
body {
  background-color: #d0e4fe;
}
h1 {
  color: orange;
  text-align: center;
}
p {
  font-family: "Times New Roman";
  font-size: 20px;
}
```

myfile.html:

```
...
<head
...
  <link rel="stylesheet" type="text/css" href="stylesheet.css" />
  ..
</head>
...
```

8.2 Bootstrap

JavaScript/HTML, CSS Framework

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.

Home Page:

<https://getbootstrap.com>

Bootstrap is a popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web

Bootstrap is a free and open-source front-end web framework for designing websites and web applications.

It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

Unlike many web frameworks, it concerns itself with client-side/front-end development only.

Additional Training:



Bootstrap Tutorial: <https://www.w3schools.com/bootstrap4/>

8.3 Font Awesome

Font Awesome 5 has a PRO edition with 7020 icons, and a FREE edition with 1535 icons. The FREE edition is a good start and has more than enough icons for most people.

To use the Free Font Awesome 5 icons, you can choose to download the Font Awesome library, or you can sign up for an account at Font Awesome and get a code (called KIT CODE) to use when you add Font Awesome to your web page.



Font Awesome Home Page: <https://fontawesome.com>

Additional Training:



Introduction to Font Awesome:

https://www.w3schools.com/icons/fontawesome5_intro.asp

8.4 Additional Resources



CSS Tutorial: <https://www.w3schools.com/css/>

9 JavaScript

JavaScript is the de facto client-side programming language. Typically, you want to use a JavaScript Framework.

Here are some popular JavaScript Frameworks:

- AngularJS, Angular2 (JavaScript Framework, Google)
- Bootstrap (JavaScript/HTML, CSS Framework), Open source framework
- JQuery
- TypeScript (Microsoft)

JavaScript is one of **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

JavaScript is the programming language of the Web.

All modern HTML pages are using JavaScript.

JavaScript is the default scripting language in all modern browsers, and in HTML5.

JavaScript is probably the most popular programming language in the world.

It is the language for HTML, for the Web, for computers, servers, laptops, tablets, smart phones, and more.

JavaScript can Change HTML Elements! – which makes it very powerful!

Note that JavaScript and Java are different languages, both in concept and design.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>
<p>JavaScript can change the content of an HTML element:</p>
<button type="button" onclick="myFunction()">Click Me!</button>
<p id="demo">This is a demonstration.</p>

<script>

function myFunction() {
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
}

</script>

</body>
</html>
```

This gives the following results in your web browser:



9.1.1 Additional Resources

W3schools.com: <https://www.w3schools.com/js/>

9.2 jQuery

jQuery is a JavaScript Library.

Additional Training:



jQuery Tutorial: <https://www.w3schools.com/jquery/>

9.3 AngularJS

JavaScript Framework developed by Google that has become very popular today.



TypeScript Home Page: <https://angularjs.org>

9.4 TypeScript (Microsoft)

TypeScript is a free and open-source programming language developed and maintained by Microsoft.

It is a superset of JavaScript that compiles to JavaScript



TypeScript Home Page: <http://www.typescriptlang.org>

10 Server-side Frameworks

We have many different so-called server-side frameworks for creating dynamic web pages:

- PHP
- Django
- ASP.NET
- Etc.

10.1 PHP

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.

10.2 Django

Django is based on the Python programming language.

Do you want to learn more about Python? Here you find lots of Python resources:

<https://www.halvorsen.blog/documents/programming/python/>

10.3 ASP.NET

ASP.NET is an open-source web framework, created by Microsoft, for building web apps and services using the .NET Framework or the .NET Core. We have both ASP.NET and ASP.NET Core. ASP.NET Core is the new approach built on .NET Core.

ASP.NET Core will be the main topic in this document. In this textbook we will go through ASP.NET Core in detail. In Part 4: ASP.NET Core we start creating ASP.NET Core Web Applications.

Part 4 : ASP.NET Core

As this point you should be familiar with Visual Studio, the C# Programming Language, and basic concepts regarding Web Programming in general. This part gives an overview of ASP.NET Core.

11 Introduction to ASP.NET Core

ASP.NET Core is based on the .NET Core Framework (not the ordinary .NET Framework).

We have:

- ASP.NET Core MVC (Model-View Controller). If you are familiar with the MVC approach, this could be your choice.
- ASP.NET Core with Razor Pages. This is the latest and recommended way. This has become the "default" approach for ASP.NET today. It mixes the best from all the others combined with PHP like syntax (PHP is probably the most popular Web Framework today)

This textbook will focus on this latest and newest approach, namely **ASP.NET Core with Razor Pages**.

Figure 11-1 shows a ASP.NET Core project in Visual Studio.

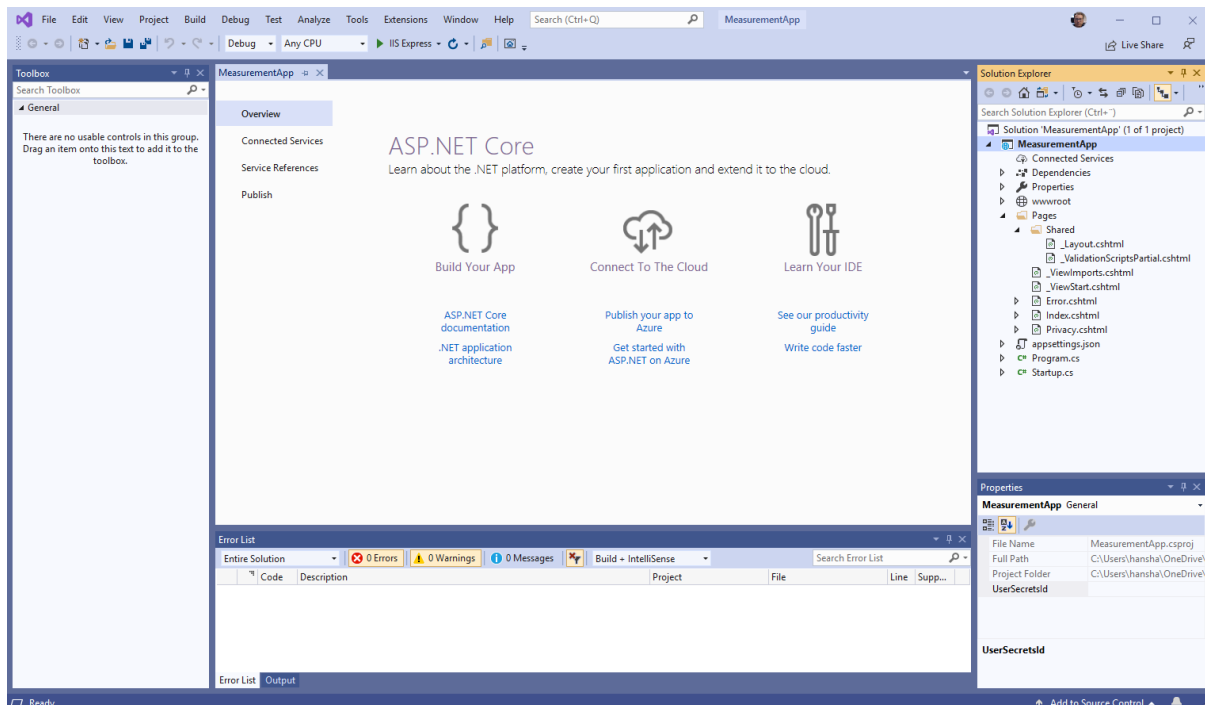


Figure 11-1: ASP.NET Core Visual Studio Project

Important Folders and Files:

- **appSettings.json** – This file contains configuration data, such as connection strings.
- **Program.cs** – This file contains the entry point for the program.
- **Startup.cs** - This file contains code that configures app behavior.
- **wwwroot** folder - Contains static files, such as Images, HTML files, JavaScript files, and CSS files.
- **Pages** folder – Here you are supposed to put your ASP.NET (".cshtml") web pages

In addition, it is standard to have a folder called “**Models**”. This folder contains C# classes that takes care of the data. The data can, e.g., be a database or a file, e.g., a JSON file.

In addition, we have what we call Supporting files. Supporting files have names that begin with an underscore (_).

- **_Layout.cshtml** file configures UI elements common to all pages. You can use this file to set up the navigation menu at the top of the page

11.1 Resources

Here are some important ASP.NET Core resources:

- ASP.NET Core fundamentals (Microsoft): <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/>
- Introduction to Razor Pages in ASP.NET Core (Microsoft): <https://docs.microsoft.com/en-us/aspnet/core/razor-pages>
- Tutorial: Create a Razor Pages web app with ASP.NET Core (Microsoft): <https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/>
- Razor syntax reference for ASP.NET Core (Microsoft): <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor>

11.2 Hello World Application

Let us start by creating the “compulsory” “Hello World” application.



ASP.NET Core Hello World: <https://youtu.be/lcQsWYgQXK4>

We start by creating a New Project in Visual Studio, see Figure 11-2.

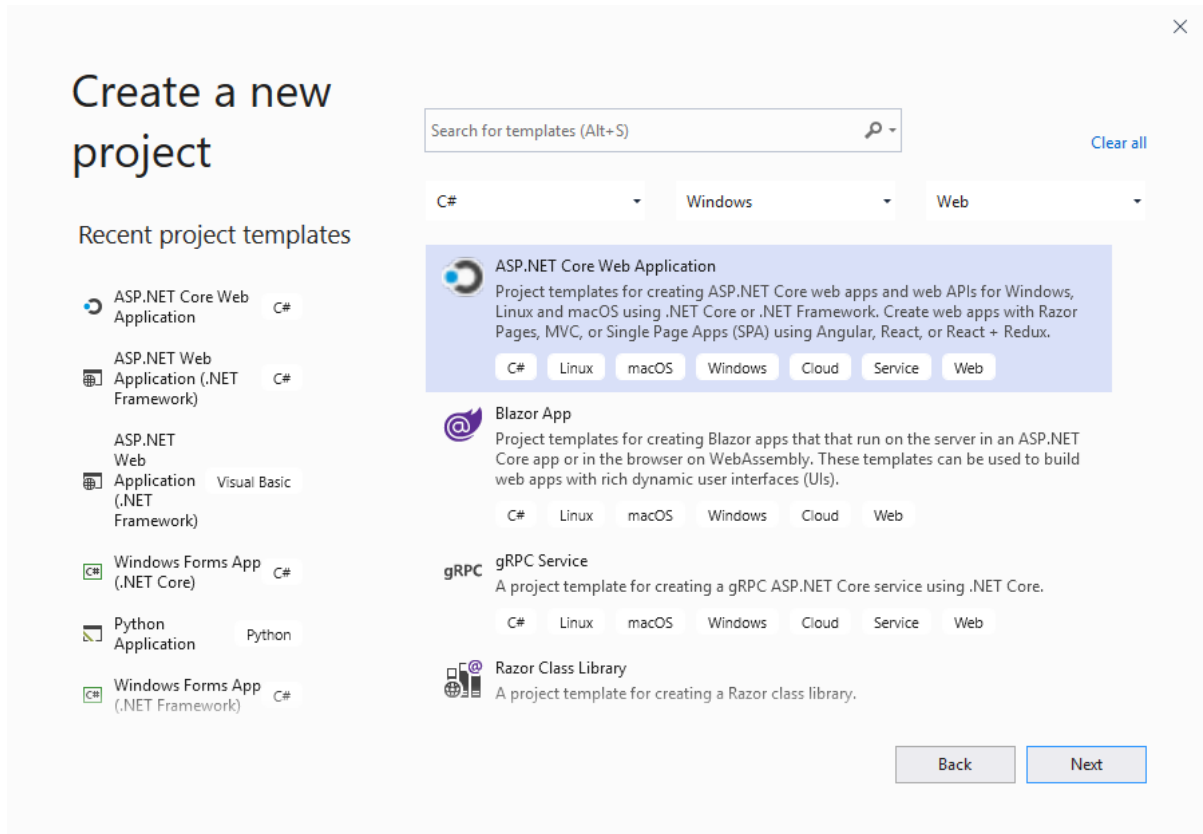


Figure 11-2: Create a New ASP.NET Core Web Application

Configure your project (see Figure 11-3):

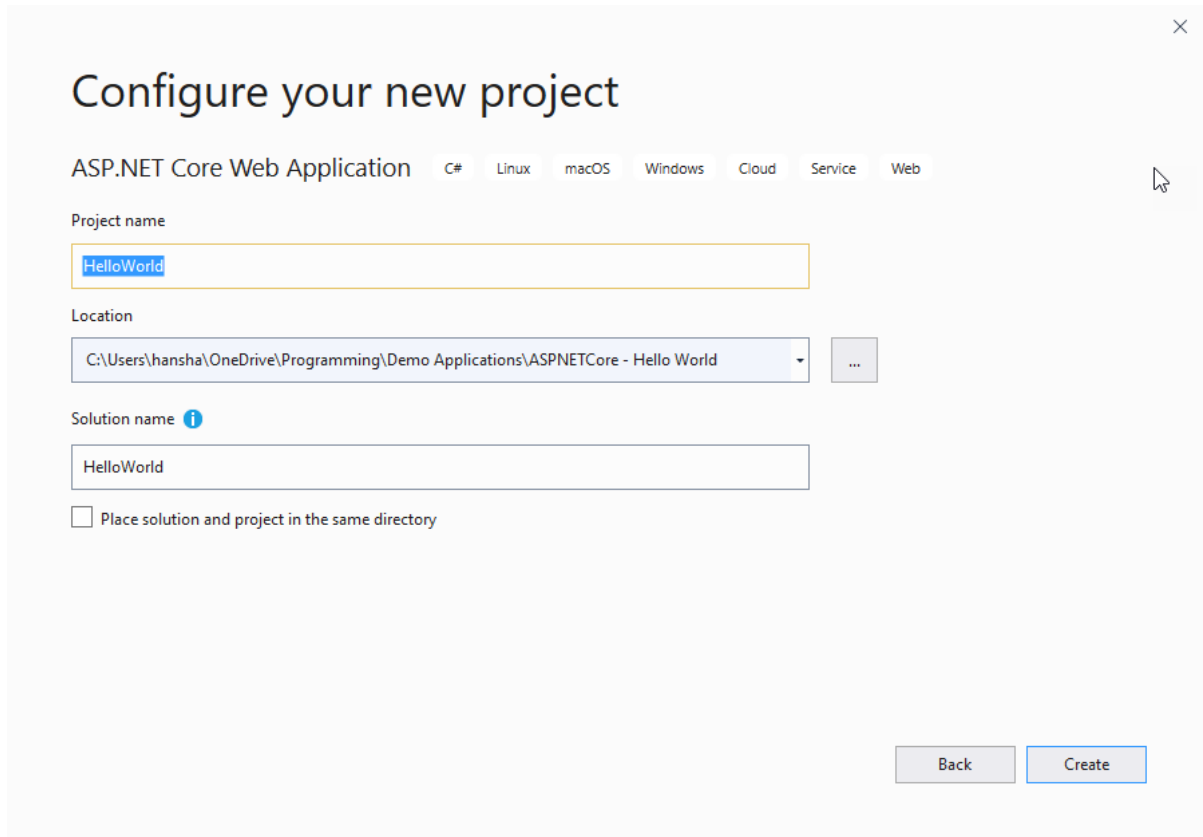


Figure 11-3: Configure the Project

Select the “Web Application” template (Figure 11-4):

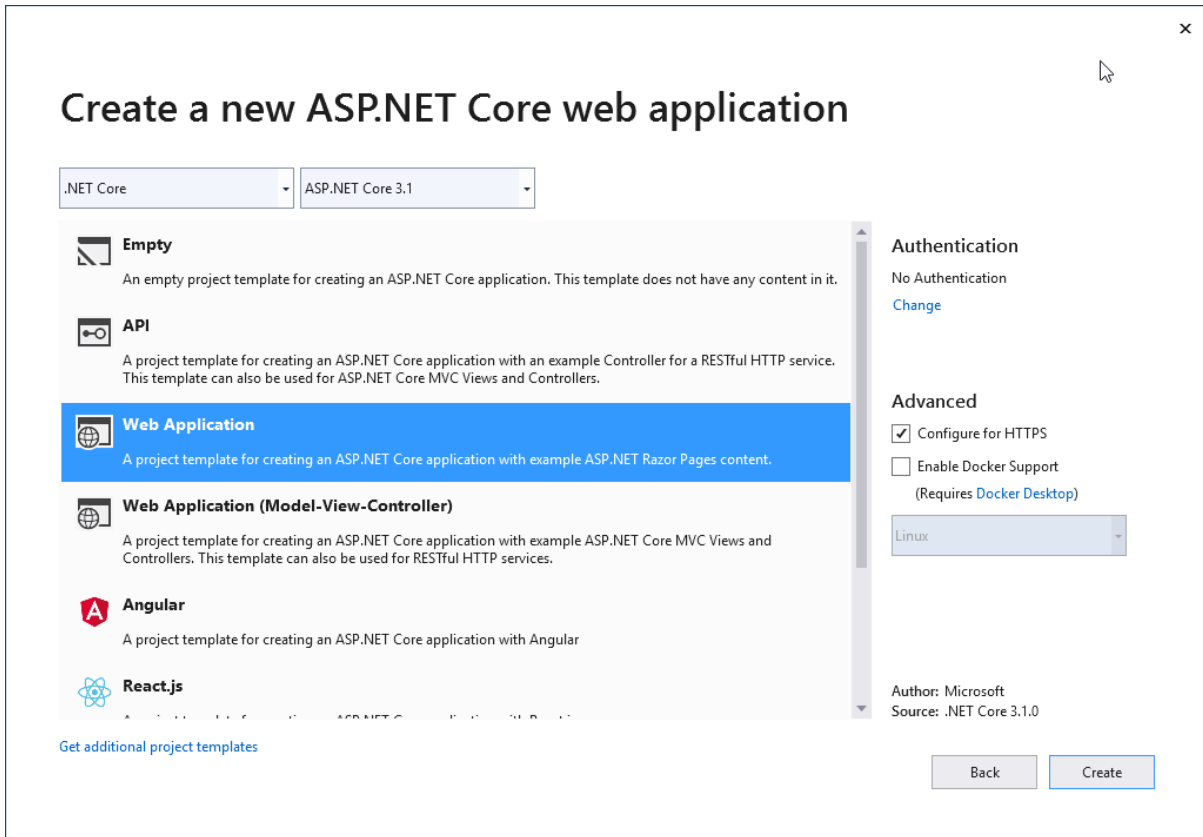


Figure 11-4: Web Application Template

After clicking “Create”, we get the following (Figure 11-5):

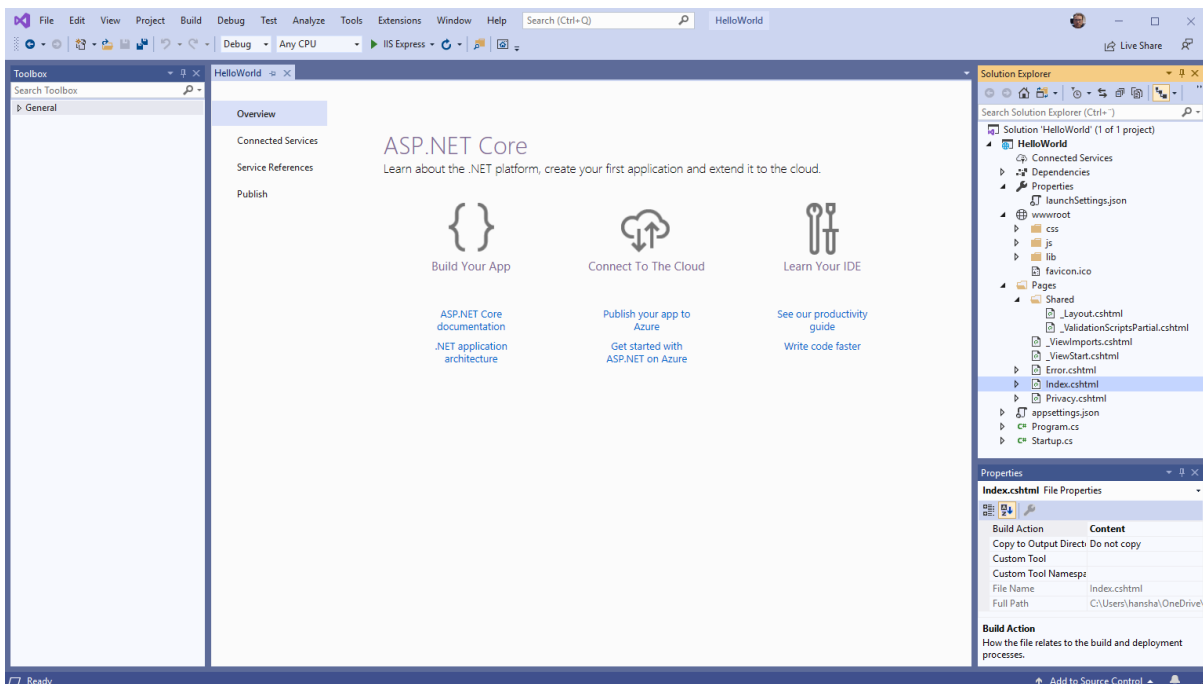


Figure 11-5: Your Project in Visual Studio

We hit F5 in order to run our web application (Figure 11-6):

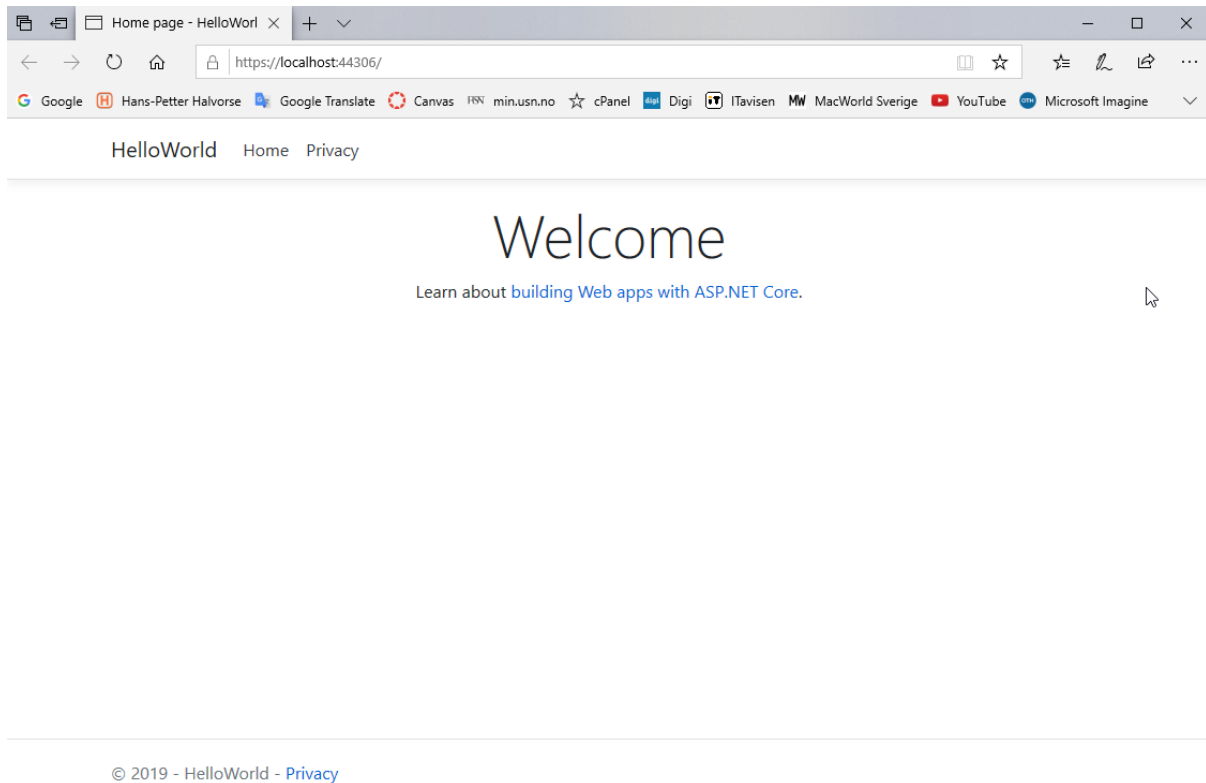


Figure 11-6: Make sure it runs in your Web Browser

Index.cshtml (Figure 11-7):

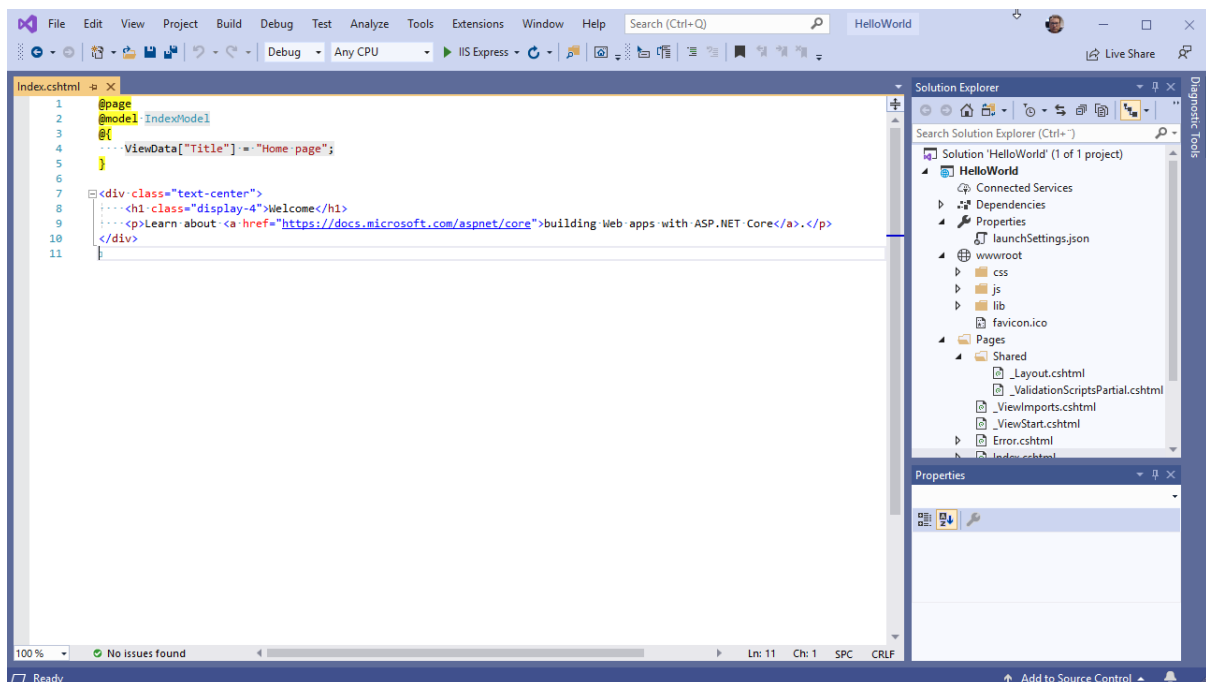


Figure 11-7: Index.cshtml

Code:

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}
<div class="text-center">
    <h1 class="display-4">Hello World</h1>
</div>
```

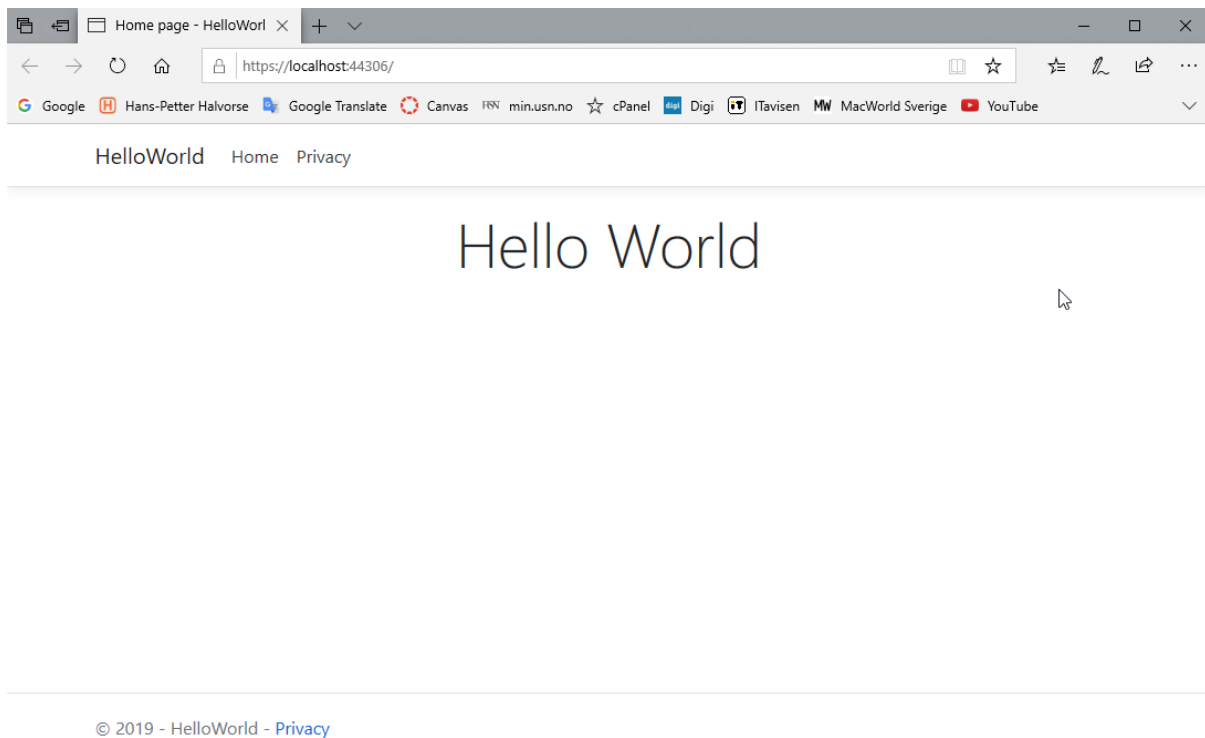
Run the application (Figure 11-8):

Figure 11-8: Final Hello World Application

11.3 ASP.NET Core with Razor

Razor is a markup syntax for embedding server-based code into webpages. The Razor syntax consists of Razor markup, C#, and HTML. Files containing Razor generally have a .cshtml file extension.

Video:

ASP.NET Core – Introduction: <https://youtu.be/zkOtiBcwo8s>

Here are some useful resources:



Introduction to Razor Pages in ASP.NET Core:

<https://docs.microsoft.com/en-us/aspnet/core/razor-pages/>



Tutorial: Create a Razor Pages web app with ASP.NET Core:

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/>



Razor syntax reference for ASP.NET Core:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor>

Razor supports C# and uses the @ symbol to transition from HTML to C#. Razor evaluates C# expressions and renders them in the HTML output.

Assume the following Razor code in the .cshtml file:

```
<p>@DateTime.Now</p>
```

This outputs the current date and time in the browser window.

11.3.1 Basic Examples

Razor supports C# and uses the @ symbol to transition from HTML to C#. Razor evaluates C# expressions and renders them in the HTML output.

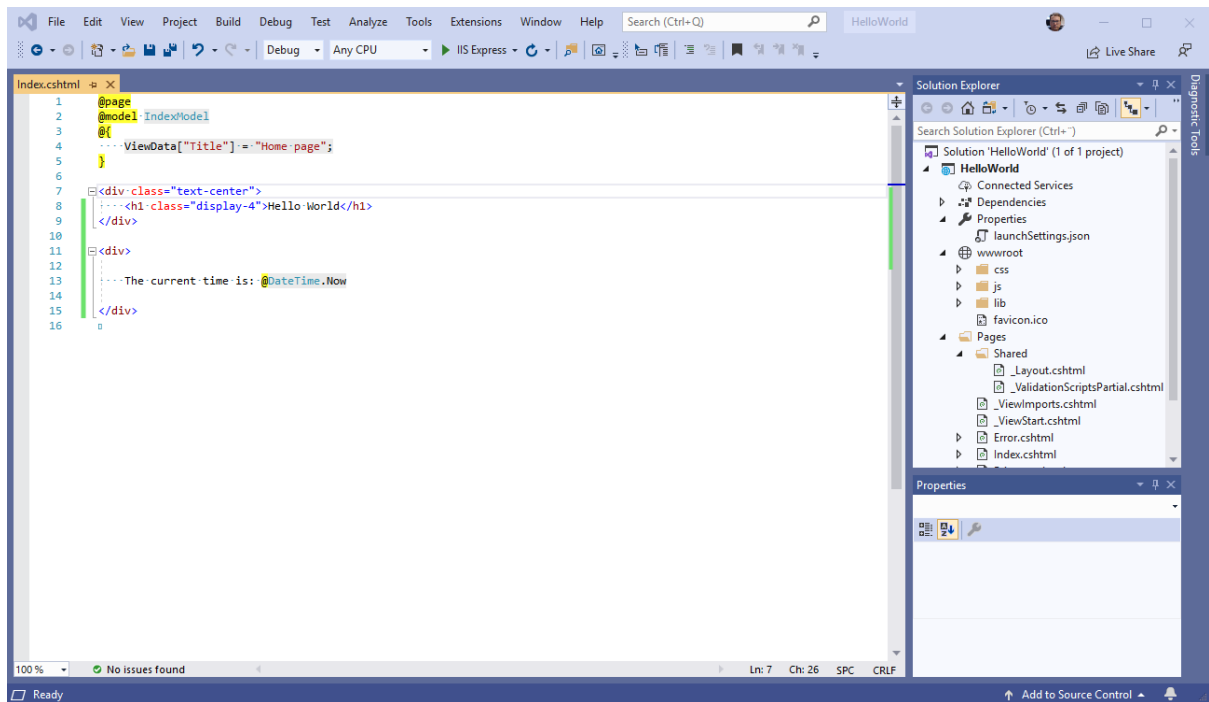


Figure 11-9: Razor Page Example

Code:

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}
<div class="text-center">
    <h1 class="display-4">Hello World</h1>
</div>
<div>
    The current time is: @DateTime.Now
</div>
```

Run the example in your web browser:

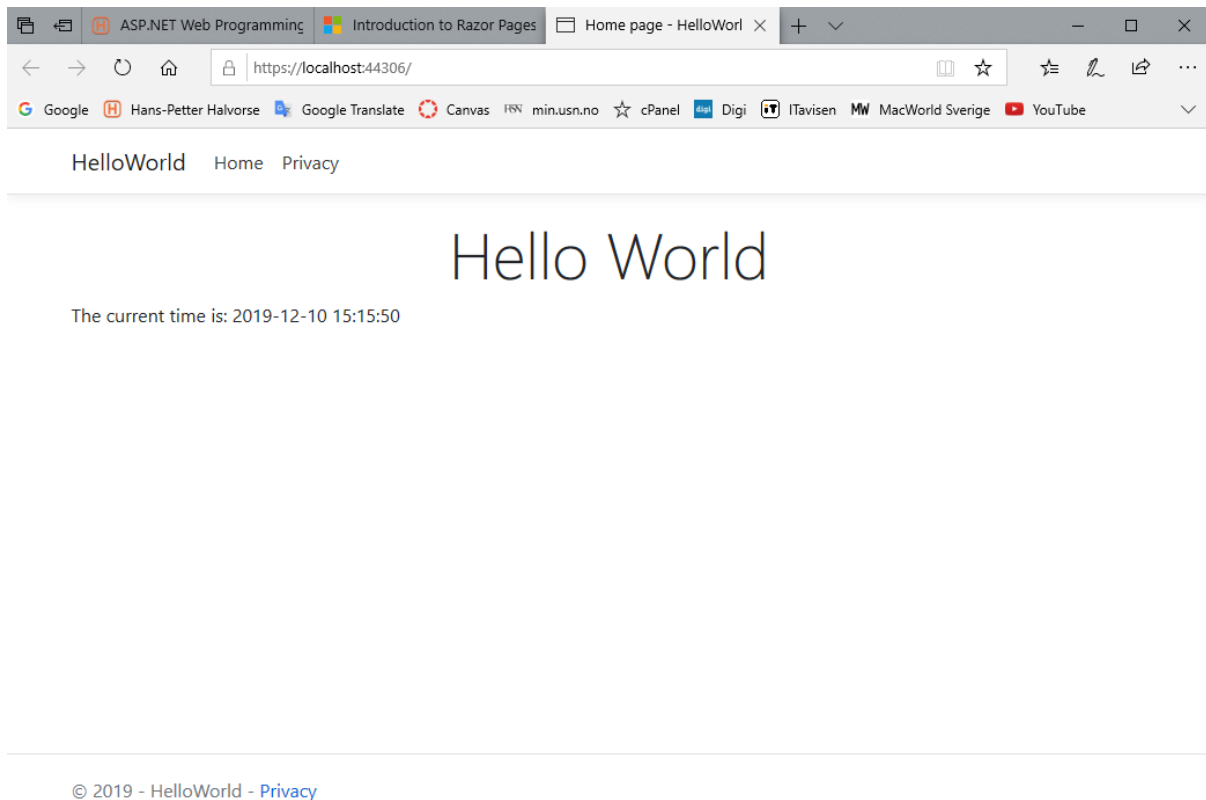


Figure 11-10: Razor Page running in the Web Browser

Let's add a message in addition to the datetime.

Index.cshtml.cs:

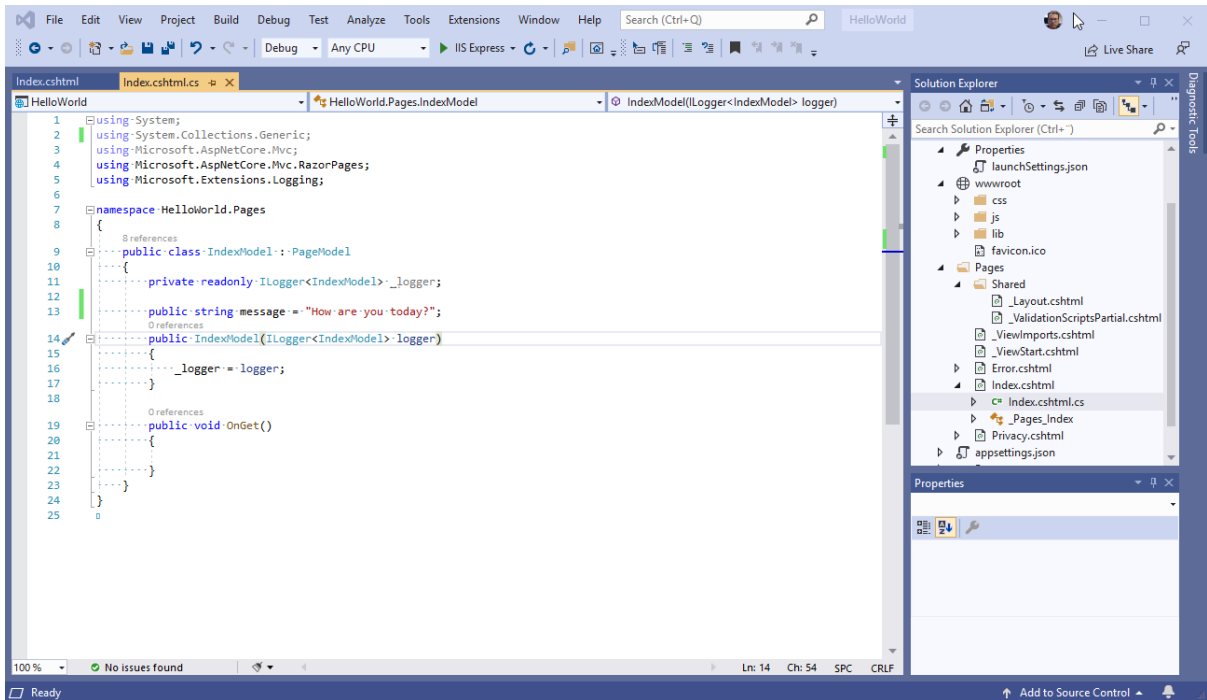


Figure 11-11: Index.cshtml.cs File

Code for “Index.cshtml.cs”:

```
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
namespace HelloWorld.Pages
{
    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;
        public string message = "How are you today?";
        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }
        public void OnGet()
        {
        }
    }
}
```

Let's make the necessary changes in the Index.cshtml file.

Figure 11-12 shows the modified Index.cshtml file.

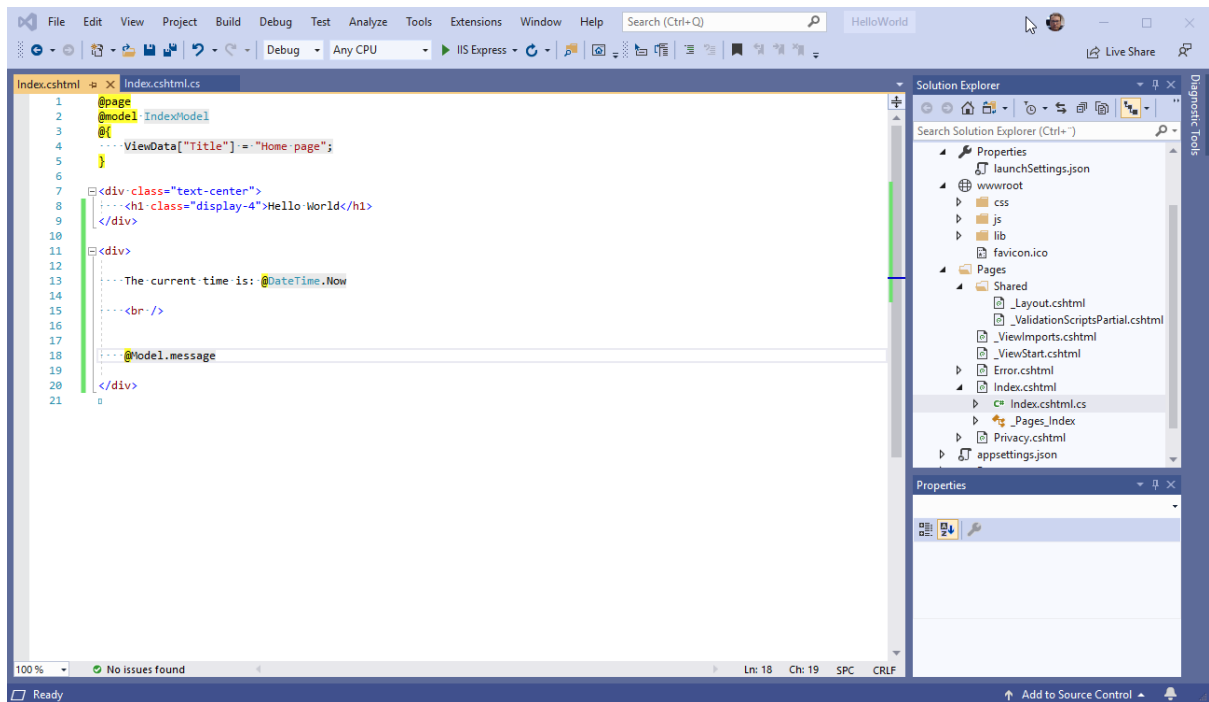


Figure 11-12: Index.cshtml

Code for “Index.cshtml”:

```
@page
@model IndexModel
@
{
    ViewData["Title"] = "Home page";
}
<div class="text-center">
    <h1 class="display-4">Hello World</h1>
</div>
<div>
    The current time is: @DateTime.Now
    <br />
    @Model.message
</div>
```

Figure 11-13 shows the results.

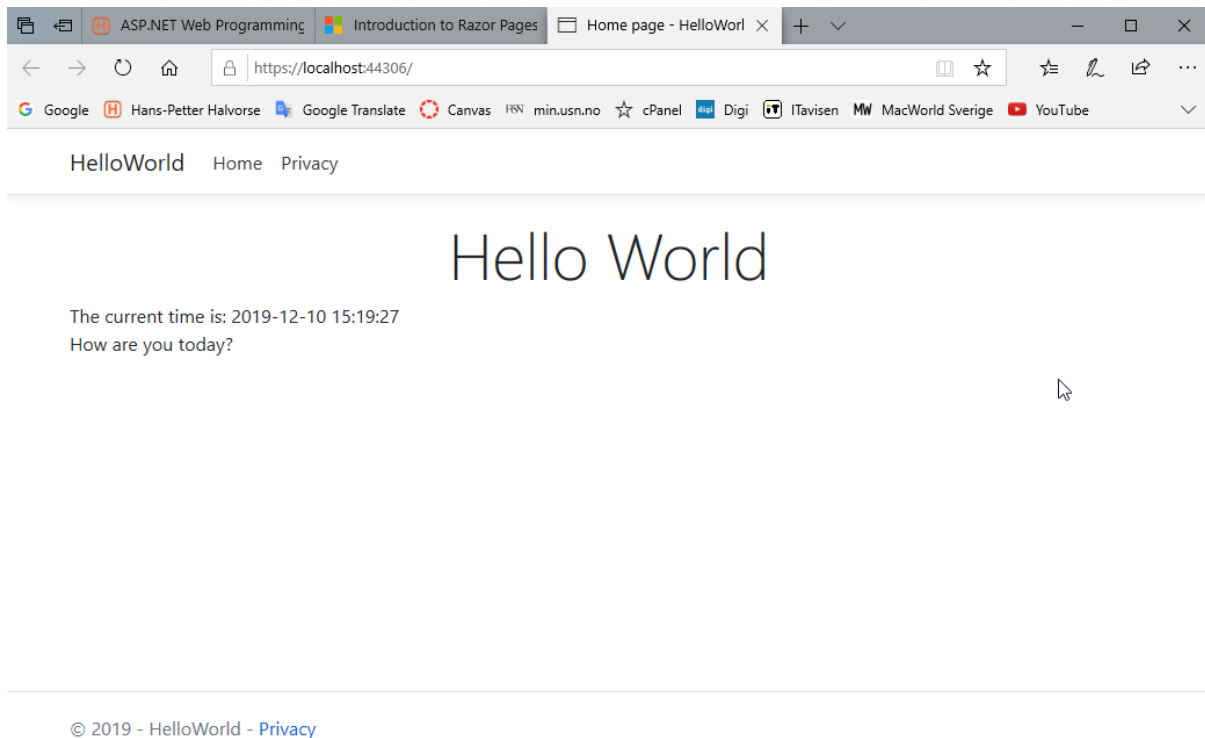


Figure 11-13: Hello World

11.4 Query String Data

Typically, we need to send data between different web pages or between the same web page. In this case we typically send data or information through the query string.

In order to manage that, we can use **`Request.Query["name"]`**.

Example:

```
public void OnGet()
{
    bookId = Convert.ToInt16(Request.Query["bookid"]);
    Book book = new Book();
    connectionString =
        _configuration.GetConnectionString("ConnectionString");
    bookdb = book.GetBookData(connectionString, bookId);
}
```

In this example the URL in the web browser could look something like this

<https://BookApp/EditBook?bookid=4>

If we want to send more than one variable, we use &, for example:

<https://BookApp/EditBook?bookid=4&booktitle=Python&author=hamsun>

11.5 Form Data

Typically, the user enter data into different fields on a web page. In order to send these data to the server for some kind of processing (e.g., store the data in a database) we use Form Data.

In order to manage that, we can use **Request.Form["name"]**.

Example

```
public void OnPost()
{
    Book book = new Book();

    book.BookId = Convert.ToInt16(Request.Form["bookId"]);
    book.Title = Request.Form["bookTitle"];
    book.Isbn = Request.Form["bookIsbn"];
    book.PublisherName = Request.Form["bookPublisher"];
    book.AuthorName = Request.Form["bookAuthor"];
    book.CategoryName = Request.Form["bookCategory"];

    connectionString =
        _configuration.GetConnectionString("ConnectionString");

    book.EditBook(connectionString, book);

    Response.Redirect("./Books");
}
```

Figure 11-14 shows a typical web form.

BookApp Home Books

New Book

Title:

ISBN:

Publisher:

Author:

Category:

© 2019 - BookApp - Privacy

Figure 11-14: Web Form Data

HTML Code:

```
<form name="bookForm" id="bookForm" method="post">

    Title:
    <br />
    <input name="bookTitle" type="text" class="form-control
    input-lg" autofocus required />
    <br />

    ISBN:
    <br />
    <input name="bookIsbn" type="text" class="form-control
    input-lg" required />
    <br />

    Publisher:
    <br />
    <input name="bookPublisher" type="text" class="form-control
    input-lg" required />
    <br />

    Author:
    <br />
    <input name="bookAuthor" type="text" class="form-control
    input-lg" required />
    <br />

    Category:
    <br />
    <input name="bookCategory" type="text" class="form-control
    input-lg" required />
```

```
<br />  
  
<input id="saveButton" type="submit" value="Save" class="btn  
btn-info" />  
  
</form>
```

12 ASP.NET Core Fundamentals

Figure 12-1 shows a typical Solution Explorer in a ASP.NET Core Web Application.

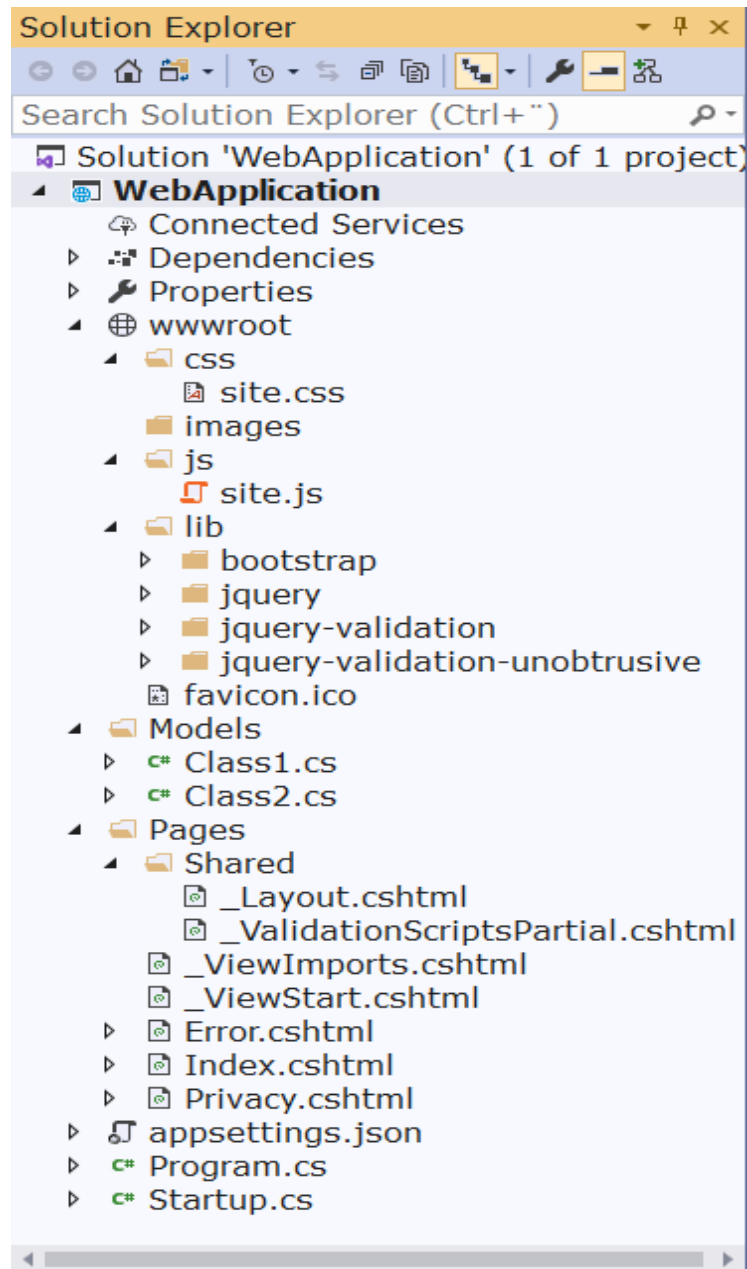


Figure 12-1: Solution Explorer ASP.NET Core Web Application

You have the following important folders:

- `wwwroot`
- `Models`
- `Pages`

In the **Models** folder you suppose to put your C# Classes.

In the **Pages** folder you should put your web pages.

In the **wwwroot** you should put files like CSS files (in the **css** folder), JavaScript files (in your **js** folder), Images in your **images** folder. Different libraries like Bootstrap, JQuery, etc. should be put in the **lib** folder.

In addition, you have the following important files:

- `appsettings.json`
- `Program.cs`
- `Startup.cs`
- `_Layout.cshtml`

They will be explained below.

12.1 Startup Class

Initial code for your application.

12.2 Web root

The web root is by default the “**wwwroot**” folder.

The web root is the base path to public, non-code, static resource files, such as:

- Stylesheets (`.css`) – Here you should put your CSS style sheets
- JavaScript (`.js`) – Here you should put your JavaScript code files
- Images (`.png`, `.jpg`, etc.) – This is the folder where you should put all your images

Static files are only served by default from the web root directory (and sub-directories).

In Razor (`.cshtml`) files, the tilde-slash (`~/`) points to the web root. A path beginning with `~/` is referred to as a virtual path.

12.3 appsettings.json

This file contains configuration data, such as connection strings.

The default “appsettings.json” looks like this:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Typically, you want to put your connection string here:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "ConnectionString": "DATA SOURCE=xxx;UID=xxx;PWD=xxx;DATABASE=xxx"
  }
}
```

Just replace “xxx” with the information for your database.

```
"ConnectionStrings": {
  "ConnectionString": "DATA SOURCE=xxx;UID=xxx;PWD=xxx;DATABASE=xxx"
}
```

Inside “ConnectionStrings” you can have one or more connection strings, let say you have a development database, a test database and a customer database. This makes it easy to switch between different connection strings.

