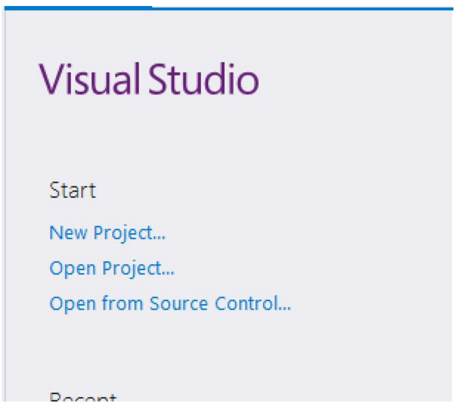
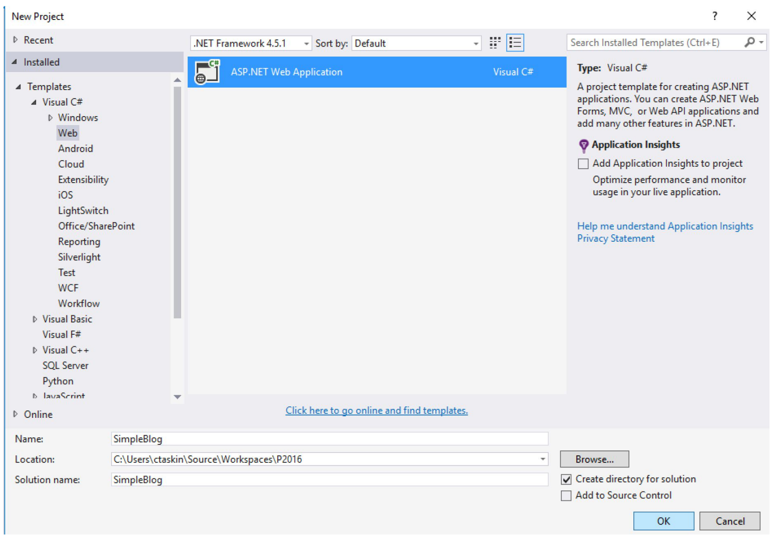
Creating Our Project

ASP.NET MVC C05





.NET Framework 4.5.1 Sort by: Default

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to project

Optimize performance and monitor usage in your live application.

Help me understand Application Insights
Privacy Statement

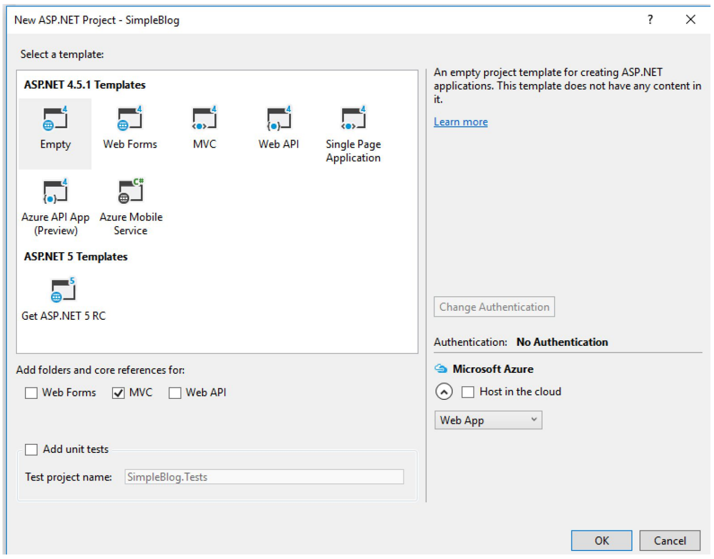
Name: SimpleBlog
Location: C:\Users\ctaskin\Source\Workspaces\P2016
Solution name: SimpleBlog

☑ Create directory for solution
☐ Add to Source Control

In this course, we are opening Visual Studio

In this course, we are going to open up Visual Studio, create a project and talk about the project structure for ASP.NET MVC.

So the first step is to open up visual studio. Then click new Project.
Template, we are goint to use it is under Templates ,Visual C# , Web.
There is only one Template, names ASP.NET Web Application. We will enter the Name, Solution Name and Location info for project and Click OK.



New ASP.NET Project - SimpleBlog

Select a template:

ASP.NET 4.5.1 Templates

ASP.NET 5 Templates

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Learn more

Authentication: **No Authentication**

Microsoft Azure

[ ] Host in the cloud

Web App

Add folders and core references for:

[ ] Web Forms [✓ MVC] [ ] Web API

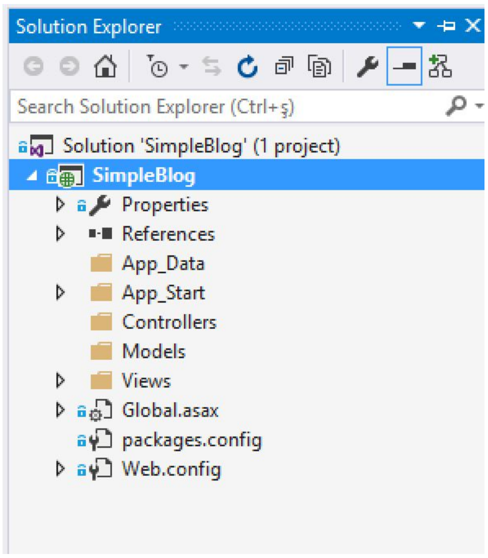
[ ] Add unit tests

Test project name: SimpleBlog.Tests

ASP.NET WEB APPLICATION PROJECT TYPES

In this course, Empty template is selected. Do not forget to select Add folders and code references for MVC.
In this project, all the things that are absolutely required for an MVC project.

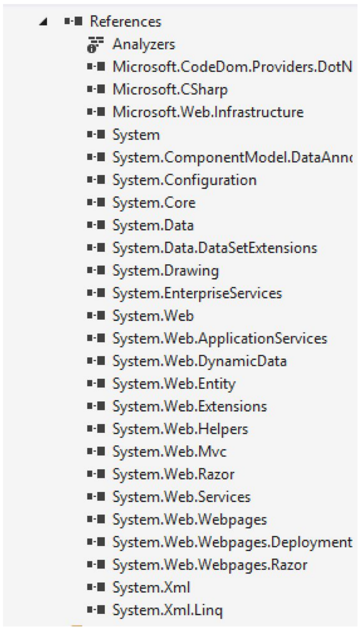
We are using Razor view Engine. It is integrated in ASP.NET. It is a view engine that creates HTML output.


Solution Explorer

Search Solution Explorer (Ctrl+ş)

```
Solution 'SimpleBlog' (1 project)
◢ SimpleBlog
    