```
"ConnectionStrings": {
  "DevelopmentDB": "DATA SOURCE=xxx;UID=xxx;PWD=xxx;DATABASE=xxx"
  "TestDB": "DATA SOURCE=xxx;UID=xxx;PWD=xxx;DATABASE=xxx"
}
```

12.4 Shared Pages

The shared pages have an underscore in their names, e.g., `_Layout.cshtml`.

Folder: `./Pages/Shared`

12.4.1 Layout

The default layout file in ASP.NET Core is “`_Layout.cshtml`”. You can modify this file so it fits the ways you want to present your files. This layout will by default be added to all your web pages (`.cshtml` files).

The default `_Layout.cshtml` looks like this:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - WebApplication</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom
    box-shadow mb-3">
      <div class="container">
        <a class="navbar-brand" asp-area="" asp-page="/Index">WebApplication</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
        target=".navbar-collapse" aria-controls="navbarSupportedContent"
        aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
          <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-page="/Index">Home</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark" asp-area="" asp-
page="/Privacy">Privacy</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2019 - WebApplication - <a asp-area="" asp-page="/Privacy">Privacy</a>
    </div>
  </footer>

  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>

  @RenderSection("Scripts", required: false)
</body>
</html>
```

A “clean” (removing “everything”, just leave the minimum) `_Layout.cshtml` may look like this

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - WebApplication</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>

  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody ()
    </main>
  </div>

  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>

  @RenderSection("Scripts", required: false)
</body>
</html>
```

If you don’t want to use the `_Layout.cshtml` on a specific file, set the following on top of your `.cshtml` web page:

```
@{
Layout = "";
}
```

You can also have multiple Layout files. If you want to use another Layout file than the default for a specific web page (`.cshtml` file), you can do something like this:

```
@{
Layout = "~/Pages/Shared/_Layout2.cshtml";
}
```

12.5 Models

In the **Models** folder you suppose to put your C# Classes. You don’t need to name this folder `Models`, but that is the recommended name. In that way it will be easier to understand and modify programs made by others.

12.6 Razor Pages

Razor pages should be placed in the default folder called “**Pages**”. Typically, your start page should be named “Index.cshtml”.

An ASP.NET Razor page has the “.cshtml” (e.g., “Index.cshtml”) file extension. This file contains a mix of HTML and Razor syntax. The Razor syntax is actually C# code mixed together with the HTML code.

The Razor parts of the file are executed on the web server before the page is sent to the client (your web browser).

The Razor page may also have a C# code file linked to it, this file has the extension “.cshtml.cs” (e.g., “Index.cshtml.cs”). The “.cshtml.cs” file is called the Page Model.

In Razor with Page Model each Razor page is a pair of files:

- A “.cshtml” file that contains HTML markup with C# code using Razor syntax.
- A “.cshtml.cs” (“code behind” or “Page model” file) file that contains C# code that handles page events.

The default Razor page is Index.cshtml. But this can of course be changed if you want to.

When we create a new Razor page (e.g., “Test”) it will look like something like this:

Test.cshtml

```
@page
@model TestModel
@{
    ViewData["Title"] = "Test Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>This is a Razor Page</p>
</div>
```

All Razor pages needs to start with the **@page** directive.

The line “@model TestModel” points to the **Page Model** file.

While the following are Razor syntax:

```
@{
    ..
}
```

The **Page Model** file (Test.cshtml.cs) or “code behind” file will look something like this:

```
using System.Collections.Generic;
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace DemoApp.Pages
{
    public class TestModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;

        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }

        public void OnGet()
        {
        }
    }
}
```

12.6.1 Sending data from the Page Model to the Razor File

Typically we need to send data between the Page Model (.cshtml.cs) and the Razor File (.cshtml). Below you see a basic example.

Page Model File (.cshtml.cs):

```
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;

namespace DemoApp.Pages
{
    public class TestModel : PageModel
    {
        public string name;

        private readonly ILogger<IndexModel> _logger;

        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }

        public void OnGet()
        {
            name = "Hans-Petter Halvorsen";
        }
    }
}
```

In this example we have declared a public variable. Then later we want to display the value in our web page (Razor Page (.cshtml)).

You typically put code that needs to run before the web page is sent to the client in the **OnGet()** method.

Razor Page (.cshtml):

```
@page
@model TestModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>

    <p>This is a Razor Page</p>

    <p>My Name is @Model.name</p>
</div>
```

As you see we need to use **@Model.xxx** where xxx is the name of a public variable in the Page Model file.

The results become as shown in Figure 12-2.

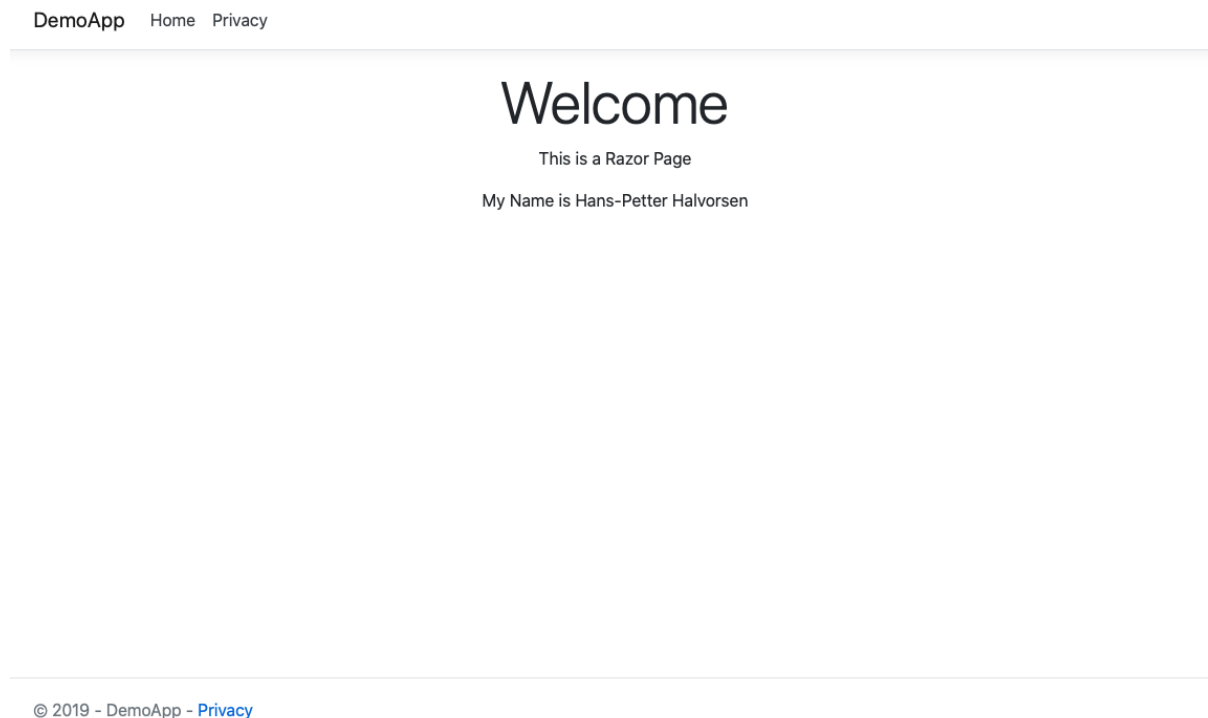


Figure 12-2: Razor Web Page

12.7 Additional Resources



ASP.NET Core fundamentals: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/>

13 Razor

Razor is a markup syntax for embedding server-based code into webpages. The Razor syntax consists of Razor markup, C#, and HTML. Files containing Razor generally have a .cshtml file extension.

The default Razor language is HTML. Rendering HTML from Razor markup is no different than rendering HTML from an HTML file. HTML markup in .cshtml Razor files is rendered by the server unchanged.



ASP.NET Documentation from Microsoft: <https://docs.microsoft.com/aspnet/>



Introduction to Razor Pages in ASP.NET Core:
<https://docs.microsoft.com/aspnet/core/razor-pages/>

13.1 Razor Syntax

Razor supports C# and uses the @ symbol to transition from HTML to C#. Razor evaluates C# expressions and renders them in the HTML output.

All code blocks must appear within @{ ... } brackets.

Example:

```
@{  
    var number = 1;  
}
```

The value of variables is rendered into HTML by prefixing them with the @ sign.

Example:

```
The number is @number
```

We can use standard C# features and built-in Classes and Methods.

Example:


```
The time is @DateTime.Now
```

A foreach loop is very handy for looping through data.

Example:

```
@{
    var numbers = Enumerable.Range(1, 10); //Get numbers from 1 - 10
    foreach(var number in numbers)
    {
        @number
    }
}
```

Or this alternative way:

```
@{
    var numbers = Enumerable.Range(1, 10);
}

@foreach(var number in numbers){
    @number
}
```

Comments can be used in different ways, either // for a single line or /* */ for multiple lines.

Example:

```
// This is a comment
...

/* ... */

/*
Multiple Lines
...
...
*/
```

13.2 Model

Using **Model** inside a foreach:

Example:

```
@foreach (var measurement in Model.measurementParameterList)
{
    <tr>
        <td> @measurement.MeasurementId</td>
        <td> @measurement.MeasurementName</td>
        <td> @measurement.MeasurementUnit</td>
    </tr>
```

}

Part 5 Database Communication

All modern applications typically communicate with a Database System. Here we will introduce Database Systems in general and especially Microsoft SQL Server. Then we will cover Database Communication with C# and ASP.NET Core in combination with SQL Server from Microsoft.

14 Database Systems

A Database is a structured way to store lots of information. The information is stored in different tables.

Some of the most popular Database Systems today are:

- SQL Server
- MySQL
- MariaDB
- Etc.

ER Diagram (Entity-Relationship Diagram) is used for design and modeling of databases. It specifies tables and relationship between them (Primary Keys and Foreign Keys), see Figure 14-1.

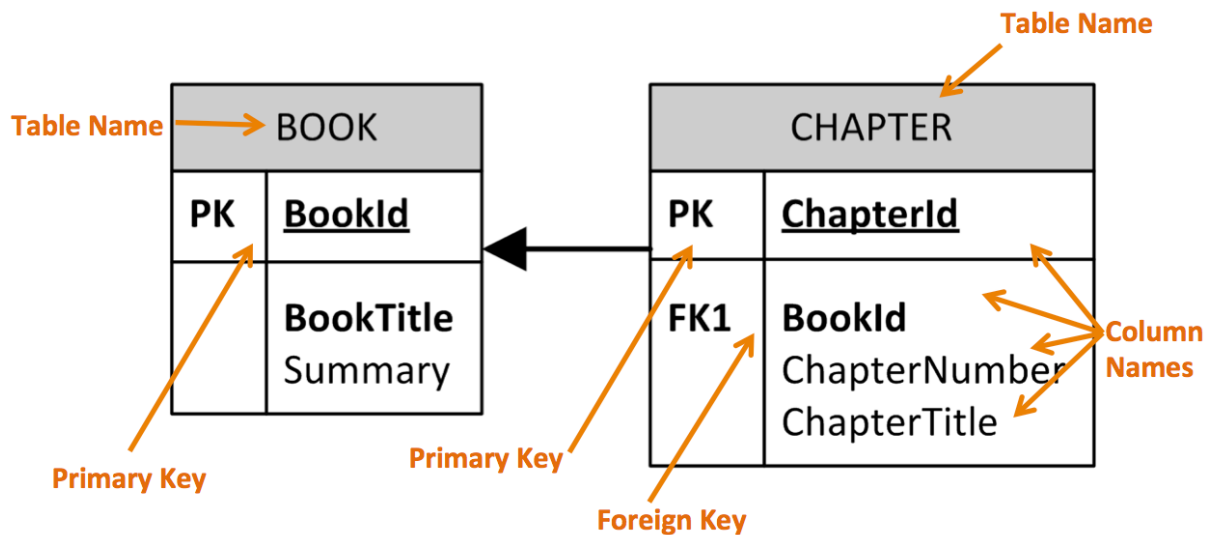


Figure 14-1: ER diagram with Primary Keys and Foreign Keys relationships

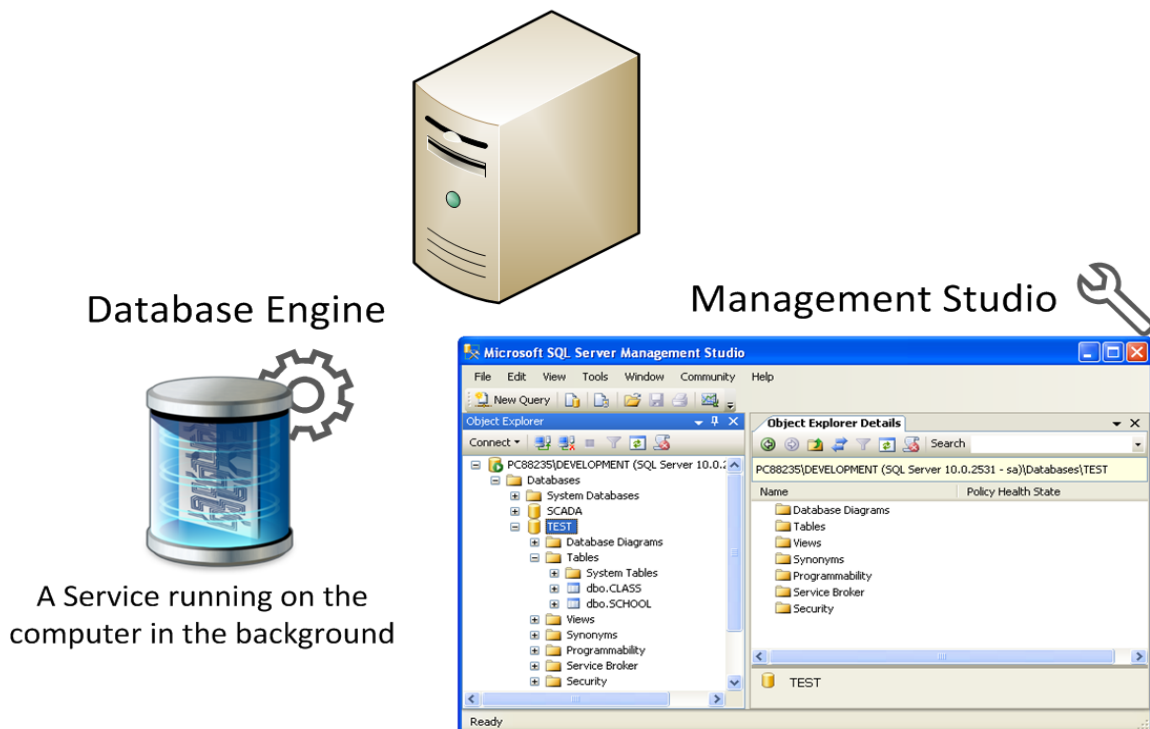


Introduction to Database Systems: <https://youtu.be/n75iPNrzN-o>

14.1 SQL Server

SQL Server is a Database System from Microsoft. SQL Server comes in different editions, for basic, personal use SQL Server Express is recommended because it is simple to use and it is free.

SQL Server consists of a Database Engine and a Management Studio (and lots of other stuff which we will not mention here). The Database engine has no graphical interface - it is just a service running in the background of your computer (preferable on the server). The Management Studio is graphical tool for configuring and viewing the information in the database. It can be installed on the server or on the client (or both).



A Graphical User Interface to the database used for configuration and management of the database

Figure 14-2: SQL Server

Videos:



SQL Server Express Installation: <https://youtu.be/hhhggAIUYo8>



Introduction to SQL Server: <https://youtu.be/SIR4KOhAG1U>

14.1.1 SQL Server Management Studio

SQL Server Management Studio (SSMS) is used to manage your databases, including creating, updating, deleting, etc. You can insert tables, create views, stored procedures, etc.

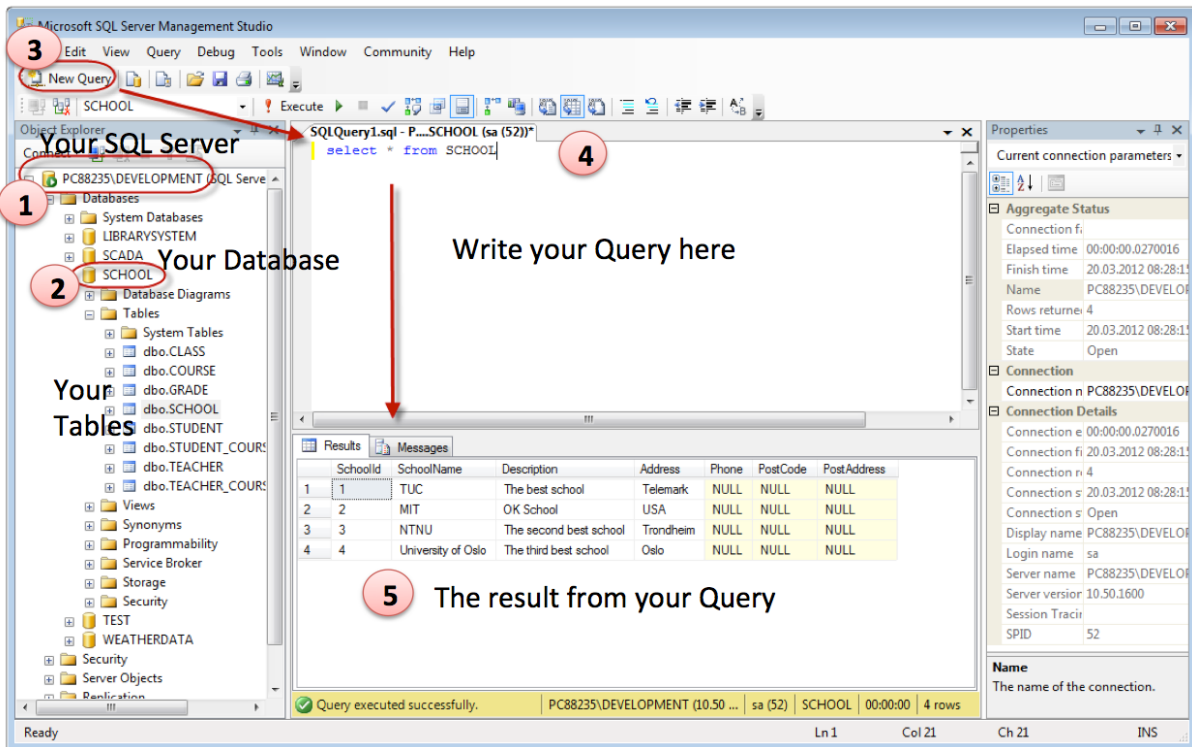


Figure 14-3: SQL Server Management Studio (SSMS)

It is quite simple to create a new database in Microsoft SQL Server. Just right-click on the “Databases” node and select “New Database...”

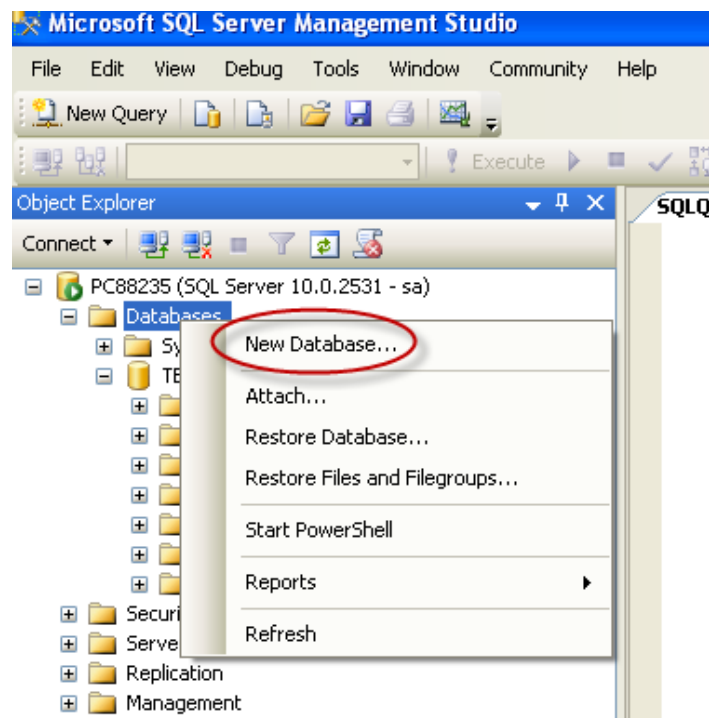


Figure 14-4: Create New Database

There are lots of settings you may set regarding your database, but the only information you must fill in is the name of your database:

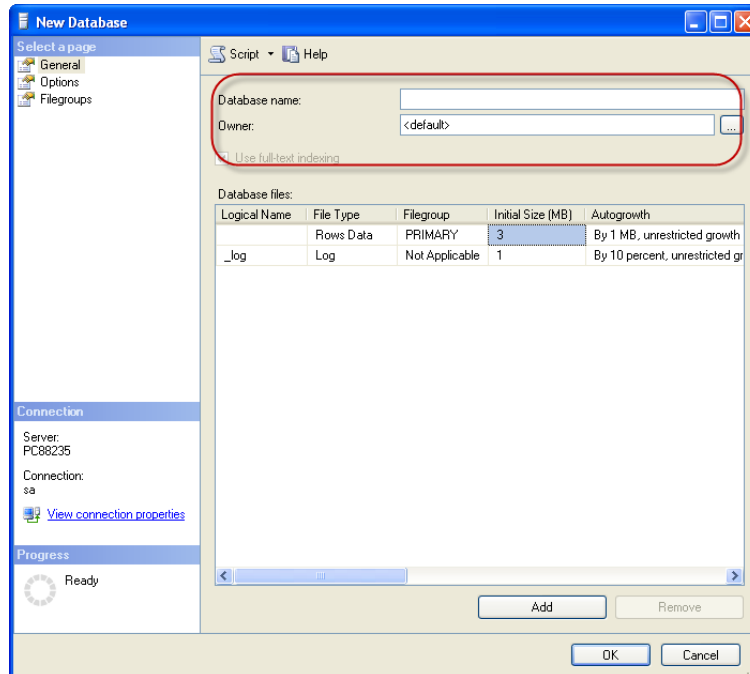


Figure 14-5: New Database Configuration

You may also use the SQL language to create a new database, but sometimes it is easier to just use the built-in features in the Management Studio.

In order to make a new SQL query, select the “New Query” button from the Toolbar.

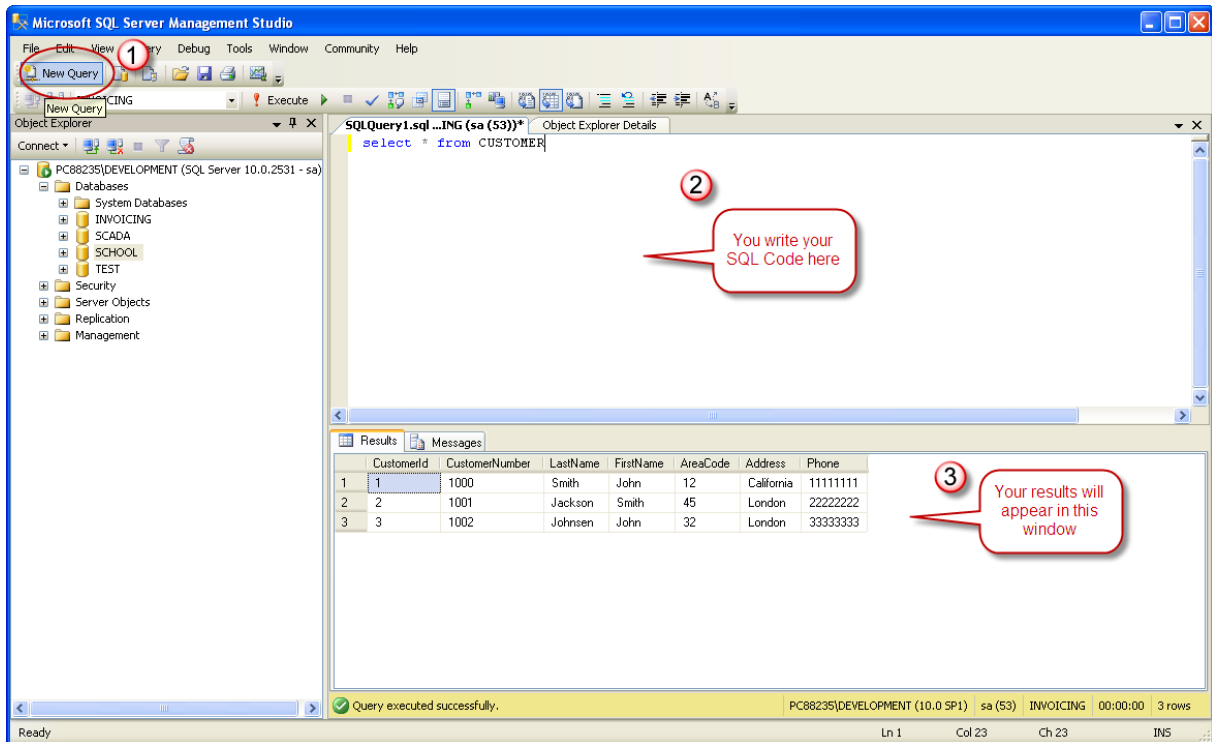


Figure 14-6: Queries

Here we can write any kind of queries that is supported by the SQL language.

14.2 Structured Query Language (SQL)

SQL (Structured Query Language) is a database computer language designed for managing data in relational database management systems (RDBMS).

In SQL, we have 4 different types of queries:

- INSERT
- SELECT
- UPDATE
- DELETE

What can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database

- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

The **Data Manipulation Language (DML)** is the subset of SQL used to add, update and delete data.

The acronym **CRUD** refers to all of the major functions that need to be implemented in a relational database application to consider it complete. Each letter in the acronym can be mapped to a standard SQL statement:

Operation	SQL	Description
Create	INSERT INTO	inserts new data into a database
Read (Retrieve)	SELECT	extracts data from a database
Update	UPDATE	updates data in a database
Delete (Destroy)	DELETE	deletes data from a database

Additional Resources:

Do you want to learn more about SQL?



Database Systems: <https://www.halvorsen.blog/documents/technology/database/>

14.2.1 Tables

In Figure 14-7 we see an example of some tables for a university or a school. These tables store information regarding the students, the teacher, the courses, the grades, etc.

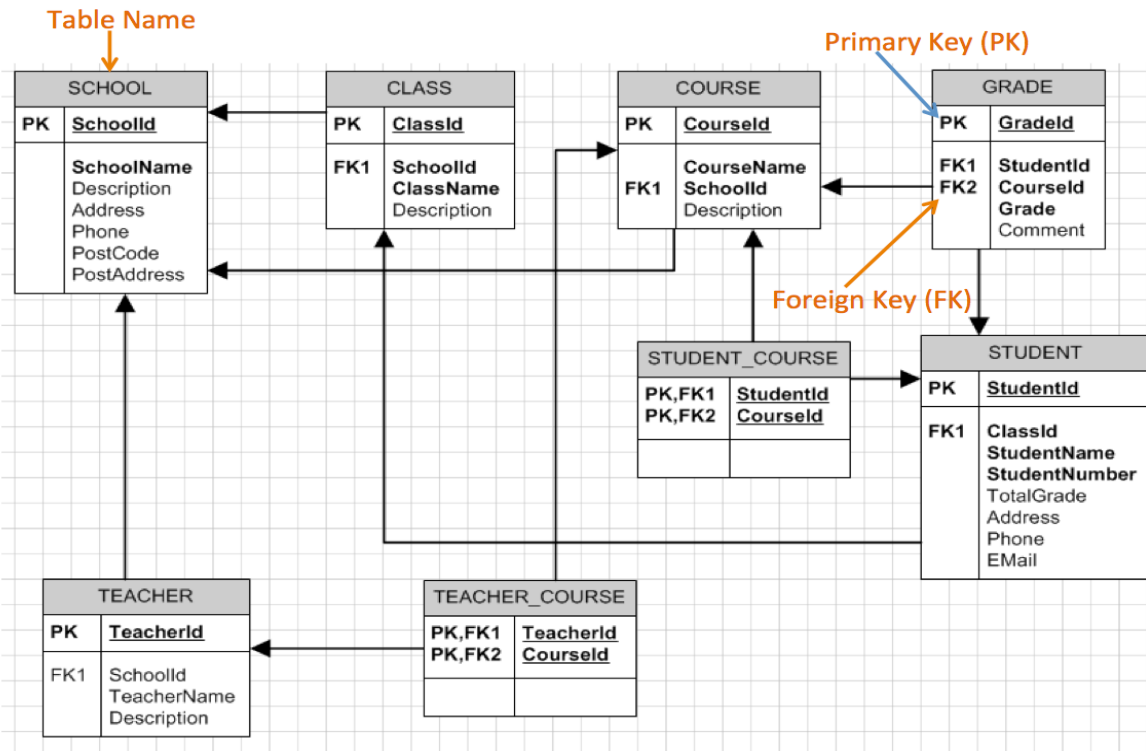


Figure 14-7: Example of Tables with relations

Best practice rules:

Here are some “Best practice” recommendations for creating tables in a database system:

- **Tables:** Use upper case and singular form in table names – not plural, e.g., “STUDENT” (not students)
- **Columns:** Use Pascal notation, e.g., “StudentId”
- **Primary Keys:**
 - If the table name is “COURSE”, name the Primary Key column “CourseId”, etc.
 - “Always” use Integer and Identity(1,1) for Primary Keys
- Specify Required Columns (NOT NULL) – i.e., which columns that need to have data or not
- **Data Types:** Standardize on these Data Types: int, float, varchar(x), datetime, bit
- Use English for table and column names
- Avoid abbreviations! (Use RoomNumber – not RoomNo, RoomNr, ...)

The **CREATE TABLE** statement is used to create a table in a database.

We want to create a table called “CUSTOMER” which has the following columns and data types:

	Column Name	Data Type	Allow Nulls
PK	CustomerId	int	<input type="checkbox"/>
	CustomerNumber	int	<input type="checkbox"/>
	LastName	varchar(50)	<input type="checkbox"/>
	FirstName	varchar(50)	<input type="checkbox"/>
	AreaCode	int	<input checked="" type="checkbox"/>
	Address	varchar(50)	<input checked="" type="checkbox"/>
	Phone	varchar(20)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Figure 14-8: Table Editor in SQL Server Management Studio

```
CREATE TABLE CUSTOMER
(
    CustomerId int IDENTITY(1,1) PRIMARY KEY,
    CustomerNumber int NOT NULL UNIQUE,
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(50) NULL,
    Phone varchar(50) NULL,
)
GO
```

Typically, you also want to use a tool for modelling the database, e.g., Erwin.

Primary keys:

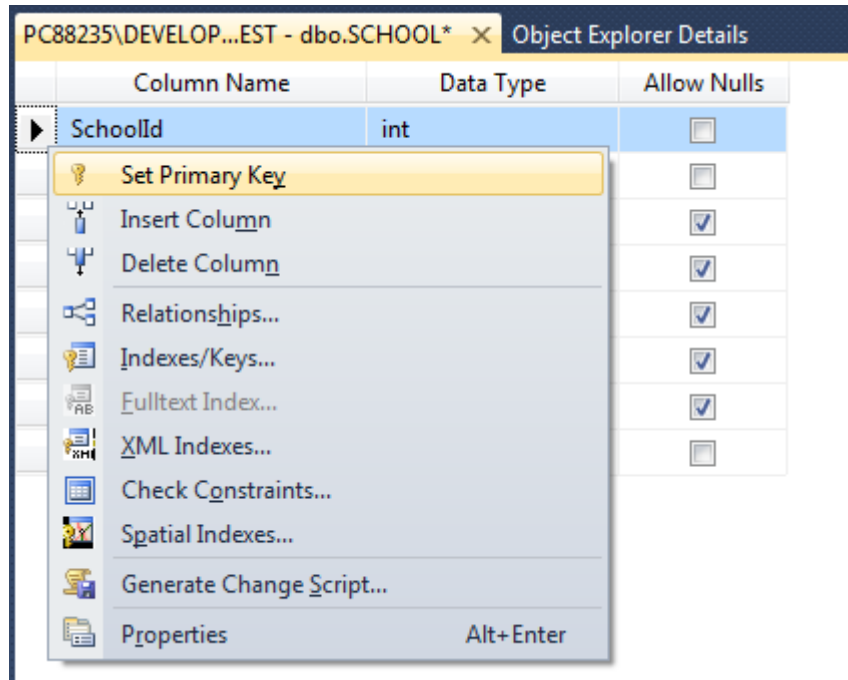
As you see we use the “Primary Key” keyword to specify that a column should be the Primary Key.


	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000					111111
2	2	1001					222222
3	3	1002	Johnson	John	32	London	33333333

Primary Keys must contain unique numbers like this

Setting Primary Keys in the Designer Tools:

If you use the Designer tools in SQL Server, you can easily set the primary Key in a table just by right-click and select “Set primary Key”.

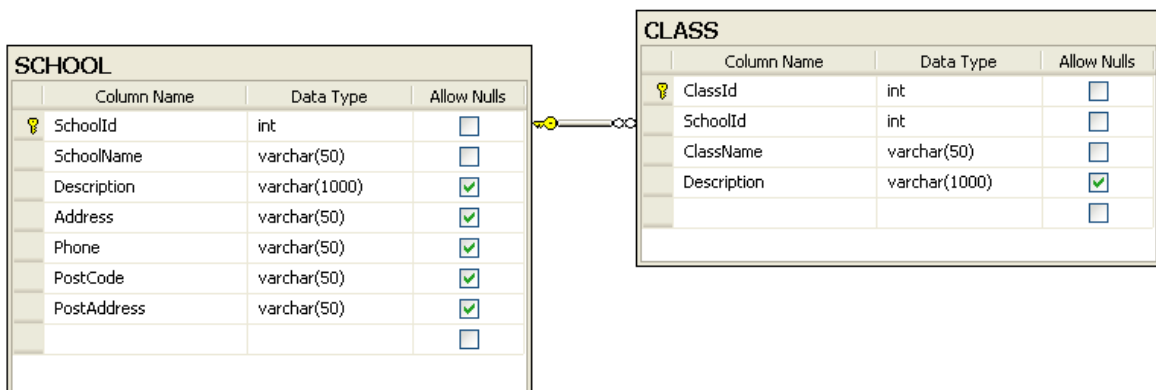


The primary Key column will then have a small key  in front to illustrate that this column is a Primary Key.

Foreign Keys:

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Example:



We will create a CREATE TABLE script for these tables:

SCHOOL:

```
CREATE TABLE SCHOOL
(
    SchoolId int IDENTITY(1,1) PRIMARY KEY,
    SchoolName varchar(50) NOT NULL UNIQUE,
```

```

Description varchar(1000) NULL,
Address varchar(50) NULL,
Phone varchar(50) NULL,
PostCode varchar(50) NULL,
PostAddress varchar(50) NULL,
)
GO

```

CLASS:

```

CREATE TABLE CLASS
(
    ClassId int IDENTITY(1,1) PRIMARY KEY,
    SchoolId int NOT NULL FOREIGN KEY REFERENCES SCHOOL (SchoolId),
    ClassName varchar(50) NOT NULL UNIQUE,
    Description varchar(1000) NULL,
)
GO

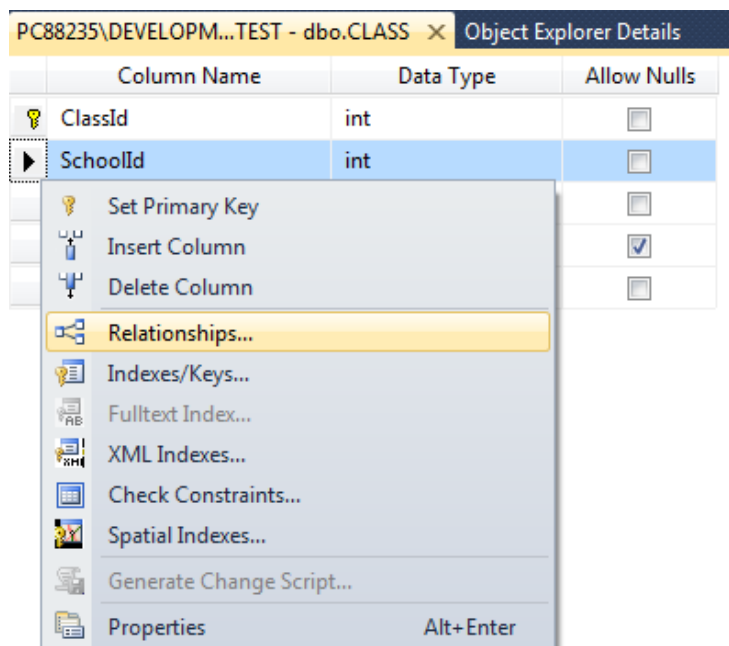
```

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

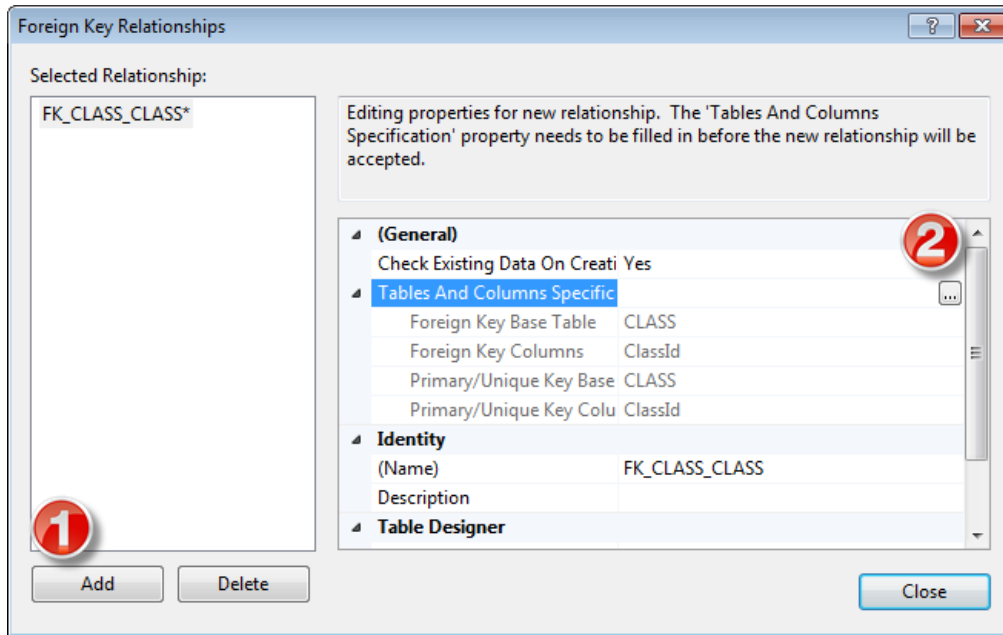
The FOREIGN KEY constraint also prevents that invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

Setting Foreign Keys in the Designer Tools:

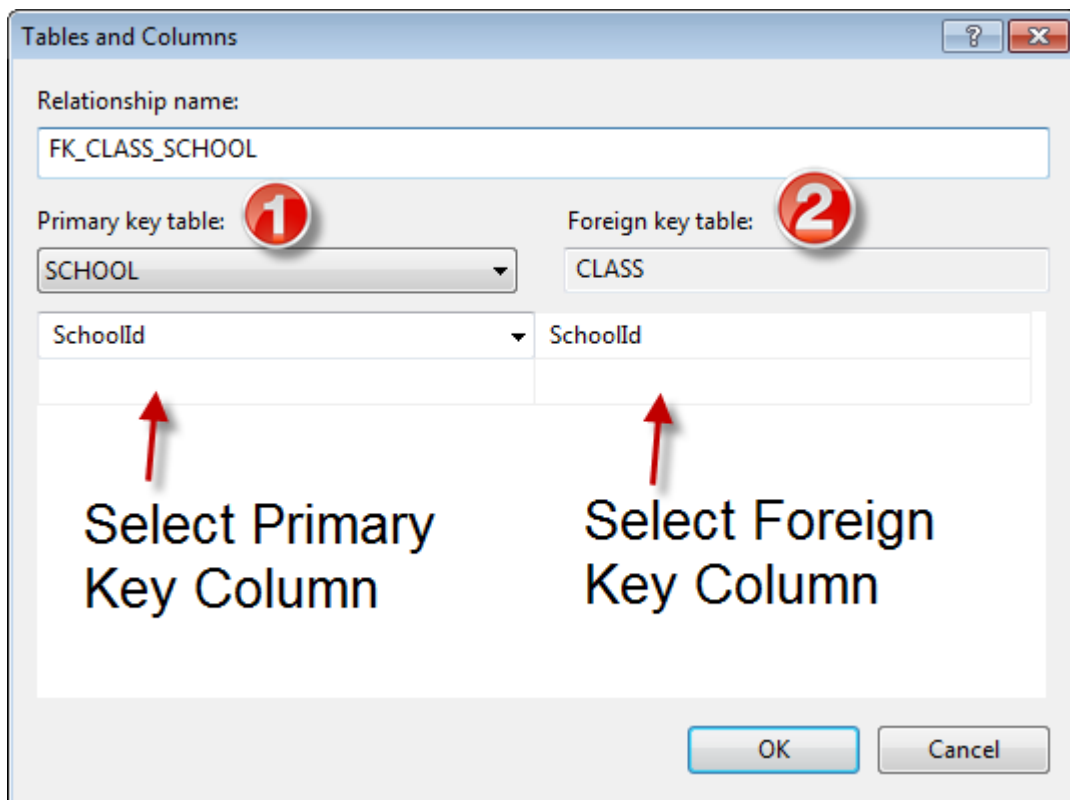
If you want to use the designer, right-click on the column that you want to be the Foreign Key and select “**Relationships...**”:



The following window pops up (Foreign Key Relationships):



Click on the “Add” button and then click on the small “...” button. Then the following window pops up (Tables and Columns):



Here you specify the primary Key Column in the Primary Key table and the Foreign Key Column in the Foreign Key table.

NOT NULL / Required Columns:

The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record or update a record without adding a value to this field.

If we take a closer look at the CUSTOMER table created earlier:

```
CREATE TABLE [CUSTOMER]
(
    CustomerId int IDENTITY(1,1) PRIMARY KEY,
    CustomerNumber int NOT NULL UNIQUE,
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(50) NULL,
    Phone varchar(50) NULL,
)
GO
```

We see that “CustomerNumber”, “LastName” and “FirstName” is set to “NOT NULL”, this means these columns needs to contain data. While “AreaCode”, “Address” and “Phone” may be left empty, i.e, they don’t need to be filled out.

Note! A primary key column cannot contain NULL values.

Setting NULL/NOT NULL in the Designer Tools:

In the Table Designer you can easily set which columns that should allow NULL or not:

Column Name	Data Type	Allow Nulls
SchoolId	int	<input type="checkbox"/>
SchoolName	varchar(50)	<input type="checkbox"/>
Description	varchar(1000)	<input checked="" type="checkbox"/>
Address	varchar(50)	<input checked="" type="checkbox"/>
Phone	varchar(50)	<input checked="" type="checkbox"/>
PostCode	varchar(50)	<input checked="" type="checkbox"/>
PostAddress	varchar(50)	<input checked="" type="checkbox"/>

UNIQUE:

The **UNIQUE** constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note! You can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

If we take a closer look at the CUSTOMER table created earlier:

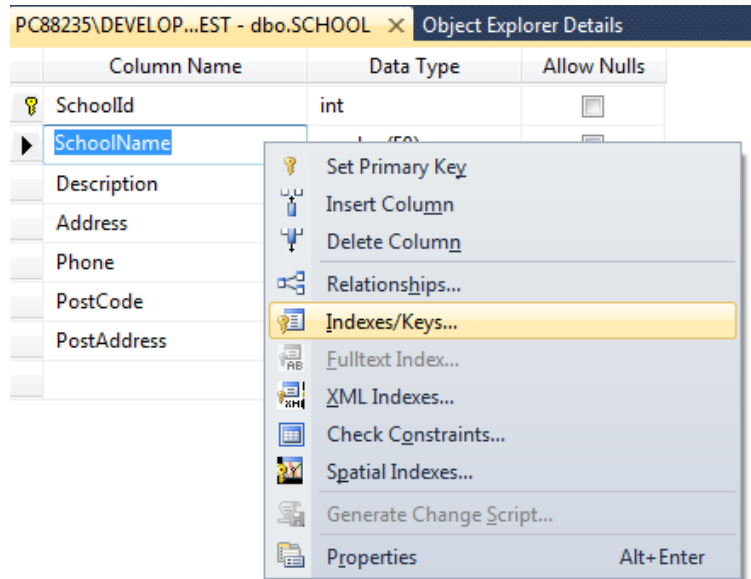
```
CREATE TABLE [CUSTOMER]
(
    CustomerId int IDENTITY(1,1) PRIMARY KEY,
    CustomerNumber int NOT NULL UNIQUE,
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(50) NULL,
    Phone varchar(50) NULL,
)
GO
```

We see that the “CustomerNumber” is set to UNIQUE, meaning each customer must have a unique Customer Number. Example:

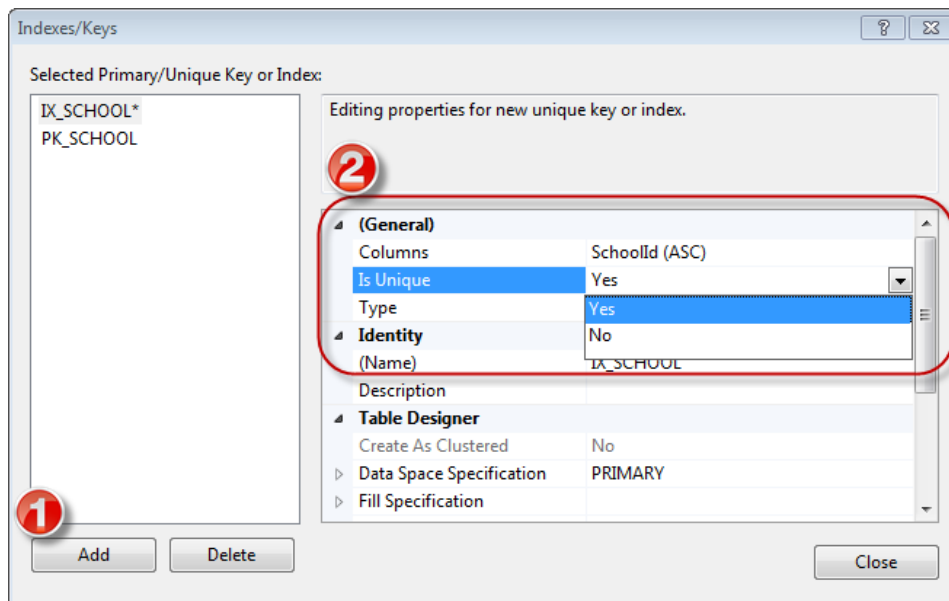
	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000	Smith	John	12	California	11111111
2	2	1001	Jackson	Smith	45	London	22222222
3	3	1002	Johnsen	John	32	London	33333333

Setting UNIQUE in the Designer Tools:

If you want to use the designer, right-click on the column that you want to be UNIQUE and select “**Indexes/Keys...**”:



Then click “Add” and then set the “Is Unique” property to “Yes”:



AUTO INCREMENT or IDENTITY:

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.

Example:

```
CREATE TABLE CUSTOMER
(
    CustomerId int IDENTITY(1,1) PRIMARY KEY,
    CustomerNumber int NOT NULL UNIQUE,
    LastName varchar(50) NOT NULL,
    FirstName varchar(50) NOT NULL,
    AreaCode int NULL,
    Address varchar(50) NULL,
```

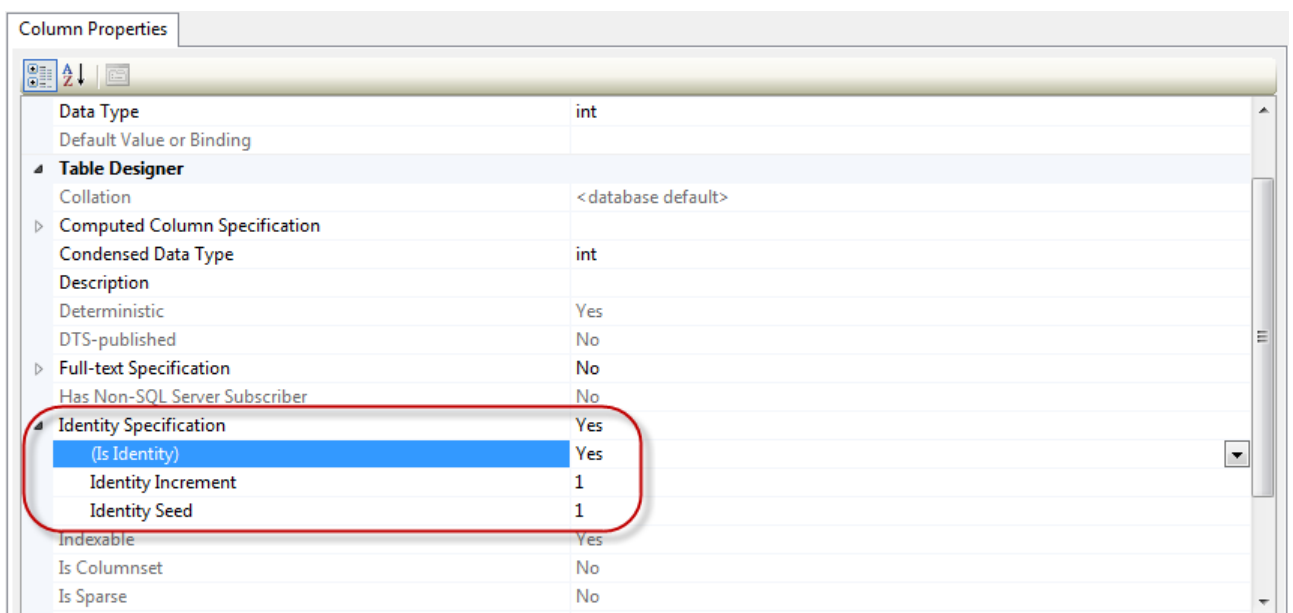
```
Phone varchar(50) NULL,
)
GO
```

As shown below, we use the IDENTITY() for this. IDENTITY(1,1) means the first value will be 1 and then it will increment by 1.

Setting identity(1,1) in the Designer Tools:

We can use the designer tools to specify that a Primary Key should be an identity column that is automatically generated by the system when we insert data in to the table.

Click on the column in the designer and go into the Column Properties window:



14.2.2 Views

Views are virtual table for easier access to data stored in multiple tables.

Create View:

```

IF EXISTS (SELECT name
           FROM sysobjects
           WHERE name = 'CourseData'
           AND type = 'V')
    DROP VIEW CourseData
GO

CREATE VIEW CourseData
AS

SELECT
    SCHOOL.SchoolId,
    SCHOOL.SchoolName,
    COURSE.CourseId,
    COURSE.CourseName,
    COURSE.Description

FROM
    SCHOOL
    INNER JOIN COURSE ON SCHOOL.SchoolId = COURSE.SchoolId
GO
    
```

A View is a “virtual” table that can contain data from multiple tables

The Name of the View

Inside the View you join the different tables together using the **JOIN** operator

You can Use the View as an ordinary table in Queries :

Using the View:

```
select * from CourseData
```

	SchoolId	SchoolName	CourseId	CourseName	Description
1	1	TUC	1	Industrial IT	The best course ever
2	1	TUC	2	Control with Implementation	Control Theory
3	1	TUC	3	Systems and Control Laboratory	Practical Lav course

Figure 14-9: Views

Example:

We use the SCHOOL and CLASS tables as an example for our View. We want to create a View that lists all the existing schools and the belonging classes.

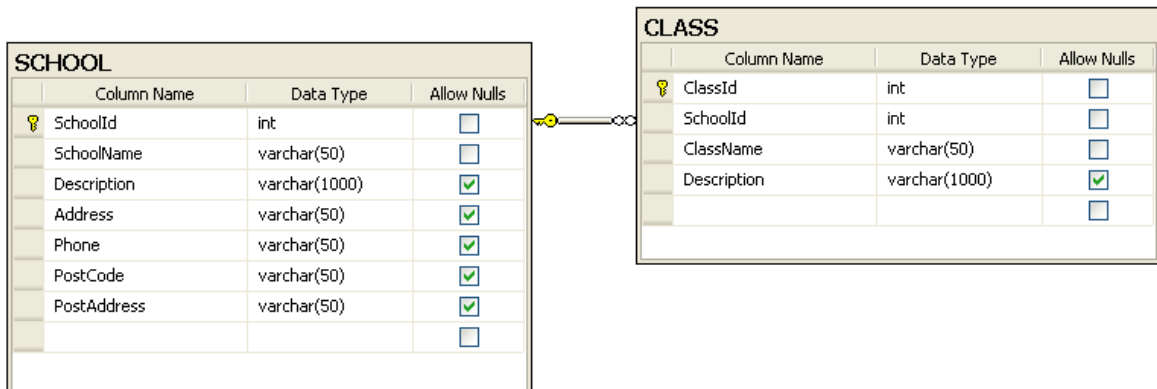


Figure 14-10: Database Example

We create the VIEW using the CREATE VIEW command:

```
CREATE VIEW SchoolView
AS

SELECT
SCHOOL.SchoolName,
CLASS.ClassName
FROM
SCHOOL
INNER JOIN CLASS ON SCHOOL.SchoolId = CLASS.SchoolId
```

Note! In order to get information from more than one table, we need to link the tables together using a JOIN.



Database Views and Stored Procedures: https://youtu.be/SHELF_iQUeU

14.2.3 Stored Procedures

A Stored Procedure is a precompiled collection of SQL statements. In a stored procedure you can use if sentence, declare variables, etc.

Create Stored Procedure:

```
IF EXISTS (SELECT name
           FROM sysobjects
           WHERE name = 'StudentGrade'
           AND type = 'P')
DROP PROCEDURE StudentGrade
GO

CREATE PROCEDURE StudentGrade
@Student varchar(50),
@Course varchar(10),
@Grade varchar(1)

AS

DECLARE
@StudentId int,
@CourseId int

select StudentId from STUDENT where StudentName = @Student

select CourseId from COURSE where CourseName = @Course

insert into GRADE (StudentId, CourseId, Grade)
values (@StudentId, @CourseId, @Grade)
GO
```

Using the Stored Procedure:

```
execute StudentGrade 'John Wayne', 'SCE2006', 'B'
```

A Stored Procedure is like Method in C# - it is a piece of code with SQL commands that do a specific task – and you reuse it

Procedure Name

Input Arguments

Internal/Local Variables
Note! Each variable starts with @

SQL Code (the “body” of the Stored Procedure)

Figure 14-11: Stored Procedure – Example



Database Views and Stored Procedures: https://youtu.be/SHELF_iQUeU

14.2.4 Triggers

A database trigger is code that is automatically executed in response to certain events on a particular table in a database.

A Trigger is executed when you insert, update or delete data in a Table specified in the Trigger.

Create the Trigger:

```

IF EXISTS (SELECT name
           FROM   sysobjects
           WHERE  name = 'CalcAvgGrade'
           AND    type = 'TR')
    DROP TRIGGER CalcAvgGrade
GO

CREATE TRIGGER CalcAvgGrade ON GRADE
FOR UPDATE, INSERT, DELETE
AS
    DECLARE
    @StudentId int,
    @AvgGrade float

    select @StudentId = StudentId from INSERTED
    select @AvgGrade = AVG(Grade) from GRADE where StudentId = @StudentId
    update STUDENT set TotalGrade = @AvgGrade where StudentId = @StudentId
GO

```

Name of the Trigger

Specify which Table the Trigger shall work on

Specify what kind of operations the Trigger shall act on

Internal/Local Variables

Inside the Trigger you can use ordinary SQL statements, create variables, etc.

SQL Code (The "body" of the Trigger)

Note! "INSERTED" is a temporarily table containing the latest inserted data, and it is very handy to use inside a trigger

Figure 14-12: Trigger - Example

15 ADO.NET

ADO.NET is the core data access technology for .NET languages.

The great thing about this is that you can use the same C# code either you are creating a desktop application or a web application.

Typically, you put your C# database code in one or more classes, and those will then be the same either you are creating a desktop application (which has direct access to the database) or a web application. If your database does not have direct access to the database (the database is e.g., located on the internet), you can create and use a so-called Web API (also called Web Service or REST API). The Web API will then be the middleware between the desktop application and the database server.

System.Data.SqlClient (or the newer **Microsoft.Data.SqlClient**) is the provider or namespace you typically use to connect to an SQL Server.

You install these packages using the NuGet Package Manager, see Figure 15-1.

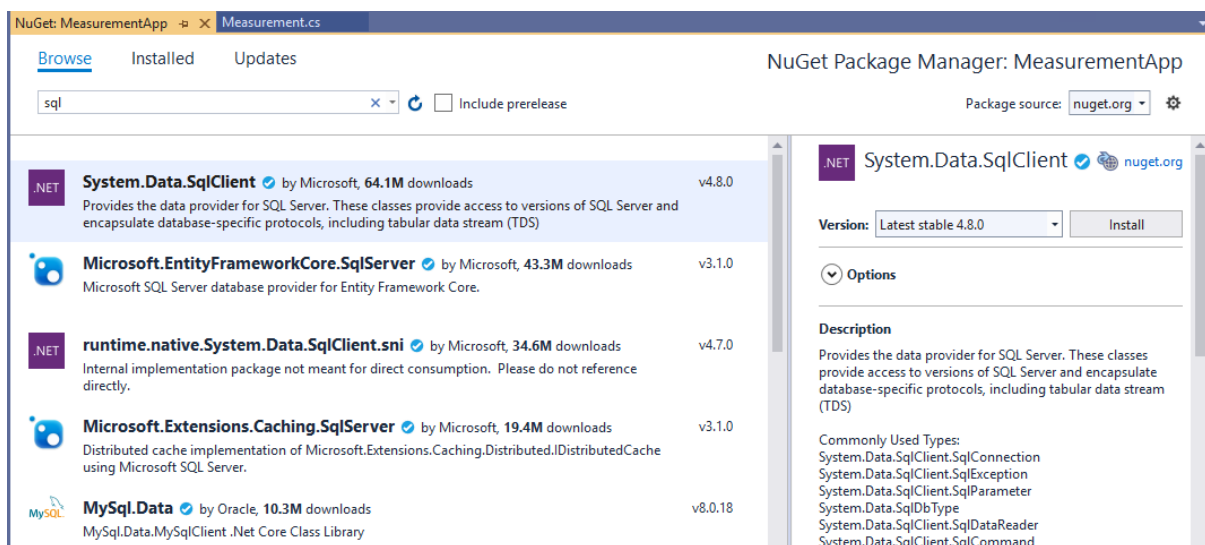


Figure 15-1: Installing using the NuGet Package Manager

16 Data from Database

To get data from a database is something you need to know about. Typically, all real applications get some data from a database.

There are many ways and methods we can use to retrieve data from a database. Here we will focus on something called **ADO.NET**. ADO.NET can use different so-called Data Providers. We will use “System.Data.SqlClient”. This will be demonstrated in the Demo Application below.

If you have used “ADO.NET” and “System.Data.SqlClient” in a desktop application (e.g., WinForm) before then there is nothing new since all this happens in basic C# code.

16.1 Demo Application

In this Demo Application we will create a basic application that gets data from an SQL Server database. Figure 16-1 shows the application we are going to create.

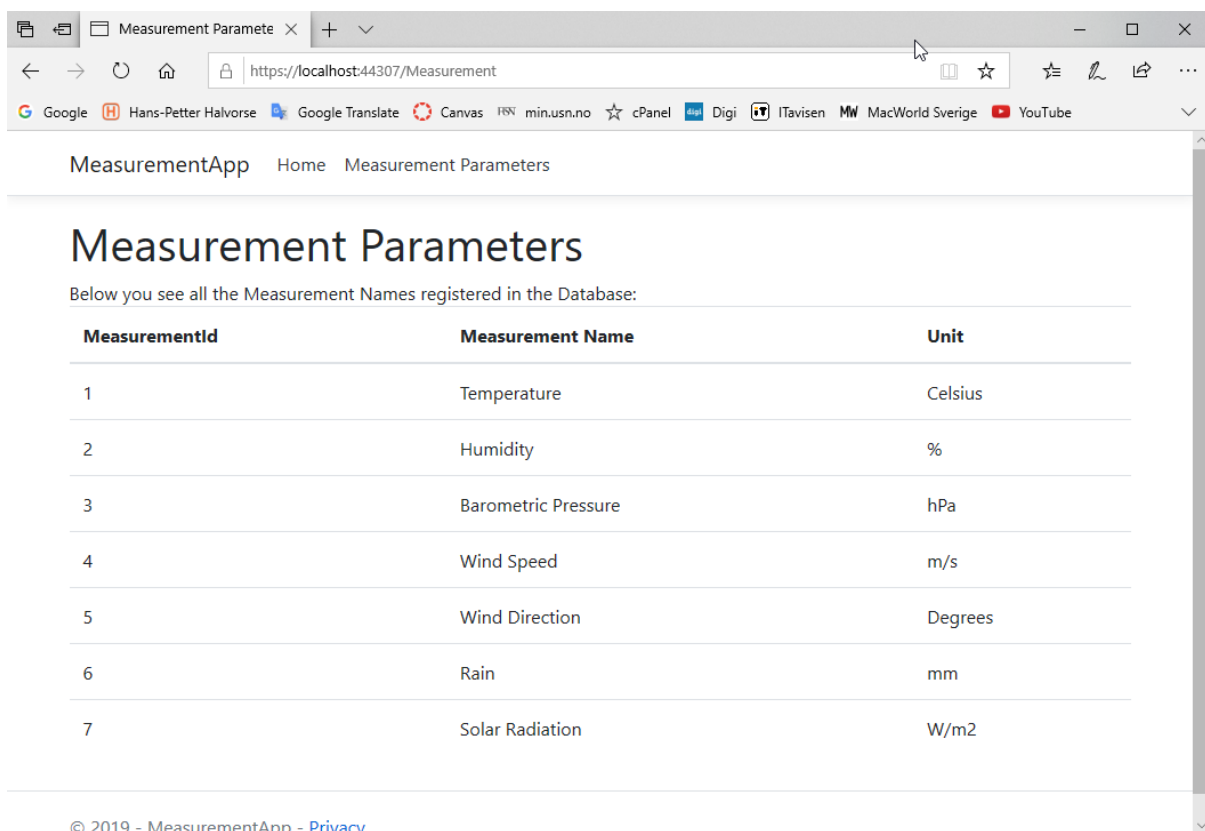


Figure 16-1: Database Application



ASP.NET Core Database Communication: <https://youtu.be/0Ta3dQ3rxzs>

The entire example can be downloaded from the home page of this textbook.

16.1.1 Database

In order to create this ASP.NET Core example we need to create a simple database that consists of a single table called “MEASUREMENT”.

We can use the following database script:

```
CREATE TABLE [MEASUREMENT]
(
    [MeasurementId]    int NOT NULL IDENTITY ( 1,1 ) Primary Key,
    [MeasurementName]  varchar(100) NOT NULL UNIQUE,
    [Unit]             varchar(50) NULL
)
go
```

We start by creating the Database in SQL Server Management Studio (Figure 16-2).

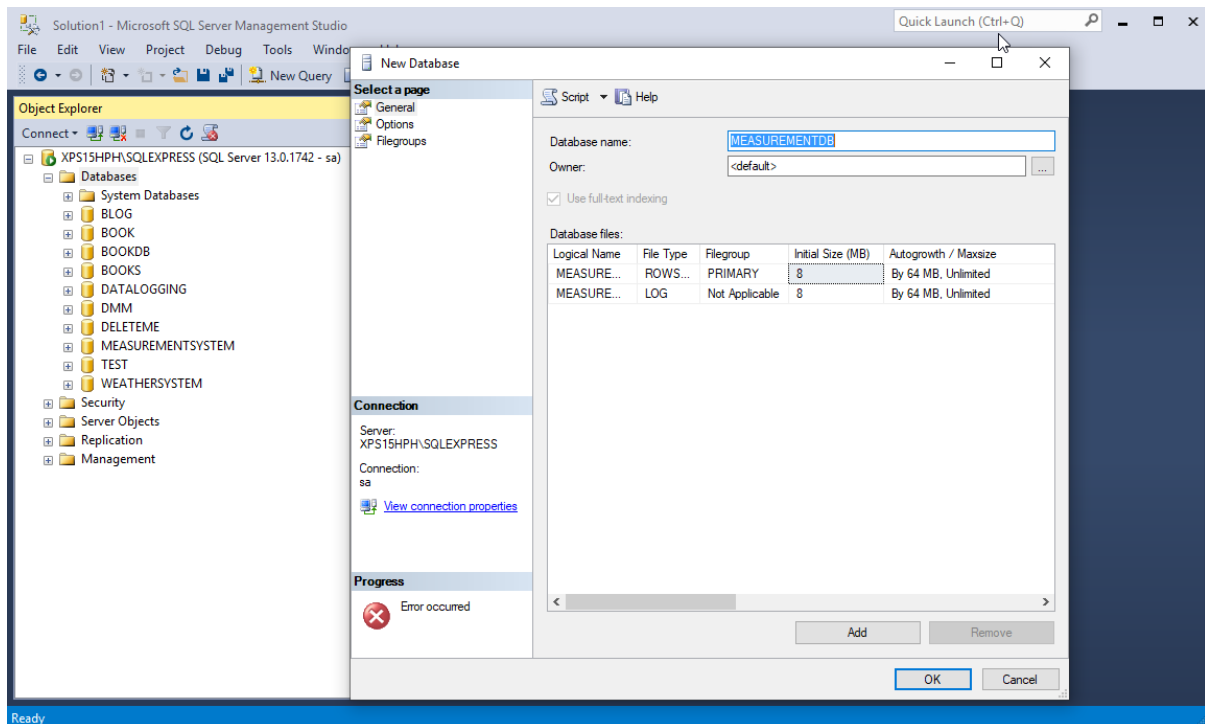


Figure 16-2: Creating the Database in SQL Server Management Studio

Then we create necessary table. We can either create the tables directly using the Table designer in the SQL Server Management Studio (not shown here) or we can open/create a SQL script that inserts the table (Figure 16-3).

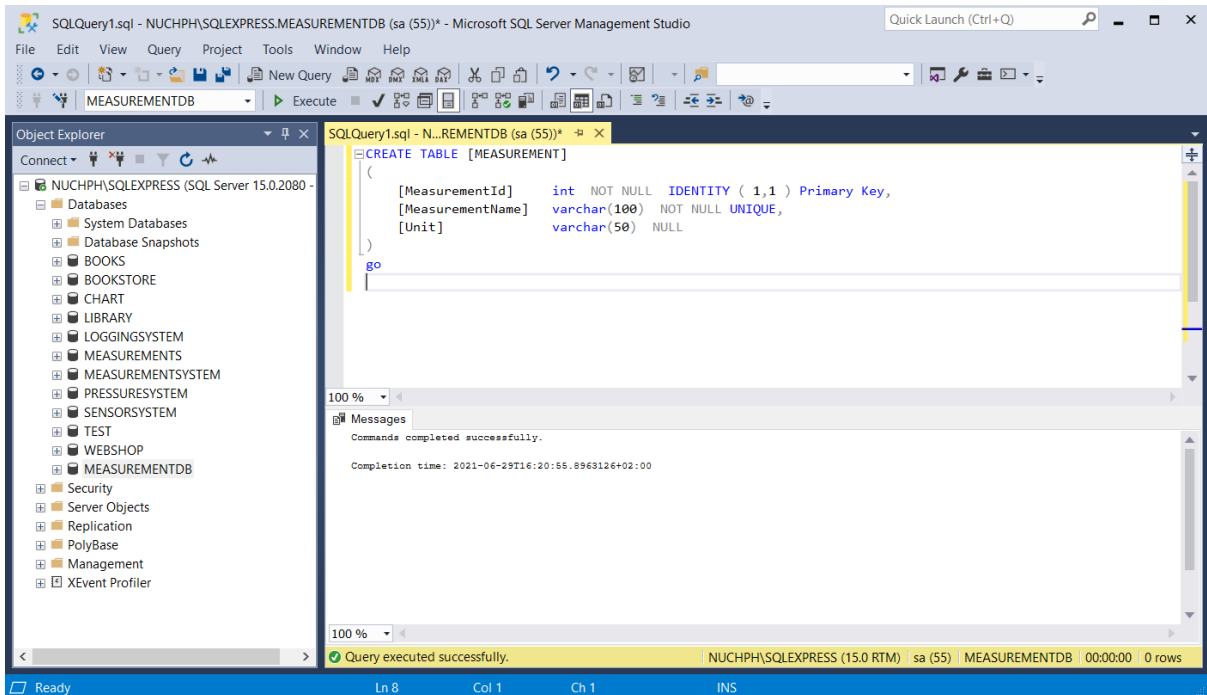


Figure 16-3: Creating Database Table

Next, we need to create some initial data into the table. We can either create the data directly using the editor (right-click on the table in the Object Explorer and select “Edit Top 200 Rows”) in the SQL Server Management Studio (See Figure 16-4) or we can open/create a SQL script that inserts the necessary data (not shown here).

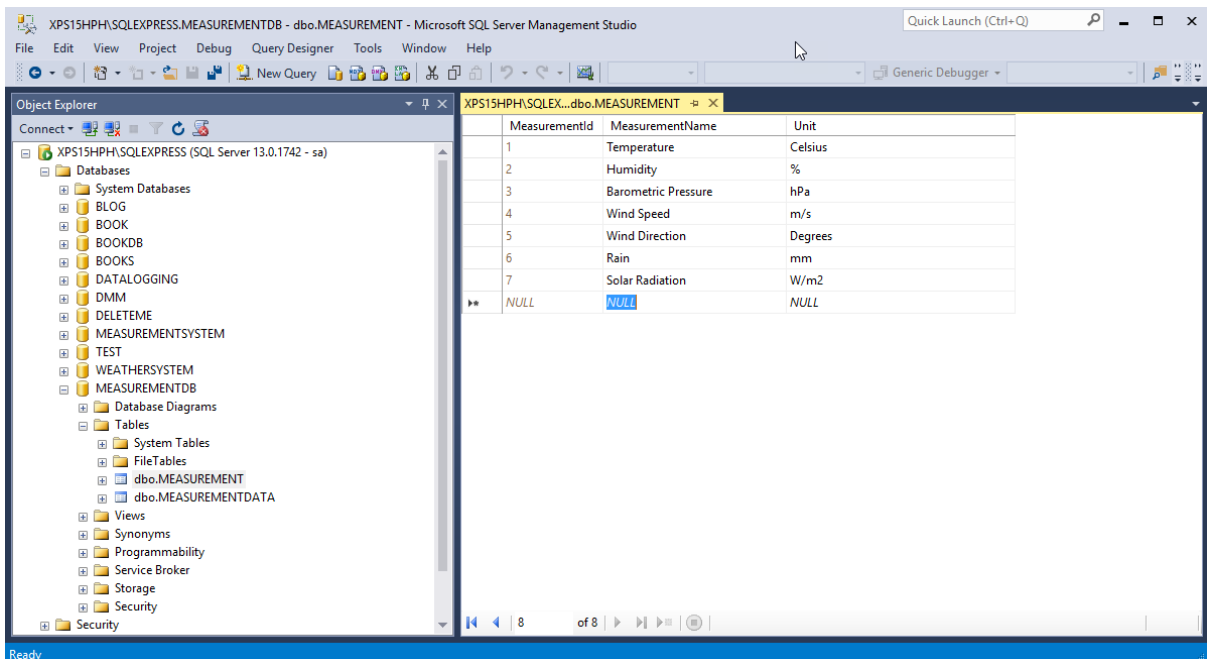


Figure 16-4: Inserting Data manually from SQL Server Management Studio

16.1.2 Visual Studio

When our database is ready, we can start the coding using Visual Studio and C#. We start by creating a New Project (Figure 16-5).

Select the “ASP.NET Core Web Application” project template.

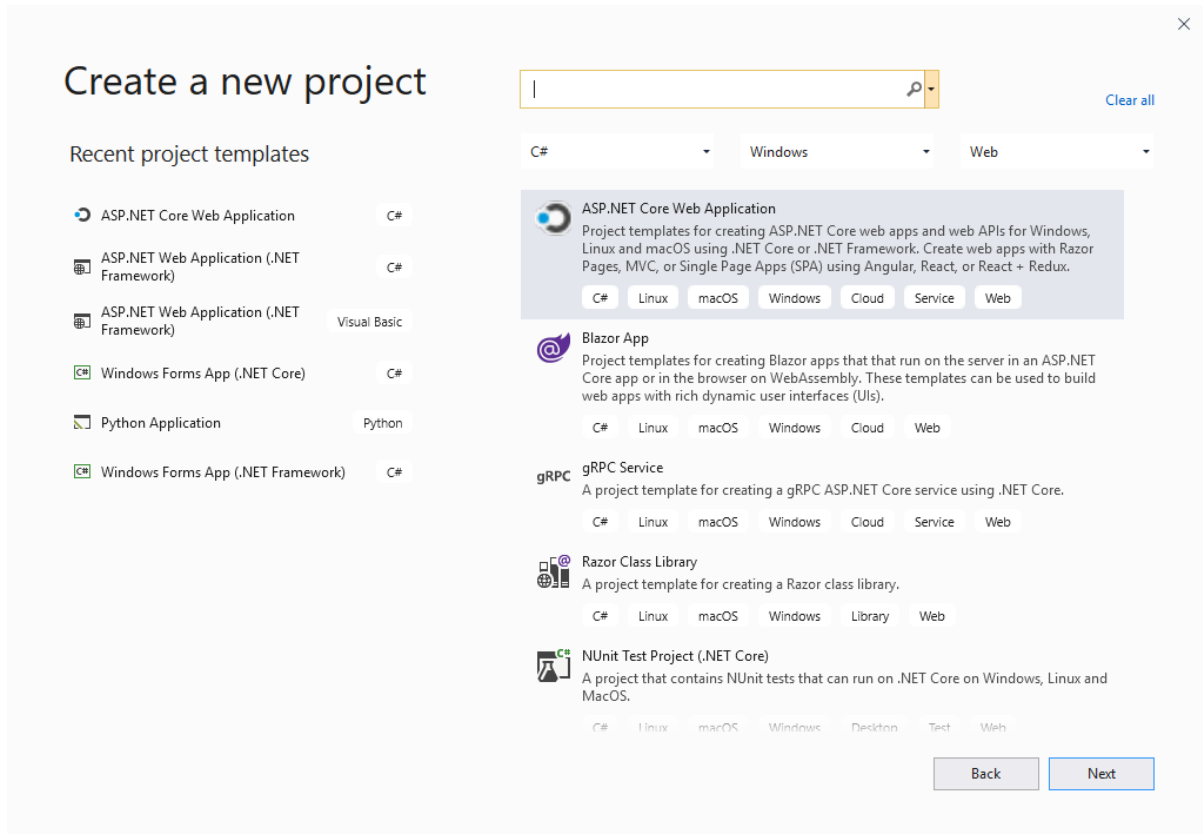
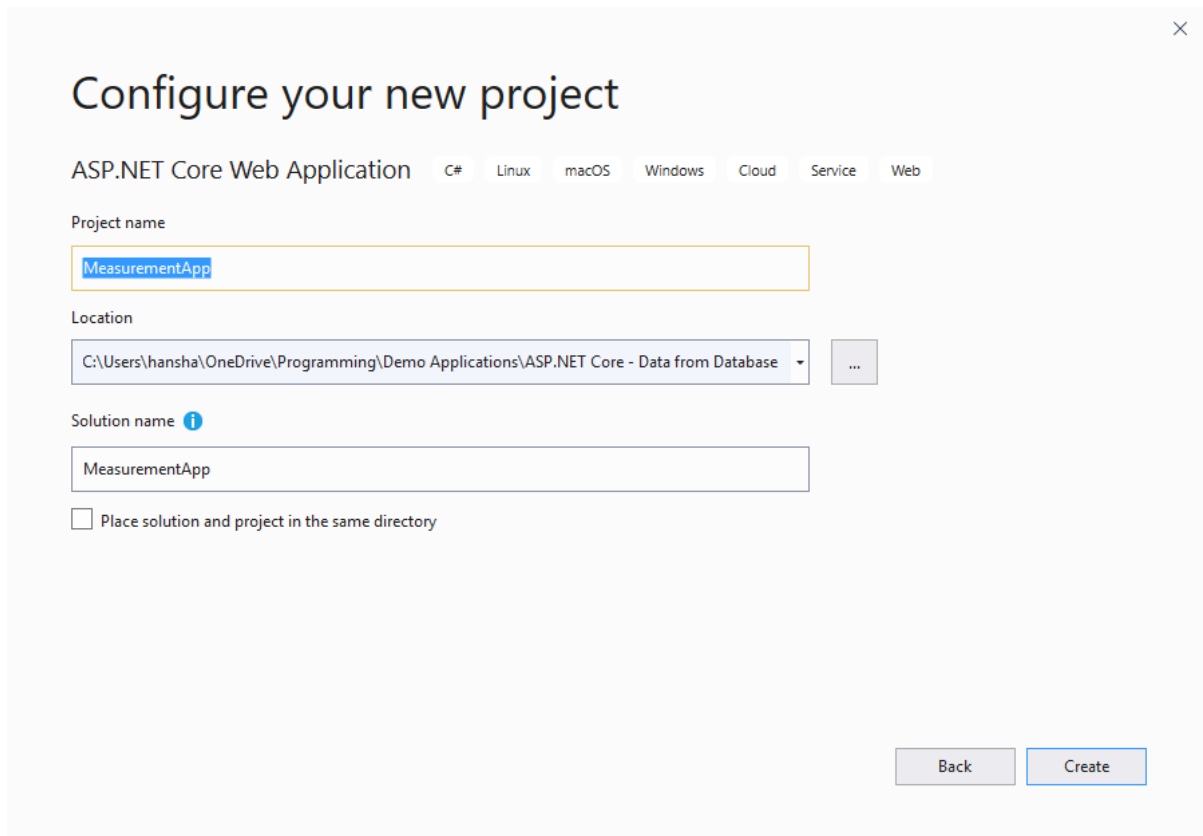


Figure 16-5: New Project – ASP.NET Core Web Application

Next you need to configure your Project (Figure 16-6) by writing the name. for the project, where it should be located on your hard drive, etc.



Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name

MeasurementApp

Location

C:\Users\hansha\OneDrive\Programming\Demo Applications\ASP.NET Core - Data from Database ...

Solution name ⓘ

MeasurementApp

Place solution and project in the same directory

Back Create

Figure 16-6: Configure your New Project

In the next window (see Figure 16-7) you need to select the proper template. We select the “Web Application” template.

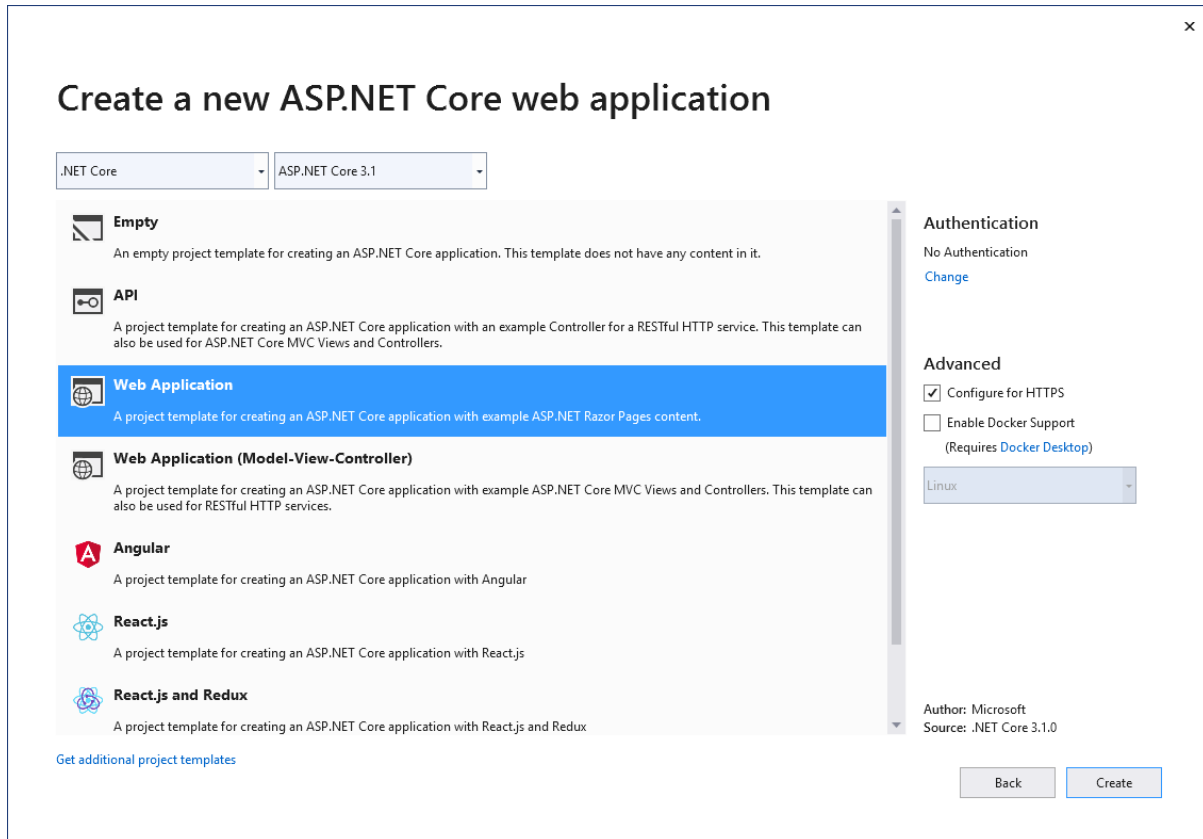


Figure 16-7: Create Web Application

Select “Create” in Figure 16-7 and the Visual studio Project will be created based on your choices. See Figure 16-8.

As you see from Figure 16-8, the following Folders and Files have been created:

- **appSettings.json** – This file contains configuration data, such as connection strings.
- **Program.cs** – This file contains the entry point for the program.
- **Startup.cs** - This file contains code that configures app behavior.
- **wwwroot** folder - Contains static files, such as HTML files, JavaScript files, and CSS files.
- **Pages** folder – Here you are supposed to put your ASP.NET (".cshtml") web pages

In addition, it is standard to have a folder called “**Models**”. This folder contains C# classes that takes care of the data. The data can, e.g., be a database or a file, e.g., a JSON file.

In addition, we have what we call Supporting files. Supporting files have names that begin with an underscore (_).

- **_Layout.cshtml** file configures UI elements common to all pages. You can use this file to set up the navigation menu at the top of the page

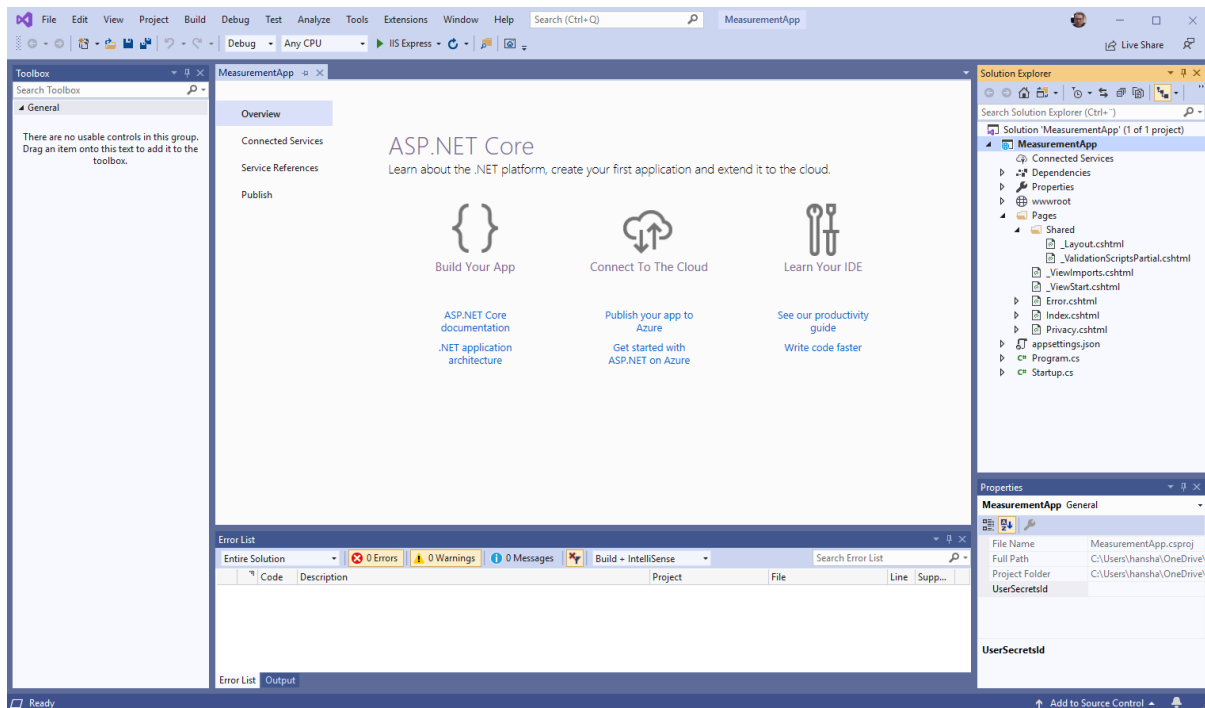


Figure 16-8: Visual Studio Project

In the Pages folder some default Razor pages have been created, like “Index.cshtml”, etc. The “Index.cshtml” file is typically the startup file for your web application, but if you want you can configure this in the Startup.cs file.

Let’s run the application and see if the application can be run inside your web browser. See Figure 16-9.

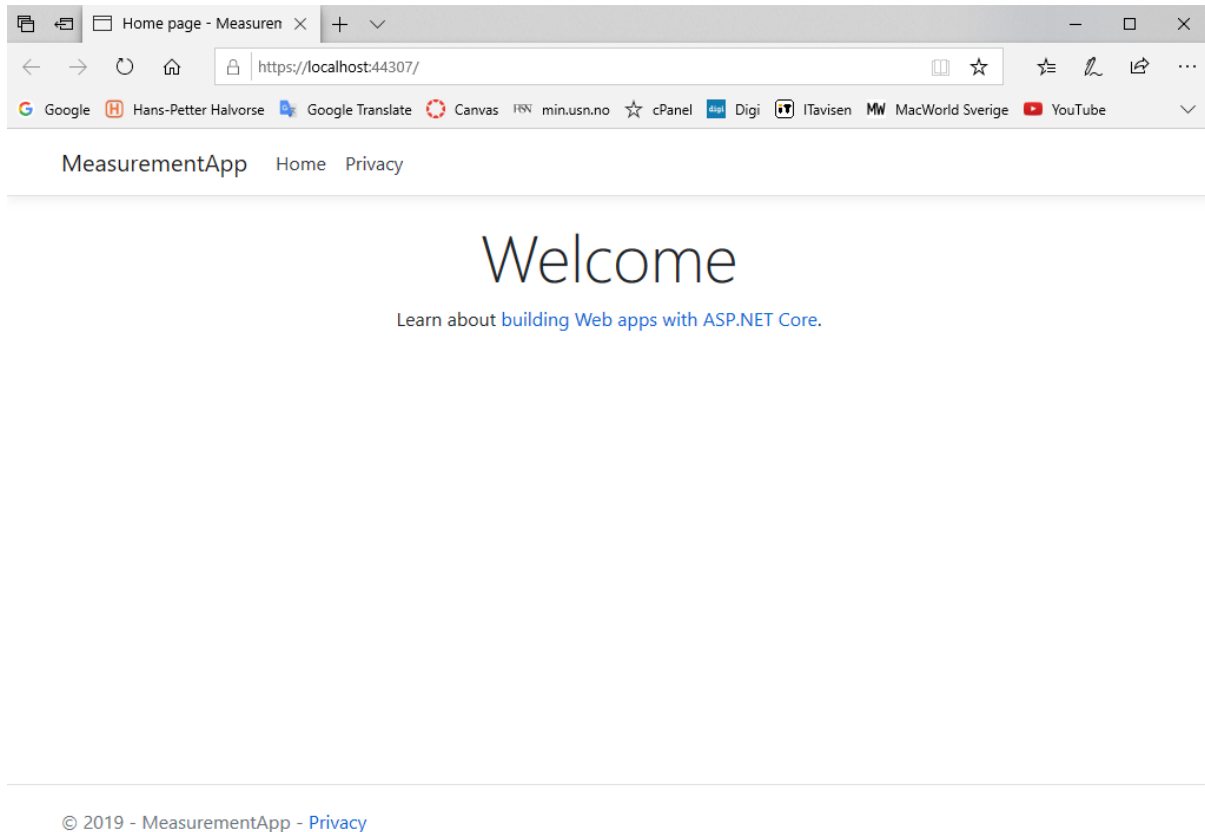


Figure 16-9: Running your Application in the Web Browser

Now we need to start creating the necessary code for our application.

Let's start to create a **Models** folder where we are making our Class that do the retrieving of data from the database. See Figure 16-10. (In the example I have written Model instead of Models, bit that's the same)

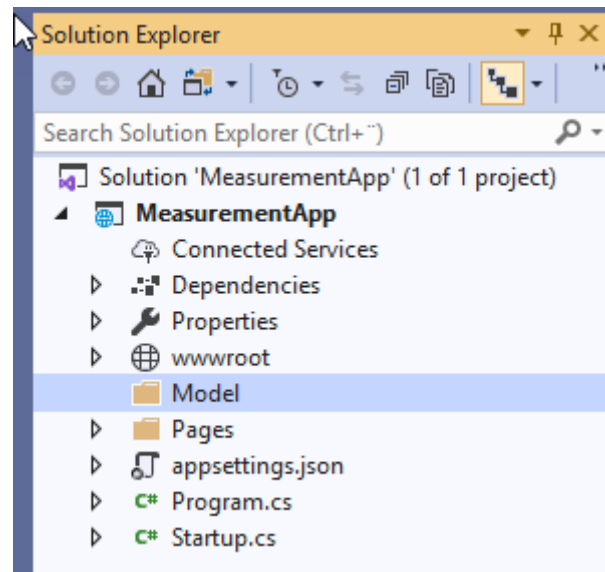


Figure 16-10: Solution Explorer

Then we create a Class called, e.g., “**Measurements.cs**” in the **Models (Model)** folder. See Figure 16-11.

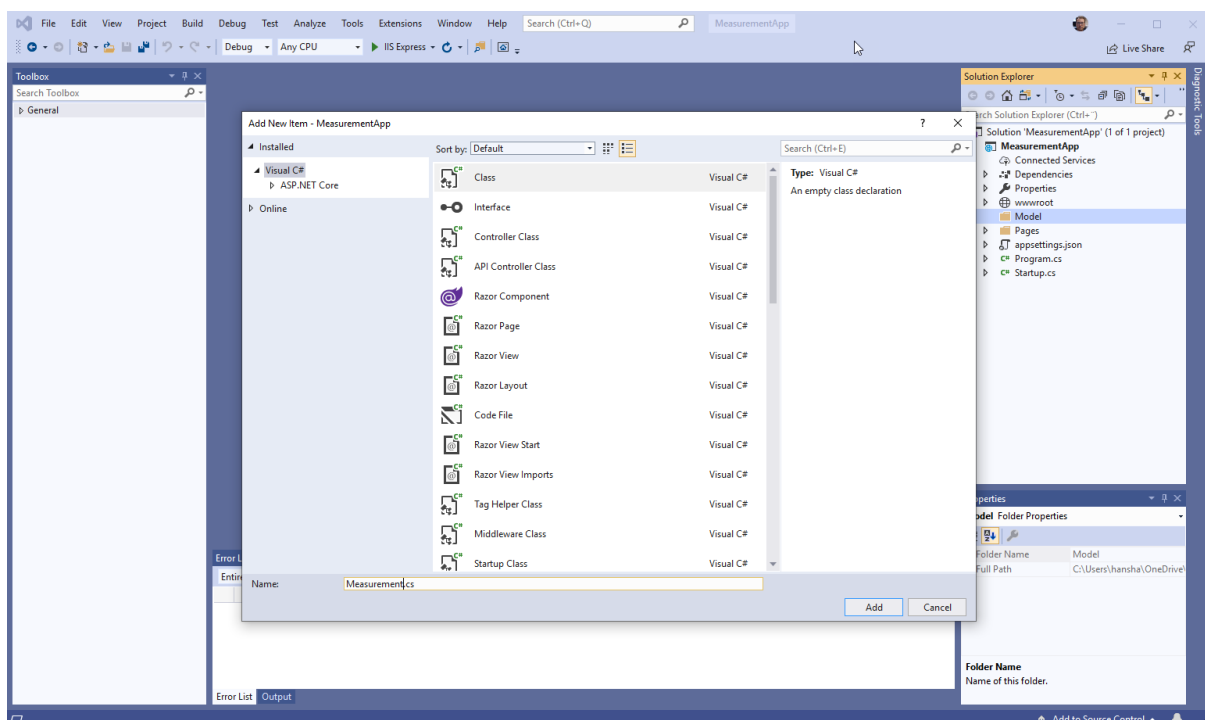


Figure 16-11: Create a new Class

Write in the following code:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Data.SqlClient;
4
5
6  namespace MeasurementApp.Model
7  {
8      public class Measurement
9      {
10
11         public int MeasurementId { get; set; }
12         public string MeasurementName { get; set; }
13         public string MeasurementUnit { get; set; }
14
15
16         public List<Measurement> GetMeasurementParameters()
17         {
18
19             List<Measurement> measurementParameterList = new List<Measurement>();
20
21             string connectionString = "DATA SOURCE=dmsserver.database.windows.net;UID=WeatherLogin;PWD=WeatherSystemUSN2019;DATABASE=WE
22
23
24             SqlConnection con = new SqlConnection(connectionString);
25
26             string sqlQuery = "select MeasurementId, MeasurementName, Unit from MEASUREMENT";
27
28             con.Open();
29
30             SqlCommand cmd = new SqlCommand(sqlQuery, con);
31
32             SqlDataReader dr = cmd.ExecuteReader();
33
34             if (dr != null)
35             {
36                 while (dr.Read())
37

```

Figure 16-12: Write Code for the Class

Complete Code Listing:

```

using System;
using System.Collections.Generic;
using System.Data.SqlClient;

namespace MeasurementApp.Model
{
    public class Measurement
    {
        public int MeasurementId { get; set; }
        public string MeasurementName { get; set; }
        public string MeasurementUnit { get; set; }

        public List<Measurement> GetMeasurementParameters()
        {
            List<Measurement> measurementParameterList = new List<Measurement>();

            string connectionString = "DATA
SOURCE=xxx;UID=sa;PWD=xxx;DATABASE=MEASUREMENTDB";

            SqlConnection con = new SqlConnection(connectionString);

            string sqlQuery = "select MeasurementId, MeasurementName, Unit from
MEASUREMENT";

            con.Open();

            SqlCommand cmd = new SqlCommand(sqlQuery, con);

            SqlDataReader dr = cmd.ExecuteReader();

```



```

        if (dr != null)
        {
            while (dr.Read())
            {
                Measurement measurementParameter = new Measurement();

                measurementParameter.MeasurementId =
Convert.ToInt32(dr["MeasurementId"]);
                measurementParameter.MeasurementName =
dr["MeasurementName"].ToString();
                measurementParameter.MeasurementUnit = dr["Unit"].ToString();

                measurementParameterList.Add(measurementParameter);
            }
        }
        return measurementParameterList;
    }
}

```

As you see the connection string to the database is hardcoded inside the “Measurement” class:

```
string connectionString = "DATA SOURCE=xxx;UID=sa;PWD=xxx;DATABASE=MEASUREMENTDB";
```

Just replace the “xxx” with the settings for your database.

Make sure to install the necessary NuGet package(s). See Figure 16-13. We need the System.Data.SqlClient.

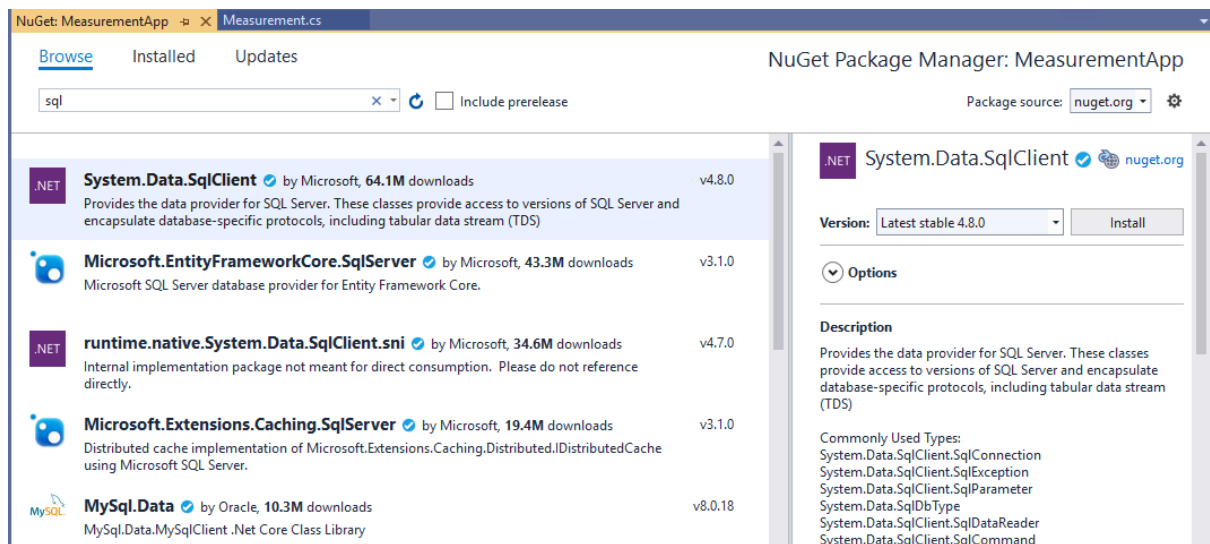


Figure 16-13: Make sure the necessary NuGet packages are installed

Then we create our Razor File in the “Pages” folder. Let’s name the file “Measurement.cshtml”. Make sure to select Razor Page (a Razor Page with a Page Model). See Figure 16-14.

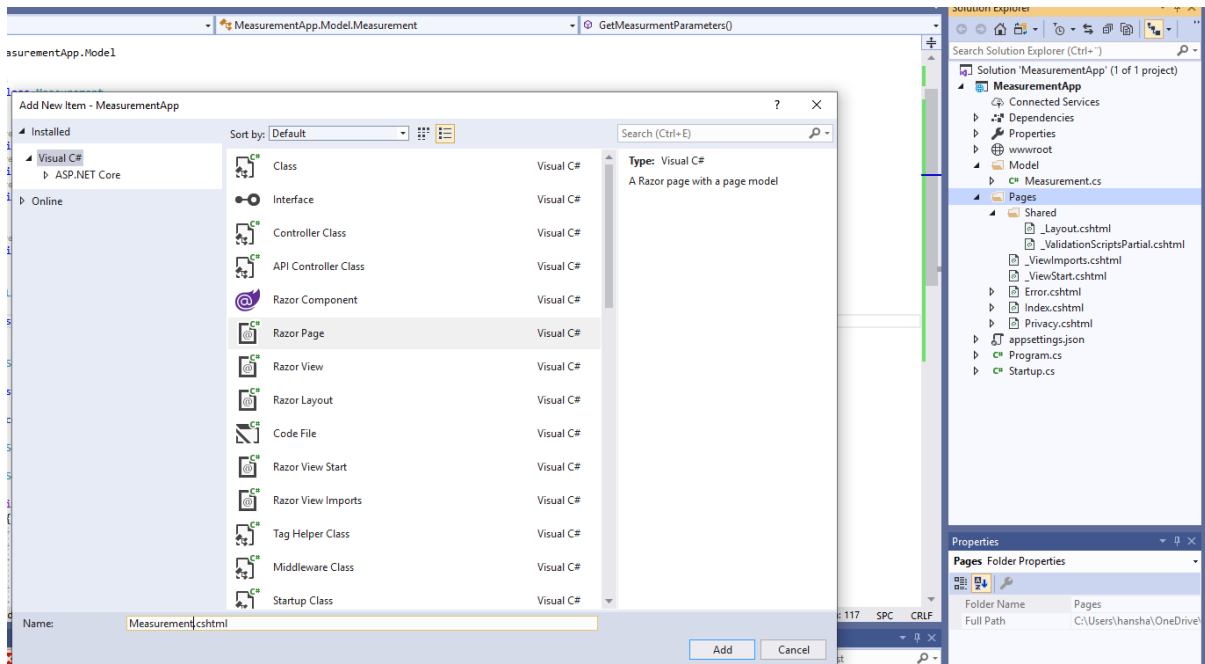


Figure 16-14: Create a Razor Page

A code behind file, also called the Page Model will also be created automatically. See Figure 16-15.

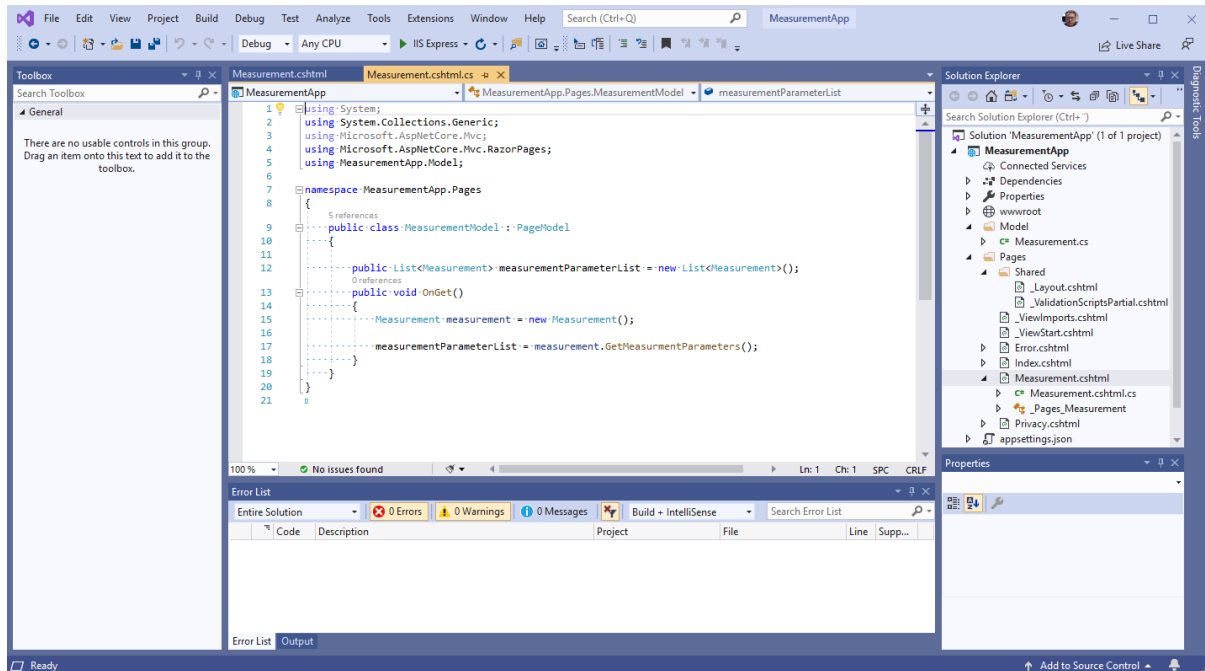


Figure 16-15: The Razor Page Model (Code behind File)

Below you see the whole contents of the “Measurement.cshtml.cs” file:

```

using System;
using System.Collections.Generic;

```

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using MeasurementApp.Model;

namespace MeasurementApp.Pages
{
    public class MeasurementModel : PageModel
    {
        public List<Measurement> measurementParameterList = new
List<Measurement>();

        public void OnGet()
        {
            Measurement measurement = new Measurement();

            measurementParameterList = measurement.GetMeasurementParameters();
        }
    }
}

```

Then we can make the contents of the “Measurement.cshtml” file. See Figure 16-16.

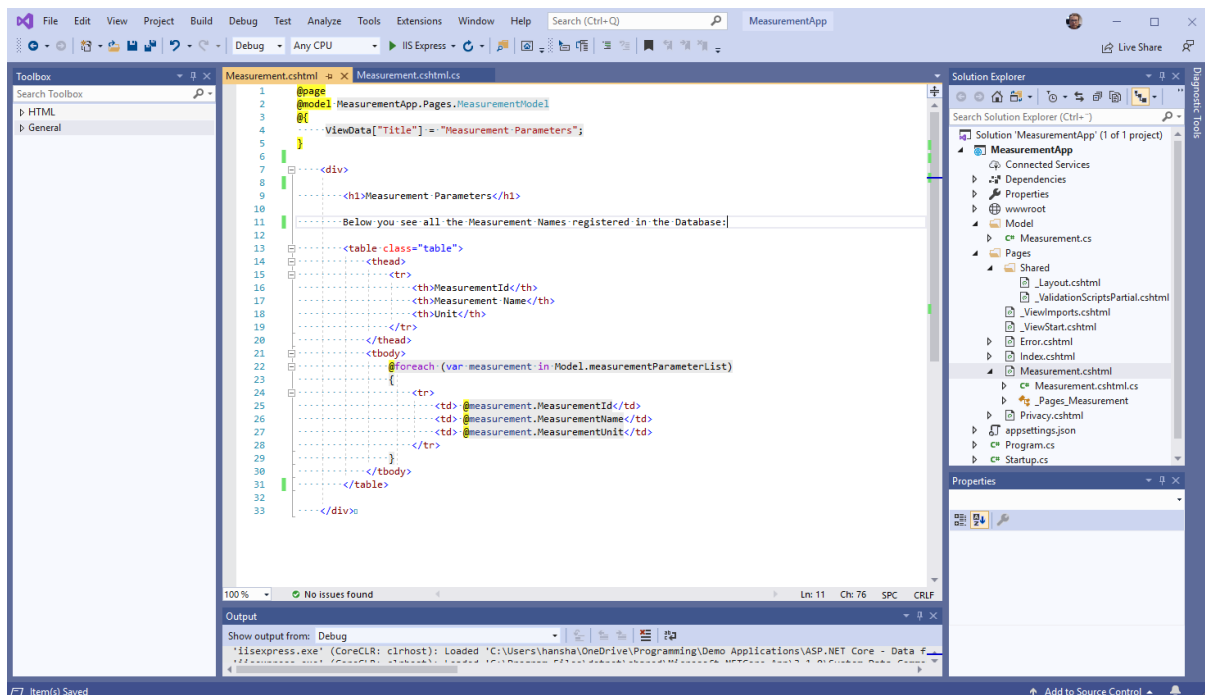


Figure 16-16: Creating the Contents in the Razor Page

Below you see the “Measurement.cshtml” file:

```

@page
@model MeasurementApp.Pages.MeasurementModel
@{
    ViewData["Title"] = "Measurement Parameters";
}

<div>

```

```
<h1>Measurement Parameters</h1>

Below you see all the Measurement Names registered in the Database:

<table class="table">
  <thead>
    <tr>
      <th>MeasurementId</th>
      <th>Measurement Name</th>
      <th>Unit</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var measurement in Model.measurementParameterList)
    {
      <tr>
        <td> @measurement.MeasurementId</td>
        <td> @measurement.MeasurementName</td>
        <td> @measurement.MeasurementUnit</td>
      </tr>
    }
  </tbody>
</table>

</div>
```

@xxx is the Razor code. The Razor code is executed on the server before the web page is sent to the client (web browser).

We use a “foreach” to create the contents inside a HTML table.

The “Model.” variable is used to retrieve data from the Page Model file (“Measurement.cshtml.cs”). All public variables that are created in the Measurement.cshtml.cs file are available in the Measurement.cshtml file by using @Model.<variablename>.

Now, our application should be finished. Let’s run the application in our web browser. See Figure 16-17.

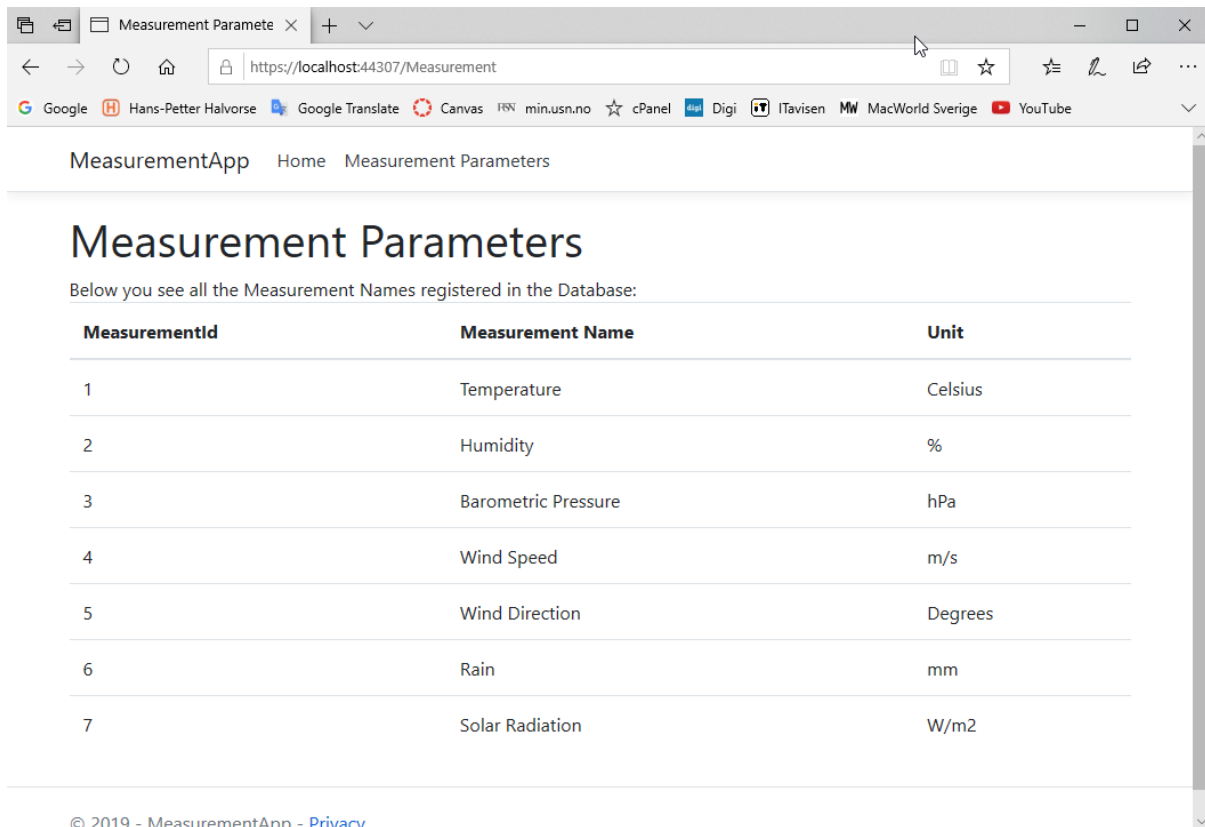


Figure 16-17: Running the Final Application in the Web Browser

16.2 Where should we put the Connection String?

As you see in the example above the connection string to the database is hardcoded inside the "Measurement" class:

```
string connectionString = "DATA SOURCE=xxx;UID=sa;PWD=xxx;DATABASE=MEASUREMENTDB";
```

where you replaced the "xxx" with the settings for your database.

The user "sa" (System Administrator) is the default user. You can use that one for testing, but for your final application you should setup and use another user.

16.2.1 appSettings.json

A better solution is to put the connection string inside the "appSettings.json" which is meant for storing configuration data, such as connection strings, etc.

Let us start by putting the connection string into the "appSettings.json" file:

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "ConnectionString": "DATA SOURCE=xxx;UID=sa;PWD=xxx;DATABASE=MEASUREMENTDB "
  }
}

```

where you replace the “xxx” with the settings for your database.

Next, we update the “Measurements.cs” class:

```

using System;
using System.Collections.Generic;
using System.Data.SqlClient;

namespace MeasurementApp.Model
{
    public class Measurement
    {
        public int MeasurementId { get; set; }
        public string MeasurementName { get; set; }
        public string MeasurementUnit { get; set; }

        public List<Measurement> GetMeasurmentParameters(string connectionString)
        {
            List<Measurement> measurementParameterList = new List<Measurement>();

            SqlConnection con = new SqlConnection(connectionString);

            string sqlQuery = "select MeasurementId, MeasurementName, Unit from
MEASUREMENT";

            con.Open();

            SqlCommand cmd = new SqlCommand(sqlQuery, con);

            SqlDataReader dr = cmd.ExecuteReader();

            if (dr != null)
            {
                while (dr.Read())
                {
                    Measurement measurmentParameter = new Measurement();

                    measurmentParameter.MeasurementId =
Convert.ToInt32(dr["MeasurementId"]);
                    measurmentParameter.MeasurementName =
dr["MeasurementName"].ToString();
                    measurmentParameter.MeasurementUnit = dr["Unit"].ToString();

                    measurementParameterList.Add(measurmentParameter);
                }
            }
            return measurementParameterList;
        }
    }
}

```

```
}

```

Then we need to add something to the “Startup.cs” file:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRazorPages();

    services.AddSingleton<IConfiguration>(Configuration);
}

```

We have added:

```
services.AddSingleton<IConfiguration>(Configuration);

```

Finally, we need to update the “Measurement.cshtml.cs” file:

```
using System;
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Configuration;
using MeasurementApp.Model;

namespace MeasurementApp.Pages
{
    public class MeasurementModel : PageModel
    {
        readonly IConfiguration _configuration;

        public List<Measurement> measurementParameterList = new
List<Measurement>();

        public string connectionString;

        public MeasurementModel(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void OnGet()
        {
            Measurement measurement = new Measurement();

            connectionString =
_configuration.GetConnectionString("ConnectionString");

            measurementParameterList =
measurement.GetMeasurementParameters(connectionString);
        }
    }
}

```

Now we can run the application. The result should be the same as before, see Figure 16-17.

17 CRUD Applications

CRUS is short for:

Create
Read
Update
Delete

The acronym CRUD refers to all the major functions that are implemented for communication with a database.

Operation	SQL	HTTP/REST API
Create	INSERT	POST
Read	SELECT	GET
Update	UPDATE	PUT
Delete	DEKETE	DELETE

We will show how we can use ASP.NET Core to get (read, retrieve, select) data from the database, insert data into the database, update the data inside the database and deleting data inside the database.

17.1 Demo Application

Let us create a Book Application with all the CRUD features included. Figure 17-1 and Figure 17-2 show the result.



ASP.NET Core - Database CRUD Application: <https://youtu.be/k5TCZDwTYcE>

The entire example can be downloaded from the home page of this textbook.

This example can be the foundation for all types of applications. All web applications typically show some data from the database in a list or table, then you will typically have features for add new data, edit/update existing data or delete data.

When I start on a new development project, I just use this application as a template or as a foundation for my new web application.

BookApp Home Books

Book Store

Welcome to my Book Store



Figure 17-1: Final Book Application – Start Page

BookApp Home Books

Books

Below you see all the Books in the Book Store:

BookId	Title	ISBN	Publisher	Author	Category	Action
1	Introduction to Linear Algebra	0-07-066781-0	Prentice Hall	Gilbert Strang	Science	Delete Book
2	Modern Control System	1-08-890781-0	Wiley	Dorf Bishop	Programming	Delete Book
3	The Lord of the Rings	2-09-066556-2	McGraw-Hill	J.R.R Tolkien	Novel	Delete Book
4	Python	55555	Halvorsen	Hans-Petter	Programming	Delete Book

New Book

© 2019 - BookApp - Privacy

Figure 17-2: Final Book Application – Books

17.1.1 Create the Visual Studio Project

We start by creating a new ASP.NET Core Web application in Visual Studio. See Figure 17-3 and Figure 17-4

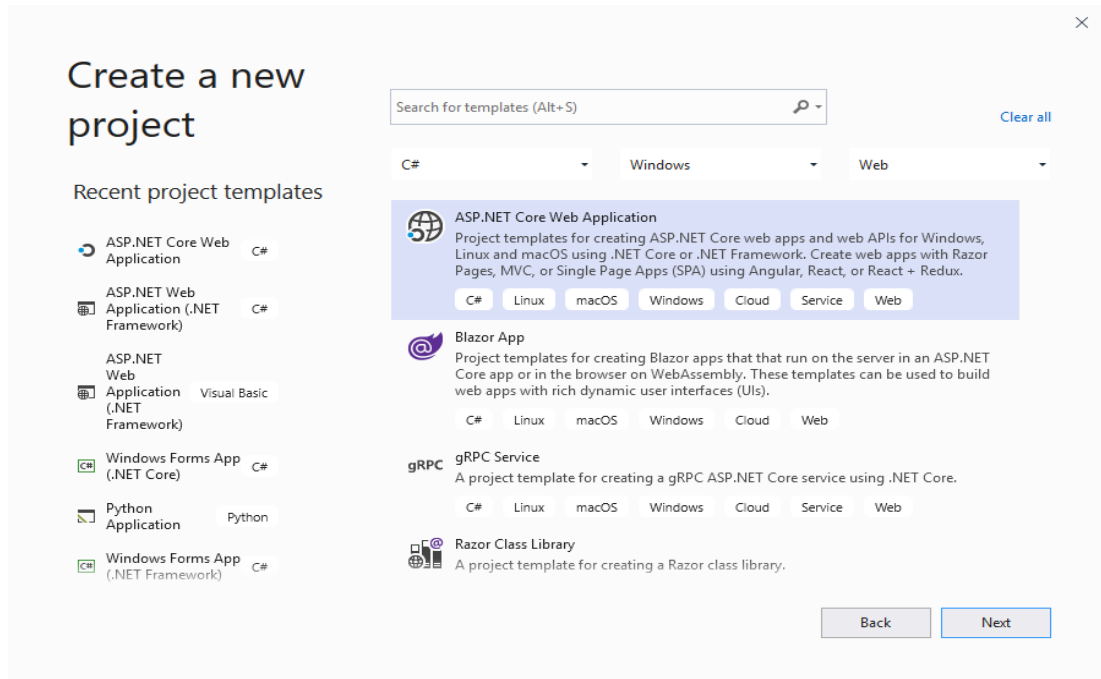


Figure 17-3: Create New Visual Studio Project

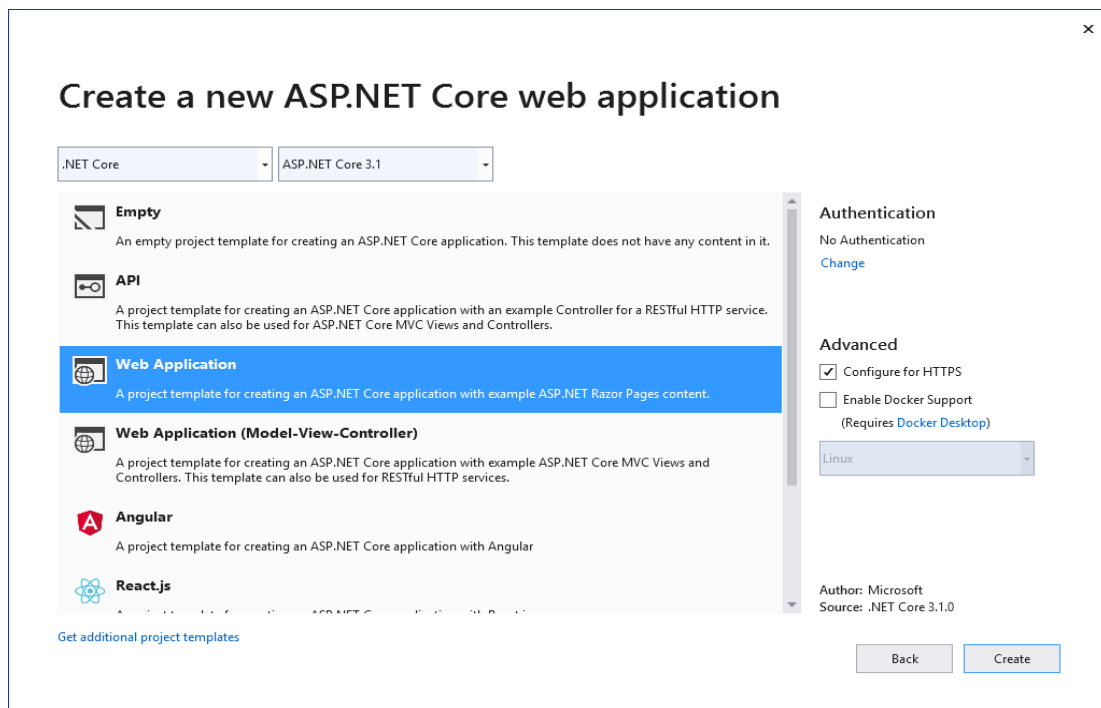


Figure 17-4: Create Web Application

The web application project becomes as shown in Figure 17-5-

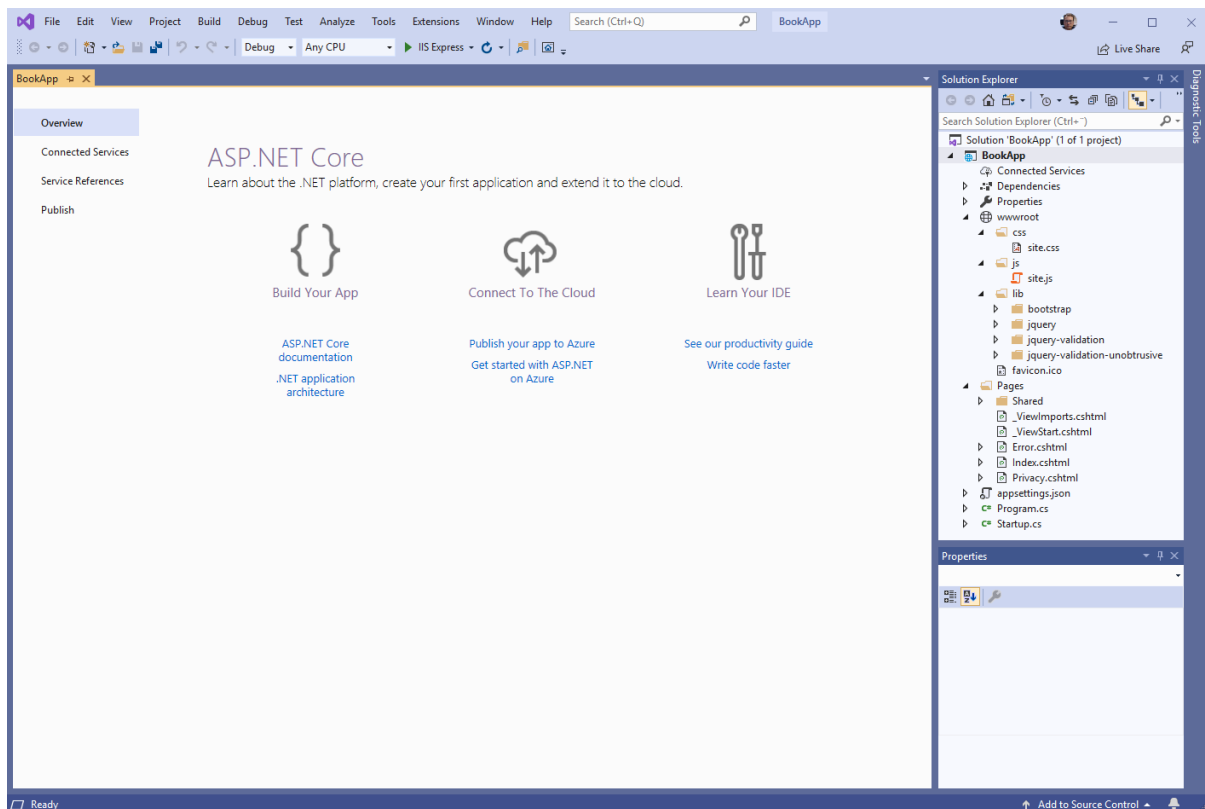


Figure 17-5: Created Project in Visual studio

We can run it without any modifications, see Figure 17-6.

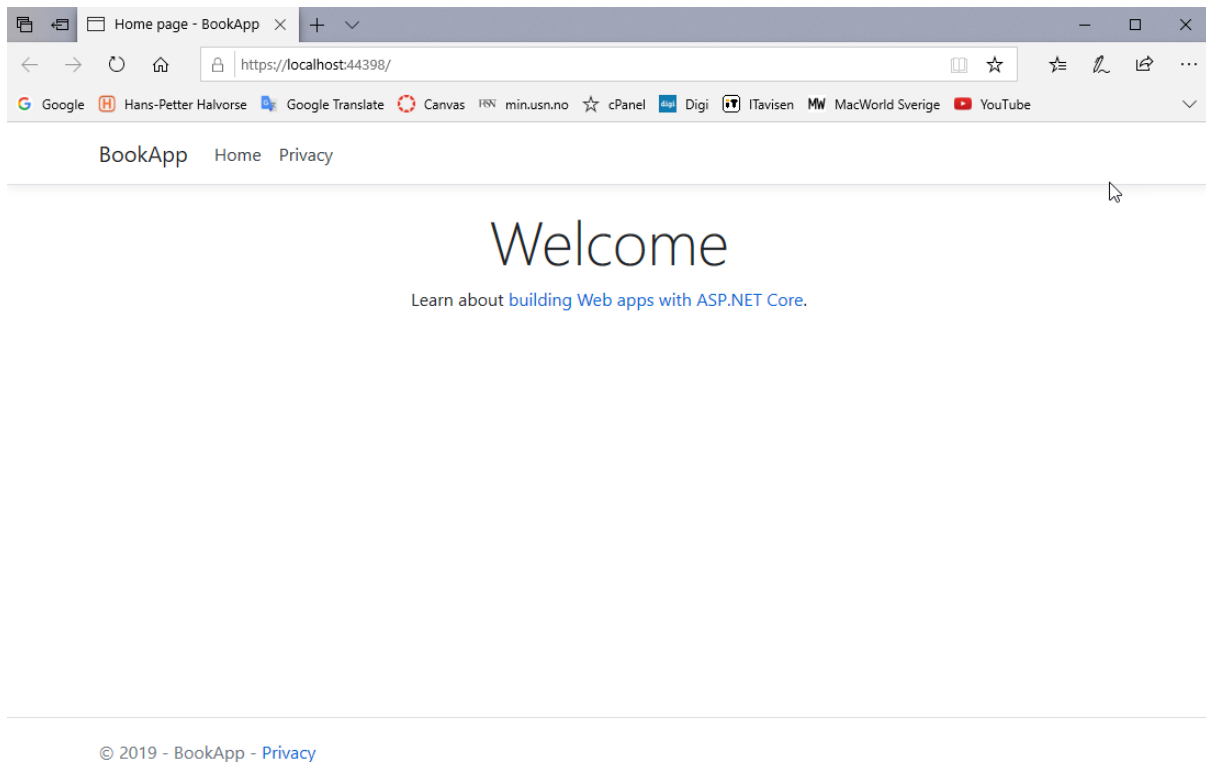


Figure 17-6: Initial Project Running in your Web Browser

17.1.2 Database

We will use the following tables:

AUTHOR:

```
if not exists (select * from dbo.sysobjects where id =
object_id(N'[AUTHOR]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [AUTHOR]
(
    [AuthorId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [AuthorName] [varchar](50) NOT NULL UNIQUE,
    [Address] [varchar](50) NULL,
    [Phone] [varchar](50) NULL,
    [PostCode] [varchar](50) NULL,
    [PostAddress] [varchar](50) NULL,
)
GO
```

PUBLISHER:

```
if not exists (select * from dbo.sysobjects where id =
object_id(N'[PUBLISHER]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [PUBLISHER]
(
    [PublisherId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
```

```

[PublisherName] [varchar](50) NOT NULL UNIQUE,
[Description] [varchar](1000) NULL,
[Address] [varchar](50) NULL,
[Phone] [varchar](50) NULL,
[PostCode] [varchar](50) NULL,
[PostAddress] [varchar](50) NULL,
[EMail] [varchar](50) NULL,
)
GO

```

CATEGORY:

```

if not exists (select * from dbo.sysobjects where id =
object_id(N'[CATEGORY]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [CATEGORY]
(
    [CategoryId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [CategoryName] [varchar](50) NOT NULL UNIQUE,
    [Description] [varchar](1000) NULL,
)
GO

```

BOOK:

```

if not exists (select * from dbo.sysobjects where id = object_id(N'[BOOK]') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
CREATE TABLE [BOOK]
(
    [BookId] [int] IDENTITY(1, 1) NOT NULL PRIMARY KEY,
    [Title] [varchar](50) NOT NULL UNIQUE,
    [ISBN] [varchar](20) NOT NULL,
    [PublisherId] [int] NOT NULL FOREIGN KEY REFERENCES [PUBLISHER] ([PublisherId]),
    [AuthorId] [int] NOT NULL FOREIGN KEY REFERENCES [AUTHOR] ([AuthorId]),
    [CategoryId] [int] NOT NULL FOREIGN KEY REFERENCES [CATEGORY] ([CategoryId]),
    [Description] [varchar](1000) NULL,
    [Year] [date] NULL,
    [Edition] [int] NULL,
    [AverageRating] [float] NULL,
)
GO

```

We insert the tables using SQL Server Management Studio:

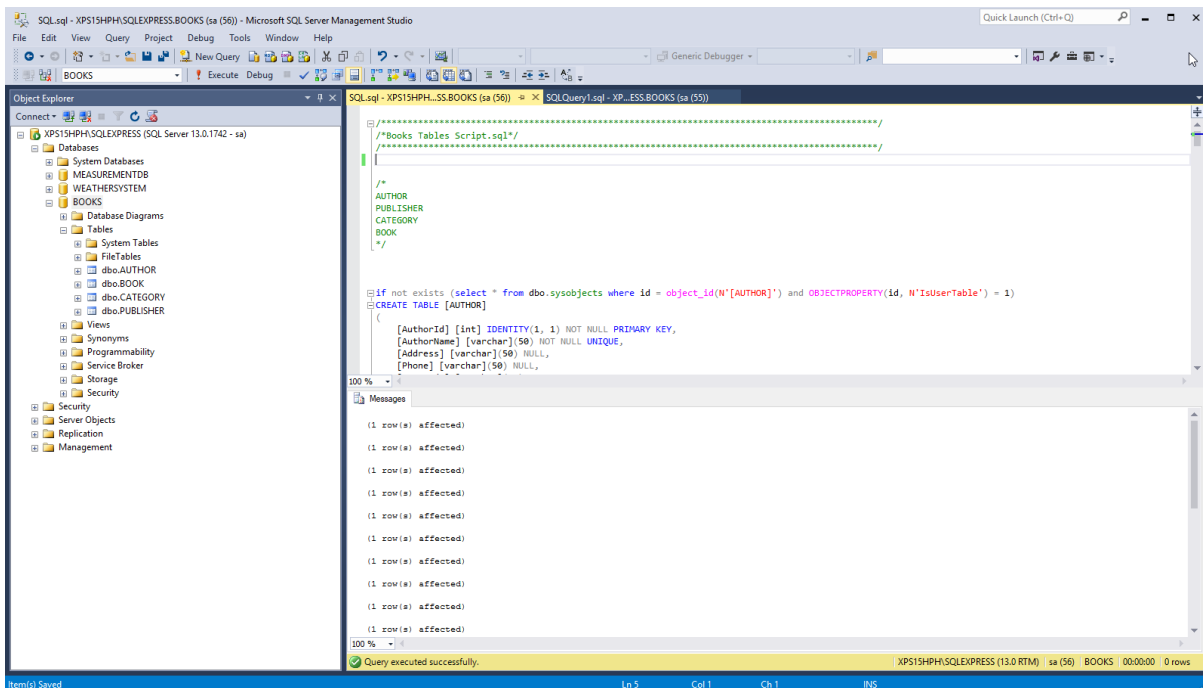


Figure 17-7: Inserting Tables using SQL Server Management Studio

We also need some Views and Stored Procedures.

“GetBookData” View:

```

CREATE VIEW GetBookData
AS

SELECT
BOOK.BookId,
BOOK.Title,
BOOK.ISBN,
PUBLISHER.PublisherName,
AUTHOR.AuthorName,
CATEGORY.CategoryName

FROM BOOK
INNER JOIN AUTHOR ON BOOK.AuthorId = AUTHOR.AuthorId
INNER JOIN PUBLISHER ON BOOK.PublisherId = PUBLISHER.PublisherId
INNER JOIN CATEGORY ON BOOK.CategoryId = CATEGORY.CategoryId

GO

```

“CreateBook” Stored Procedure:

```

CREATE PROCEDURE CreateBook
@Title varchar(50),
@Isbn varchar(20),
@PublisherName varchar(50),
@AuthorName varchar(50),
@CategoryName varchar(50)
AS

```

```

if not exists (select * from CATEGORY where CategoryName = @CategoryName)
INSERT INTO CATEGORY (CategoryName) VALUES (@CategoryName)

if not exists (select * from AUTHOR where AuthorName = @AuthorName)
INSERT INTO AUTHOR (AuthorName) VALUES (@AuthorName)

if not exists (select * from PUBLISHER where PublisherName = @PublisherName)
INSERT INTO PUBLISHER (PublisherName) VALUES (@PublisherName)

if not exists (select * from BOOK where Title = @Title)
INSERT INTO BOOK (Title, ISBN, PublisherId, AuthorId, CategoryId)
VALUES
(
@Title,
@ISBN,
(select PublisherId from PUBLISHER where PublisherName=@PublisherName),
(select AuthorId from AUTHOR where AuthorName=@AuthorName),
(select CategoryId from CATEGORY where CategoryName=@CategoryName)
)
GO

```

“UpdateBook” Stored Procedure:

```

CREATE PROCEDURE UpdateBook
@BookId int,
@Title varchar(50),
@ISBN varchar(20),
@PublisherName varchar(50),
@AuthorName varchar(50),
@CategoryName varchar(50)
AS

if not exists (select * from CATEGORY where CategoryName = @CategoryName)
INSERT INTO CATEGORY (CategoryName) VALUES (@CategoryName)

if not exists (select * from AUTHOR where AuthorName = @AuthorName)
INSERT INTO AUTHOR (AuthorName) VALUES (@AuthorName)

if not exists (select * from PUBLISHER where PublisherName = @PublisherName)
INSERT INTO PUBLISHER (PublisherName) VALUES (@PublisherName)

UPDATE BOOK SET
Title = @Title,
ISBN = @ISBN,
PublisherId = (select PublisherId from PUBLISHER where
PublisherName=@PublisherName),
AuthorId = (select AuthorId from AUTHOR where AuthorName=@AuthorName),
CategoryId = (select CategoryId from CATEGORY where CategoryName=@CategoryName)
WHERE BookId = @BookId
GO

```

“DeleteBook” Stored Procedure:

```

CREATE PROCEDURE DeleteBook
@BookId int
AS

```

```
delete from BOOK where BookId=@BookId  
GO
```

We insert the Stored Procedures using SQL Server Management Studio in the same way as for the tables.

17.1.3 Index (Start Page)

We start by creating the start page of our application (Index.cshtml). See Figure 17-8

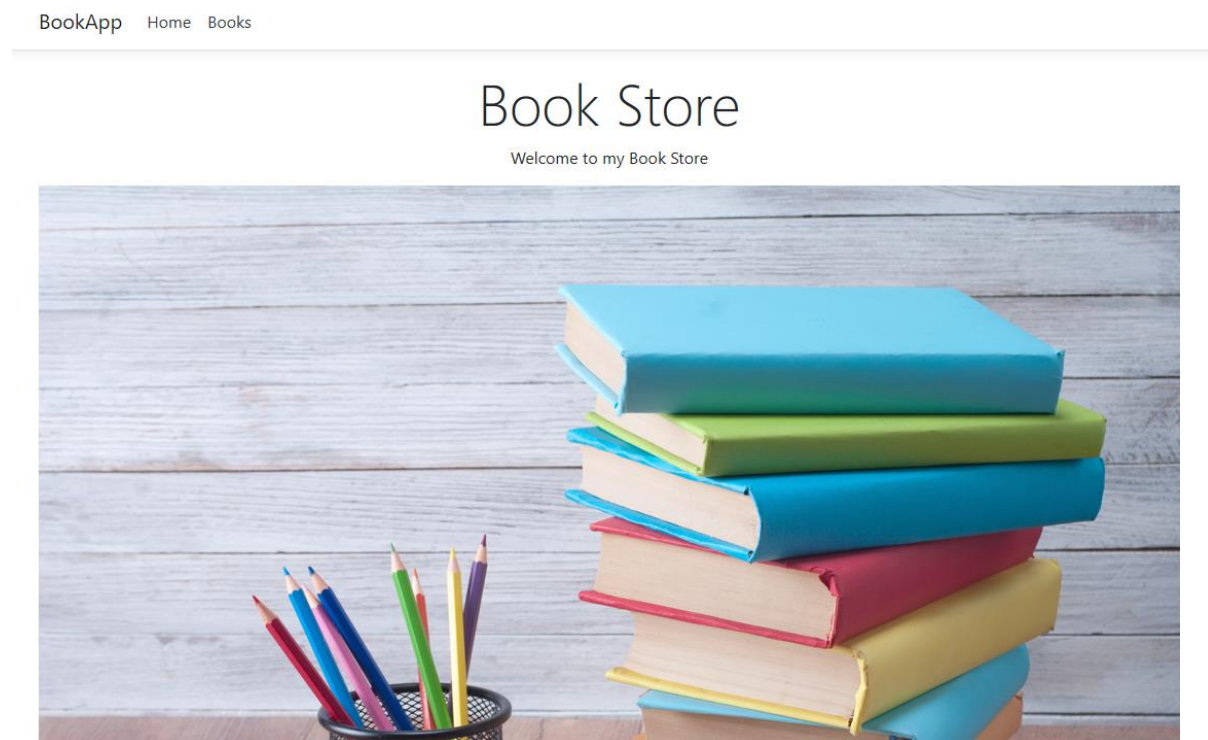


Figure 17-8: Start Page - Index

17.1.4 Models

Then we create our models or classes.

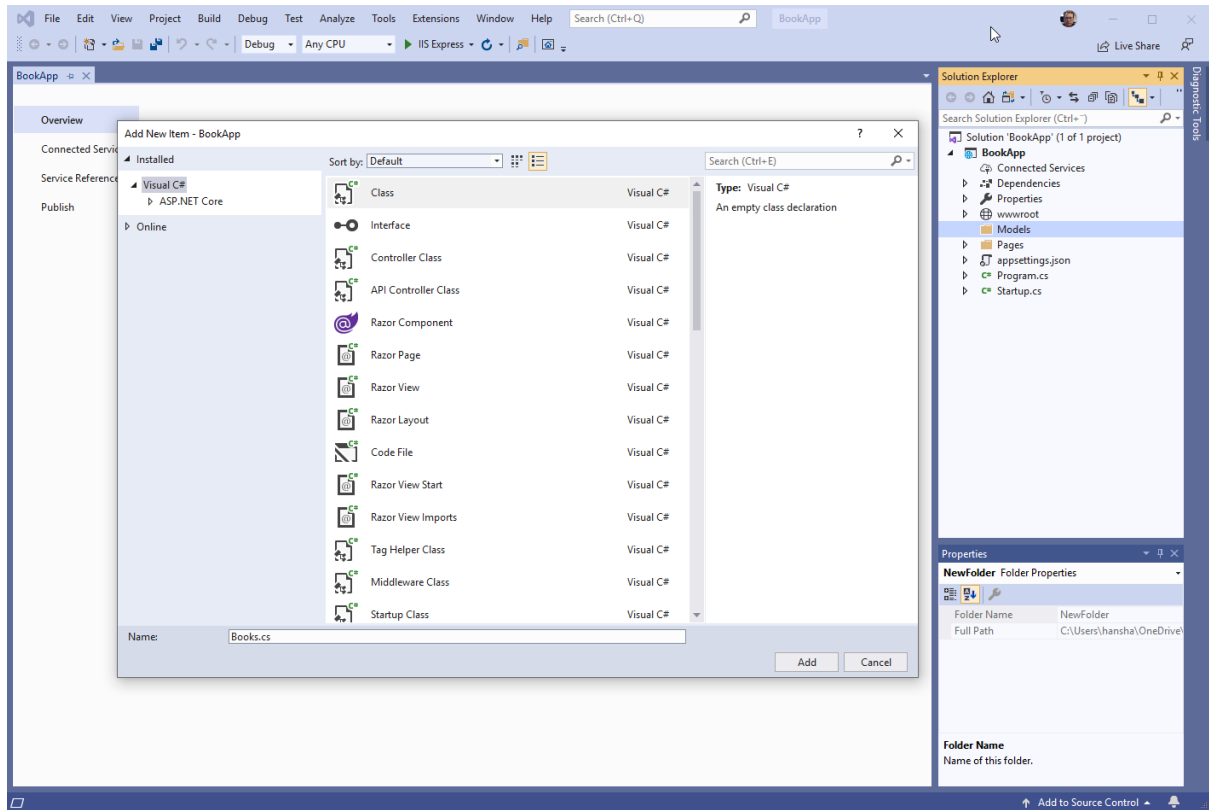


Figure 17-9: Creating the Models

Code:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Data;
4  using System.Data.SqlClient;
5
6  namespace BookApp.Models
7  {
8      public class Book
9      {
10         public int BookId { get; set; }
11         public string Title { get; set; }
12         public string ISBN { get; set; }
13         public string PublisherName { get; set; }
14         public string AuthorName { get; set; }
15         public string CategoryName { get; set; }
16
17         public List<Book> GetBooks(string connectionString) ...
18
19         public Book GetBookData(string connectionString, int bookId) ...
20
21         public void CreateBook(string connectionString, Book book) ...
22
23         public void EditBook(string connectionString, Book book) ...
24
25         public void DeleteBook(string connectionString, int bookId) ...
26     }
27 }

```

Figure 17-10: Book Class

Book Class:

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;

namespace BookApp.Models
{
    public class Book
    {
        public int BookId { get; set; }
        public string Title { get; set; }
        public string ISBN { get; set; }
        public string PublisherName { get; set; }
        public string AuthorName { get; set; }
        public string CategoryName { get; set; }

        ...

        public List<Book> GetBooks(string connectionString) {}
    }
}

```

```

public Book GetBookData(string connectionString, int bookId){}
public void CreateBook(string connectionString, Book book){}
public void EditBook(string connectionString, Book book){}
public void DeleteBook(string connectionString, int bookId){}

}
}

```

GetBooks Method:

```

public List<Book> GetBooks(string connectionString)
{
    List<Book> bookList = new List<Book>();
    SqlConnection con = new SqlConnection(connectionString);

    string selectSQL = "select BookId, Title, Isbn, PublisherName,
                        AuthorName, CategoryName from GetBookData";

    con.Open();

    SqlCommand cmd = new SqlCommand(selectSQL, con);

    SqlDataReader dr = cmd.ExecuteReader();

    if (dr != null)
    {
        while (dr.Read())
        {
            Book book = new Book();

            book.BookId = Convert.ToInt32(dr["BookId"]);
            book.Title = dr["Title"].ToString();
            book.Isbn = dr["ISBN"].ToString();
            book.PublisherName = dr["PublisherName"].ToString();
            book.AuthorName = dr["AuthorName"].ToString();
            book.CategoryName = dr["CategoryName"].ToString();

            bookList.Add(book);
        }
    }
    return bookList;
}

```

GetBookData Method:

```

public Book GetBookData(string connectionString, int bookId)
{
    SqlConnection con = new SqlConnection(connectionString);

    string selectSQL = "select BookId, Title, Isbn, PublisherName,
                        AuthorName, CategoryName
                        from GetBookData where BookId = " + bookId;

    con.Open();

    SqlCommand cmd = new SqlCommand(selectSQL, con);

    SqlDataReader dr = cmd.ExecuteReader();

    Book book = new Book();

```

```

if (dr != null)
{
    while (dr.Read())
    {
        book.BookId = Convert.ToInt32(dr["BookId"]);
        book.Title = dr["Title"].ToString();
        book.Isbn = dr["ISBN"].ToString();
        book.PublisherName = dr["PublisherName"].ToString();
        book.AuthorName = dr["AuthorName"].ToString();
        book.CategoryName = dr["CategoryName"].ToString();
    }
}
return book;
}

```

CreateBook Method:

```

public void CreateBook(string connectionString, Book book)
{
    try
    {
        using (SqlConnection con = new
                SqlConnection(connectionString))
        {
            SqlCommand cmd = new SqlCommand("CreateBook", con);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.Add(new SqlParameter("@Title", book.Title));
            cmd.Parameters.Add(new SqlParameter("@Isbn", book.Isbn));
            cmd.Parameters.Add(new SqlParameter("@PublisherName",
                book.PublisherName));
            cmd.Parameters.Add(new SqlParameter("@AuthorName",
                book.AuthorName));
            cmd.Parameters.Add(new SqlParameter("@CategoryName",
                book.CategoryName));

            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}

```

EditBook Method:

```

public void EditBook(string connectionString, Book book)
{
    try
    {
        using (SqlConnection con = new
                SqlConnection(connectionString))
        {
            SqlCommand cmd = new SqlCommand("UpdateBook", con);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.Add(new SqlParameter("@BookId", book.BookId));

```

```
cmd.Parameters.Add(new SqlParameter("@Title", book.Title));
cmd.Parameters.Add(new SqlParameter("@Isbn", book.Isbn));
cmd.Parameters.Add(new SqlParameter("@PublisherName",
    book.PublisherName));
cmd.Parameters.Add(new SqlParameter("@AuthorName",
    book.AuthorName));
cmd.Parameters.Add(new SqlParameter("@CategoryName",
    book.CategoryName));

con.Open();
cmd.ExecuteNonQuery();
con.Close();
}
}
catch (Exception ex)
{
    throw ex;
}
}
```

DeleteBook Method:

```
public void DeleteBook(string connectionString, int bookId)
{
    try
    {
        using (SqlConnection con = new
            SqlConnection(connectionString))
        {
            SqlCommand cmd = new SqlCommand("DeleteBook", con);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.Add(new SqlParameter("@BookId", bookId));
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

NuGet Packages:

We need to install the System.Data.SqlClient.

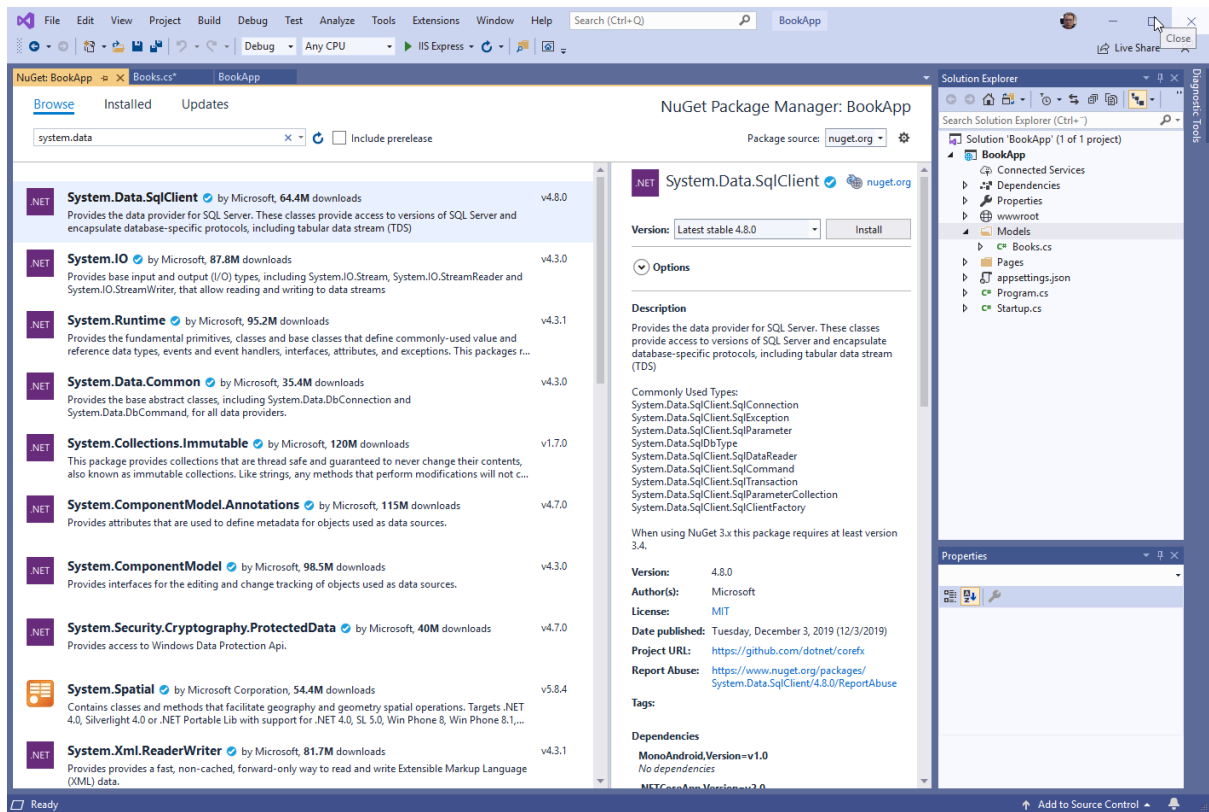


Figure 17-11: Install NuGet Packages

17.1.5 Show Books

We start creating our web pages.

BookApp Home Books

Books

Below you see all the Books in the Book Store:

BookId	Title	ISBN	Publisher	Author	Category	Action
1	Introduction to Linear Algebra	0-07-066781-0	Prentice Hall	Gilbert Strang	Science	Delete Book
2	Modern Control System	1-08-890781-0	Wiley	Dorf Bishop	Programming	Delete Book
3	The Lord of the Rings	2-09-066556-2	McGraw-Hill	J.R.R Tolkien	Novel	Delete Book
4	Python	55555	Halvorsen	Hans-Petter	Programming	Delete Book

[New Book](#)

© 2019 - BookApp - [Privacy](#)

Figure 17-12: Show Lists of Books stored in the Database

Books.cshtml:

```
@page
@model BookApp.Pages.BooksModel
@{
    ViewData["Title"] = "Books";
}

<div>

    <h1>Books</h1>

    Below you see all the Books in the Book Store:<br />

    <table class="table">
        <thead>
            <tr>
                <th>BookId</th>
                <th>Title</th>
                <th>ISBN</th>
                <th>Publisher</th>
                <th>Author</th>
                <th>Category</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var book in Model.books)
            {
                <tr>
                    <td> @book.BookId </td>

                    <td>
                        <a href="."/EditBook?bookid=@book.BookId">
                            @book.Title
                        </a> </td>

                    <td> @book.Isbn </td>
            }
        </tbody>
    </table>
```

```

        <td> @book.PublisherName </td>
        <td> @book.AuthorName </td>
        <td> @book.CategoryName </td>
        <td><a href="./DeleteBook?bookid=@book.BookId"
            class="btn btn-danger" role="button">Delete
            Book</a> </td>
    </tr>
    }
</tbody>
</table>

    <a href="./NewBook" class="btn btn-info" role="button">New Book</a>

</div>

```

Books.cshtml.cs:

```

...
using BookApp.Models;

namespace BookApp.Pages
{
    public class BooksModel : PageModel
    {
        readonly IConfiguration _configuration;

        public List<Book> books = new List<Book>();

        string connectionString;

        public BooksModel(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void OnGet()
        {
            books = GetBookList();
        }

        private List<Book> GetBookList()
        {
            connectionString =
                _configuration.GetConnectionString("ConnectionString");

            List<Book> bookList = new List<Book>();

            Book book = new Book();

            bookList = book.GetBooks(connectionString);

            return bookList;
        }
    }
}

```

17.1.6 New Book

We create the «New Book» page.

BookApp Home Books

New Book

Title:

ISBN:

Publisher:

Author:

Category:

© 2019 - BookApp - [Privacy](#)

Figure 17-13: New Book

NewBook.cshtml:

```
@page
@model BookApp.Pages.NewBookModel
@{
    ViewData["Title"] = "New Book";
}

<div>

    <h1>New Book</h1>

    <form name="bookForm" id="bookForm" method="post">

        Title:
        <br />
        <input name="bookTitle" type="text" class="form-control
            input-lg" autofocus required />
        <br />

        ISBN:
        <br />
        <input name="bookIsbn" type="text" class="form-control
            input-lg" required />
        <br />

        Publisher:
        <br />
        <input name="bookPublisher" type="text" class="form-control
            input-lg" required />
        <br />

        Author:
        <br />
```

```



```

NewBook.cshtml.cs:

```

...
using BookApp.Models;

namespace BookApp.Pages
{
    public class NewBookModel : PageModel
    {
        readonly IConfiguration _configuration;

        public string connectionString;

        public NewBookModel(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void OnGet()
        {
        }

        public void OnPost()
        {
            Book book = new Book();

            book.Title = Request.Form["bookTitle"];
            book.Isbn = Request.Form["bookIsbn"];
            book.PublisherName = Request.Form["bookPublisher"];
            book.AuthorName = Request.Form["bookAuthor"];
            book.CategoryName = Request.Form["bookCategory"];

            connectionString =
                _configuration.GetConnectionString("ConnectionString");

            book.CreateBook(connectionString, book);

            Response.Redirect("./Books");
        }
    }
}

```

17.1.7 Edit Book

We create the «Edit Book» page.

BookApp Home Books

Edit Book

Title:

ISBN:

Publisher:

Author:

Category:

© 2019 - BookApp - [Privacy](#)

Figure 17-14: Edit Book

EditBook.cshtml:

```
@page
@model BookApp.Pages.EditBookModel
@{
    ViewData["Title"] = "Edit Book";
}

<div>

<h1>Edit Book</h1>

<form name="bookForm" id="bookForm" method="post">

    <input name="bookId" type="text"
        value="@Model.bookdb.BookId" hidden/>

    Title:
    <br />
    <input name="bookTitle" type="text"
        value="@Model.bookdb.Title" class="form-control input-lg"
        autofocus required />
    <br />

    ISBN:
    <br />
    <input name="bookIsbn" type="text"
        value="@Model.bookdb.Isbn" class="form-control input-lg"
        required />
    <br />

    Publisher:
    <br />
    <input name="bookPublisher" type="text"
        value="@Model.bookdb.PublisherName" class="form-control
```

```

        input-lg" required />
    <br />

    Author:
    <br />
    <input name="bookAuthor" type="text"
        value="@Model.bookdb.AuthorName" class="form-control input-
        lg" required />
    <br />

    Category:
    <br />
    <input name="bookCategory" type="text"
        value="@Model.bookdb.CategoryName" class="form-control
        input-lg" required />
    <br />

    <input id="saveButton" type="submit" value="Save" class="btn
        btn-info" />

</form>

</div>

```

EditBook.cshtml.cs:

```

...
using BookApp.Models;

namespace BookApp.Pages
{
    public class EditBookModel : PageModel
    {
        readonly IConfiguration _configuration;

        public Book bookdb = new Book();

        public string connectionString;

        public int bookId;

        public EditBookModel(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void OnGet()
        {
            bookId = Convert.ToInt16(Request.Query["bookid"]);

            Book book = new Book();

            connectionString =
                _configuration.GetConnectionString("ConnectionString");

            bookdb = book.GetBookData(connectionString, bookId);
        }

        public void OnPost()
        {
            Book book = new Book();

            book.BookId = Convert.ToInt16(Request.Form["bookId"]);
            book.Title = Request.Form["bookTitle"];
        }
    }
}

```

```

        book.Isbn = Request.Form["bookIsbn"];
        book.PublisherName = Request.Form["bookPublisher"];
        book.AuthorName = Request.Form["bookAuthor"];
        book.CategoryName = Request.Form["bookCategory"];

        connectionString =
            _configuration.GetConnectionString("ConnectionString");

        book.EditBook(connectionString, book);

        Response.Redirect("./Books");
    }
}

```

17.1.8 Delete Book

We add functionality for deleting existing books.

BookApp Home Books

Books

Below you see all the Books in the Book Store:

BookId	Title	ISBN	Publisher	Author	Category	Action
1	Introduction to Linear Algebra	0-07-066781-0	Prentice Hall	Gilbert Strang	Science	Delete Book
2	Modern Control System	1-08-890781-0	Wiley	Dorf Bishop	Programming	Delete Book
3	The Lord of the Rings	2-09-066556-2	McGraw-Hill	J.R.R Tolkien	Novel	Delete Book
4	Python	55555	Halvorsen	Hans-Petter	Programming	Delete Book

[New Book](#)

© 2019 - BookApp - [Privacy](#)

Figure 17-15: Delete Book

DeleteBook.cshtml.cs:

```

...
using BookApp.Models;
using Microsoft.Extensions.Configuration;

namespace BookApp.Pages
{
    public class DeleteBookModel : PageModel
    {
        readonly IConfiguration _configuration;
    }
}

```

```
public string connectionString;

public int bookId;

public DeleteBookModel(IConfiguration configuration)
{
    _configuration = configuration;
}

public void OnGet()
{
    bookId = Convert.ToInt16(Request.Query["bookid"]);

    connectionString =
        _configuration.GetConnectionString("ConnectionString");

    Book book = new Book();

    book.DeleteBook(connectionString, bookId);

    Response.Redirect("./Books");
}
}
```

Part 6 Additional ASP.NET Core Features

Here will some additional ASP.NET Core features be presented. like the use of Session variables.

18 Session Data

HTTP is a stateless protocol. Without taking additional steps, HTTP requests are independent messages that do not retain user values or app state. We will see how we can use something called **Session variables** for sharing information between different web pages.

Typically, we need to share information between 2 different web pages as shown in Figure 18-1.



Figure 18-1: Share information between 2 different web pages

Very often we also need to share data between multiple web pages, this could be login information, etc. See Figure 18-2.

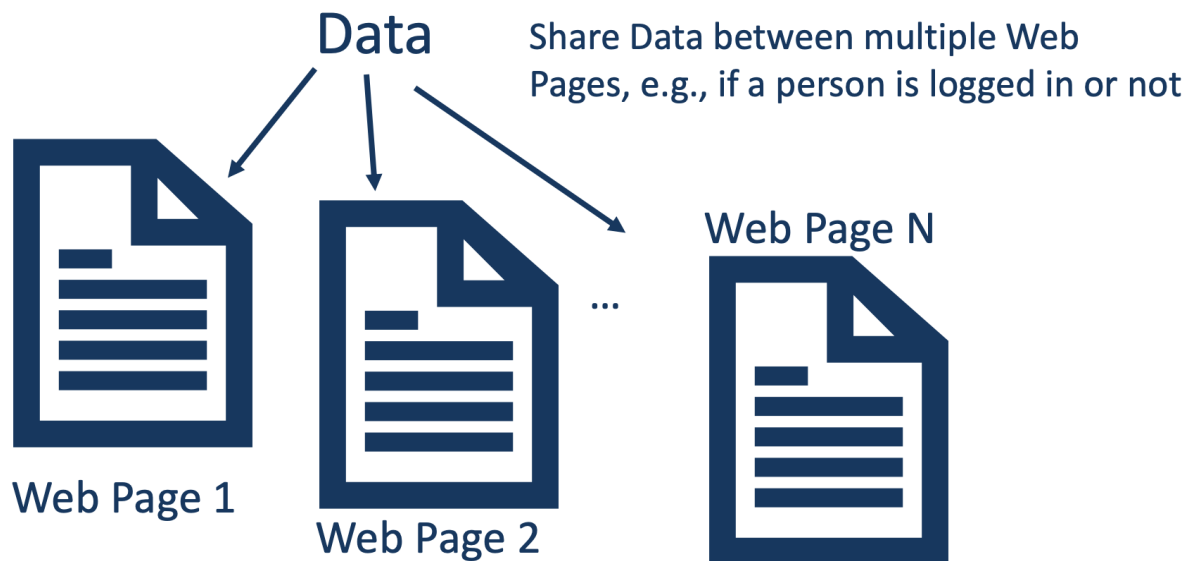


Figure 18-2: Share Data between multiple Web Pages

In a web page, the state can be stored using several approaches:

- Cookies
- **Session State**
- TempData
- Query String
- Hidden fields
- Etc.

Here we will focus on Session state and Session variables.

Session state is a mechanism that enables you to store and retrieve user specific values temporarily. The default time is 20 minutes (but can be configured if necessary).

These values can be stored for the duration of the visitor's session on your site. It can be used to share data between different web pages. It can, e.g., be used to store information if the user is logged on to the application or not and the Username.

18.1 Session State in ASP.NET Core

Session management in ASP.NET Core is not enabled by default.

You need to do the following in order to enable Session management in ASP.NET Core:

- You need to install the **Microsoft.AspNetCore.Session** NuGet Package in order to use Session state.
- You need to **enable Session State** in the **Startup.cs** file
- You need to include the Namespace **Microsoft.AspNetCore.Http**

Download Microsoft.AspNetCore.Session NuGet package (Figure 18-3):

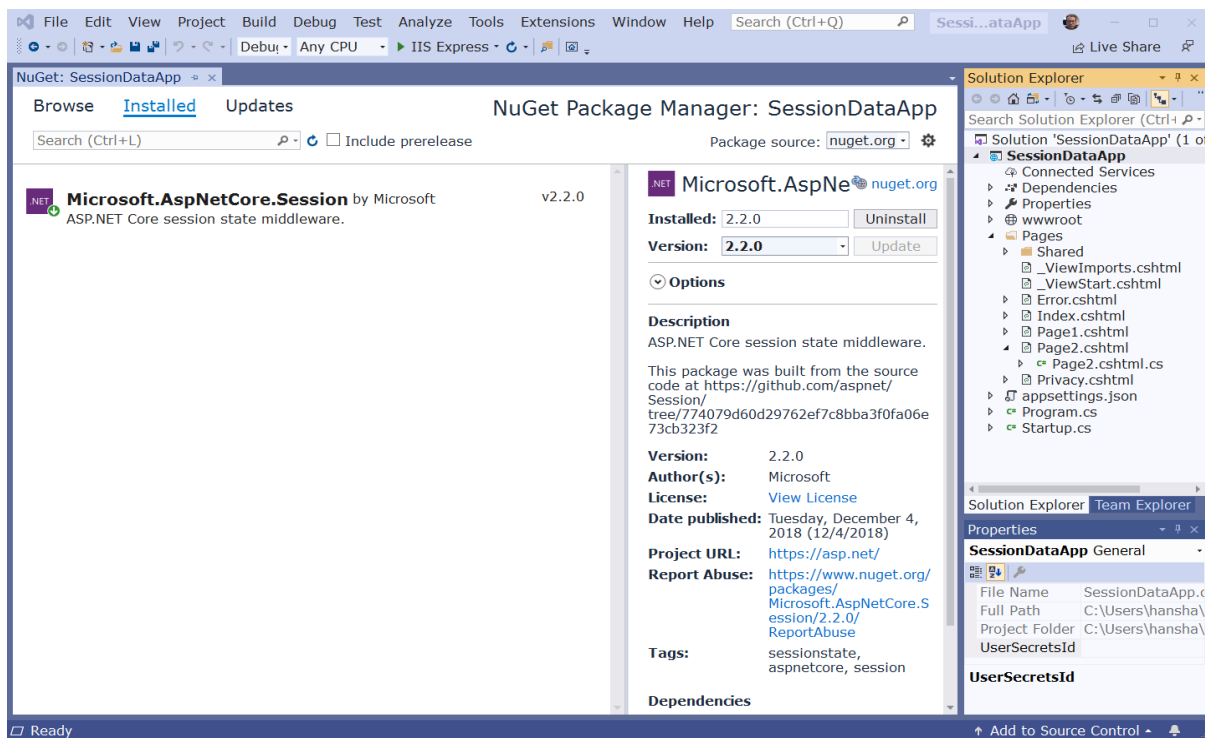


Figure 18-3: Download Microsoft.AspNetCore.Session NuGet Package

The following needs to be added to the “**Startup.cs**” file:

```
public void ConfigureServices(IServiceCollection services)
{
    ...
    services.AddSession() ;
    services.AddMemoryCache() ;
    ...
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    ...
    app.UseSession() ;
    ...
}
```

In the page model files (.cshtml.cs) where you shall use Session variables you need to add the “**Microsoft.AspNetCore.Http**” Namespace:

```
...
using Microsoft.AspNetCore.Http;
...

//Your C# Code goes here
...
```

Write to Session Variables in C#

In order to write to Session Variables in C# we use the **SetString()** and **SetInt32()** methods. Below you see an example:

```
...
string name;
int age;

HttpContext.Session.SetString("Name", name);

HttpContext.Session.SetInt32("Age", age);
```

Read from Session Variables in C#

In order to read from Session Variables in C# we use the **GetString()** and **GetInt32()** methods. Below you see an example:

```
string name;
int age;

name = HttpContext.Session.GetString("Name");

age = (Int32)HttpContext.Session.GetInt32("Age");
```

18.2 Example - Share Data between 2 Web Pages

Below we will go through a basic example that use Session in order to share data.

This application consists of 2 web pages. In Page 1 (Page1.cshtml) the user enters some data into 2 textboxes; Name and Age. When the user clicks the "Save" button, the data is stored in 2 Session variables. The Name is a String, and the Age is an Integer. Page 2 (Page2.cshtml) reads the Session variables and present them to the user. See Figure 18-4.

Example Application

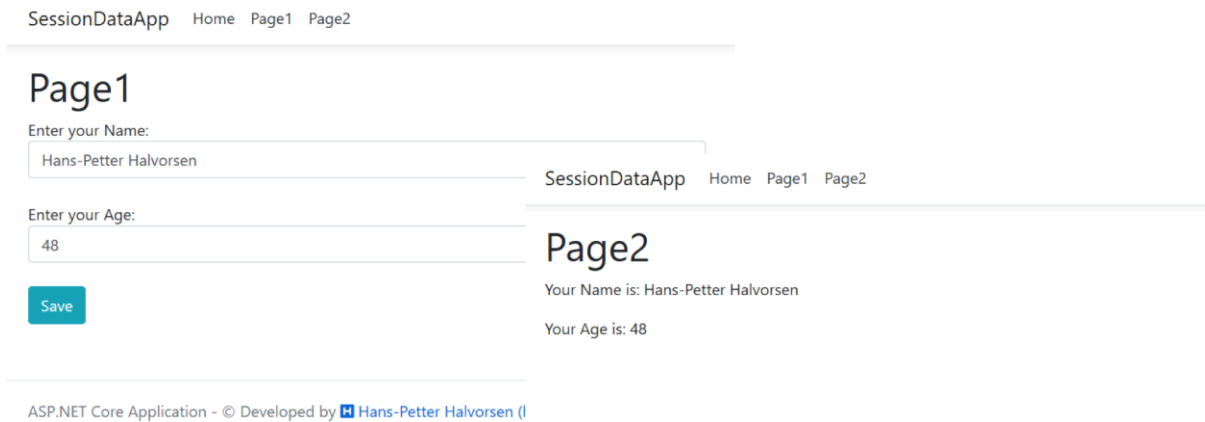


Figure 18-4: Session Data Example Application

Below we go through the example step by step.

18.2.1 Page1

Let us start creating Page1. Sometimes we start development of the Razor file (.cshtml) and other times it is smart to start with the Code Behind file (cshtml.cs).

Page 1 is shown in Figure 18-5.

SessionDataApp Home Page1 Page2

Page1

Enter your Name:

Enter your Age:

ASP.NET Core Application - © Developed by [H Hans-Petter Halvorsen \(https://www.halvorsen.blog\)](https://www.halvorsen.blog)

Figure 18-5: Page1 in Example

For Page1 we start with Razor file (Page1.cshtml):

```
@page
@model SessionDataApp.Pages.Page1Model
@{
    ViewData["Title"] = "Page1";
}
<div>
<h1>Page1</h1>
<form method="post">
    Enter your Name:
    <br />
    <input name="Name" type="text" class="form-control input-lg" required />
    <br />
    Enter your Age:
    <br />
    <input name="Age" type="number" min="1" max="100" class="form-control input-
    lg" required />
    <br />
    <input id="SaveButton" type="submit" value="Save" class="btn btn-info">
</form>
</div>
```

Then we create the code for the “Page1.cshtml.cs” file:

```
..
using Microsoft.AspNetCore.Http;
namespace SessionDataApp.Pages
{
    public class Page1Model : PageModel
    {
        string name;
        int age;
    }
}
```

```
public void OnGet()  
{  
}  
public void OnPost()  
{  
    name = Request.Form["Name"];  
    HttpContext.Session.SetString("Name", name);  
    age = Convert.ToInt32(Request.Form["Age"]);  
    HttpContext.Session.SetInt32("Age", age);  
}  
}
```

Explanation of the code:

From the webpage (Page1.cshtml) we retrieve the value the user enters in the "Name" textbox and put the value into the variable "name":

```
string name;  
name = Request.Form["Name"];
```

Then the value is stored into a Session variable called "Name":

```
HttpContext.Session.SetString("Name", name);
```

From the webpage (Page1.cshtml) we retrieve the value the user enters in the "Age" input field and put the value into the variable "age":

```
int age;  
age = Convert.ToInt32(Request.Form["Age"]);
```

Since age is an Integer, we need to convert the value to Int32.

Then the value is stored into a Session variable called "Age":

```
HttpContext.Session.SetInt32("Age", age);
```

18.2.2 Page2

Let us start creating Page2.

Page 2 is Figure 18-6.

SessionDataApp Home Page1 Page2

Page2

Your Name is: Hans-Petter Halvorsen

Your Age is: 48

ASP.NET Core Application - © Developed by [H Hans-Petter Halvorsen \(https://www.halvorsen.blog\)](https://www.halvorsen.blog)

Figure 18-6: Page2 in Example

For Page2 we start with Code Behind file (Page2.cshtml.cs):

```
..
using Microsoft.AspNetCore.Http;
namespace SessionDataApp.Pages
{
    public class Page2Model : PageModel
    {
        public string name;
        public int age;
        public void OnGet()
        {
            if (HttpContext.Session.GetString("Name") != null)
                name = HttpContext.Session.GetString("Name");
            if (HttpContext.Session.GetInt32("Age") != null)
                age = (Int32)HttpContext.Session.GetInt32("Age");
        }
    }
}
```

Then we put contents into the “Page2.cshtml” file:

```
@page
@model SessionDataApp.Pages.Page2Model
@{
    ViewData["Title"] = "Page2";
}
<div>
    <h1>Page2</h1>
```

```
<p>  
  Your Name is: @Model.name  
</p>  
<p>  
  Your Age is: @Model.age  
</p>  
</div>
```

18.3 Additional Resources

Additional Resources for Session state and Session variables:

- <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state>
- <https://www.learnrazorpages.com/razor-pages/session-state>

Part 7 Charting

Presenting data in charts is important. Here we will see how we can create charts in an ASP.NET Core application.

19 Charting

Typically, you want to use charts in your web application. Here we have many options. As far as I know ASP.NET Core has still not any built-in features for creating charts, so you need to use a 3.part tool for the job.

Some chart tools are:

- Google Charts
- Charts.js

19.1 Introduction

In Figure 19-1 you see an example where a chart is displayed on web page or a web application.

From: 2019-11-01 To: 2019-12-06



Temperature (mtTemp1)

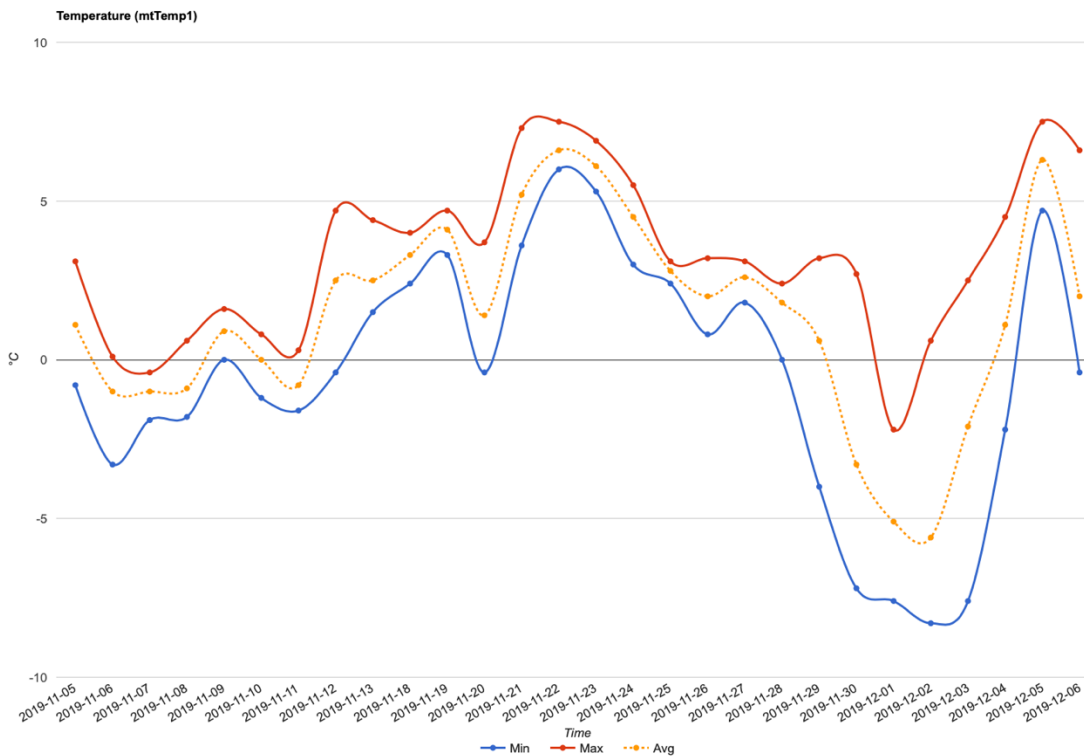


Figure 19-1: Charting Example

20 Google Charts

Google Charts is a JavaScript based charting library for presenting different types of charts on a web page.

Google Charts is an API (or framework) for creating Charts in your web pages. It is free to use. It is easy to use (when you first know how to use it).

Web Site:

<https://developers.google.com/chart>

Google Charts offers many different types of charts:

- Line Chart
- Bar Chart
- Column Chart
- Pie Chart
- etc.

Figure 20-1 shows an example.

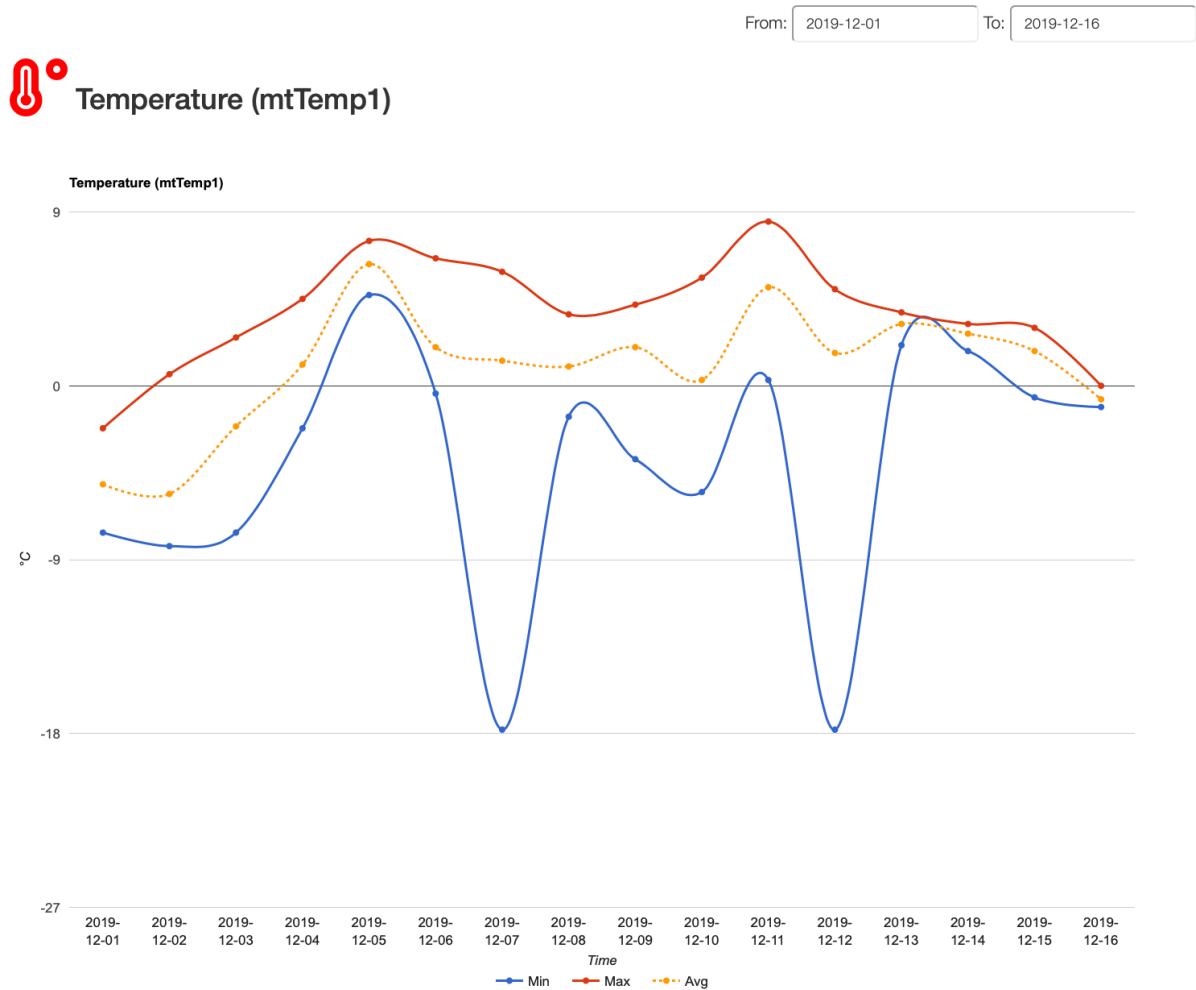


Figure 20-1: Google Charts

20.1 Google Charts Implementation

The most common way to use Google Charts is with simple JavaScript that you embed in your web page.

In your web page you need to implement the following steps:

1. You **load** some Google Chart libraries,
2. List the data to be charted,
3. Select options to **customize** your chart,
4. Finally **create a chart object** with an id that you choose.

5. Then, later in the web page, you **create a <div>** with that id to display the Google Chart.

First, you need to load the Google Chart libraries:

```
<script src="https://www.gstatic.com/charts/loader.js"></script>
<script>
  google.charts.load('current', {packages: ['corechart']});
  google.charts.setOnLoadCallback(drawChart);
  ...
</script>
```

20.2 Google Charts Examples

Some basic Charts examples using ASP.NET Core will be demonstrated. The examples use the Google Charts framework. An ASP.NET Core Chart Application has been created in order to demonstrate how to implement and use Google Charts. See Figure 20-2.

ChartApp Home LineChart MultiLineChart BarChart ColumnChart

Chart Examples

This ASP.NET Core Web Application demonstrates how to implements Charts in your Web Application.

These examples use Google Charts. [Here you see a Basic Google Chart Example](#)

Examples that gets the Data from a SQL Server Database:

- [Line Chart](#)
- [MultiLine Chart](#)
- [Bar Chart](#)
- [Column Chart](#)

The most common way to use Google Charts is with simple JavaScript that you embed in your web page. You load some Google Chart libraries, list the data to be charted, select options to customize your chart, and finally create a chart object with an id that you choose. Then, later in the web page, you create a <div> tag with that id to display the Google Chart.

[Google Charts](#)

Developed by [Hans-Petter Halvorsen \(https://www.halvorsen.blog\)](https://www.halvorsen.blog)

Figure 20-2: Google Chart Examples



ASP.NET Core - Charts: <https://youtu.be/mksUls9fx-Q>

The entire example can be downloaded from the home page of this textbook.

The application shown in Figure 20-2 will demonstrate different chart types like “Line Chart”, “Multi-Line Chart”, “Bar Chart” and “Column Chart”.

20.2.1 Basic Chart Example

We start with a basic Google Chart examples, see Figure 20-3.

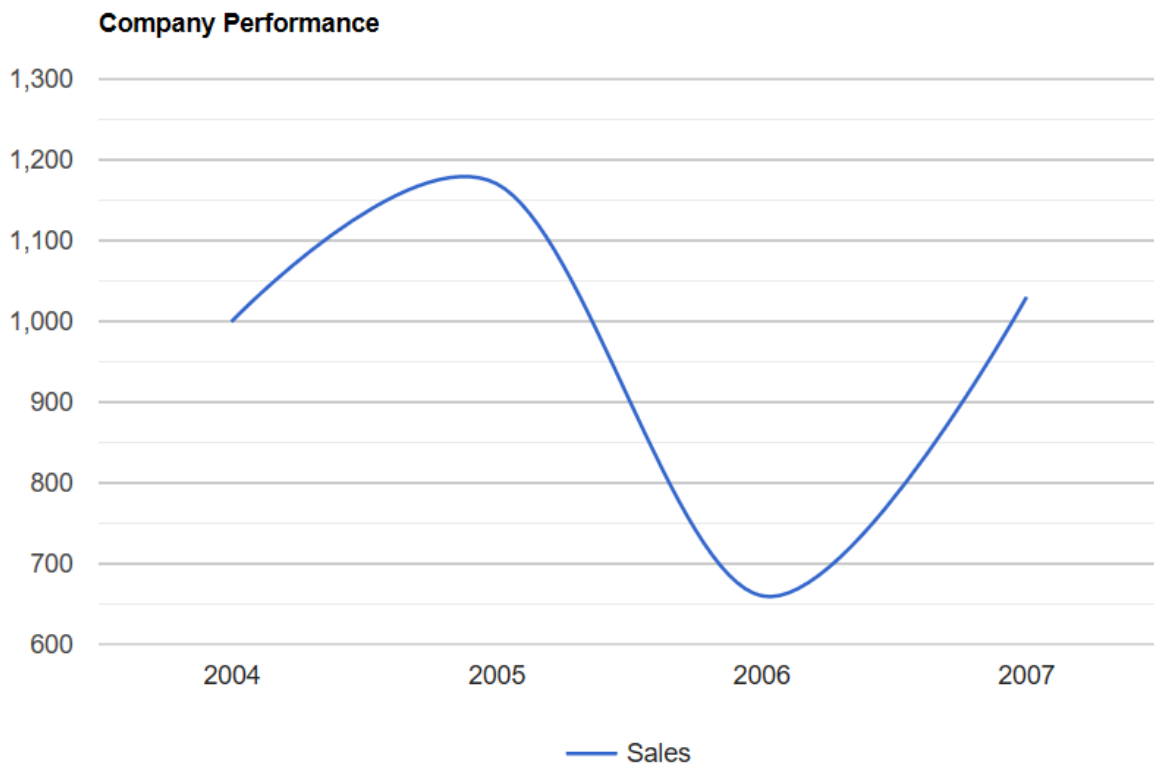


Figure 20-3: Basic Google Chart Example

Web Page:

```
<html>
<head>
  <script type="text/javascript"
    src="https://www.gstatic.com/charts/loader.js"></script>
  <script type="text/javascript">

google.charts.load('current', {'packages':['corechart']});
google.charts.setOnLoadCallback(drawChart);
function drawChart() {
  var data = google.visualization.arrayToDataTable([
    ['Year', 'Sales'],
    ['2004', 1000],
    ['2005', 1170],
    ['2006', 660],
    ['2007', 1030]
  ]);
  var options = {
    title: 'Company Performance',
    curveType: 'function',
```

```
    legend: { position: 'bottom' }
  };
  var chart = new
    google.visualization.LineChart(
      document.getElementById('mychart'));
  chart.draw(data, options);
}
</script>
</head>
<body>
  <div id="mychart" style="width: 900px; height: 500px"></div>
</body>
</html>
```

20.2.2 Database Examples

Most of the time you will need to use data that are stored in a SQL Server. Typically, you want to plot some of the data inside your database

You should have SQL Server locally installed on your computer (for these examples). SQL Server Express is recommended for these simple examples, but if you have another version already, you can of course use that.

For the examples provided here, a simple database will be created, see below:

```
CREATE TABLE [CHARTDATA]
(
  [ChartDataId] int NOT NULL IDENTITY ( 1,1 ) Primary Key,
  [ChartTimeStamp] datetime NOT NULL DEFAULT GETDATE(),
  [ChartValue] float NOT NULL
)
go
```

This is a simple database with just one table. The table contains 2 important columns, one with timestamps, which is typically the x-axis in your chart. The other column is the data itself, which is typically the y-axis.

Next, we need to add some data in the database. You can just enter some data manually in order to have some data for the examples. See Figure 20-4.

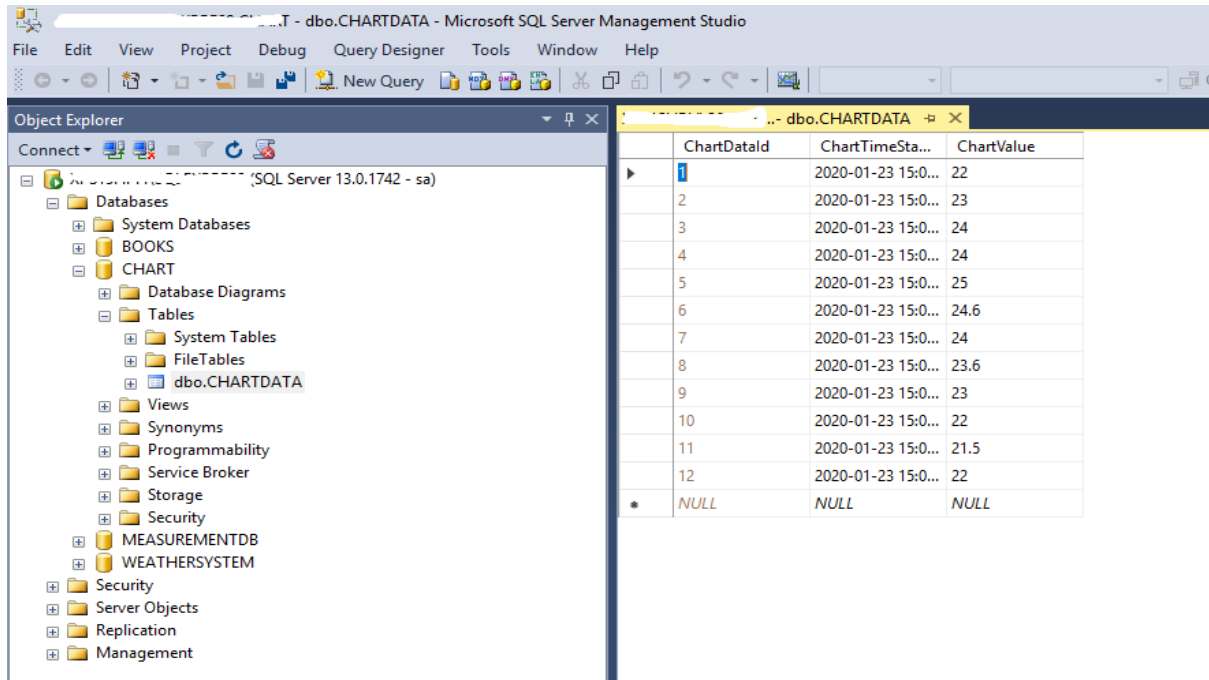


Figure 20-4: SQL Server

We will create some basic Examples:

- Line Chart
- Bar Chart
- Column Chart
- Multi Line Chart

We create a new ASP.NET Core application to demonstrate the different charts. Figure 20-5 shows the Solution Explorer in Visual Studio for our project ("ChartApp").

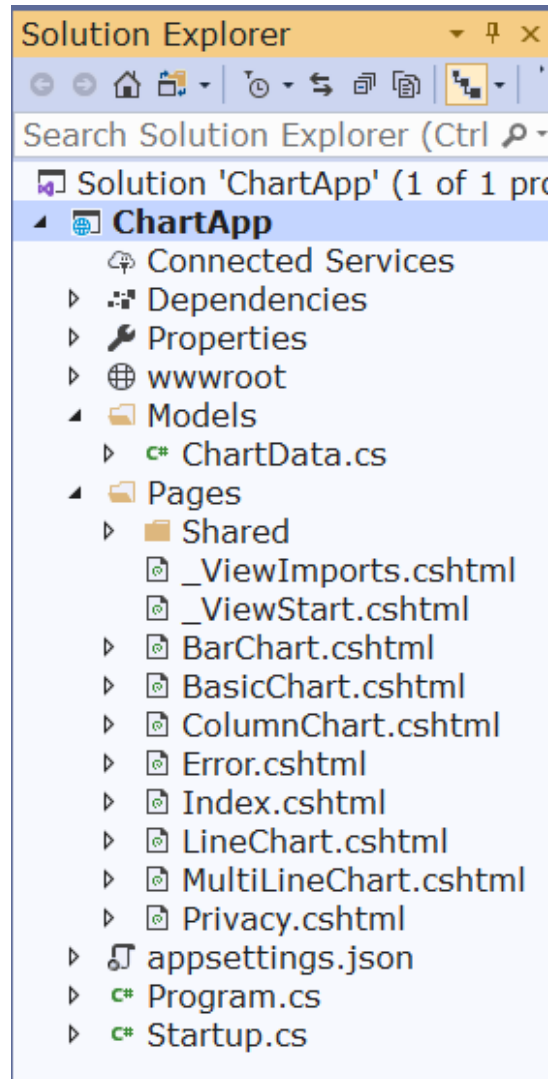


Figure 20-5: Solution Explorer

As in previously ASP.NET Core examples we have created 2 folders, namely “Models” and “Pages”.

In the “Models” folder we create our C# code that communicates and retrieve data from the database. We create a class called “ChartData.cs” for this.

C# Code for “ChartData.cs”

```
using System.Data.SqlClient;
namespace ChartApp.Models
{
    public class ChartData
    {
        public int ChartDataId { get; set; }
        public string ChartTimeStamp { get; set; }
        public double ChartValue { get; set; }
        public List<ChartData> GetChartData(string connectionString)
        {
            List<ChartData> chartDataList = new List<ChartData>();
        }
    }
}
```

```

SqlConnection con = new SqlConnection(connectionString);
string selectSQL = "SELECT ChartDataId,
FORMAT(ChartTimeStamp, 'MM.dd HH:mm') AS ChartTimeStamp,
ChartValue FROM CHARTDATA";
con.Open();
SqlCommand cmd = new SqlCommand(selectSQL, con);
SqlDataReader dr = cmd.ExecuteReader();
if (dr != null)
{
    while (dr.Read())
    {
        ChartData chartData = new ChartData();
        chartData.ChartDataId =
            Convert.ToInt32(dr["ChartDataId"]);
        chartData.ChartTimeStamp =
            dr["ChartTimeStamp"].ToString();
        chartData.ChartValue =
            Convert.ToDouble(dr["ChartValue"]);
        chartDataList.Add(chartData);
    }
}
return chartDataList;
}
}
}

```

Next, we start creating the web pages for displaying the charts. We start with a “Line Chart”.

20.2.3 Line Chart Example

Based on the data we created inside our database we will plot them in a “Line Chart”. Figure 20-6 shows the result.

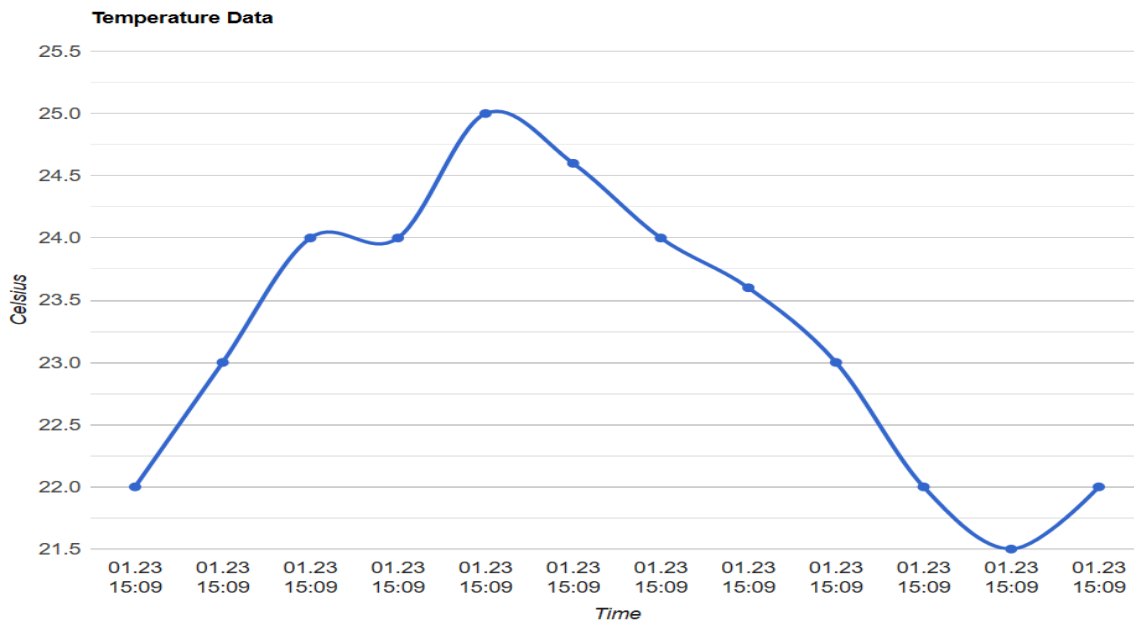


Figure 20-6: Line Chart Example

Lets starts creating the web page. As you now, a ASP.NET Core web page consists of 2 different files, a Page model file (“.cshtml.cs”), or a “code-behind file and the web page itself with Razor syntax (“.cshtml”).

C# Code (LineChart.cshtml.cs):

```
using ChartApp.Models;
namespace ChartApp.Pages
{
    public class LineChartModel : PageModel
    {
        public List<ChartData> chartDataList = new
            List<ChartData>();

        string connectionString;
        readonly IConfiguration _configuration;
        public LineChartModel(IConfiguration configuration)
        {
            _configuration = configuration;
        }
        public void OnGet()
        {
            chartDataList = ChartData();
        }
        private List<ChartData> ChartData()
        {
            connectionString =
                _configuration.GetConnectionString("ConnectionString");

            List<ChartData> chartDataList = new List<ChartData>();
            ChartData chartData = new ChartData();
            chartDataList =
                chartData.GetChartData(connectionString);
            return chartDataList;
        }
    }
}
```

Web Page (LineChart.cshtml):

```
@page
@model ChartApp.Pages.LineChartModel
@{
    ViewData["Title"] = "Line Chart";
    string chartTitle = "Temperature Data";
    string chartUnit= "Celsius";
}
<div class="text-center">
    <h1 class="display-4">Line Chart</h1>
</div>

<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
    google.charts.load('current', { 'packages': ['corechart'] });
    google.charts.setOnLoadCallback(drawChart);
    function drawChart() {
        var data = google.visualization.arrayToDataTable([
            ['Time', 'Data'],
            @foreach (var data in Model.chartDataList) {
                <text>['@data.ChartTimeStamp', @data.ChartValue],</text>
            }
        ])
    }
}
```

```

    });
    var options = {
      title: '@chartTitle',
      curveType: 'function',
      pointsVisible: true,
      lineWidth: 3,
      legend: 'none',
      hAxis: {title: 'Time'},
      vAxis: {title: '@chartUnit'},
      width: '100%',
      height: '100%',
      chartArea: {width: '85%', height: '75%'}
    };
    var chart = new
        google.visualization.LineChart(document
            .getElementById('line_chart'));
    chart.draw(data, options);
  }
</script>
<div class="container-fluid lead">
  <div id="line_chart" style="width: 800px; height: 600px"></div>
</div>

```

The new part in this example compared to the previous example are the part where we get the build an array with the data from the database:

```

var data = google.visualization.arrayToDataTable([
  ['Time', 'Data'],
  @foreach (var data in Model.chartDataList) {
    <text>['@data.ChartTimeStamp', @data.ChartValue],</text>
  }
]);

```

When running your application, the plot in Figure 20-6 should be the result.

20.2.4 Bar Chart Example

Figure 20-7 shows a “Bar Chart”.

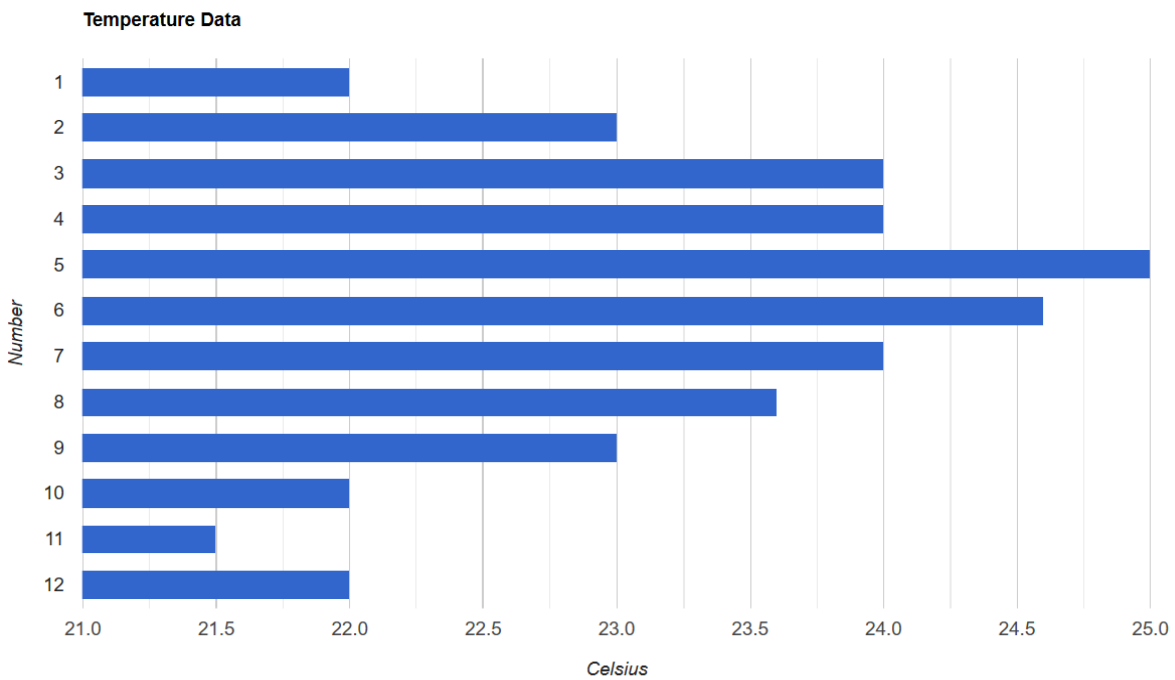


Figure 20-7: Bar Chart Example

The code is almost identical, except for one line.

Change the following from “LineChart” to “BarChart”:

```
var chart = new  
google.visualization.LineChart(document.getElementById('line_chart'));
```

For a “Bar Chart” we set:

```
var chart = new  
google.visualization.BarChart(document.getElementById('line_chart'));
```

20.2.5 Column Chart Example

Figure 20-8 shows a “Column Chart”.

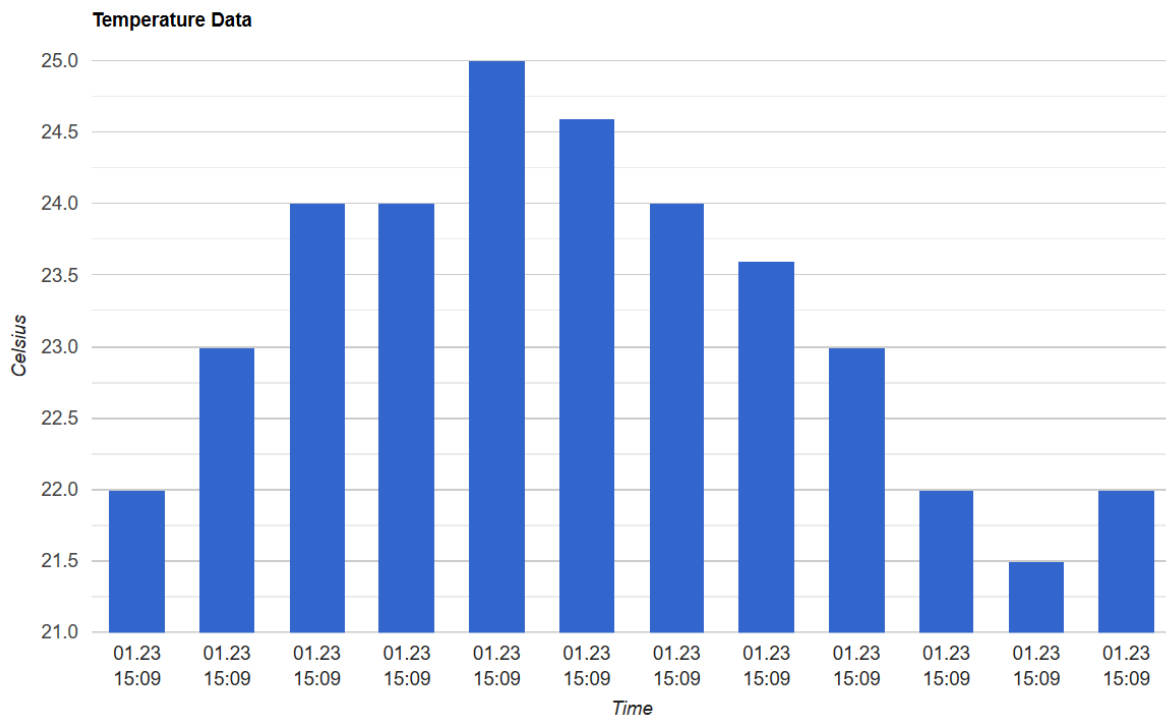


Figure 20-8: Column Chart Example

The code is almost identical, except for one line.

Change the following from “LineChart” to “BarChart”:

```
var chart = new
google.visualization.LineChart(document.getElementById('line_chart'));
```

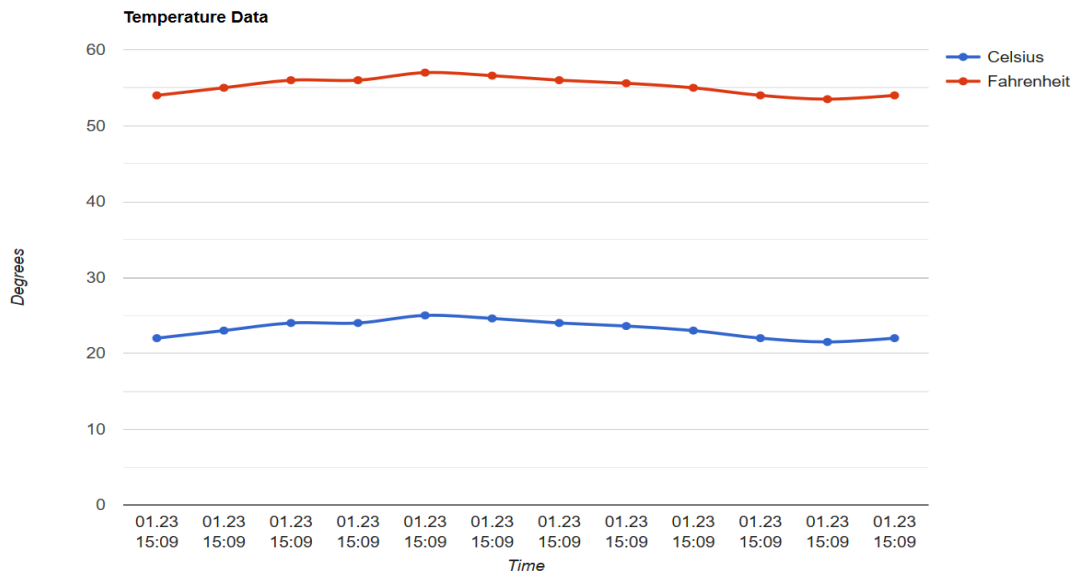
For a “Bar Chart” we set:

```
var chart = new
google.visualization.ColumnChart(document.getElementById('line_chart'));
```

20.2.6 Multi-Line Chart Example

Figure 20-9 shows a “Multi-Line Chart”.

Multi Line Chart



26

Figure 20-9: Multi-Line Example

In this example we plot 2 lines in the same chart. In this example, the blue line is the same as in Figure 20-6, while the red line is just the calculated Fahrenheit value.

Web Page (MultiLineChart.cshtml):

```
@page
@model ChartApp.Pages.MultiLineChartModel
@{
    ViewData["Title"] = "MultiLine Chart";

    string chartTitle = "Temperature Data";
    string chartUnit = "Degrees";
}

<div class="text-center">
    <h1 class="display-4">MultiLine Chart</h1>
</div>

<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
    google.charts.load('current', { 'packages': ['corechart'] });
    google.charts.setOnLoadCallback(drawChart);

    function drawChart() {
        var data = google.visualization.arrayToDataTable([
            ['Time', 'Celsius', 'Fahrenheit'],

            @foreach (var data in Model.chartDataList) {
```



```

        double fahrenheitValue = data.ChartValue * (9/5) + 32;

        <text>['@data.ChartTimeStamp', @data.ChartValue,
            @fahrenheitValue],</text>

    }

    ]);

    var options = {
        title: '@chartTitle',
        curveType: 'function',
        pointsVisible: true,
        lineWidth: 3,
        legend: 'right',
        hAxis: {title: 'Time'},
        vAxis: {title: '@chartUnit'},
        width: '100%',
        height: '100%',
        chartArea: {width: '70%', height: '75%'}
    };

    var chart = new
        google.visualization.LineChart (document
            .getElementById('line_chart'));

    chart.draw(data, options);
}
</script>

<div class="container-fluid lead">

    <div id="line_chart" style="width: 1000px; height: 600px"></div>

</div>

```

The only difference from the other examples, is that we need to make 3 columns instead of 2. One column for the “TimeStamp” values, one column for the “Celsius” values, and one column for the “Fahrenheit” value.

In addition, there is a simple formula for converting from Celsius to Fahrenheit.

```
double fahrenheitValue = data.ChartValue * (9/5) + 32;
```

21 Chart.js

Chart.js is similar to Google Charts.

Web Site:

<https://www.chartjs.org>

Part 8 APIs

Overview of APIs

22 Class Libraries

A class library defines types and methods that are called by an application. When you finish your class library, you can decide whether you want to distribute it as a third-party component or whether you want to include it as a bundled component with one or more applications. You can also create a NuGet package of it.

Benefits:

- Make a better code structure
- Sharing code between multiple Visual Studio Projects.
- Share an assembly (.dll) that is doing a specific task to others, e.g., you can make a Class Library that have Classes and Methods for communication with a database or an OPC server, or a math library with Classes and Methods for dealing with matrices, statistics, etc. There are endless possibilities.



Introduction to Class Libraries in ASP.NET Core: <https://youtu.be/emUiMd1zRrY>

Figure 22-1 show the .NET Core Class Library template for creating a new Class Library in Visual Studio.

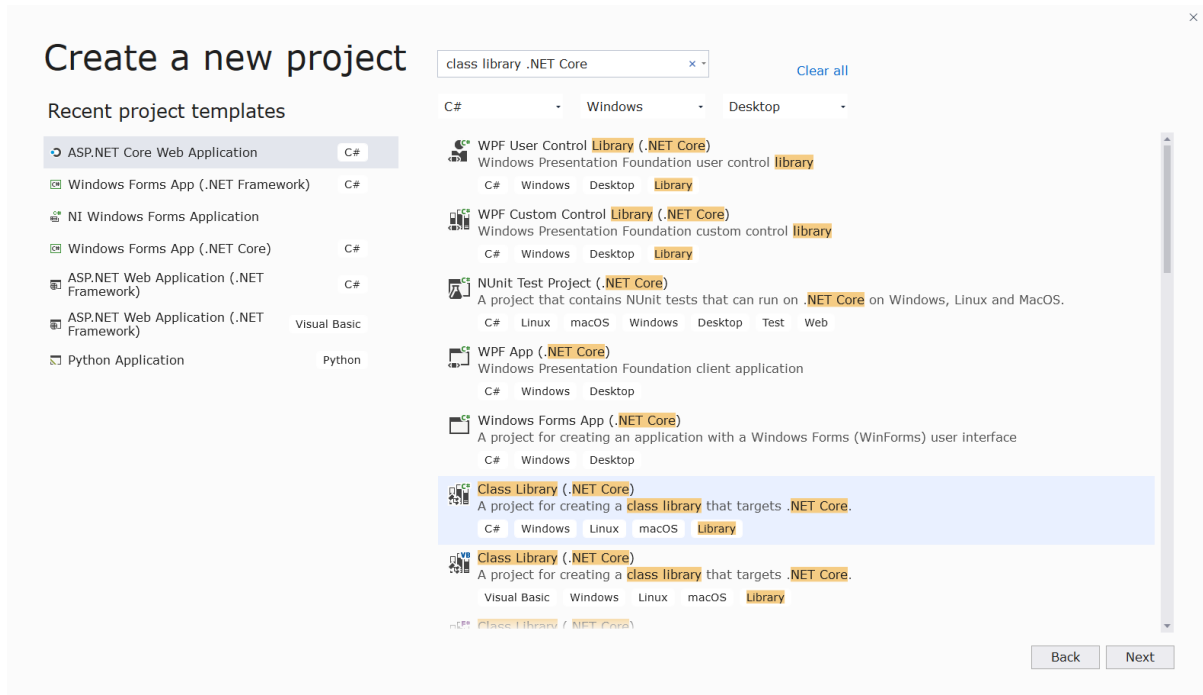


Figure 22-1: Class Library – New Project

22.1 Demo Application

Here will creating and using a Class Library be demonstrated. A video is also provided that goes through the example in detail.



Introduction to Class Libraries in ASP.NET Core: <https://youtu.be/emUiMd1zRrY>

Here the following steps will be shown:

1. Make the Class Library
2. Use the Class Library in another Project/Application.

....

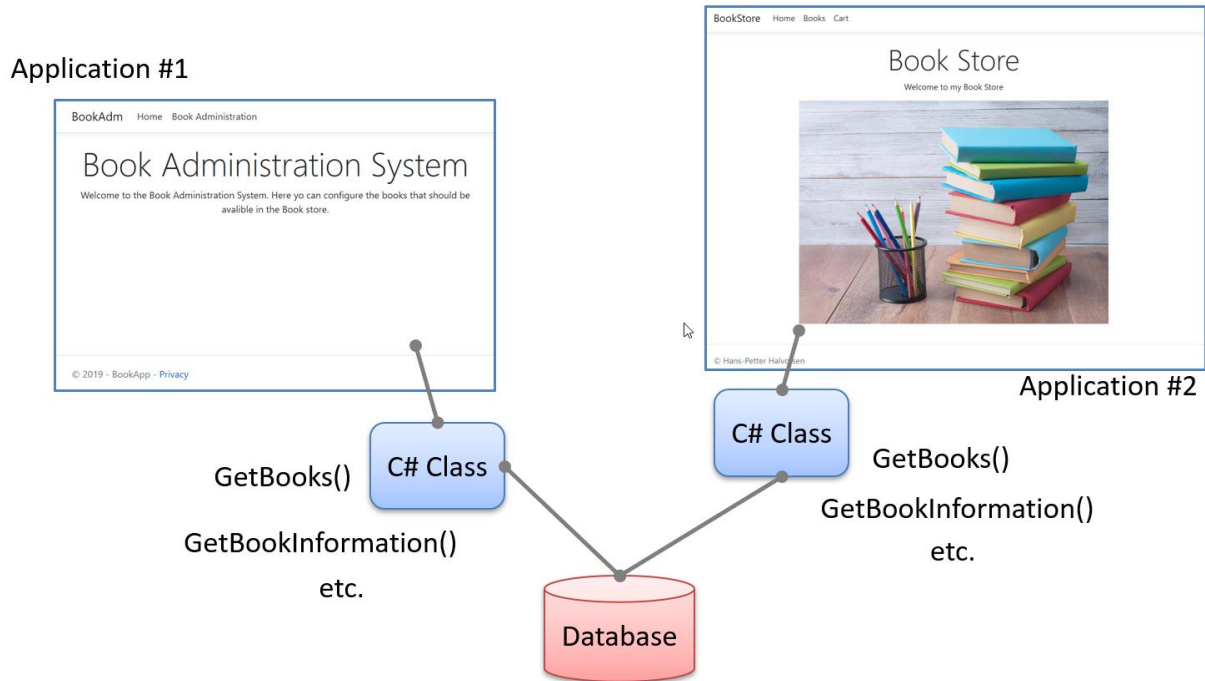


Figure 22-2: Applications NOT using a common Class Library

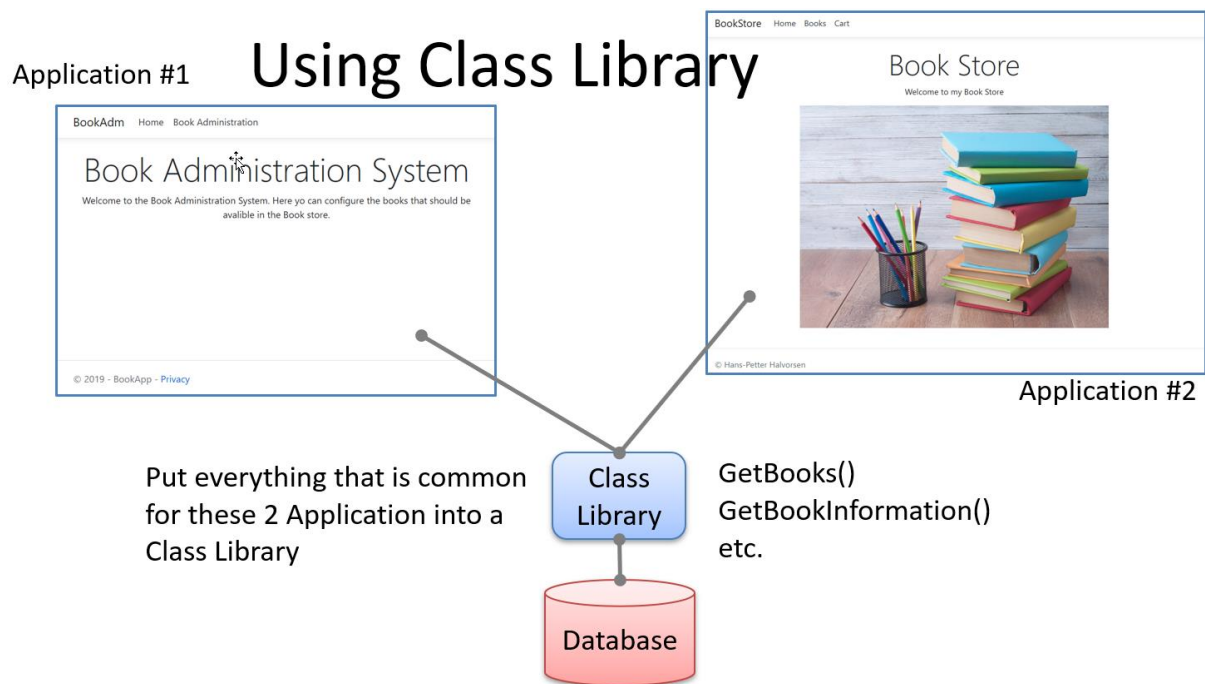


Figure 22-3: Applications using a common Class Library

In this example we will first create a Class Library, then we will create Applications that are using the Class Library.

We will create the following:

- The **Class Library**, which contains common C# code for the 2 applications
- The **BookAdm App**. This app is used to configure the books that should be sold in the BookStore App. The application uses the C# code in the Class Library.
- The **BookStore App**. This app is used by the customers that want to buy books in the Book Store. The application uses the C# code in the Class Library.

22.1.1 Class Library

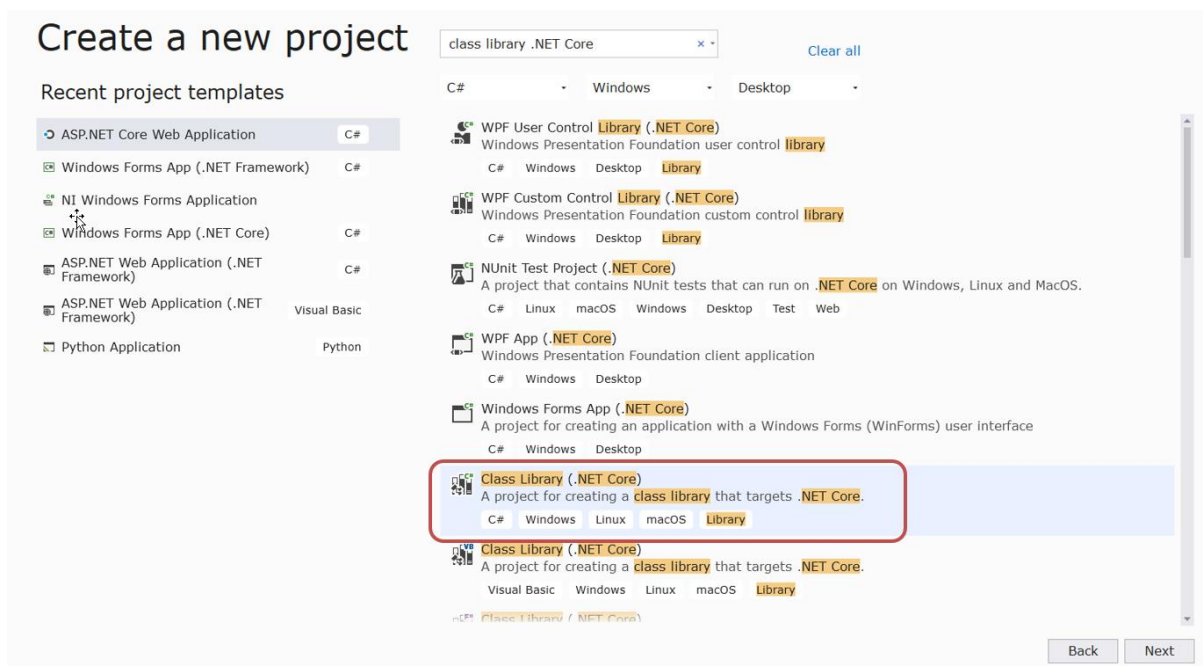


Figure 22-4: New Project – Class Library (.NET Core) in Visual Studio

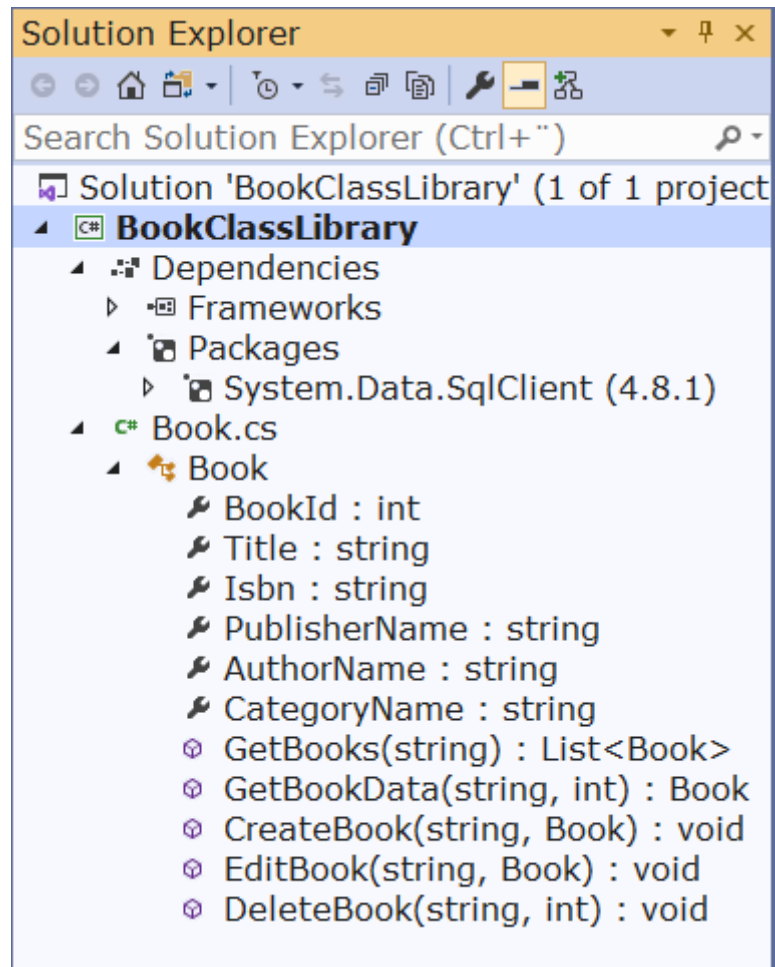
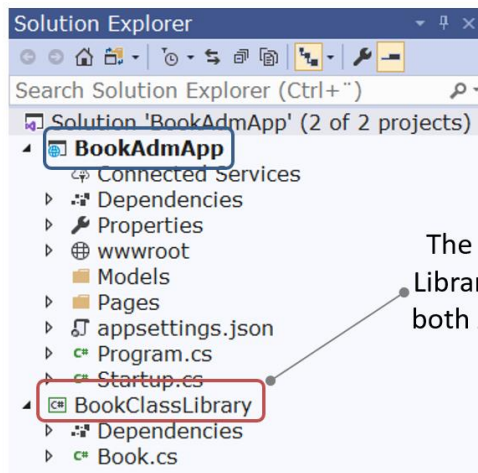


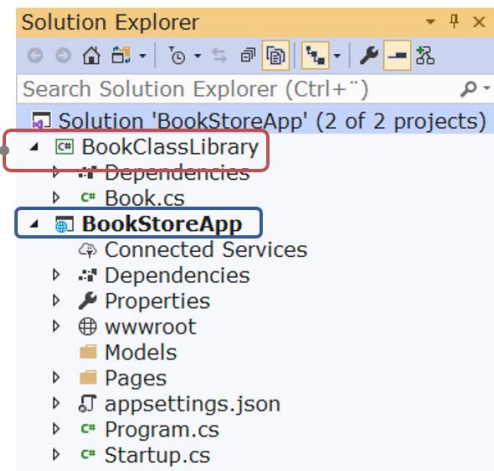
Figure 22-5: Class Library Project

Solution Explorer

Application #1



Application #2



The same Class Library is used by both Applications

Figure 22-6: Visual Studio Solution with Web Application Project and Class Library Project

22.1.2 BookAdm App

BookAdm Home Book Administration

Book Administration System

Welcome to the Book Administration System. Here yo can configure the books that should be available in the Book store.

Figure 22-7: Book Administration Web Application

...

BookAdm Home Book Administration

Books

Below you see all the Books in the Book Store:

BookId	Title	ISBN	Publisher	Author	Category	Action
1	Introduction to Linear Algebra	0-07-066781-0	Prentice Hall	Gilbert Strang	Science	Delete Book
2	Modern Control System	1-08-890781-0	Wiley	Dorf Bishop	Programming	Delete Book
3	The Lord of the Rings	2-09-066556-2	McGraw-Hill	J.R.R Tolkien	Novel	Delete Book
4	Python	5555568	Halvorsen	Hans-Petter	Programming	Delete Book

New Book

© 2019 - BookApp - Privacy

Figure 22-8: Books Page

Add Existing Project:

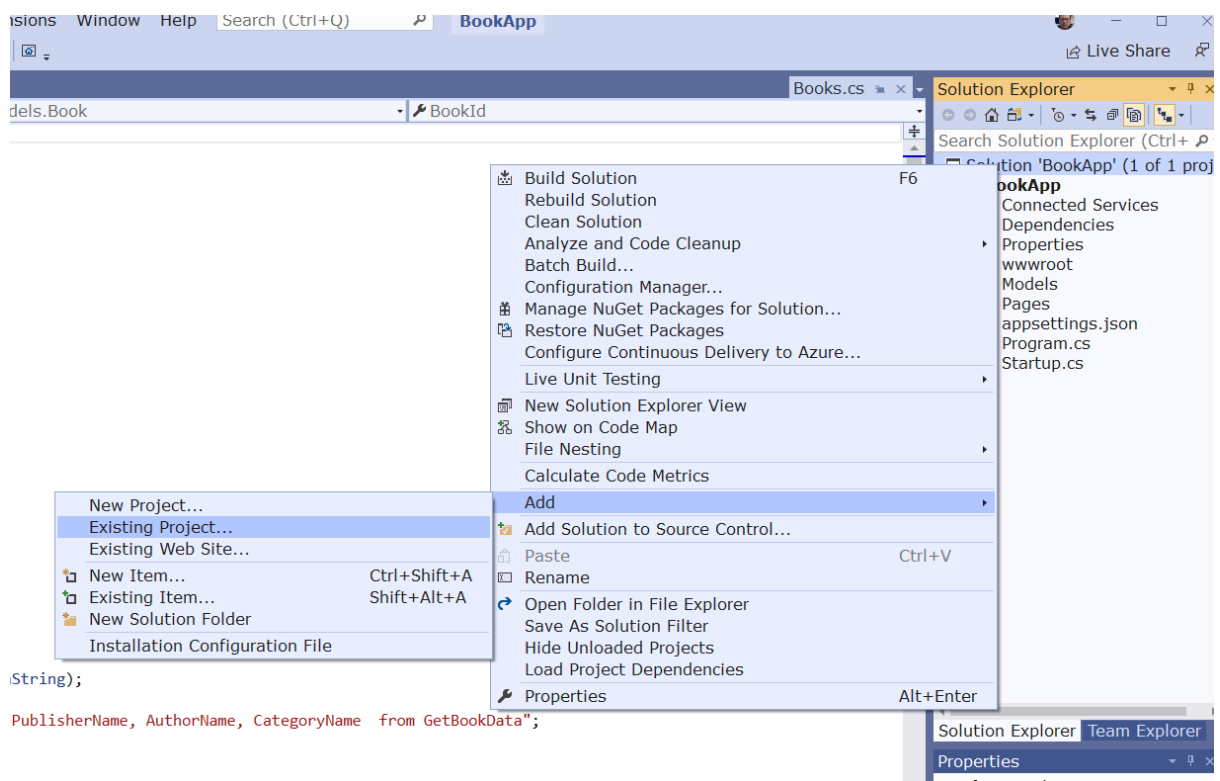
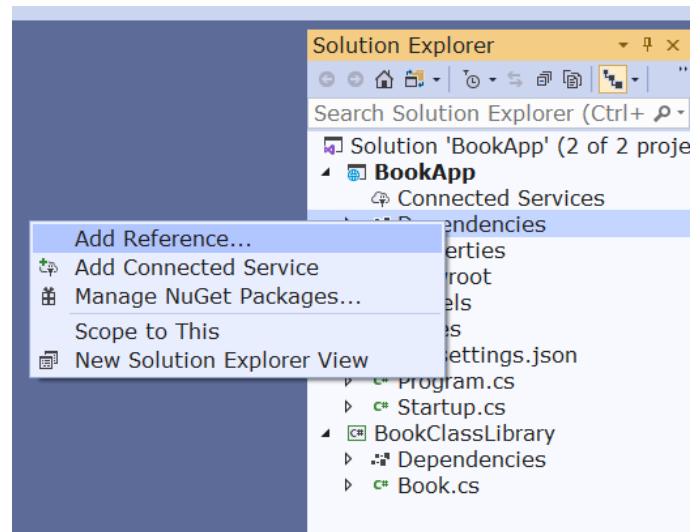
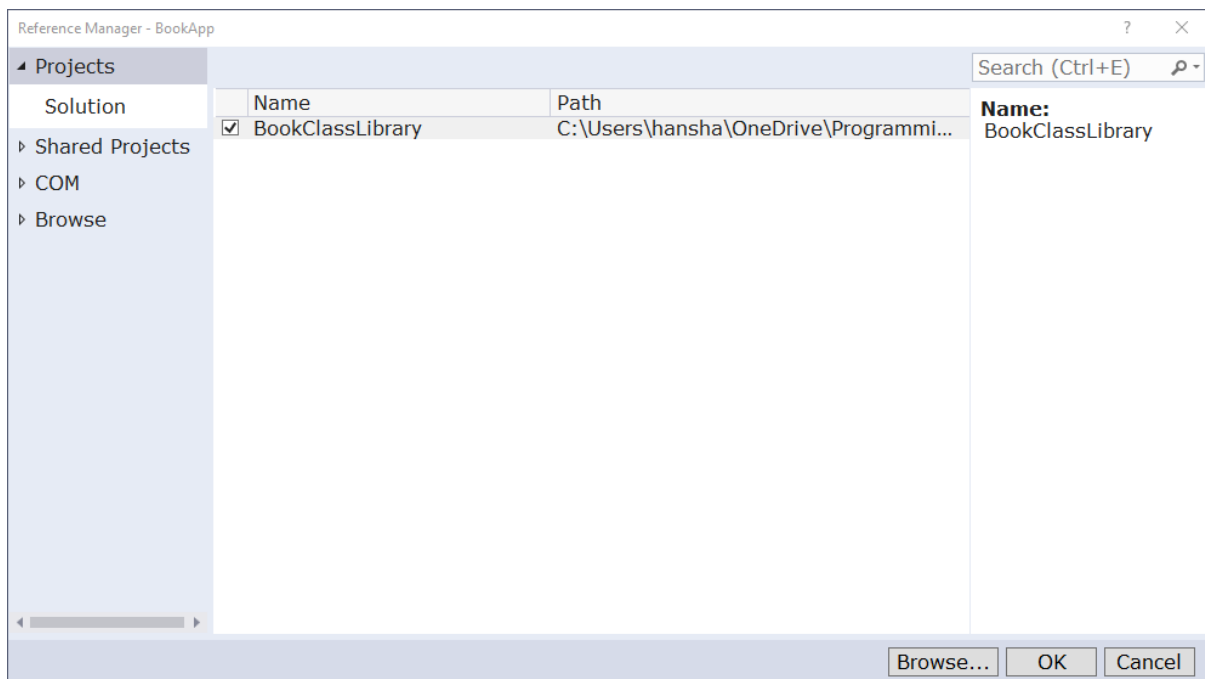


Figure 22-9

...

Add Reference:*Figure 22-10**Figure 22-11*

Solution Explorer

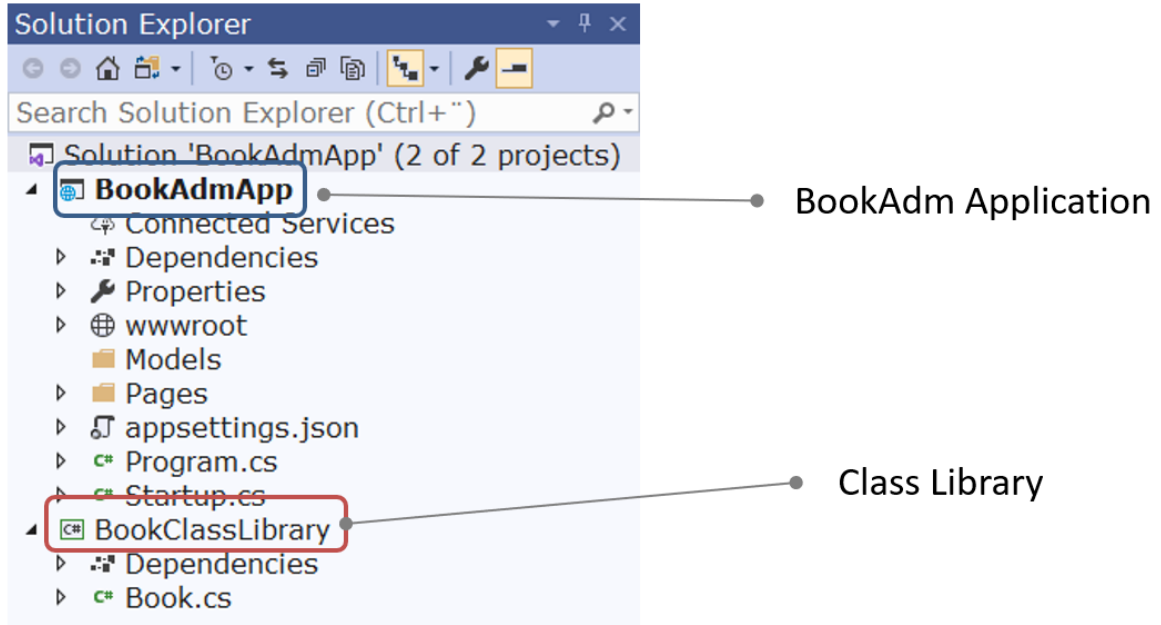


Figure 22-12

22.1.3 BookStore App

BookStore Home Books Cart

Book Store

Welcome to my Book Store



© Hans-Petter Halvorsen

Figure 22-13

...

BookStore Home Books Cart

Books

Below you see all the Books in the Book Store:

The screenshot displays a web page for a book store. At the top, there is a navigation bar with links for 'BookStore', 'Home', 'Books', and 'Cart'. Below this is a section titled 'Books'. A sub-header reads 'Below you see all the Books in the Book Store:'. Two book listings are shown, each in a yellow rectangular box. The first listing is for 'Introduction to Linear Algebra' by Gilbert Strang, published by Prentice Hall, with ISBN 0-07-066781-0, categorized under Science, and a blue 'Buy' button. The second listing is for 'Modern Control System' by Dorf Bishop, published by Wiley, with ISBN 1-08-890781-0, categorized under Programming, and a blue 'Buy' button.

Introduction to Linear Algebra
Gilbert Strang
Prentice Hall
0-07-066781-0
Science
Buy

Modern Control System
Dorf Bishop
Wiley
1-08-890781-0
Programming
Buy

Figure 22-14

...

Solution Explorer

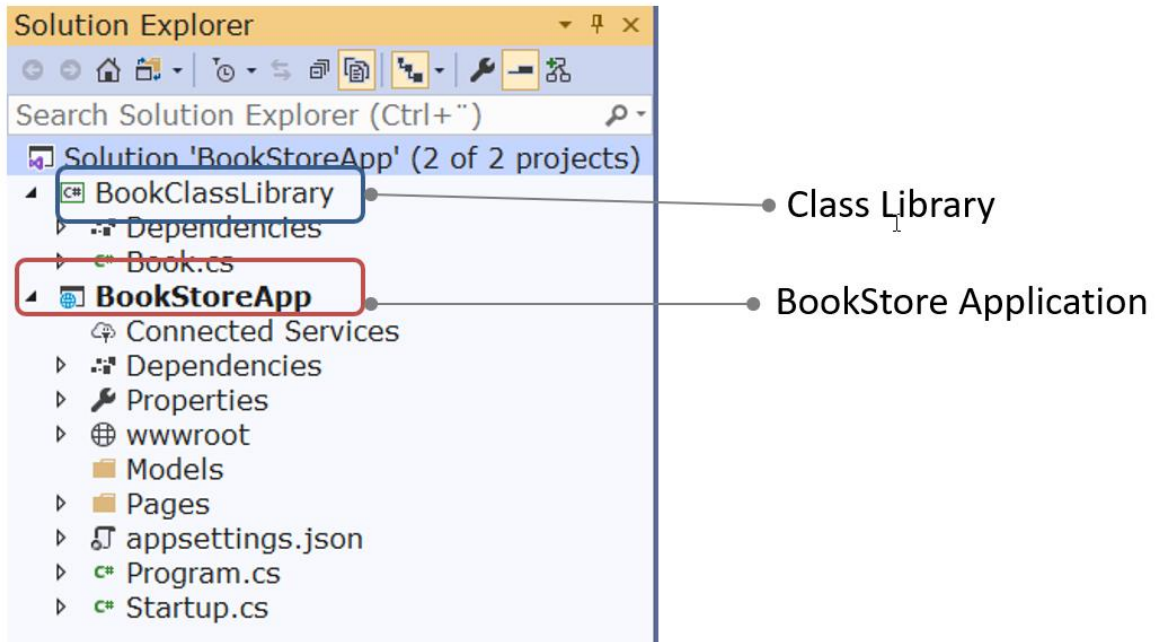
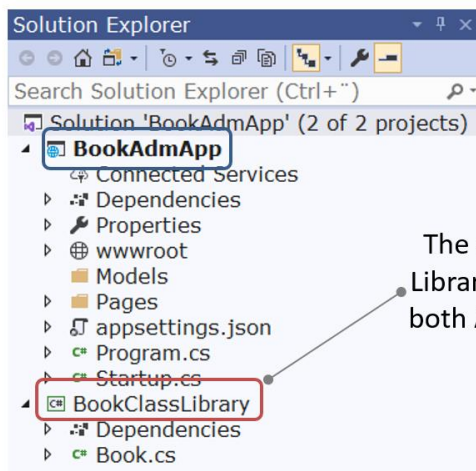


Figure 22-15

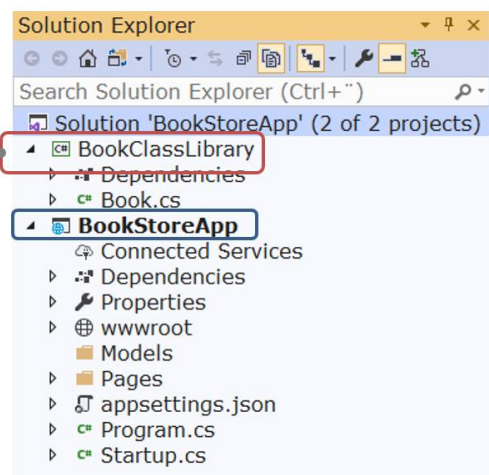
22.2 Final System

Solution Explorer

Application #1



Application #2



The same Class Library is used by both Applications

Figure 22-16

File Explorer

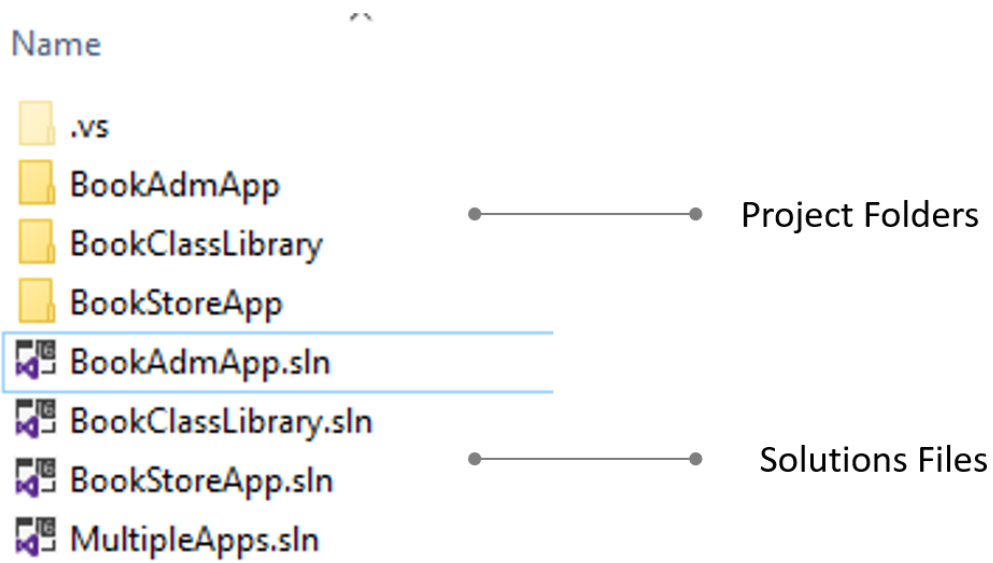


Figure 22-17

23 Web API

Or REST API, Web Service – many names for the same.

Part 9 User Login and ASP.NET Core Identity

Introduction to User Login and ASP.NET Core Identity.

24 User Identity and Login

In this chapter we will see how we can create and use login functionality in your ASP.NET Core Web Applications.

Typically, you need to create functionality for User Registration, Login, etc. Here you will see how this can be done from scratch. If you do it from scratch, you will have full control of your code.

If you use something called "ASP.NET Core Identity" (which will be explained and demonstrated in the next chapter) lots of "magic" happens behind the curtains. If something not working, it may be more complicated to figure out why.

24.1 Password Security

Keeping your passwords safe is important and all software systems should take this seriously.

Password security mechanism:

- Encryption and Decrypting
- Hashing
- Salting
- 2 Factor Authentication
- Etc.

These password security mechanisms will be described in more detail below.

24.1.1 Encryption and Decrypting

Encryption is the practice of scrambling information in a way that only someone with a corresponding key can unscramble and read it.

Encryption is a two-way function. When you encrypt something, you are doing so with the intention of decrypting it later.

To encrypt data, you use an algorithm. Many different encryption algorithms do exist

Figure 24-1 gives an overview of the concepts of Encryption and Decryption.

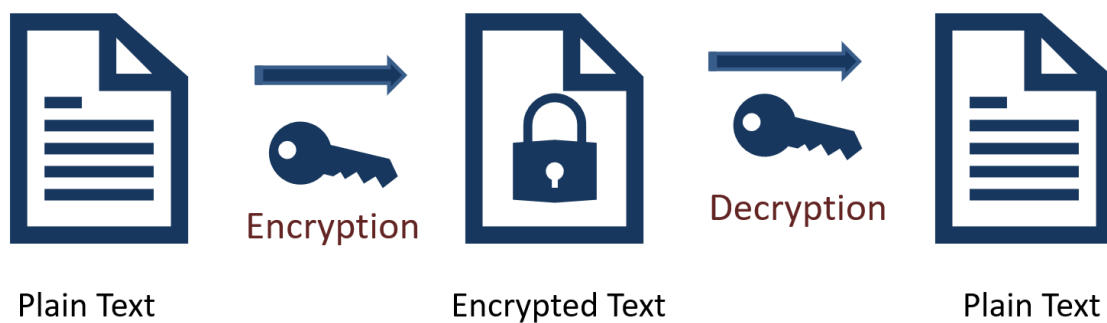


Figure 24-1: Encryption and Decryption

When should encryption be used? Here are some examples:

- Encryption is a two-way function.
- You encrypt information with the intention of decrypting it later.
- Examples when to use encryption:
 - Protecting Files and Information on your Computer
 - Protecting your Cloud data
 - Transmitting Data between 2 Computers
 - Etc.

The key is that Encryption is reversible. Hashing is not.

24.1.2 Hashing

Hashing is the practice of using an algorithm to map data of any size to a fixed length. Encryption is a two-way function. Hashing is a one-way function.

While it is technically possible to reverse-hash something, the computing power required makes it unfeasible. Hashing is one-way. See Figure 24-2.

Encryption is meant to protect data in transit, hashing is meant to verify that a file or piece of data has not been altered—that it is authentic. In other words, it serves as a checksum. Every hash value is unique.

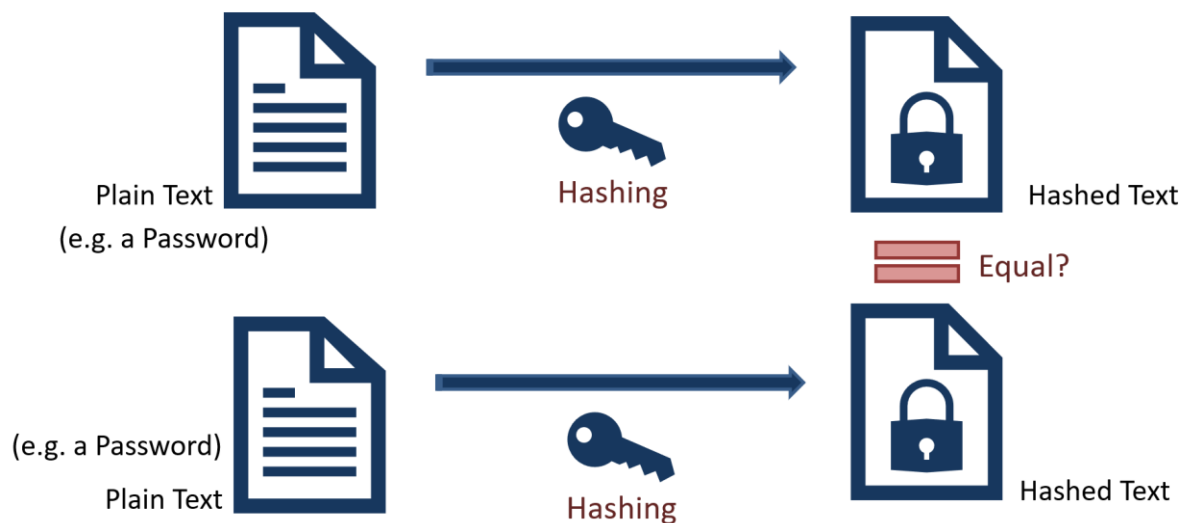


Figure 24-2: Hashing

Is it possible for a hacking to get access to Hashed Passwords?

By using something called “Rainbow Table” the hacker can get access to your hashed password, see Figure 24-3.

Password Table for System X

UserName	HashedPassword
Mike	4420d1918bbcf7
Bob	73fb51a0c9be7d
Peter	4420d1918bbcf7

If a Hacker gets access to this Database, he can see that Mike and Peter have the same password. But he does not know the actual password

Password	HashedPassword
tesla	4420d1918bbcf7
friendship	73fb51a0c9be7d
bicycle	7420e1618abcf6

If the Hacker has access to so-called “Rainbow table” (which is essentially a pre-computed database of hashes), he may also be able to find the Password (as seen here)

Rainbow table

If you have a complicated password, it is less likely that your password is in such a Rainbow table

Figure 24-3: Using Rainbow Table for Hacking your Hashed Password

If a Hacker gets access to this Database, he can see that Mike and Peter have the same password, but he does not know the actual password. If the Hacker has access to a so-called “Rainbow table” (which is essentially a pre-computed database of hashes), he may also be able to find the Password, as seen in Figure 24-3. If you have a complicated password, it is less likely that your password is in such a Rainbow table.

24.1.3 Salting

Salting is a technique typically used for Password Hashing. It is a unique value that can be added to the end of the password to create a different hash value. The additional value is referred to as a “salt”. This is done to make it even more secure. Typically, the Hashing Algorithm uses a Random salt. This prevents an attacker from seeing whether users have the same password. See Figure 24-4.

```
password = "Password123"
salt = "Tesla"

passwordHashed = HashPassword(password, salt);
```

Typically, Salting is built into the Hashing Algorithm and it is changed every time

```
password = "Password123"

ph1 = HashPassword(password);
ph2 = HashPassword(password);
```

ph1 \neq ph2

This means if 2 different Users use the same Password, the Hashed Password will be different!

Figure 24-4: Salting

Is it possible to hack “Hashing with Salt”?

Assume Mike and Peter use the same Password, see Table 24-1. If a hacker gets access to this database, he cannot see that Mike and Peter have the same password. This is because a random Salt has made these 2 Hashed Passwords different!

Table 24-1: Examples of Hashed Passwords with Salt

User Name	Hashed Password with Salt
Mike	4420d1918bbcf7
Bob	73fb51a0c9be7d

Peter	4520d1818cbcf7
-------	----------------

Figure 24-5 shows a typical Flow when Creating User and Login.

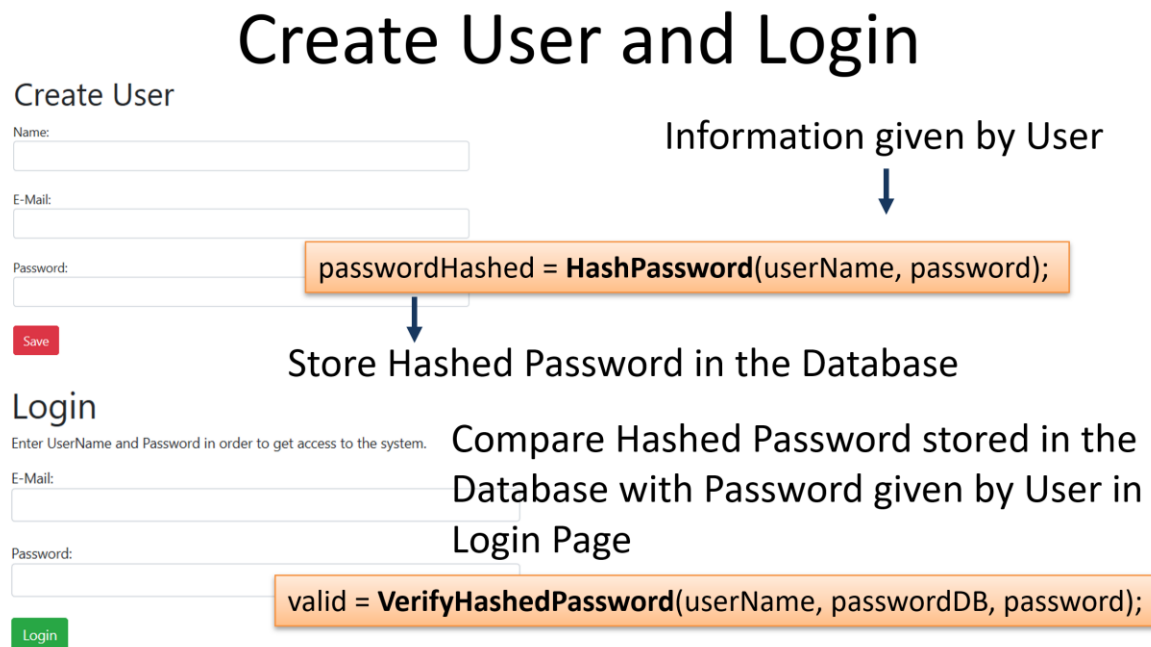


Figure 24-5: Typical Flow when Creating User and Login

24.2 Microsoft.AspNetCore.Identity

This Namespace contains different Classes and Methods for Identity handling. We will use the **PasswordHasher<TUser>** Class.

24.2.1 PasswordHasher<TUser> Class

Namespace: Microsoft.AspNetCore.Identity

2 important Methods:

- **HashPassword**(TUser, String)

Returns a hashed representation of the supplied password for the specified user.

- **VerifyHashedPassword**(TUser, String, String)

Returns a PasswordVerificationResult indicating the result of a password hash comparison.

Example:

```
using Microsoft.AspNetCore.Identity;
...
string username; //UserName given by user when creating a User

string passwordHashed;

PasswordHasher<string> pw = new PasswordHasher<string>();

passwordHashed = pw.HashPassword(userName, password);
```

24.3 Session State in ASP.NET Core

We need to store information whether the User is logged in or not. We can use Session variables in order to share that information between multiple web pages.

Session management in ASP.NET Core is not enabled by default.

- You need to install the **Microsoft.AspNetCore.Session** NuGet Package in order to use Session state.
- You need to **enable Session State** in the **Startup.cs** file
- You need to include the Namespace using **Microsoft.AspNetCore.Http**;

24.4 Demo Application

Here we will demonstrate how we can create a web application (see Figure 24-6) with “Login”, including “Create New User”, “Update User Information”.

Figure 24-6 shows the main page of the “Login” application.

LoginApp Home Weather User

Welcome

ASP.NET Core Web Application

© Developed by [Hans-Petter Halvorsen](https://www.halvorsen.blog) (<https://www.halvorsen.blog>)

© Developed by [Hans-Petter Halvorsen](https://www.halvorsen.blog) (<https://www.halvorsen.blog>)

Figure 24-6: LoginApp - Welcome Web Page

LoginApp Home Weather User

User


Please Login.

Login

Figure 24-7: User needs to Login before he can see Information

24.4.1 Login

LoginApp Home Weather User



Login

Enter UserName and Password in order to get access to the system.

E-Mail:

Password:

[Create User](#)



© Developed by  Hans-Petter Halvorsen (<https://www.halvorsen.blog>)

Figure 24-8: Login Web Page

24.4.2 Create User

LoginApp Home Weather User



Create User

The Password will be hashed before it is stored in the database. This means that no one can find your password even if the database was hacked.

Name:

E-Mail:

Password:


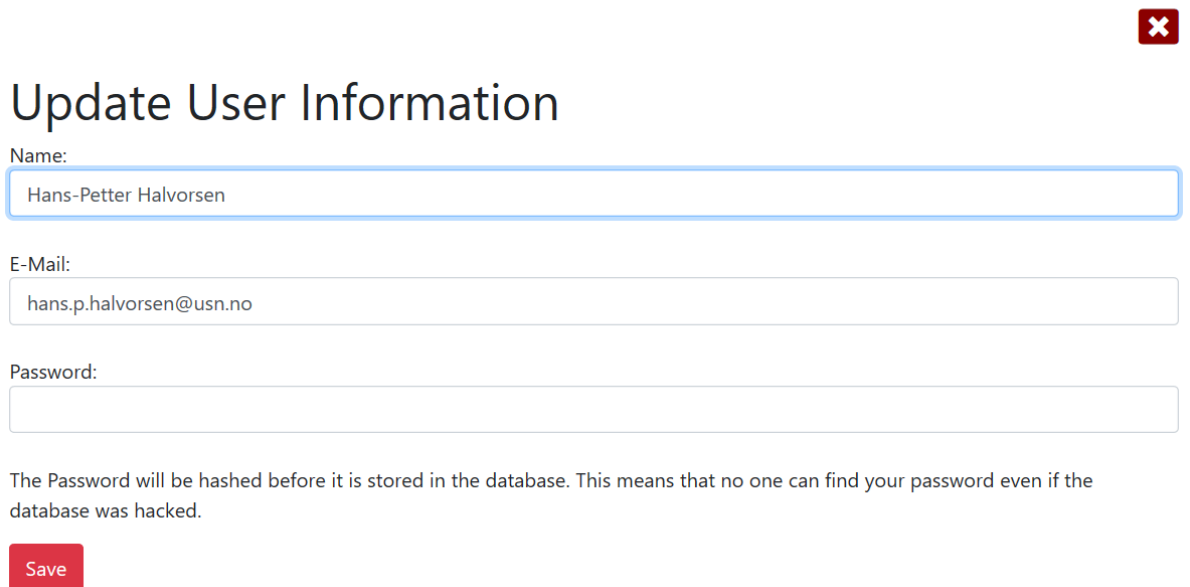
© Developed by  Hans-Petter Halvorsen (<https://www.halvorsen.blog>)

Figure 24-9: Create User Web Page

...

24.4.3 Update User Information

LoginApp Home Weather User



Name:
Hans-Petter Halvorsen

E-Mail:
hans.p.halvorsen@usn.no

Password:

The Password will be hashed before it is stored in the database. This means that no one can find your password even if the database was hacked.

Save

Figure 24-10: Update User Information Web Page

24.4.4 More Features

The web application presented is very basic and only to illustrate the basic principles.

All modern systems offer what we call “2 Factor Authentication”. This means in addition to enter the password, the user needs to enter a one-time password received on E-Mail or SMS.

An alternative is to use an Authenticator App like “Google Authenticator” or “Microsoft Authenticator” available on iPhone and Android.

Another basic feature is “Forgot Password?”. Today we have lots of accounts on different systems. It is recommended that we use different Passwords for these accounts, and it is easy to forget the password for one or more of these accounts. Because of that we need to have “Forgot Password” functionality. This means that the user needs to enter his e-mail address and then the system should send an email (or SMS or similar) with a new temporary Password that the user needs to change once he is able to logon to the system again.

To increase security it is also normal to have some kind of keyword (What is your Nickname?, What is your favorite Pet?, etc.) that the user needs to remember before he can receive a new password.

25 ASP.NET Core Identity

25.1 Introduction

We will use ASP.NET Core Identity for creating an Application with built-in Authentication that has the following features:

- User Registration
- User Login
- Check if User is Authenticated/Logged into your Application
- 2FA

ASP.NET Core Identity is an API that supports user interface (UI) login functionality out of the box. You can manage users, passwords, roles, email confirmation, 2FA, and more.

Users can create an account with the login information stored in Identity or they can use an external login provider.

Supported external login providers include Facebook, Google, Microsoft Account, and Twitter.

ASP.NET Core Identity offers GUI for creating Users and User Login, 2FA, etc. If you need to change the layout and behaviors of those “out of the box” provided GUIs, you need to use “Scaffolding”. Scaffolding is explained below in more detail.

25.1.1 Scaffold Identity in ASP.NET Core Projects

What is Scaffolding?

In general, Scaffolding, also called scaffold or staging is a temporary structure used to support a work crew and materials to aid in the construction, maintenance, and repair of buildings, etc.

<https://en.wikipedia.org/wiki/Scaffolding>

Scaffolding, as used in computing, refers to one of two techniques: The first is a code generation technique related to database access in some model–view–controller frameworks; the second is a project generation technique supported by various tools.

[https://en.wikipedia.org/wiki/Scaffold_\(programming\)](https://en.wikipedia.org/wiki/Scaffold_(programming))

Scaffold Identity in ASP.NET Core Projects

Applications that include Identity can apply the scaffolder to selectively add the source code contained in the Identity Razor Class Library (RCL).

You might want to generate source code so you can modify the code and change the behavior. For example, you could instruct the scaffolder to generate the code used in login or registration.

Generated code takes precedence over the same code in the Identity RCL.

25.2 Demo Application

Below an ASP.NET Core Web Application implementing and using ASP.NET Core Identity will be presented.

25.2.1 Create Project in Visual Studio with Identity Enabled

We start by creating an ordinary ASP.NET Core Web Application, see Figure 25-1.

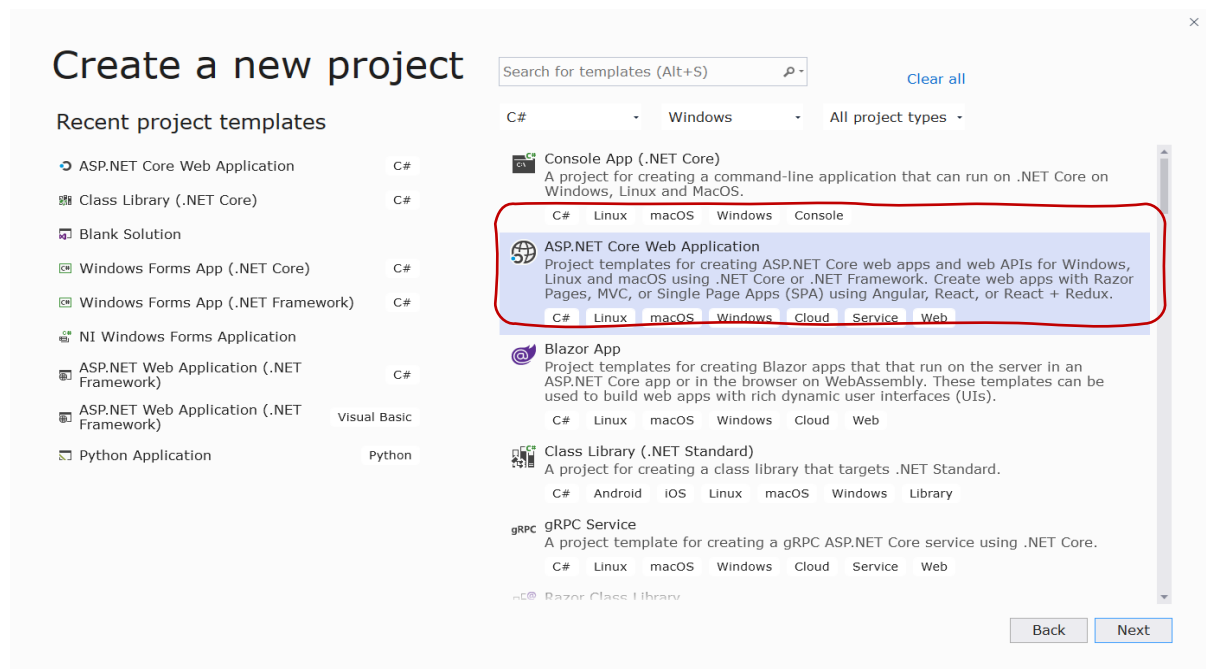


Figure 25-1: ASP.NET Core Web Application Template

Select Authentication:

Then, in the next window (see Figure 25-2) you need to select “Authentication”.

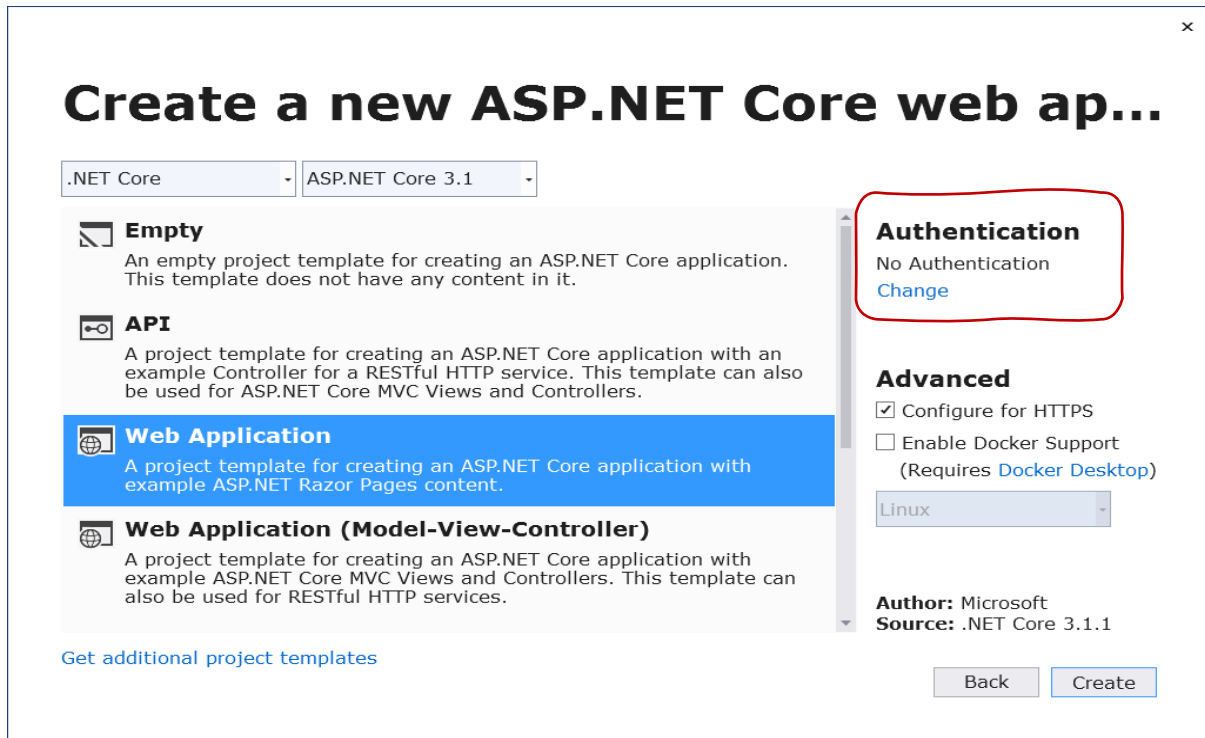


Figure 25-2: Select Authentication

Change Authentication:

When you click “Authentication” in Figure 25-2, the “Change Authentication” window appears (see Figure 25-3). Here you then should typically select “Individual User Accounts”.

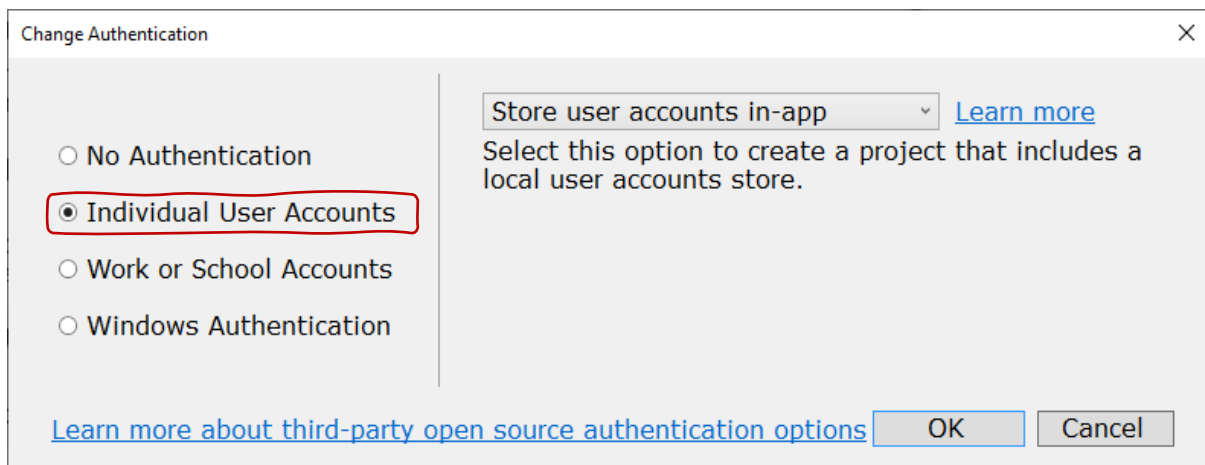


Figure 25-3: Change Authentication

25.2.3 Register New Account and Log In

Create New Account:

IdentityApp Home Privacy Register Login

Register

Create a new account.

Email

Password

Confirm password

[Register](#)

Use another service to register.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

© 2020 - IdentityApp - [Privacy](#)

Figure 25-7: Create New Account

Confirm Registration:

IdentityApp Home Privacy

Register Login

Register confirmation

This app does not currently have a real email sender registered, see [these docs](#) for how to configure a real email sender. Normally this would be emailed: [Click here to confirm your account](#)

© 2020 - IdentityApp - [Privacy](#)

Figure 25-8: Confirm Registration

Login:

IdentityApp Home Privacy

Register Login

Log in

Use a local account to log in.

Email

Password

 Remember me?[Forgot your password?](#)[Register as a new user](#)

Use another service to log in.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

Figure 25-9: Login

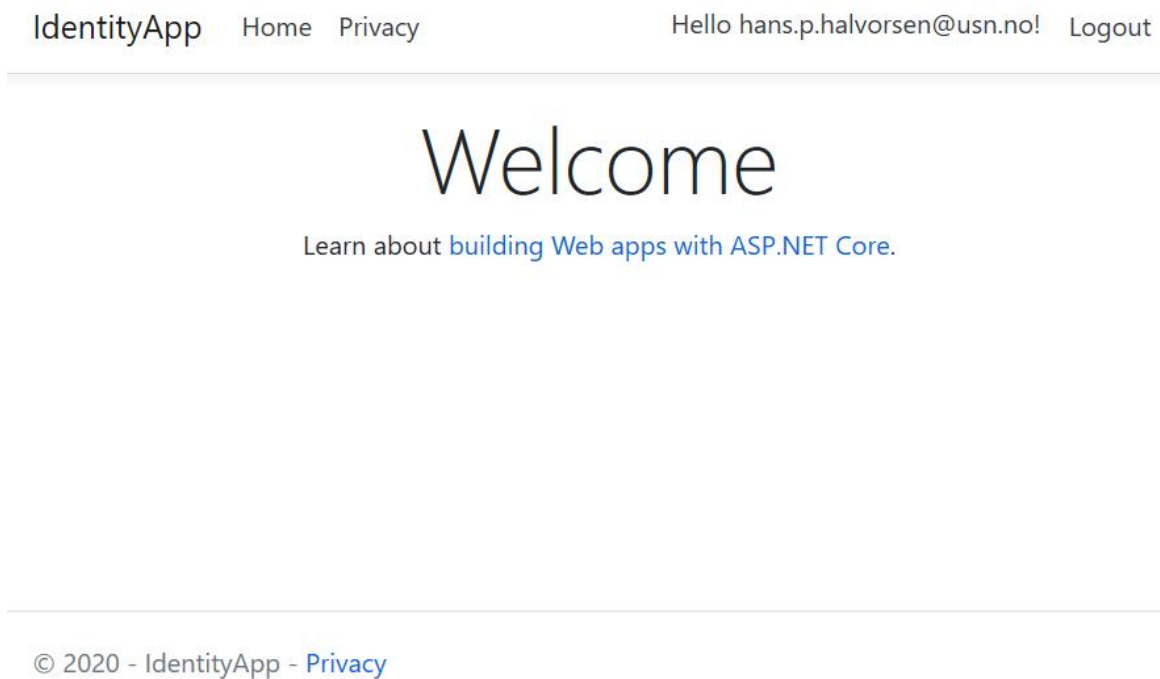


Figure 25-10: You are Logged In

Manage your Account:

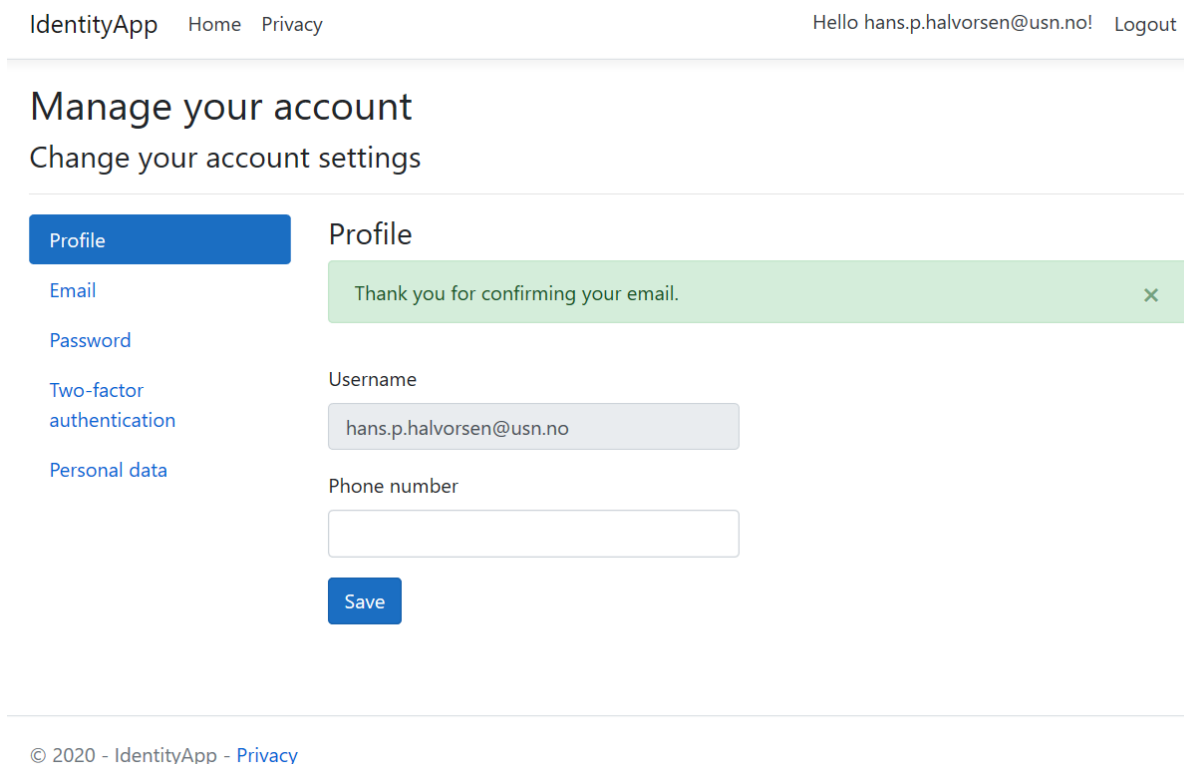


Figure 25-11: Manage your Account

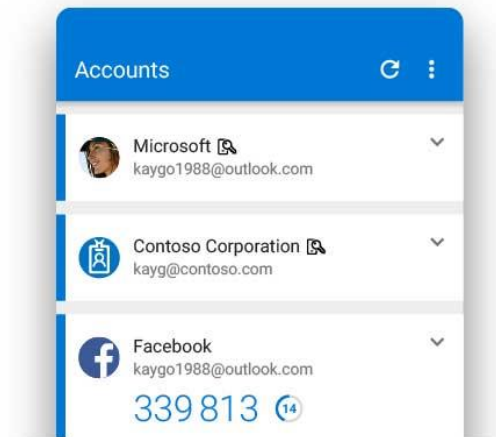


Figure 25-13: Microsoft Authenticator App for 2FA

Microsoft Authenticator App

<https://www.microsoft.com/en-us/account/authenticator>

Log In using 2FA:

Enter Code from Authentication App

IdentityApp Home Privacy Register Login

Two-factor authentication

Your login is protected with an authenticator app. Enter your authenticator code below.

Authenticator code

Remember this machine

[Log in](#)

Don't have access to your authenticator device? You can [log in with a recovery code](#).

© 2020 - IdentityApp - [Privacy](#)

Figure 25-14: Log In using 2FA

25.2.5 Start Creating your Application

When the Identity features are installed, configured, and set up, you can start creating the rest of your application

Typically, you need to check in your different web pages if the user is logged in (authenticated) or not

@User.Identity

Typically, you want to use the following:

- @User.Identity.IsAuthenticated
- @User.Identity.Name

Check if you are Logged In or not in your Code:

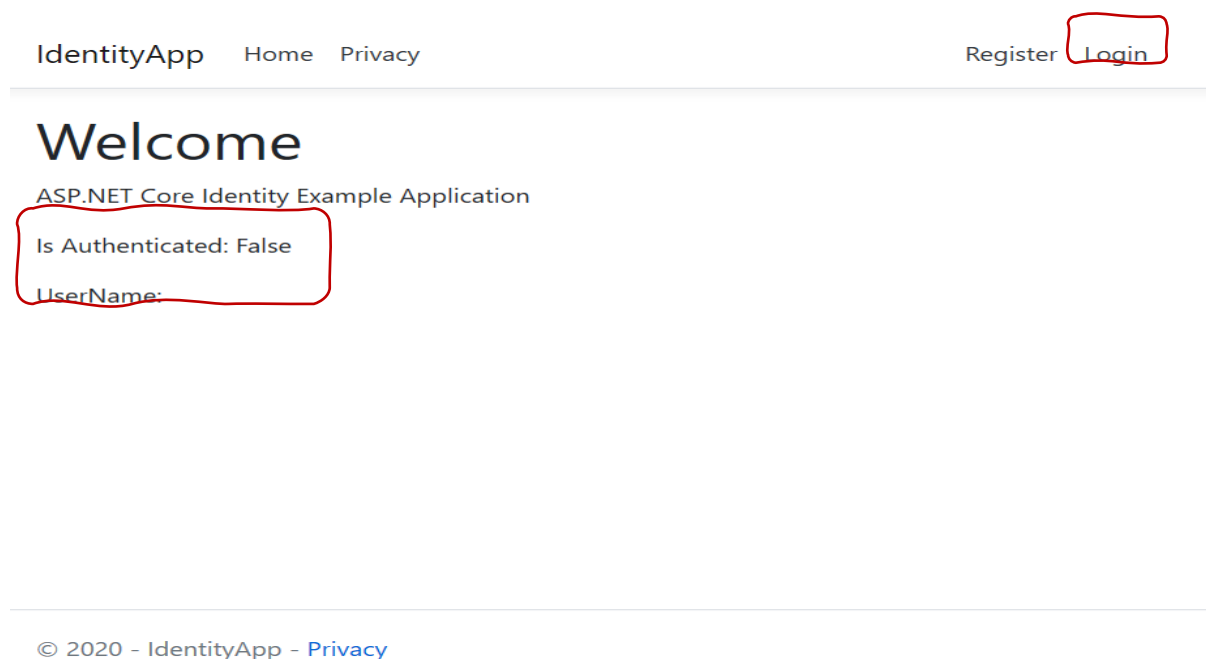


Figure 25-15: Check if you are Logged In or not in your Code

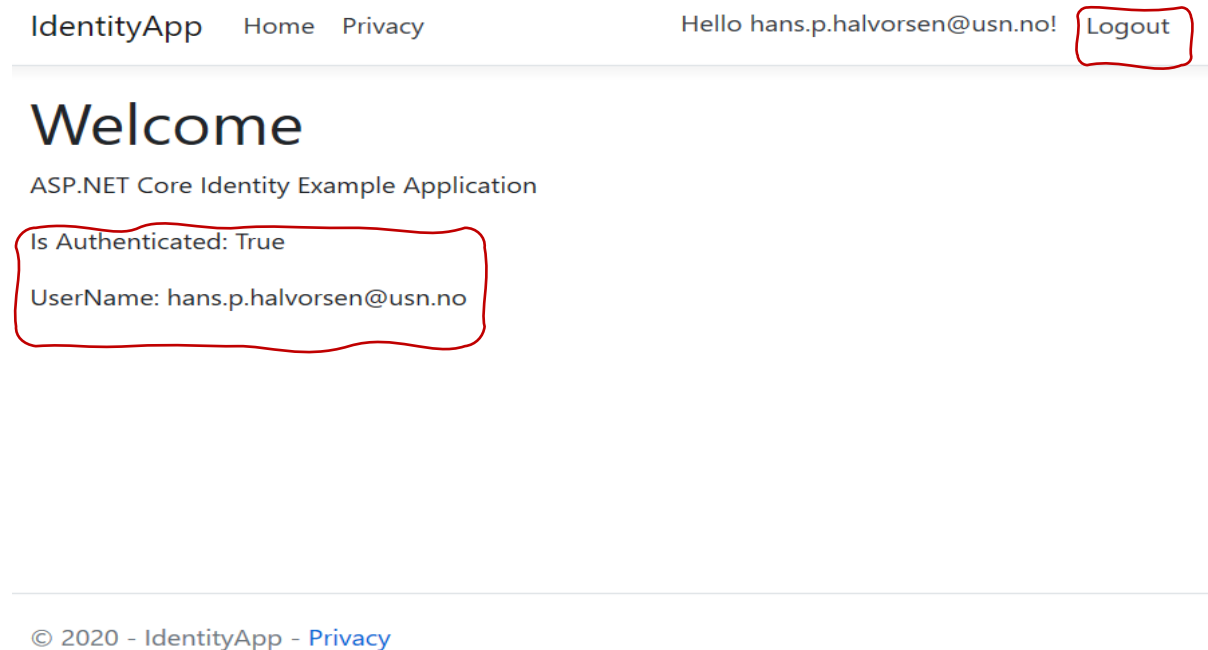


Figure 25-16: User are Logged In

Razor Code:

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}
<div>
    <h1>Welcome</h1>
    <p>ASP.NET Core Identity Example Application</p>
    <p>Is Authenticated: @User.Identity.IsAuthenticated</p>
    <p>UserName: @User.Identity.Name</p>
</div>
```

Typically, we want to use “@User.Identity.IsAuthenticated” for checking if Logged In or not and then show, e.g., data from the database, if the user are logged in.

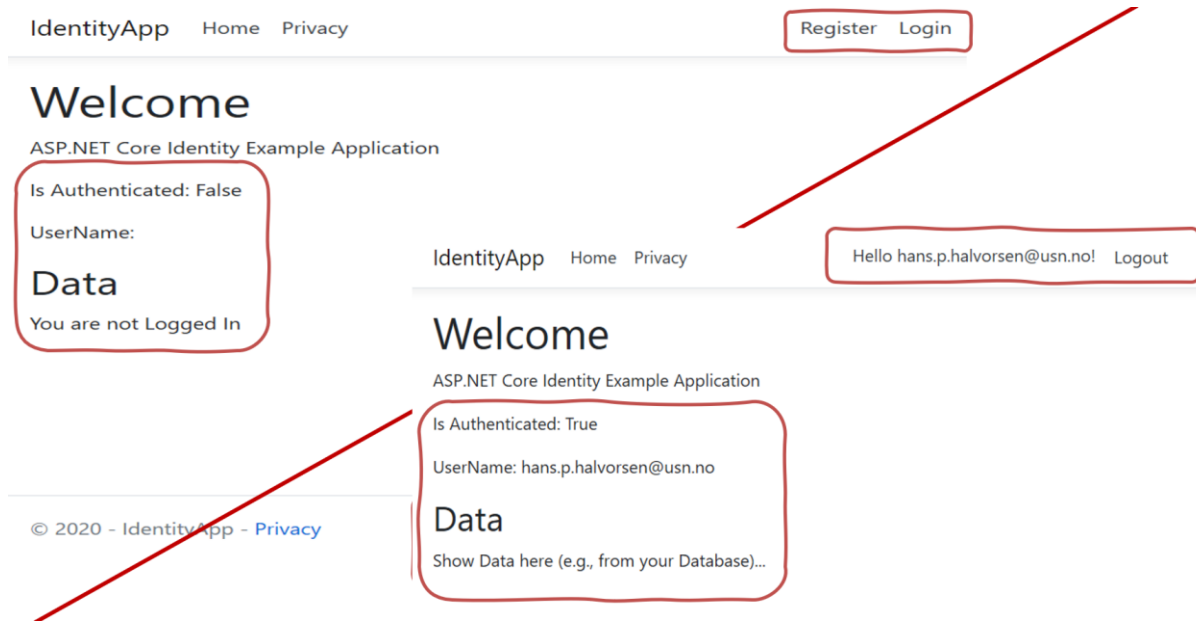


Figure 25-17

Razor Code:

```
<div>
  <h2>Data</h2>
  @if (User.Identity.IsAuthenticated)
  {
    <p>Show Data here (e.g., from your Database)...</p>
  }
  else
  {
    <p>You are not Logged In</p>
  }
</div>
```

25.2.6 Scaffolding

If you are not happy with the default Layout of the different Identity web pages (Register, Login, etc.) You can override the default Scaffolding and modify these web pages, so they fit your needs.

Scaffold Identity in ASP.NET Core Projects:

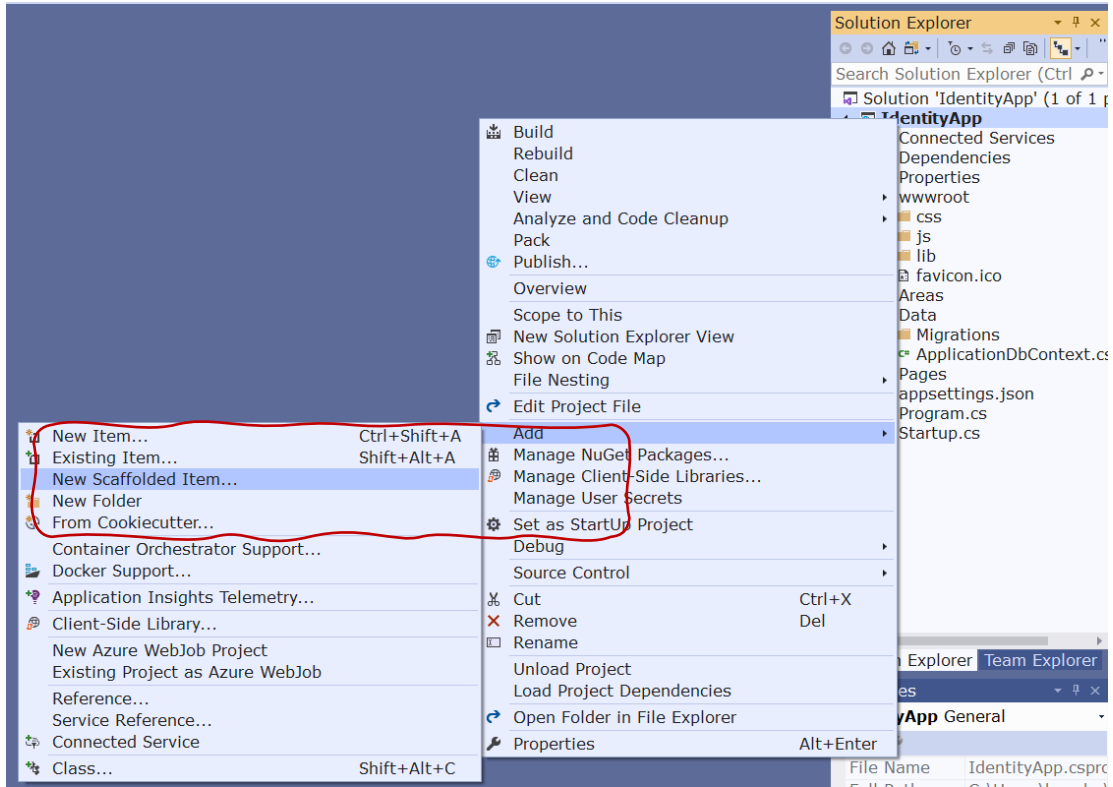


Figure 25-18

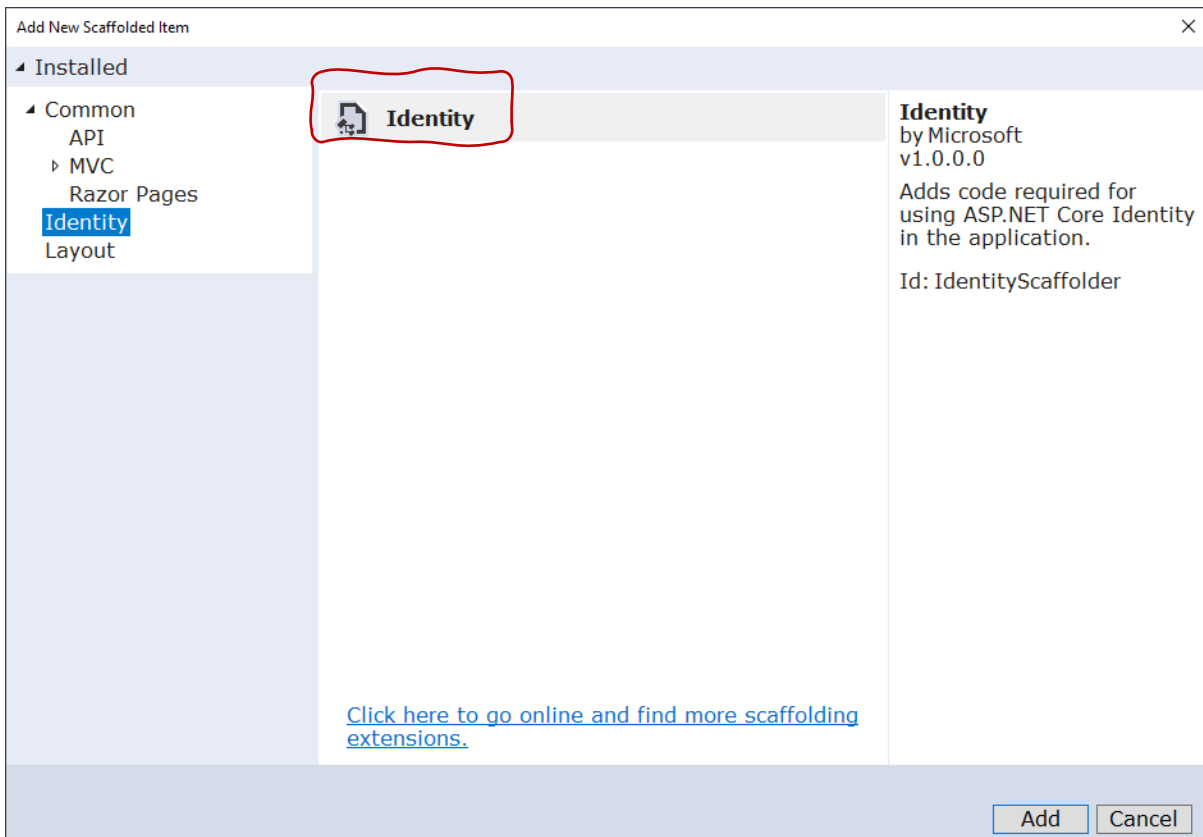


Figure 25-19

Select the Identity page(s) you want to override

Add Identity ✕

Select an existing layout page, or specify a new one:

...

(Leave empty if it is set in a Razor _viewstart file)

Override all files

Choose files to override

<input type="checkbox"/> Account\StatusMessage	<input type="checkbox"/> Account\AccessDenied	<input type="checkbox"/> Account\ConfirmEmail
<input type="checkbox"/> Account\ConfirmEmailChange	<input type="checkbox"/> Account\ExternalLogin	<input type="checkbox"/> Account\ForgotPassword
<input type="checkbox"/> Account\ForgotPasswordConfirmation	<input type="checkbox"/> Account\Lockout	<input checked="" type="checkbox"/> Account\Login
<input type="checkbox"/> Account\LoginWith2fa	<input type="checkbox"/> Account>LoginWithRecoveryCode	<input type="checkbox"/> Account\Logout
<input type="checkbox"/> Account\Manage\Layout	<input type="checkbox"/> Account\Manage\ManageNav	<input type="checkbox"/> Account\Manage\StatusMessage
<input type="checkbox"/> Account\Manage\ChangePassword	<input type="checkbox"/> Account\Manage\DeletePersonalData	<input type="checkbox"/> Account\Manage\Disable2fa
<input type="checkbox"/> Account\Manage\DownloadPersonalData	<input type="checkbox"/> Account\Manage\Email	<input type="checkbox"/> Account\Manage\EnableAuthenticator
<input type="checkbox"/> Account\Manage\ExternalLogins	<input type="checkbox"/> Account\Manage\GenerateRecoveryCode	<input type="checkbox"/> Account\Manage\Index
<input type="checkbox"/> Account\Manage\PersonalData	<input type="checkbox"/> Account\Manage\ResetAuthenticator	<input type="checkbox"/> Account\Manage\SetPassword
<input type="checkbox"/> Account\Manage\ShowRecoveryCodes	<input type="checkbox"/> Account\Manage\TwoFactorAuthenticatio	<input type="checkbox"/> Account\Register
<input type="checkbox"/> Account\RegisterConfirmation	<input type="checkbox"/> Account\ResetPassword	<input type="checkbox"/> Account\ResetPasswordConfirmation

Data context class: +

Use SQLite instead of SQL Server

User class: +

Figure 25-20

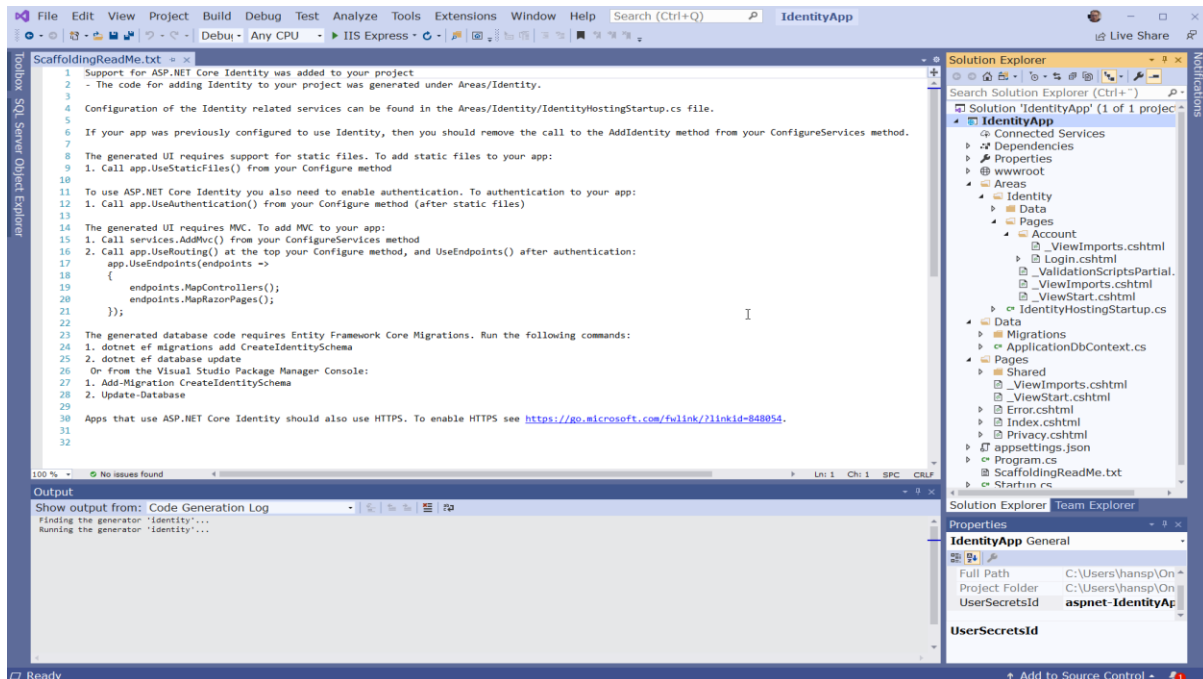


Figure 25-21

Existing Login.cshtml

IdentityApp Home Privacy Register Login

Log in

Use a local account to log in.

Email

Password

Remember me?

[Log in](#)

[Forgot your password?](#)

[Register as a new user](#)

Use another service to log in.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

Want to remove this

Figure 25-22

Updated Login.cshtml

IdentityApp Home Privacy Register Login

Log in

Use a local account to log in.

Email

Password

Remember me?

[Log in](#)

[Forgot your password?](#)

[Register as a new user](#)

Figure 25-23

25.3 Additional Resources

Here are some additional resources if you want to dig deeper into ASP.NET Core Identity:

- Introduction to Identity on ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity>
- Scaffold Identity in ASP.NET Core projects: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity>
- Account confirmation and password recovery in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/acconconfirm>

Part 10 Testing

Introduction to Software Testing.

26 Unit Testing



Introduction to Unit Testing in ASP.NET Core: <https://youtu.be/EzeDCEQ2qMs>

Part 11 Deployment

Introduction to Deployment.

27 Web Servers

When running our web application inside Visual Studio, the IIS Express Web Server is running in the background (without our notice).

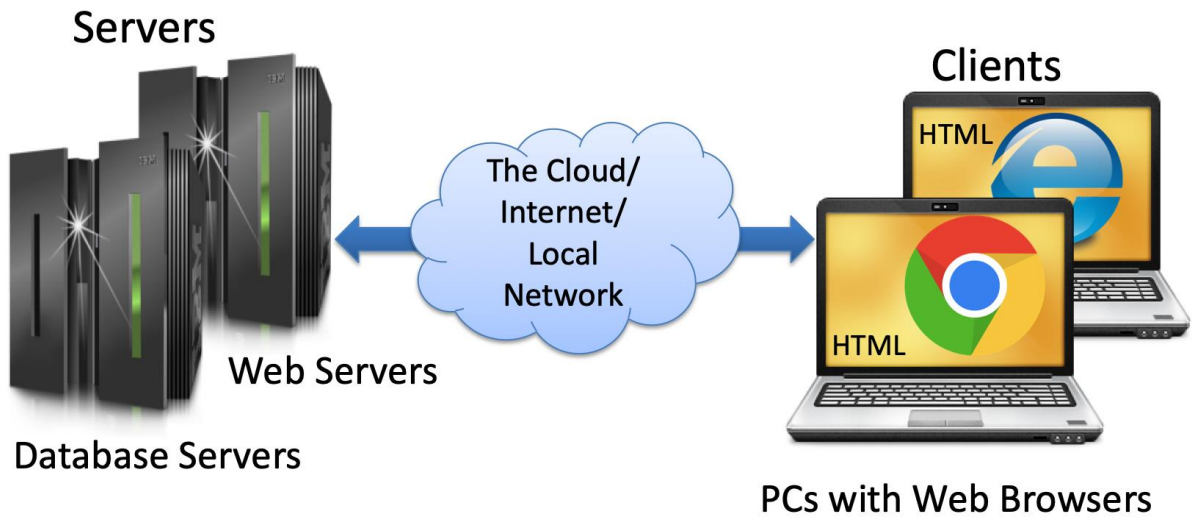
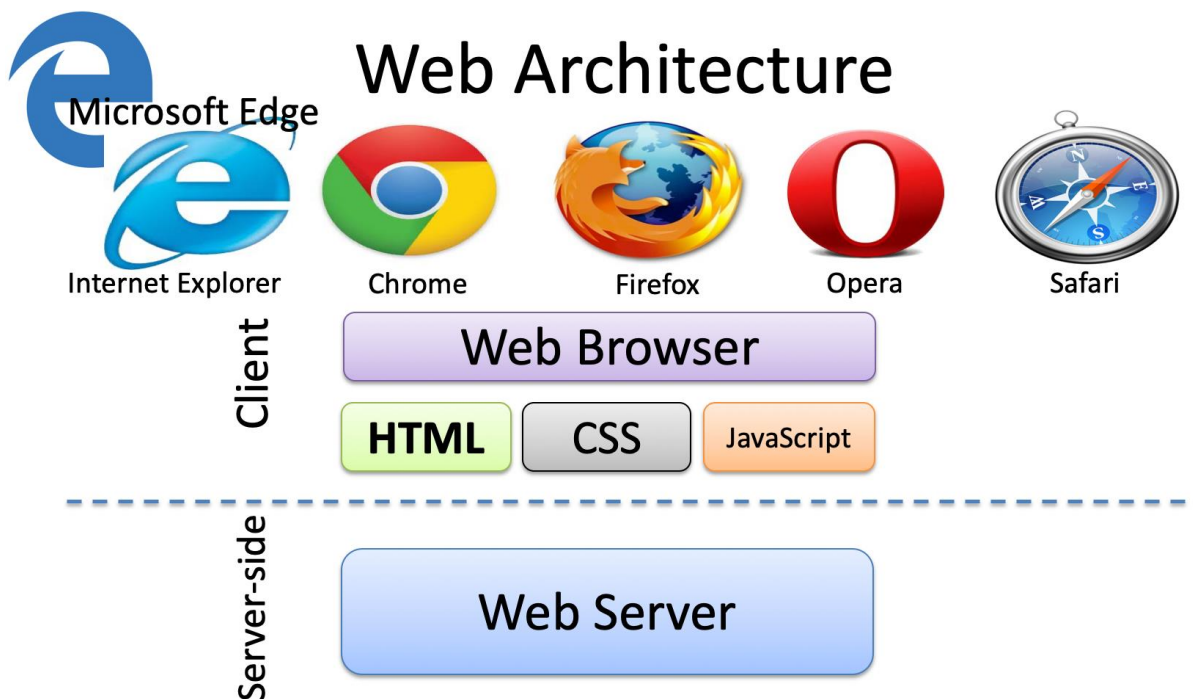
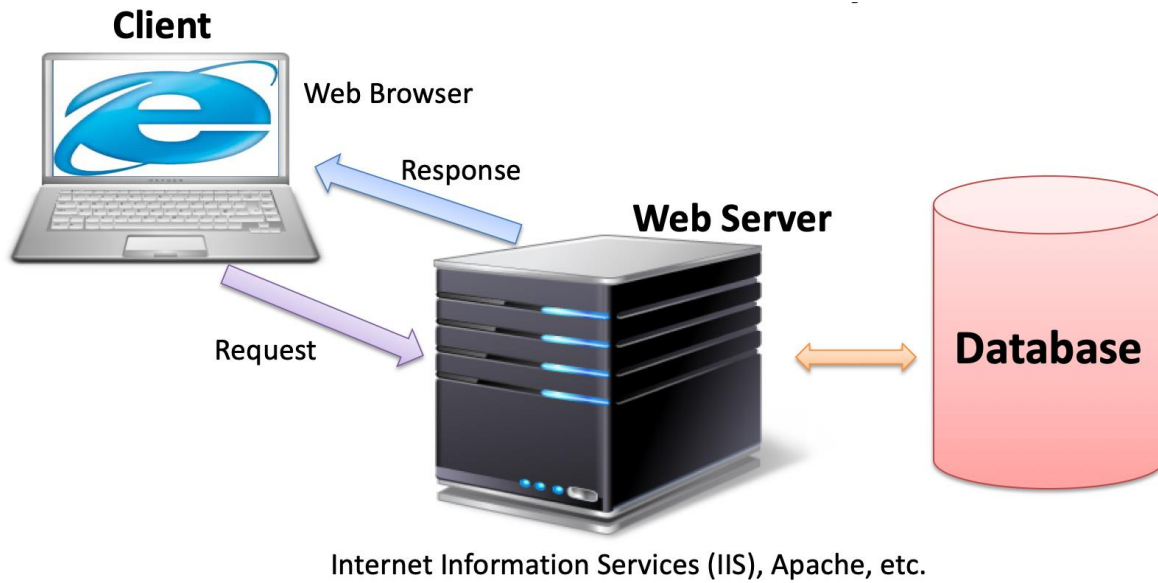


Figure 27-1

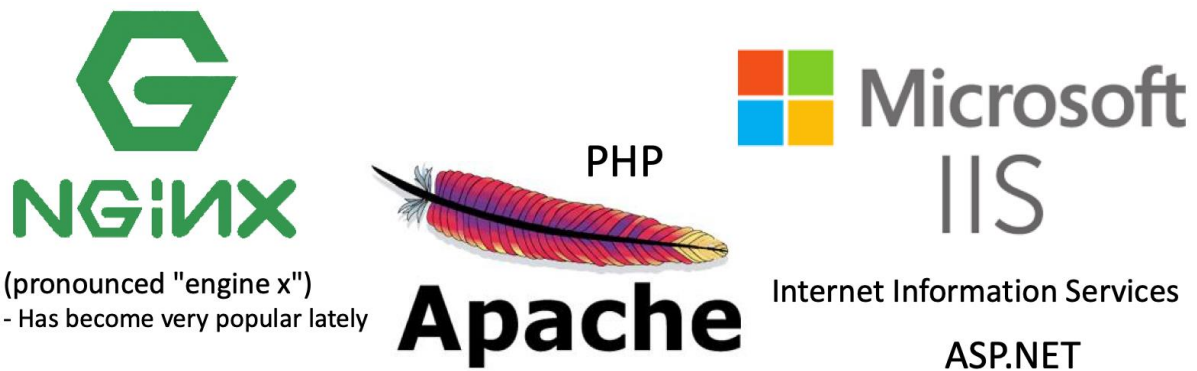


Web Servers are used to host web sites and web pages. The term web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet.



The following web servers (software) are very popular today:

- Internet Information Services (IIS) (included with Windows)
- Apache
- Nginx (pronounced "engine x")



Cross-platform: UNIX, Linux, OS X, Windows, ...

Reference: https://en.wikipedia.org/wiki/Web_server

Product	Vendor	Platform	Percent
Apache	Apache	Open Source Cross Platform	45.9%
Nginx	Nginx, inc	Free + Paid versions	39.0%

		Cross Platform	
IIS (Internet Information Services)	Microsoft	Windows, Included with Windows (Windows Server, Windows 10 Pro)	9.5%
LiteSpeed	LiteSpeed	Proprietary, Linux	3.4%
GWS (Google Web Server)	Google	Custom Linux-based Web server that Google uses for its online services	1.0%

28 Deployment in Visual Studio

Publish your application from Visual Studio.

29 Internet Information Services (IIS)

Internet Information Services (IIS) is a Web Server from Microsoft. IS is integrated with Windows.

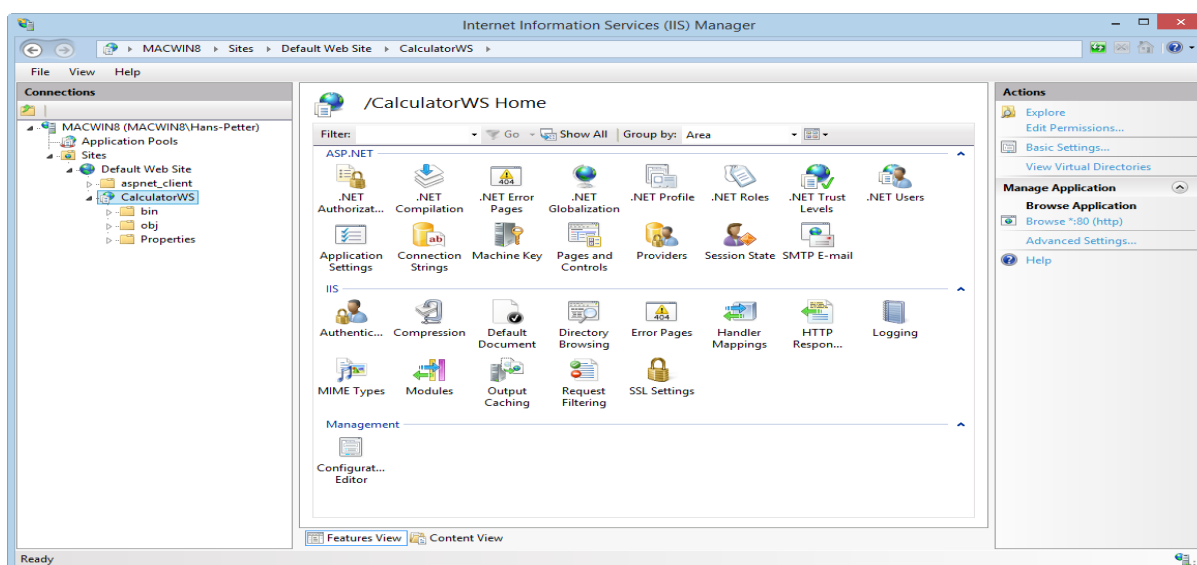


Figure 29-1: Internet Information Services (IIS) Manager

29.1 Installation

IIS – Internet Information Services

Web Server that host the Web Pages/Web Site

Make sure to have the IIS Role installed with ASP.NET subcomponents

Default IIS Directory:

C:\inetpub\wwwroot

In order to install IIS and make sure it work with ASP.NET Core Web Applications, you need to do the following.

- Turn on **Internet Information Services** in Windows Features
- Install the **.NET Core Hosting Bundle** on the IIS server. If the Hosting Bundle is installed before IIS, the bundle installation must be repaired. Run the Hosting Bundle installer again after installing IIS.

29.1.1 Windows Features

Installation/ Configuration

Search for «Windows Features» or open the Control Panel

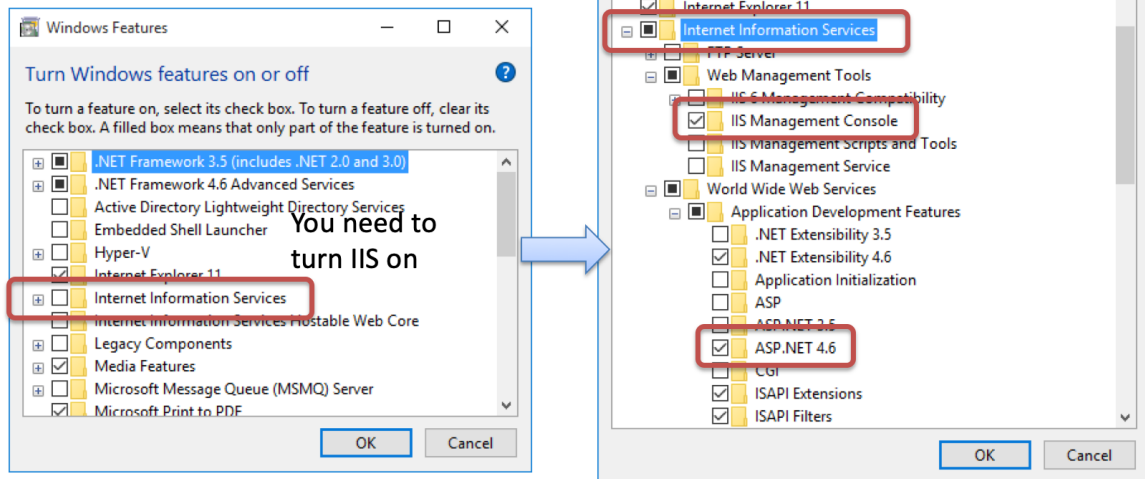


Figure 29-2: Installation of Internet Information Services (IIS)

29.1.2 .NET Core Hosting Bundle

29.2 Demo Application



ASP.NET Core – Web Server IIS deployment: <https://youtu.be/MoI9SSLV4B4>

In the video the ASP.NET Core Database CRUD Application is used as an example. See Chapter 17.

The entire example can be downloaded from the home page of this textbook.

29.2.1 Add Application

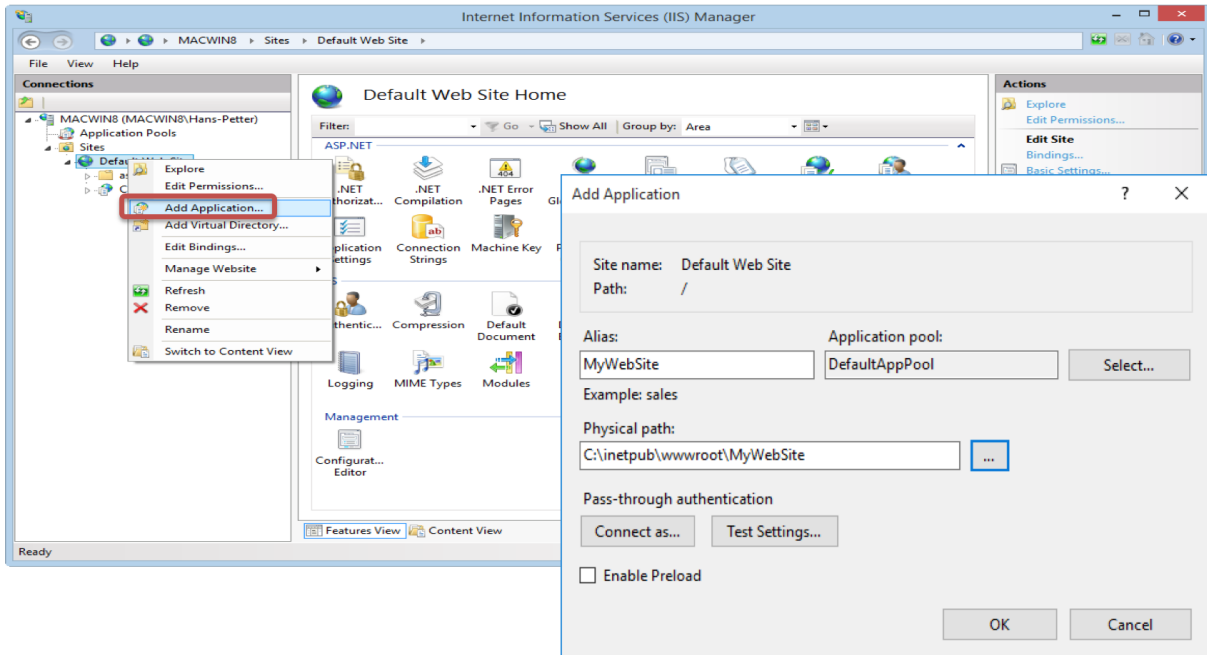


Figure 29-3: IIS – Add Application

Copy your Files to default IIS Directory: C:\inetpub\wwwroot

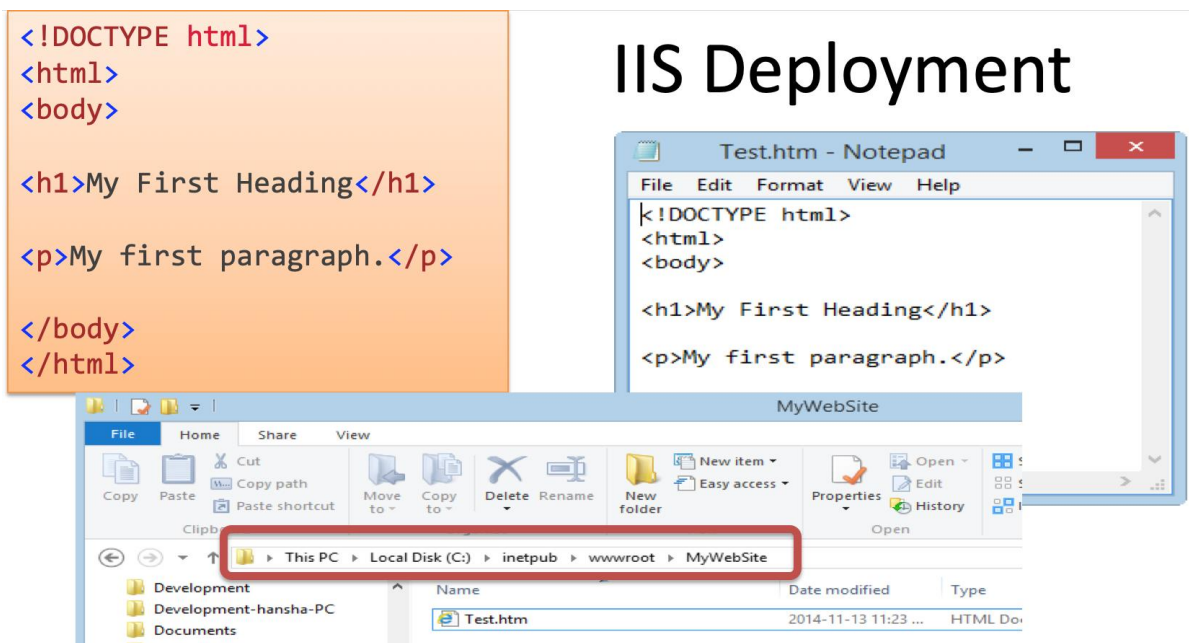


Figure 29-4: Copy Files

IIS Deployment

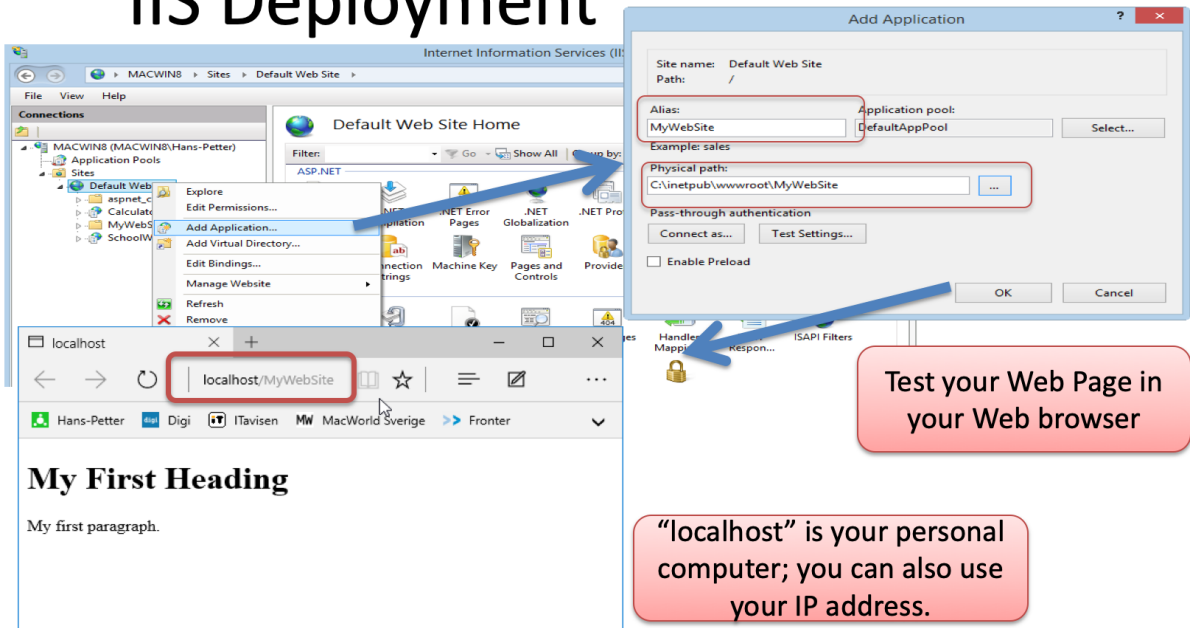


Figure 29-5: IIS Deployment

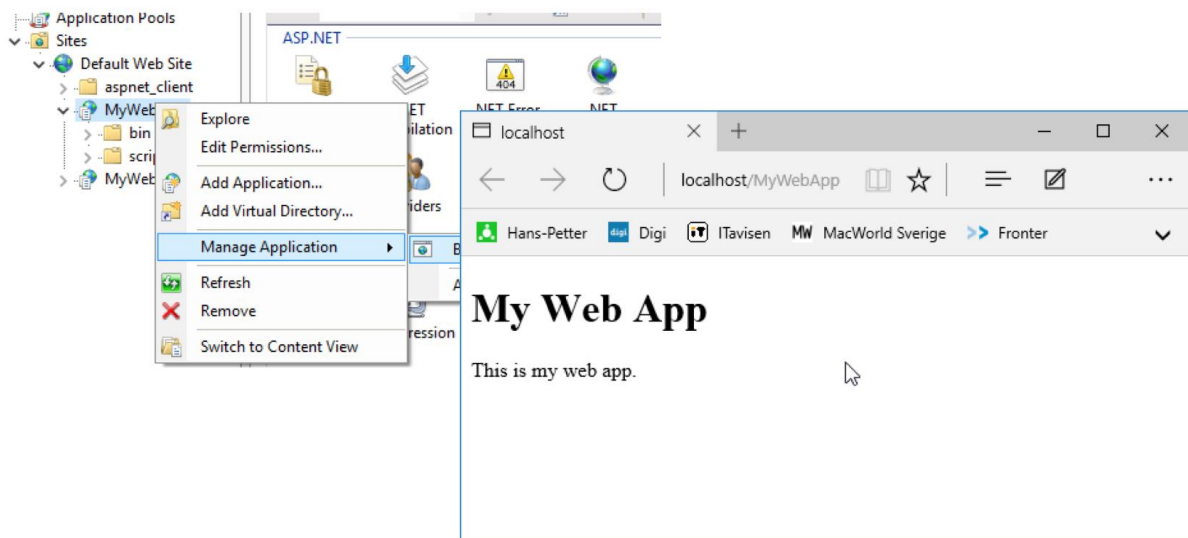


Figure 29-6: Run your Application

Part 12 Microsoft Azure

Introduction to Microsoft Azure.

30 Introduction to Azure

You could say Microsoft Azure is "Windows running in the Cloud".

A cloud platform from Microsoft, a competitor to AWS, Google Cloud Platform, etc.

Web:

<https://azure.microsoft.com/>

30.1 Azure Web Portal

Using the Microsoft Azure Portal, you can configure and setup the different services provided by Microsoft Azure, such as Virtual Machines (VM), databases, web servers, etc.

Web:

<https://portal.azure.com>

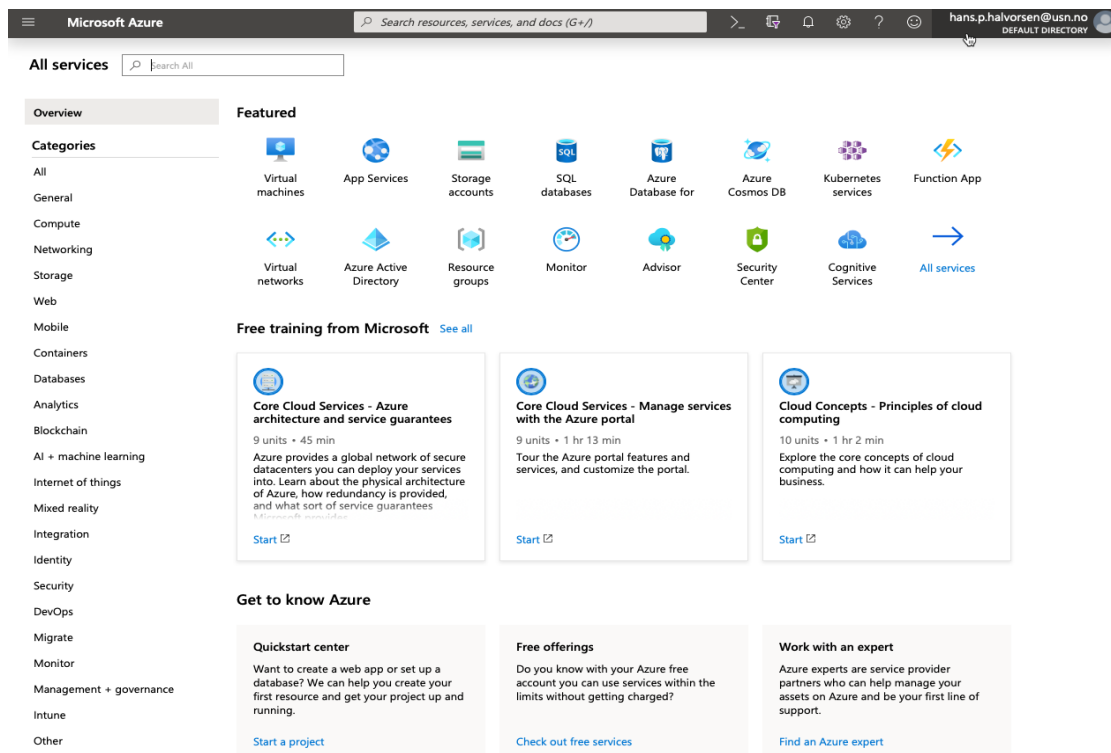


Figure 30-1: Microsoft Azure Portal

31 Databases in Azure

31.1 Create the Database

You use the Microsoft Azure Portal to create and configure your Azure SQL Databases, see Figure 31-1 and Figure 31-2.

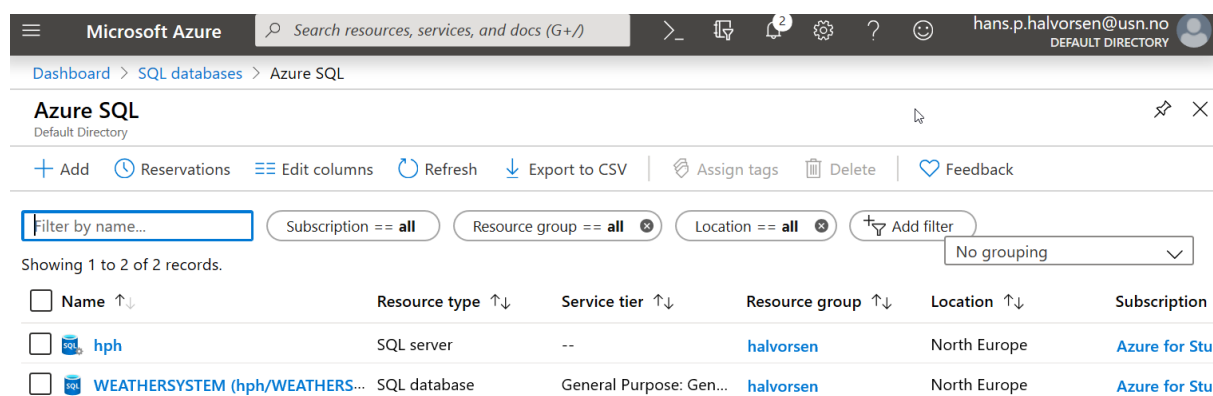


Figure 31-1: Create and Configure your Azure SQL Databases in Azure Portal

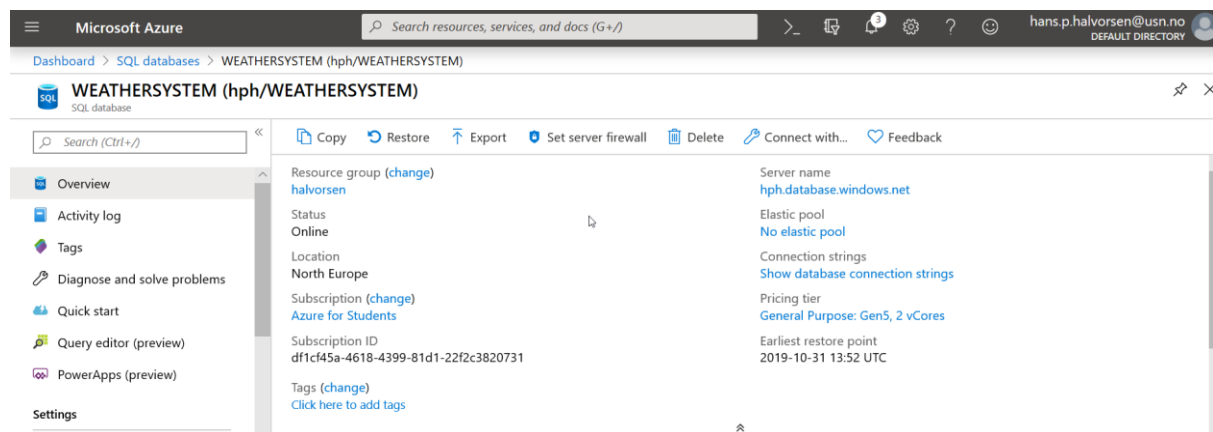


Figure 31-2: Azure SQL Database Configuration

31.1.1 Azure Data Studio

Then we can use Azure Data Studio (or SQL Server Management Studio) to insert the necessary Tables, Stored Procedures, etc. See Figure 31-3.

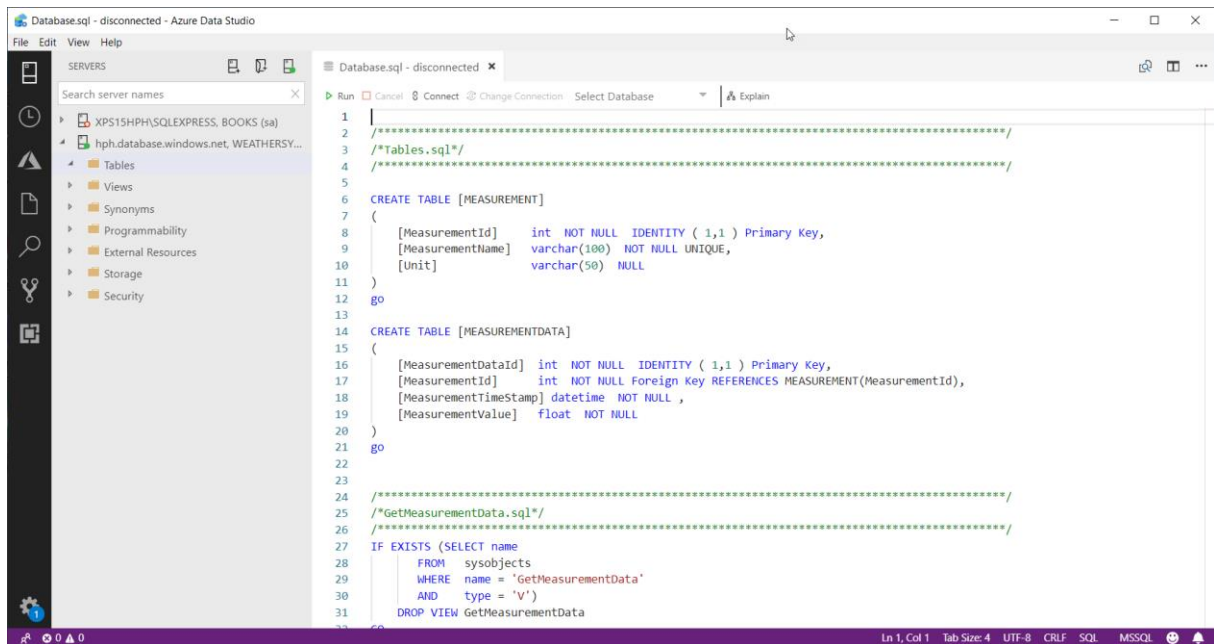


Figure 31-3: Azure Data Studio

Download:

<https://docs.microsoft.com/en-us/sql/azure-data-studio/>

31.2 Create Tables, etc.

Connect to the Windows Azure SQL Server from your local SQL Management Studio.

Configure **Firewall** Setting in Azure Web Portal

Your local Management Studio: You connect to the Windows Azure SQL Server Database in the same way as you connect to a local Database

Create Tables, Views, Stored Procedures, etc. -> using a SQL Script is recommended!

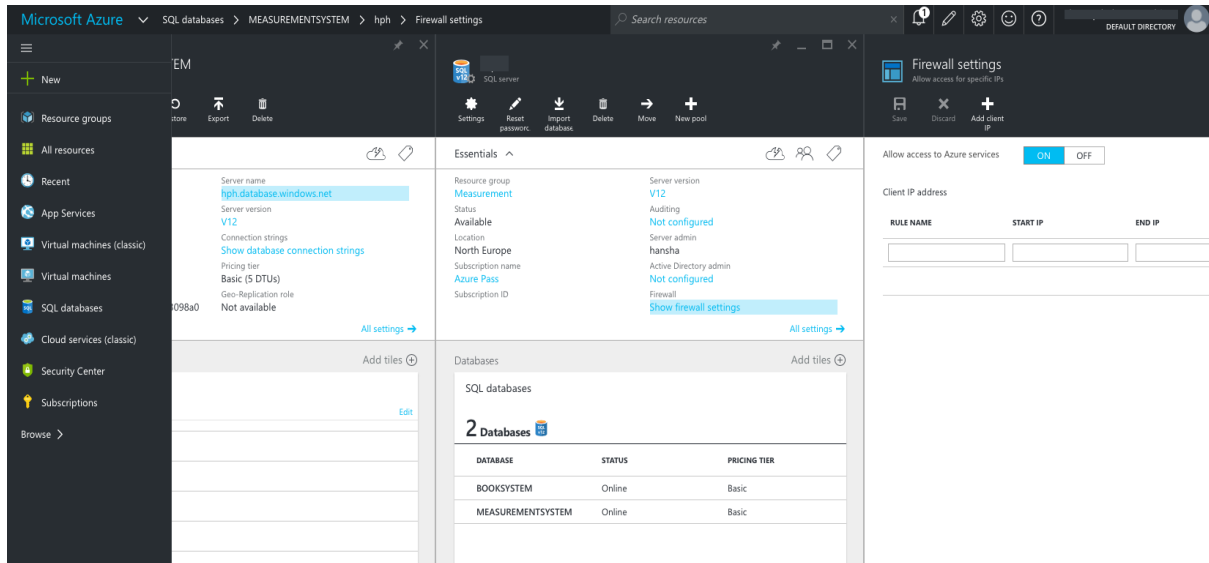


Figure 31-4: Azure Firewall Settings

31.2.1 SQL Server Management Studio

Connect to your Azure database from your local SQL Server Management Studio (SSMS).

31.2.2 Azure Data Studio

You can also use Azure Data Studio.

Azure Data Studio is a cross-platform (Windows, macOS, Linux) down-scaled version of SQL Server Management Studio (Windows only).

Web Site:

<https://docs.microsoft.com/sql/azure-data-studio>

32 Web Applications in Azure

App Services is a container (Web Server) for web pages and other web services like REST API, etc.

You could say it is the Azure version of Internet Information Services (IIS). IIS is a web server software from Microsoft.

32.1 App Service

An “App Service” can be create from Azure Portal (or directly in Visual Studio) as seen in Figure 32-1

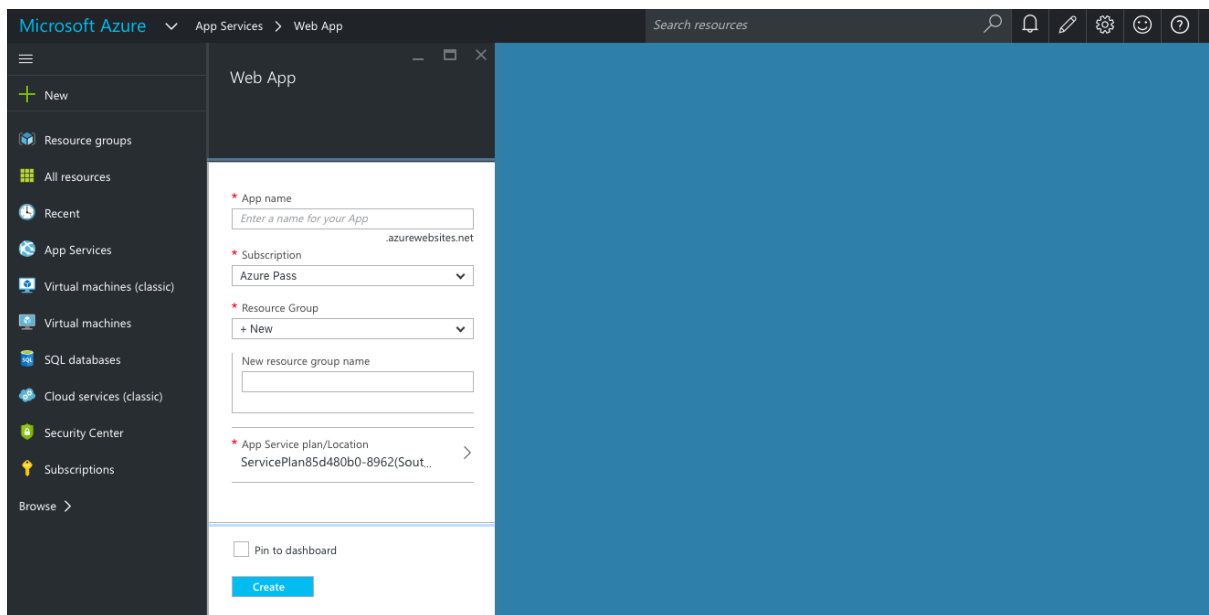


Figure 32-1: Azure App Service

Deploy the Web Project to the Azure Web App from Visual Studio.

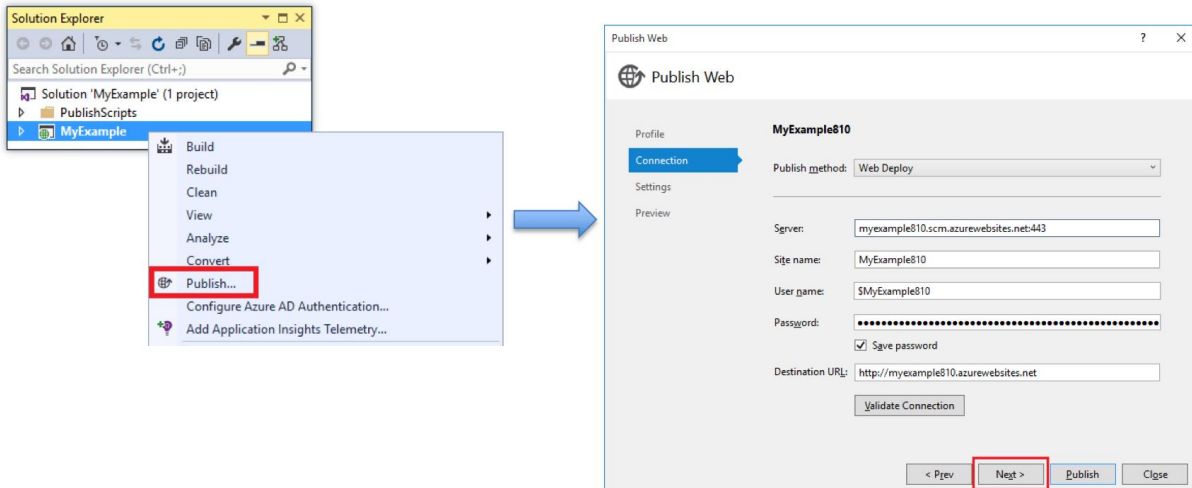


Figure 32-2: Deployment from Visual Studio

32.2 Default Document

Configure Default Document

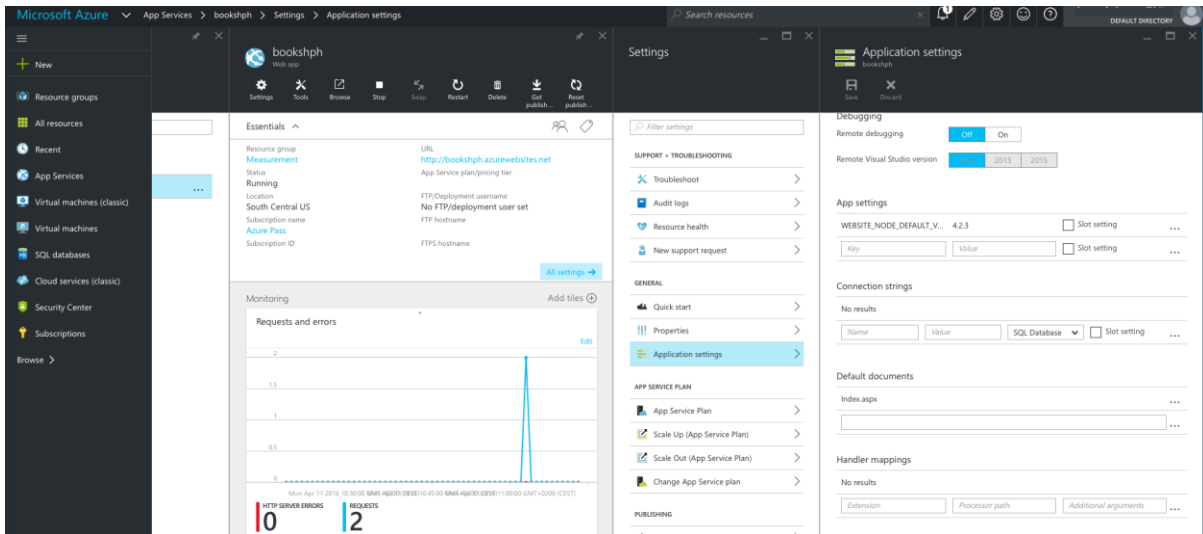


Figure 32-3: Configure Default Document

Part 13 Resources

Additional resources.

33 Bootstrap

Bootstrap is a JavaScript/HTML, CSS Framework.

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.

Bootstrap is included in the standard ASP.NET Core Web Application.

Web Site:

<https://getbootstrap.com>

Bootstrap is a popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web

Bootstrap is a free and open-source front-end web framework for designing websites and web applications.

It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

Unlike many web frameworks, it concerns itself with client-side/front-end development only.

Additional Training:

Bootstrap Tutorial: <https://www.w3schools.com/bootstrap4/>

34 Font Awesome

Font Awesome 5 has a PRO edition with 7020 icons, and a FREE edition with 1535 icons. The FREE edition is a good start and has more than enough icons for most people.

To use the Free Font Awesome 5 icons, you can choose to download the Font Awesome library, or you can sign up for an account at Font Awesome and get a code (called KIT CODE) to use when you add Font Awesome to your web page.

Web Site:

<https://fontawesome.com>

Additional Training:

https://www.w3schools.com/icons/fontawesome5_intro.asp

Part 14 Applications

This part gives an overview of real applications created in ASP.NET Core. A Weather System will be presented and a Data Management System (DMS).

35 Weather System

The Weather Station is located in Porsgrunn (Latitude 59.1386° N, Longitude 9.6555° E), Norway at University of South-Eastern Norway (USN).

Web Site:

<https://www.halvorsen.blog/documents/software/weather/>

The Weather System is based on a Capricorn 2000ex Weather Station with Weather MicroServer from Columbia Weather Systems.

Figure 35-1 shows an overview of the system.

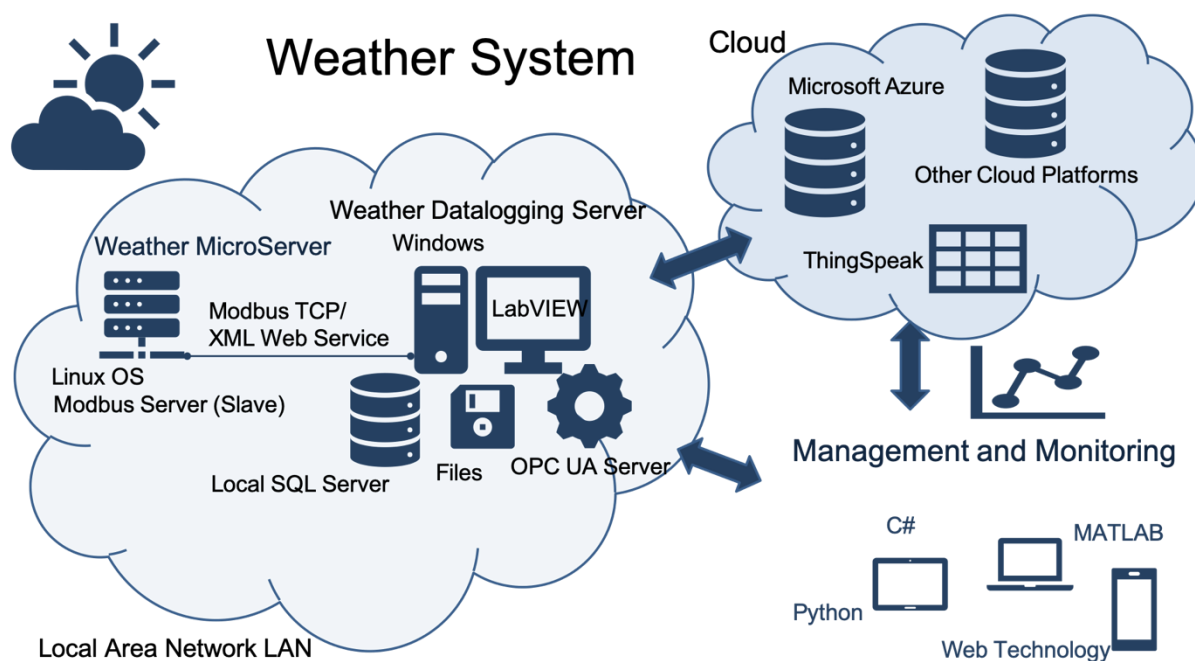


Figure 35-1: Weather System Overview

The system consists of several modules, applications and services.

Main applications:

- Data Logging Applications
- Data Management Applications
- Data Monitoring Applications

Multiple programming languages and frameworks has been used in the development if this system, some examples are LabVIEW, C#, ASP.NET, PHP, JavaScript, CSS, Bootstrap, etc.

Different database systems are also in use, e.g., SQL Server (both local and in Microsoft Azure), MySQL/MariaDB.

Web Servers that are used are Apache running on a Linux server, Internet Information Services (App Service) running on Microsoft Azure.

35.1 ASP.NET Core Web Application

Here, we will focus on the applications that has been made with Visual Studio, C# and ASP.NET Core.

The source code for the application presented below is available from the home page of this textbook.

Figure 35-2 shows the start page for the application.

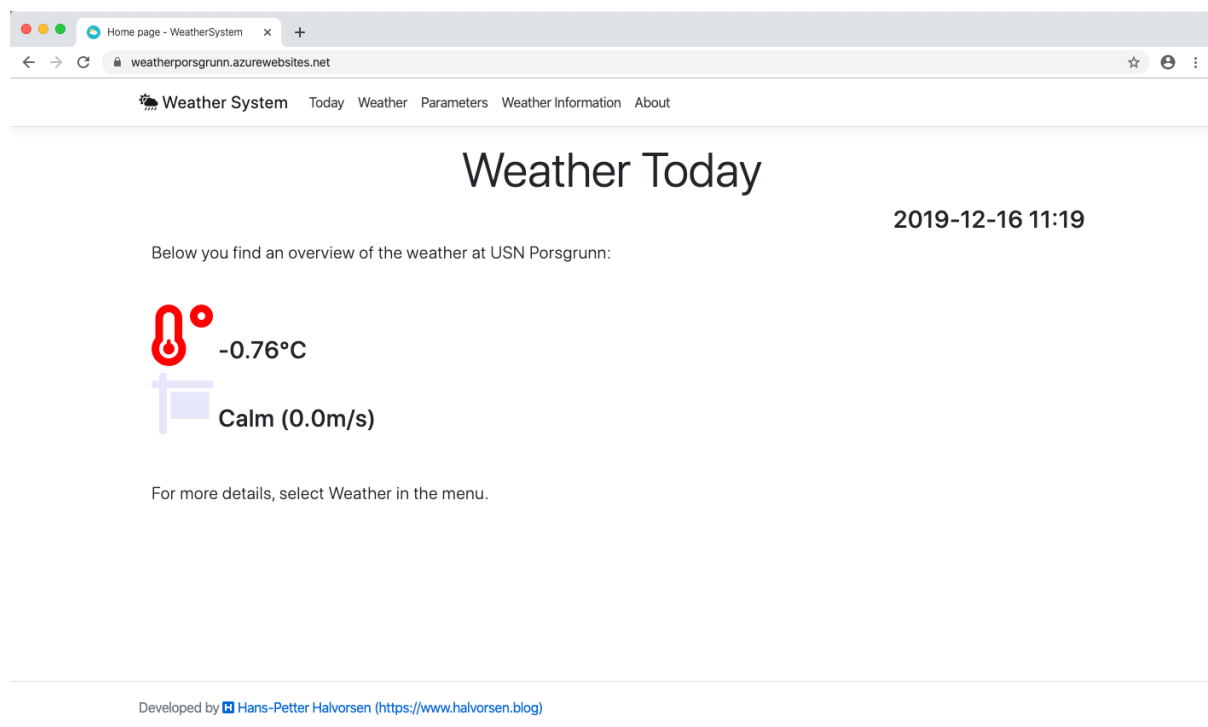


Figure 35-2: Weather System ASP.NET Core Web Application

In the menu we have different files for the different features like:

- Today
- Weather
- Charts
- Parameters
- Weather Information

- About

The development of each of those will be described below.

35.2 Database

The database structure is very basic.

MEASUREMENT table:

```
CREATE TABLE [MEASUREMENT]
(
    [MeasurementId]    int NOT NULL IDENTITY ( 1,1 ) Primary Key,
    [MeasurementName]  varchar(100) NOT NULL UNIQUE,
    [MeasurementAlias] varchar(100) NULL,
    [Unit]             varchar(50) NULL
)
```

MEASUREMENTDATA table:

```
CREATE TABLE [MEASUREMENTDATA]
(
    [MeasurementDataId] int NOT NULL IDENTITY ( 1,1 ) Primary Key,
    [MeasurementId]    int NOT NULL Foreign Key REFERENCES MEASUREMENT(MeasurementId),
    [MeasurementTimeStamp] datetime NOT NULL ,
    [MeasurementValue] float NOT NULL
)
```

Database View (“GetMeasurementData”):

```
IF EXISTS (SELECT name
           FROM sysobjects
           WHERE name = 'GetMeasurementData'
           AND type = 'V')
    DROP VIEW GetMeasurementData
GO

CREATE VIEW GetMeasurementData
AS

SELECT
    MEASUREMENTDATA.MeasurementDataId,
    MEASUREMENT.MeasurementId,
    MEASUREMENT.MeasurementName,
    MEASUREMENT.MeasurementAlias,
    MEASUREMENTDATA.MeasurementTimeStamp,
    MEASUREMENTDATA.MeasurementValue,
    MEASUREMENT.Unit

FROM MEASUREMENTDATA
INNER JOIN MEASUREMENT ON MEASUREMENTDATA.MeasurementId = MEASUREMENT.MeasurementId
GO
```

Stored Procedure (“SaveMeasurementData”):

```
IF EXISTS (SELECT name
           FROM sysobjects
           WHERE name = 'SaveMeasurementData')
```

```

AND type = 'P')
DROP PROCEDURE SaveMeasurementData
GO

CREATE PROCEDURE SaveMeasurementData
@MeasurementName varchar(100),
@Unit varchar(50),
@MeasurementValue float
AS

DECLARE
@MeasurementId int

if not exists (select * from MEASUREMENT where MeasurementName = @MeasurementName)
insert into MEASUREMENT (MeasurementName, Unit) values (@MeasurementName,
@Unit)

select @MeasurementId = MeasurementId from MEASUREMENT where MeasurementName =
@MeasurementName

insert into MEASUREMENTDATA (MeasurementId, MeasurementValue, MeasurementTimeStamp)
values (@MeasurementId, @MeasurementValue, getdate())

GO

```

35.3 Visual Studio Project

We create a new ASP.NET Core Web Application in Visual Studio.

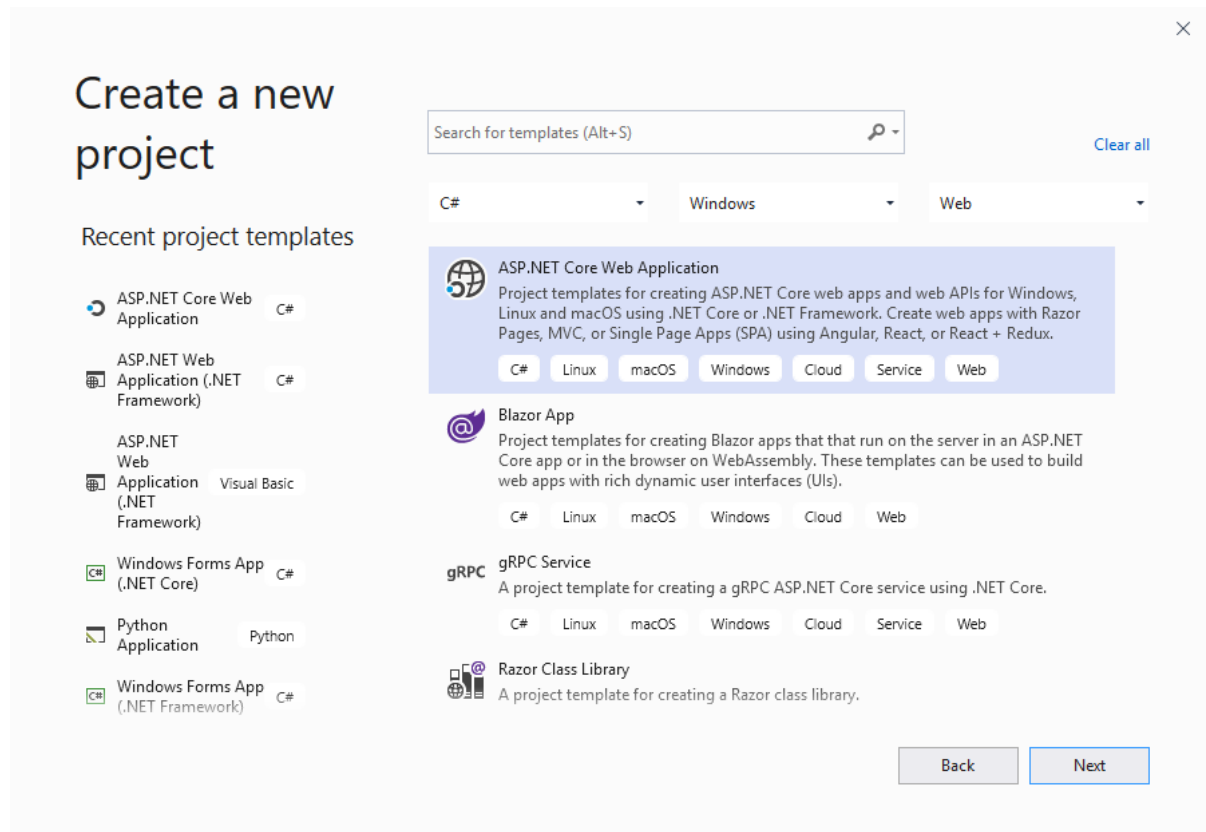


Figure 35-3 ASP.NET Core Web Application Project in Visual Studio

In the “Pages” folder we create our web pages (.cdhtml files).

In this application we have the following Pages:

- Index.cshtml
- Weather.cshtml
- Chart.cshtml
- WeatherParameters.cshtml
- WeatherInformation.cshtml
- About.cshtml

We create a “Models” folder where we put our Classes.

In this application we have the following Classes:

- WeatherParameters.cs
- WeatherData.cs
- ChartData.cs

35.4 Connection String

You can hardcode the connection string to the database in your C# code like this:

```
string connectionString = "DATA SOURCE=xxx;UID=sa;PWD=xxx;DATABASE=xxx";
```

where you replaced the “xxx” with the settings for your database.

The user “sa” (System Administrator) is the default user. You can use that one for testing, but for your final application you should setup and use another user.

35.4.1 appSettings.json

A better solution is to put the connection string inside the “**appSettings.json**” which is meant for storing configuration data, such as connection strings, etc.

Below you see a typical “appSettings.json” file:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
```

```

"ConnectionString": "DATA SOURCE=xxx;UID=sa;PWD=xxx;DATABASE=xxx"
}
}

```

where you replace the “xxx” with the settings for your database.

35.5 Index Page (Start Page)

The start page of an ASP.NET Core web application is by default “Index.cshtml”

In this web application, the index file shows the current weather in a formatted way.

Code for “Index.cshtml”:

```

@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Weather Today</h1>
</div>

<div class="text-lg-right">
    <h3>@DateTime.Now.ToString("yyyy-MM-dd HH:mm")</h3>
</div>

<div class="container-fluid lead">

    Below you find an overview of the weather at USN Porsgrunn:

    <p>&nbsp;</p>

    <h3>@Html.Raw(Model.weatherTemperatureText)</h3>

    <h3>@Html.Raw(Model.weatherWindSpeedText)</h3>

    <p>&nbsp;</p>

    For more details, select Weather in the menu.

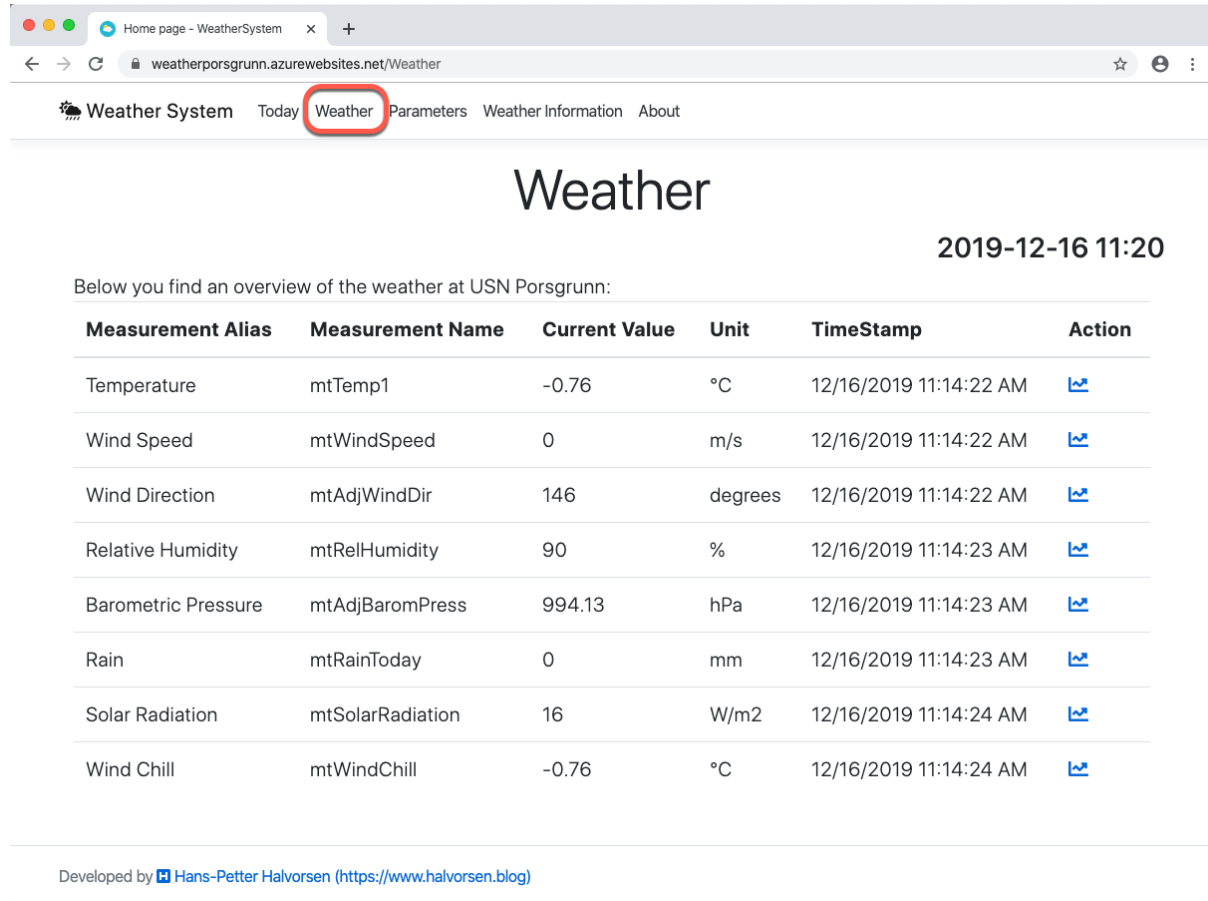
</div>

```

Here more weather information can be added.

35.6 Weather Overview Page

Weather.cshtml



Home page - WeatherSystem x +

weatherporsgrunn.azurewebsites.net/Weather

Weather System Today **Weather** Parameters Weather Information About

Weather

2019-12-16 11:20

Below you find an overview of the weather at USN Porsgrunn:

Measurement Alias	Measurement Name	Current Value	Unit	TimeStamp	Action
Temperature	mtTemp1	-0.76	°C	12/16/2019 11:14:22 AM	Chart
Wind Speed	mtWindSpeed	0	m/s	12/16/2019 11:14:22 AM	Chart
Wind Direction	mtAdjWindDir	146	degrees	12/16/2019 11:14:22 AM	Chart
Relative Humidity	mtRelHumidity	90	%	12/16/2019 11:14:23 AM	Chart
Barometric Pressure	mtAdjBaromPress	994.13	hPa	12/16/2019 11:14:23 AM	Chart
Rain	mtRainToday	0	mm	12/16/2019 11:14:23 AM	Chart
Solar Radiation	mtSolarRadiation	16	W/m2	12/16/2019 11:14:24 AM	Chart
Wind Chill	mtWindChill	-0.76	°C	12/16/2019 11:14:24 AM	Chart

Developed by [Hans-Petter Halvorsen](https://www.halvorsen.blog) (<https://www.halvorsen.blog>)

Figure 35-4: Weather Overview Page

35.7 Charts Page

Each of the weather parameters can be plotted. In order to open the different charts, you just click on the “chart” button in the list as shown in Figure 35-4.

Error! Reference source not found.) we see an example where Google Charts are used in an ASP.NET Core Web Weather System Application.

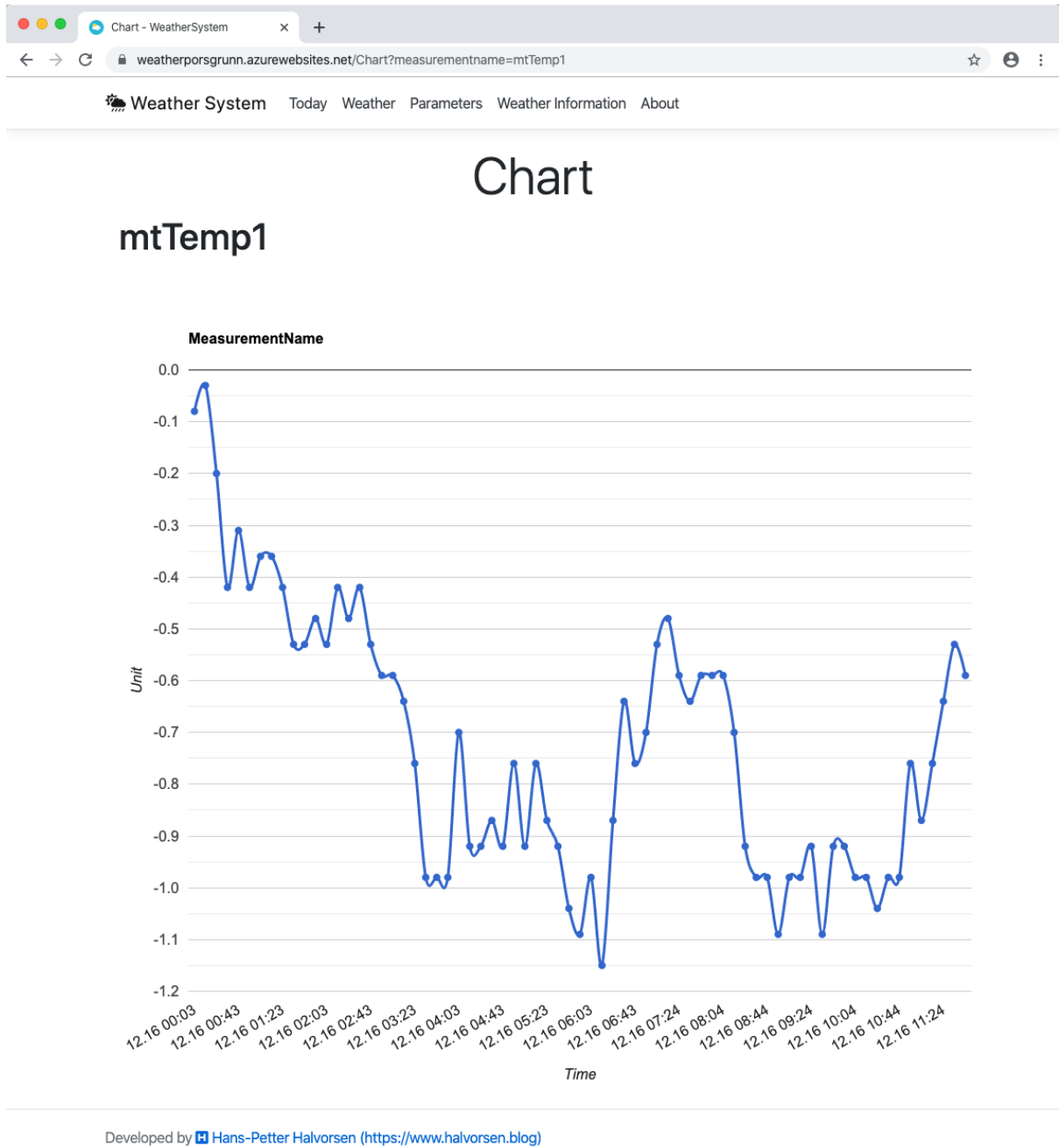


Figure 35-5: Weather Charts

Source Code for Chart.cshtml

```
@page
@model WeatherSystem.Pages.ChartModel
@{
    ViewData["Title"] = "Chart";
}

<div class="text-center">
    <h1 class="display-4">Chart</h1>
</div>
```

```

<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
    google.charts.load('current', { 'packages': ['corechart'] });
    google.charts.setOnLoadCallback(drawChart);

    function drawChart() {
        var data = google.visualization.arrayToDataTable([
            ['Time', 'Data'],

            @foreach (var data in Model.chartDataList) {

                <text>['@data.MeasurementTimeStamp',
                    @data.MeasurementValue],</text>

            }

        ]);

        var options = {
            title: 'MeasurementName',
            curveType: 'function',
            pointsVisible: true,
            lineWidth: 3,
            legend: 'none',
            hAxis: {title: 'Time'},
            vAxis: {title: 'Unit'},
            width: '100%',
            height: '100%',
            chartArea: {width: '85%', height: '75%'}
        };

        var chart = new
        google.visualization.LineChart(document.getElementById('curve_chart'));

        chart.draw(data, options);
    }
</script>

<div class="container-fluid lead">

    <h1>@Model.measurementName</h1>

    <div id="curve_chart" style="width: 1000px; height: 900px"></div>

</div>

```

Page Model (Chart.cshtml.cs):

```

...
namespace WeatherSystem.Pages
{
    public class ChartModel : PageModel
    {
        public string measurementName;

        public List<ChartData> chartDataList = new List<ChartData>();

        string connectionString;
    }
}

```

```

        readonly IConfiguration _configuration;

        public ChartModel(IConfiguration configuration)
        {
            _configuration = configuration;
        }

        public void OnGet()
        {
            measurementName = Request.Query["measurementname"];

            chartDataList = ChartData();
        }

        public void OnPut()
        {
        }

        private List<ChartData> ChartData()
        {
            connectionString =
                _configuration.GetConnectionString("ConnectionString");

            List<ChartData> chartDataList = new List<ChartData>();

            ChartData chartData = new ChartData();

            chartDataList = chartData.GetChartData(connectionString,
                measurementName);

            return chartDataList;
        }
    }
}

```

Model ("ChartData.cs"):

```

...
using System.Data.SqlClient;

namespace WeatherSystem.Models
{
    public class ChartData
    {
        public int MeasurementDataId { get; set; }
        public string MeasurementTimeStamp { get; set; }
        public string MeasurementValue { get; set; }

        public List<ChartData> GetChartData(string connectionString, string
            measurementName)
        {
            List<ChartData> chartDataList = new List<ChartData>();

            SqlConnection con = new SqlConnection(connectionString);

```

```
        string selectSQL = "SELECT MeasurementDataId,
FORMAT(MeasurementTimeStamp, 'MM.dd HH:mm') AS MeasurementTimeStamp,
MeasurementValue FROM GetMeasurementData WHERE MeasurementTimeStamp between
CONVERT(DATE, GETDATE()) AND GETDATE() AND MeasurementName='" +
measurementName + "' ORDER BY MeasurementDataId";

        con.Open();

        SqlCommand cmd = new SqlCommand(selectSQL, con);

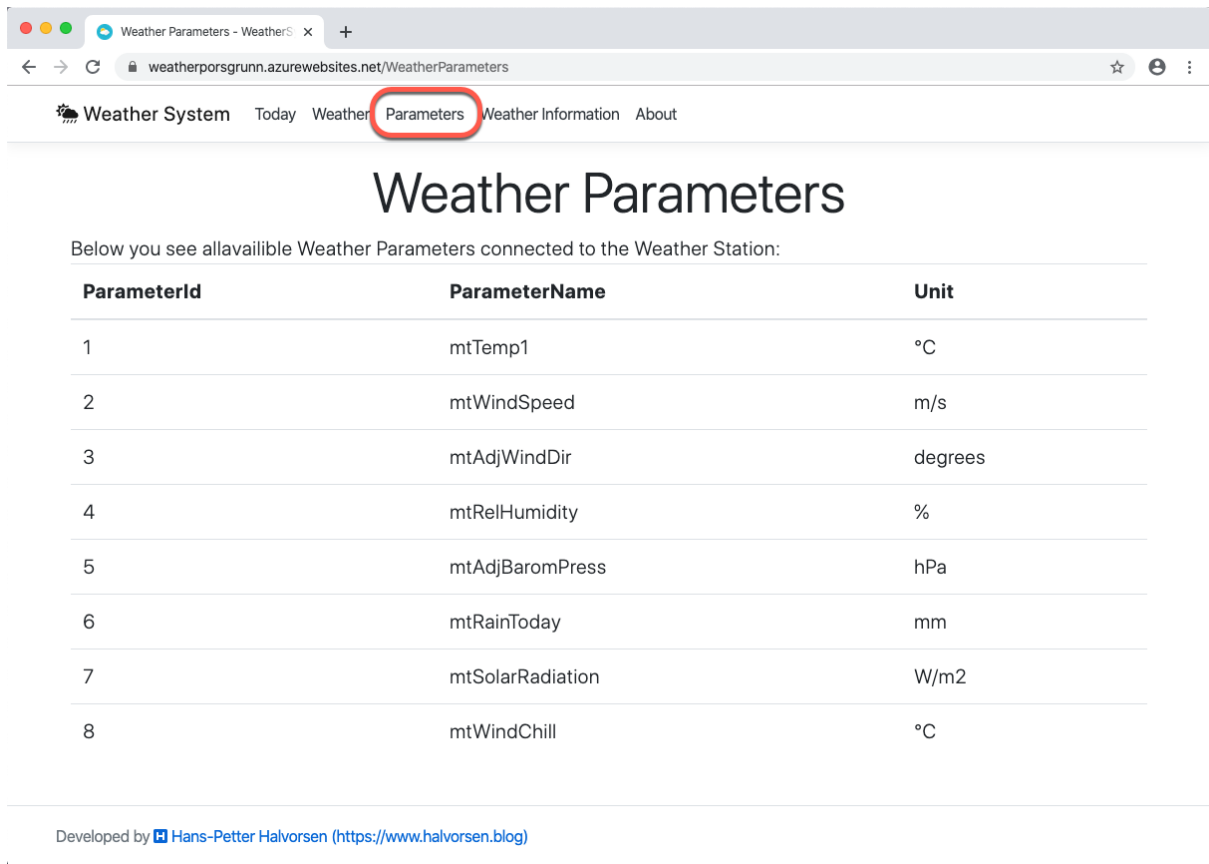
        SqlDataReader dr = cmd.ExecuteReader();

        if (dr != null)
        {
            while (dr.Read())
            {
                ChartData chartData = new ChartData();

                chartData.MeasurementDataId =
                    Convert.ToInt32(dr["MeasurementDataId"]);
                chartData.MeasurementTimeStamp =
                    dr["MeasurementTimeStamp"].ToString();
                chartData.MeasurementValue =
                    dr["MeasurementValue"].ToString();

                chartDataList.Add(chartData);
            }
        }
        return chartDataList;
    }
}
```

35.8 Weather Parameters Page



Weather Parameters - WeatherS x +

weatherporsgrunn.azurewebsites.net/WeatherParameters

Weather System Today Weather Parameters Weather Information About

Weather Parameters

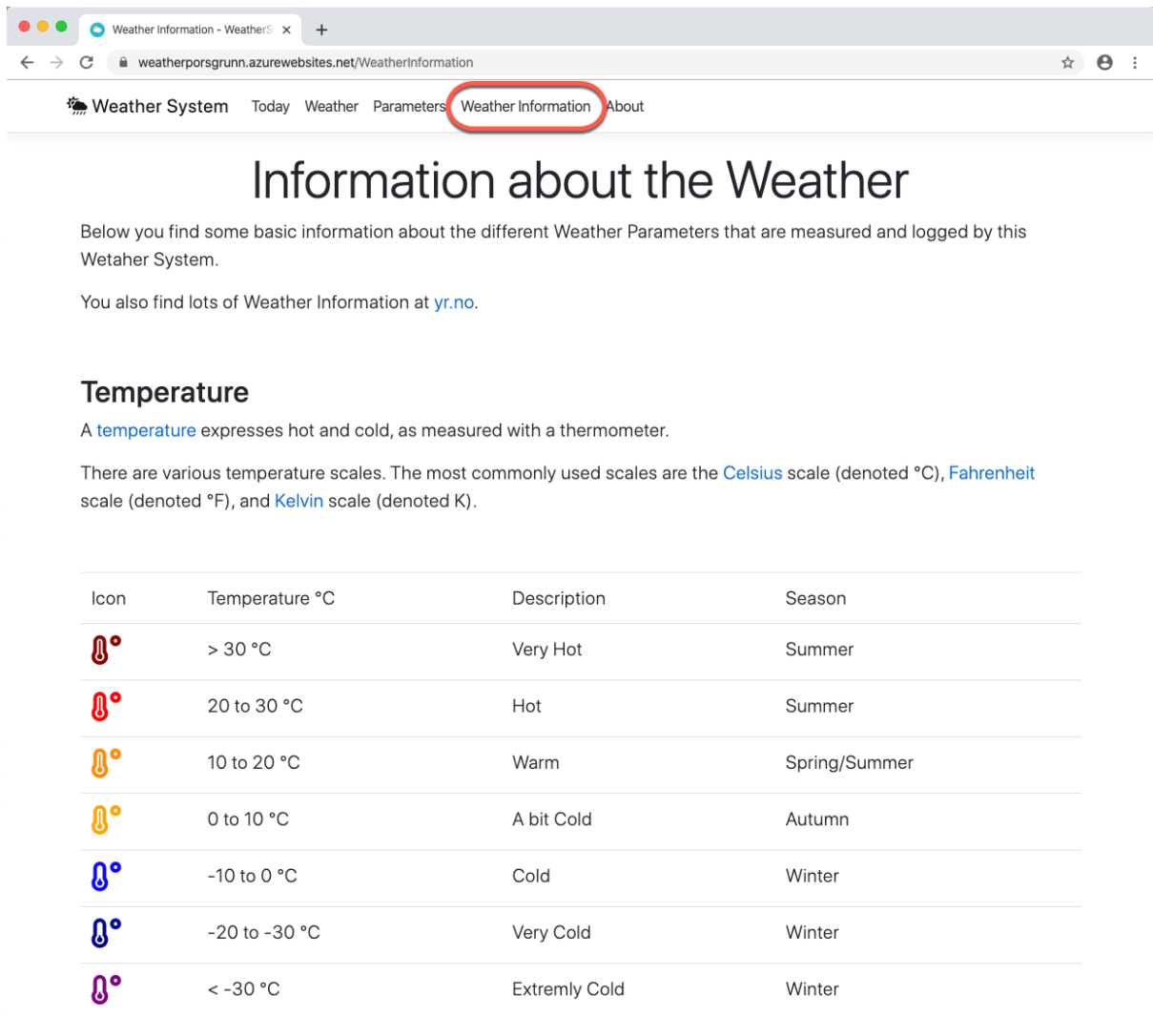
Below you see allavailable Weather Parameters connected to the Weather Station:

ParameterId	ParameterName	Unit
1	mtTemp1	°C
2	mtWindSpeed	m/s
3	mtAdjWindDir	degrees
4	mtRelHumidity	%
5	mtAdjBaromPress	hPa
6	mtRainToday	mm
7	mtSolarRadiation	W/m2
8	mtWindChill	°C

Developed by [Hans-Petter Halvorsen](https://www.halvorsen.blog) (<https://www.halvorsen.blog>)

Figure 35-6: Weather Parameters

35.9 Weather Information Page



Weather Information - WeatherS x +

weatherporsgrunn.azurewebsites.net/WeatherInformation

Weather System Today Weather Parameters **Weather Information** About

Information about the Weather

Below you find some basic information about the different Weather Parameters that are measured and logged by this Weather System.

You also find lots of Weather Information at yr.no.

Temperature

A [temperature](#) expresses hot and cold, as measured with a thermometer.

There are various temperature scales. The most commonly used scales are the [Celsius](#) scale (denoted °C), [Fahrenheit](#) scale (denoted °F), and [Kelvin](#) scale (denoted K).








Icon	Temperature °C	Description	Season
	> 30 °C	Very Hot	Summer
	20 to 30 °C	Hot	Summer
	10 to 20 °C	Warm	Spring/Summer
	0 to 10 °C	A bit Cold	Autumn
	-10 to 0 °C	Cold	Winter
	-20 to -30 °C	Very Cold	Winter
	< -30 °C	Extremely Cold	Winter

Figure 35-7: Weather Information

35.10 About Page

The screenshot shows a web browser window with the URL `weatherporsgrunn.azurewebsites.net/About`. The page title is "Information about the Weather System". The content includes the following text:

This Weather Station is located at [University of South-Eastern Norway](#), campus Porsgrunn.

The Weather System is based on a [Capricorn 2000ex Weather Station](#) with [Weather MicroServer](#) from [Columbia Weather Systems](#).

Below we see an overview of the Weather System at [University of South-Eastern Norway](#), campus Porsgrunn. The Weather System is Developed by [Hans-Petter Halvorsen](https://www.halvorsen.blog) - <https://www.halvorsen.blog>

The diagram illustrates the system architecture, divided into a Local Area Network LAN and a Cloud. The LAN includes a Weather Datalogging Server (Windows) with LabVIEW, a Weather MicroServer, a Linux OS Modbus Server (Slave), a Local SQL Server, Files, and an OPC UA Server. The Cloud includes Microsoft Azure and Other Cloud Platforms, with ThingSpeak. The Cloud is connected to Management and Monitoring, which includes C#, Python, MATLAB, and Web Technology.

Figure 35-8: About – Information about the Weather System

35.11 Deployment to Azure

In this example I have deployed the application to Microsoft Azure, but the application can be “anywhere”.

Below, the necessary steps for deploying the application to Microsoft Azure is presented.

Both the database and the web application will be deployed to Microsoft Azure.

When you have setup the web application, you need to make sure it has access to the database. See Figure 35-9.

Microsoft Azure

Dashboard > All resources > WEATHER (dmsserver/WEATHER) > Firewall settings

Firewall settings

dmsserver (SQL server)

Save Discard + Add client IP

Connections from the IPs specified below provides access to all the databases in dmsserver.

Allow Azure services and resources to access this server

ON OFF

Client IP address 128.39.132.158

Rule name	Start IP	End IP
<input type="text"/>	<input type="text"/>	<input type="text"/> ...

Figure 35-9: Make sure the Web Application has Access to the Database

36 Voting System

Here will the “Voting System” application be presented. It can be used as an online Voting System, e.g., vote for best athlete, best book, best song, best web page, etc. Only the imagination limits what this application can be used for.

This application contains features typically needed in a professional ASP.NET Core applications. Some examples are:

- CRUD functionality
- Query String Data
- Form Data
- Session Data
- Charts
- Login Features
- Etc.

The entire example can be downloaded from the home page of this textbook.

Figure 36-1 shows the Start Page (Index) page of the application.

Voting System

Use the following Features as foundation for your voting:

- HTML skills
- Contents
- Creativity
- Design and Layout
- User friendly

For each System: For each of the Features above, give a Score between 0-100%. Then calculate the Average Score the specific System and enter the Score in the "Vote" section. You should also write an overall Comment for the specific System.

Here are some resources:

- [HTML Web Site Requirements](#)
- [Vote Sheet](#)

Make sure you are Logged In before you start entering the different Scores.

This system has been made with **ASP.NET Core**. Here you find [ASP.NET Core Resources](#).

This system has been made with **ASP.NET Core**. Here you can download the entire Application: [Download Voting System](#).

ASP.NET Core Application - © Developed by [Hans-Petter Halvorsen \(https://www.halvorsen.blog\)](https://www.halvorsen.blog)

Figure 36-1: Voting System

Figure 36-2 show the "New Vote" feature.

[Voting System](#) [Home](#) [Systems](#) [Vote](#) [Chart](#) [Results](#) [User](#)

New Vote

Your Name: Please Login before you go further.
Note! You can only Vote once for each System, but it is possible to change your existing Votes.
[My existing Votes](#)

System:

Score (0-100%):

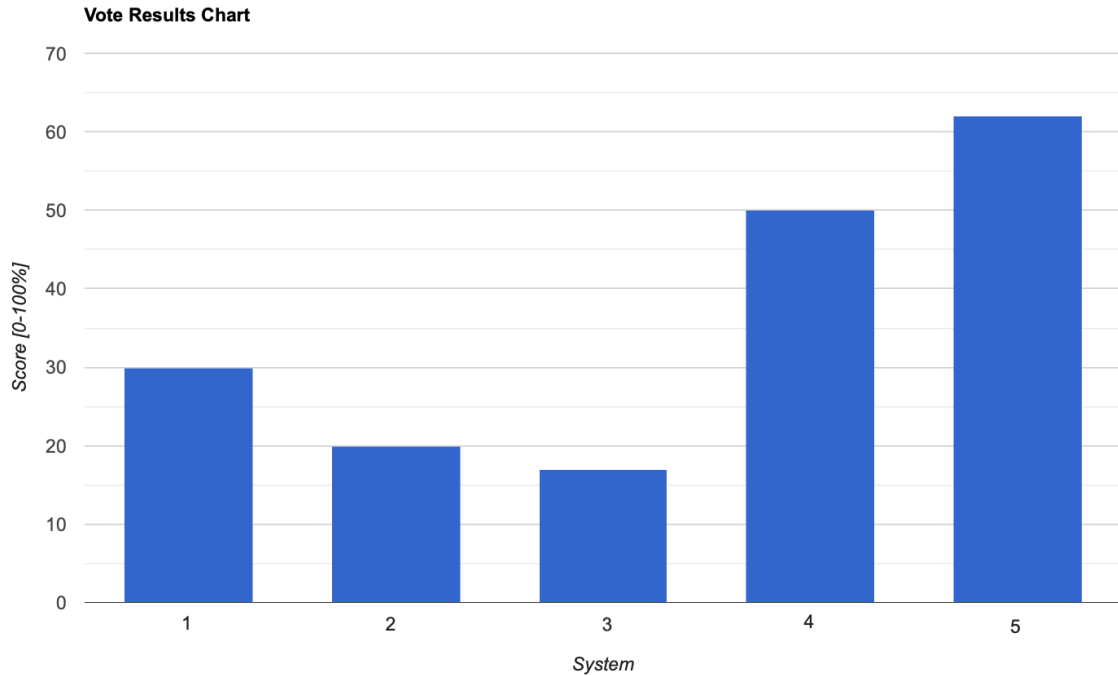
Comment:

ASP.NET Core Application - © Developed by [Hans-Petter Halvorsen \(https://www.halvorsen.blog\)](https://www.halvorsen.blog)

Figure 36-2: New Vote

Figure 36-3 show the web page for presenting the results from the votes.

Vote Results Chart



[Click here to see Bar Chart for a specific System](#)

Figure 36-3: Vote Results

37 Data Management System (DMS)

The intention is to create an Open Source Cloud Platform for Data Management, Datalogging and Data Analysis based on open standards.

Main Features:

- Database storage, both locally and cloud
- Support for different Cloud platforms, both general cloud platforms like Azure, etc. specific IoT platforms like ThingSpeak, Azure Sphere, etc. and Industrial Cloud platforms like MindSphere from Siemens, etc.
- Management features
- Logging features
- Monitoring features
- IoT devices and applications
- Support different protocols like Modbus TCP, OPC UA, MQTT, REST, etc.
- Support for different languages like LabVIEW, C#, Python, MATLAB

Figure 37-1 shows an overview of the Data Management System (DMS).

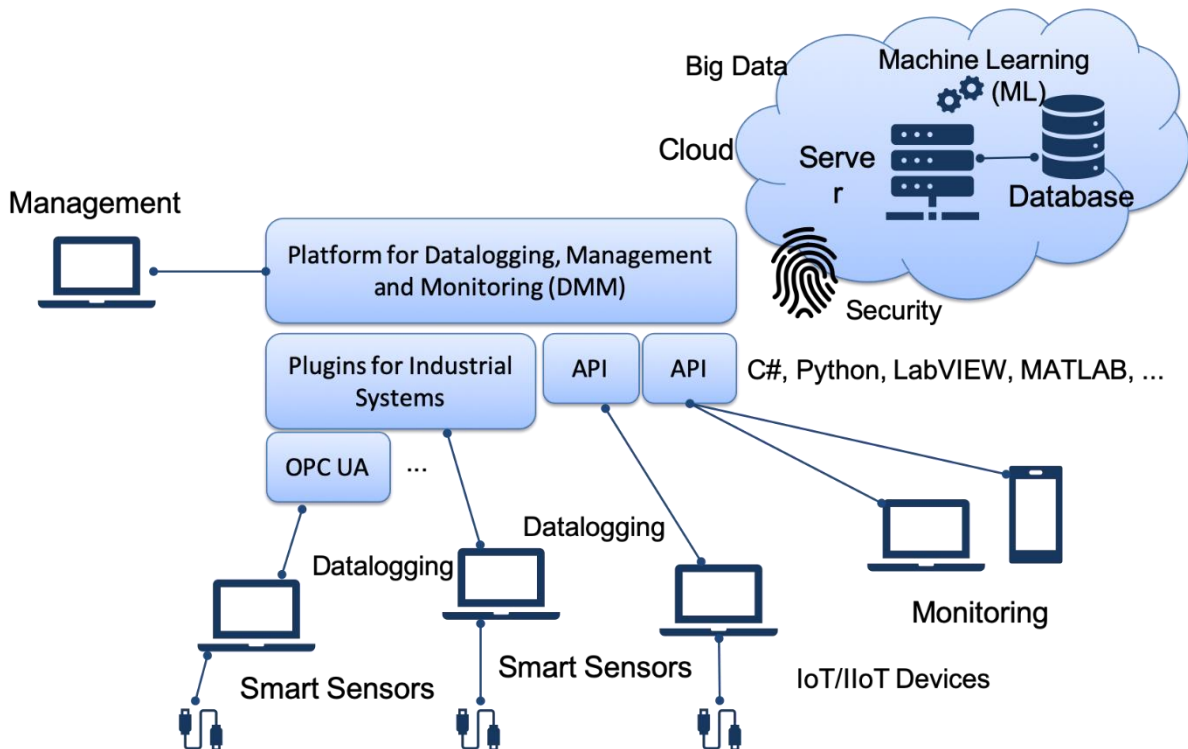


Figure 37-1: Data Management System (DMS)

Figure 37-2 shows the Data Management and Monitoring web application.

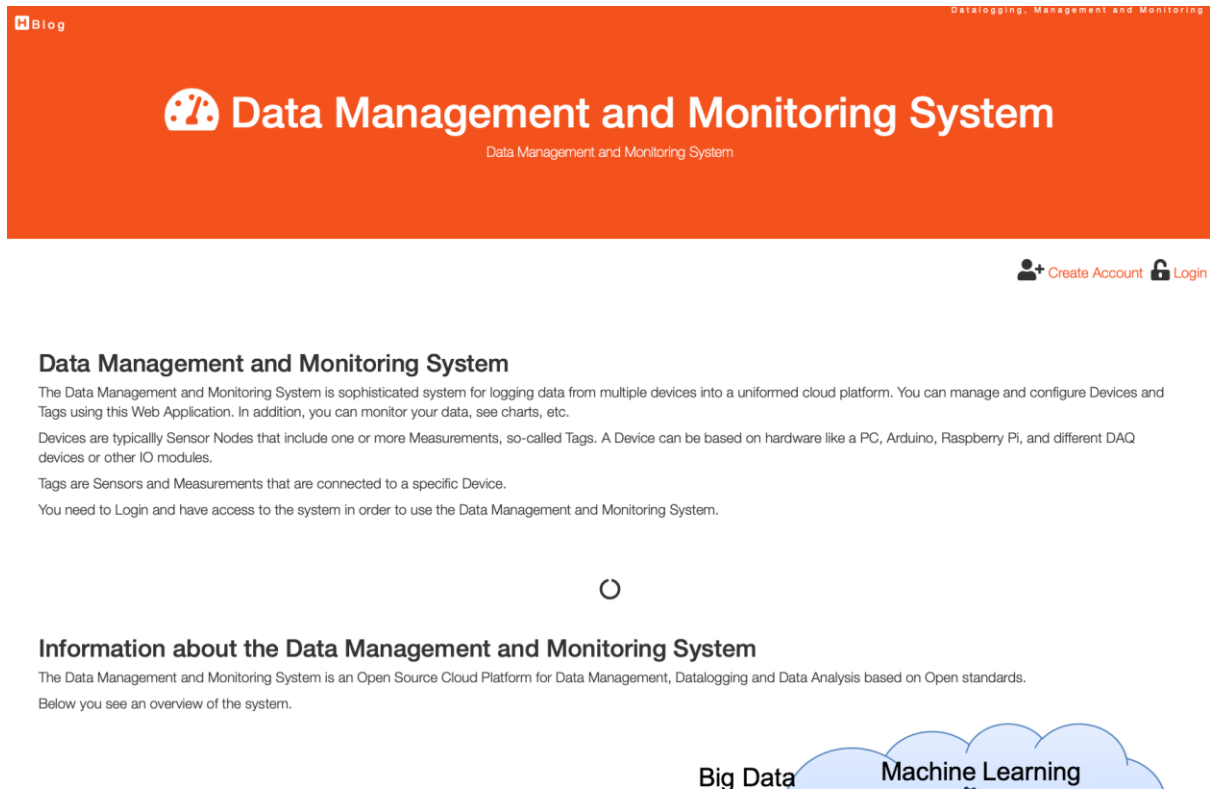


Figure 37-2: Data Management and Monitoring [Hans-Petter Halvorsen]

Web Site:

<https://www.halvorsen.blog/documents/software/dmm/>

Figure 37-3 shows the web interface for the Data Management System (DMS) after Login.

Data Management and Monitoring System

The Data Management and Monitoring System is sophisticated system for logging data from multiple devices into a uniformed cloud platform. You can manage and configure Devices and Tags using this Web Application. In addition, you can monitor your data, see charts, etc.

Devices are typically Sensor Nodes that include one or more Measurements, so-called Tags. A Device can be based on hardware like a PC, Arduino, Rasperry Pi, and different DAQ devices or other IO modules.

Tags are Sensors and Measurements that are connected to a specific Device.

You need to Login and have access to the system in order to use the Data Management and Monitoring System.

Management and Configuration

Below you can manage your Devices and Tags.

- Device Management** - Create new Devices for Datalogging
- Tag Management** - Create Tags and link them to a specific Device.


Monitoring Data

Below you can search for data, see charts, etc.

- Search Data**
- Chart**
- View Data**

Figure 37-3: DMS Platform – Web Interface

Figure 37-4 and Figure 37-6 shows Web-based Data Management for setting up Devices and respective Tags for the different Devices. Here we have configured the Device Weather Station to have the Tag Temperature.



Device Management

























Data Management and Monitoring System

[Home](#) [New Device](#)

Devices


Devices are typically Sensor Nodes that include one or more Measurements, so-called Tags. A Device can be based on hardware like Arduino, Raspberry Pi, and different DAQ devices or other IO modules.

List of Devices registered in the database:

Device Name	Location	Action
 Office	Porsgrunn	    
 Separation Rig	Process Hall	    
 TestDevice1	B-236a	    
 Weather Station	Porsgrunn	    

[New Device](#)

Figure 37-4: Web-based Data Management – Devices



Device Dashboard

Data Management and Monitoring System

[Home](#) [Device Management](#)

Weather Station

Porsgrunn

List of Tags connected to the specific Device. You may click on the Tag Names in order to see a Chart with data for the specific Tag.
See [Charts for all Tags](#) connected to this Device in one page - or on a [Dashboard \(4x2\)](#)


























Tag Name	Tag Alias	Current Value	TimeStamp	Action
 mtAdjBaromPress	Barometric Pressure	1015.12 hPa	15:30	 
 mtAdjWindDir	Wind Direction	51 °	15:30	 
 mtRainToday	Rain	6.86 mm	15:30	 
 mtRelHumidity	Humidity	85 %	15:30	 
 mtSolarRadiation	Solar Radiation	0 W/m2	15:30	 
 mtTemp1	Temperature	6.02 °C	15:30	 
 mtWindChill	Wind Chill	-3.33 °C	15:30	 
 mtWindSpeed	Wind Speed	6.71 m/s	15:30	 

Figure 37-5: Device Dashboard



Tag Management

Data Management and Monitoring System

[Home](#) | [Device Management](#) | [New Tag](#)

































Tags

Devices are typically Sensor Nodes that include one or more Measurements, so-called Tags. A Device can be based on hardware like Arduino, Raspberry Pi, and different DAQ devices or other IO modules.

Select Device:

Weather Station

List of Tags registered in the database:

TagName	Tag Alias	Unit	Device	Action
 mtAdjBaromPress	Barometric Pressure	hPa	Weather Station	  
 mtAdjWindDir	Wind Direction	°	Weather Station	  
 mtRainToday	Rain	mm	Weather Station	  
 mtRelHumidity	Humidity	%	Weather Station	  
 mtSolarRadiation	Solar Radiation	W/m2	Weather Station	  
 mtTemp1	Temperature	°C	Weather Station	  
 mtWindChill	Wind Chill	°C	Weather Station	  
 mtWindSpeed	Wind Speed	m/s	Weather Station	  

+ Add Tag

Figure 37-6: Web-based Data Management – Tags

Figure 37-7 shows Web-based Data Monitoring.

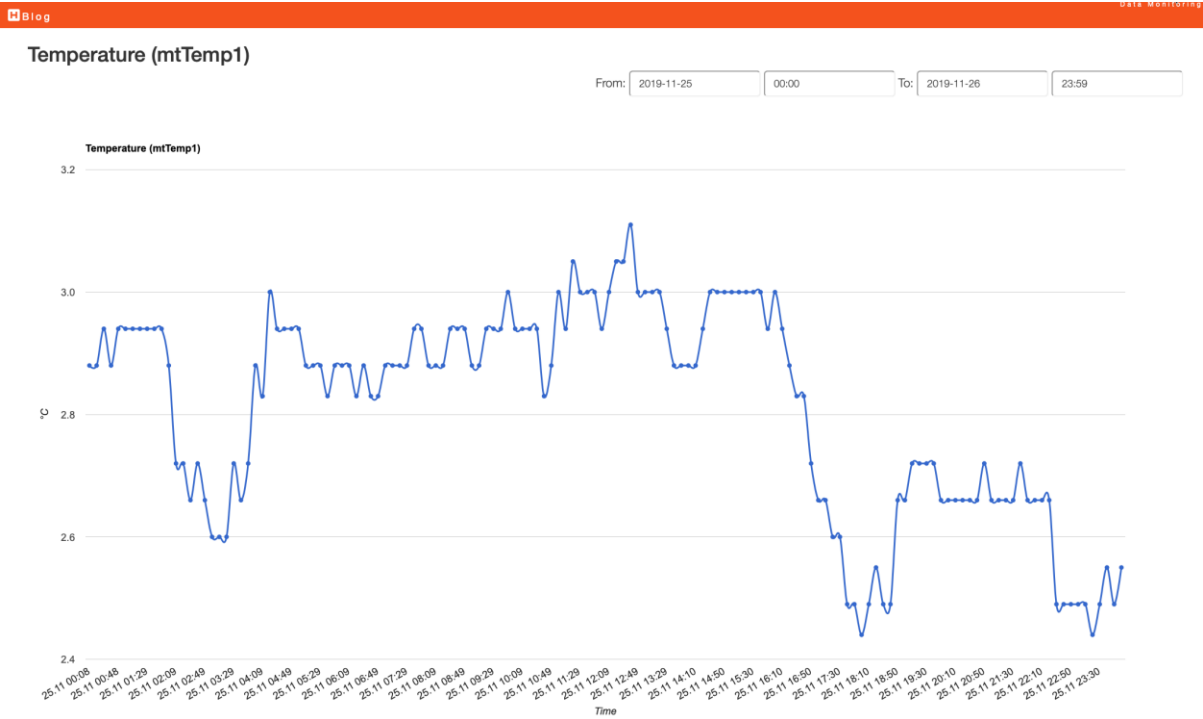
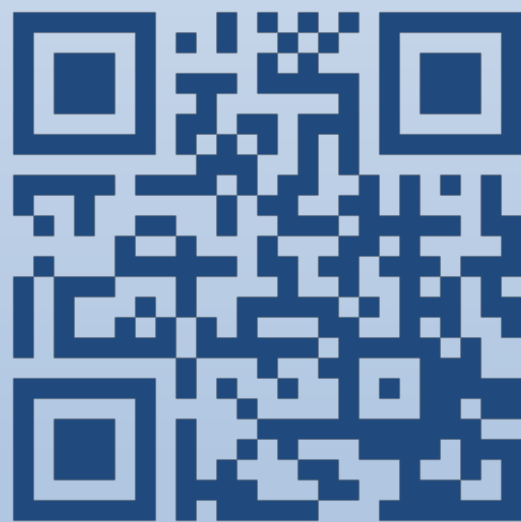


Figure 37-7: Web-based Data Monitoring – Charting



Web Programming

ASP.NET Core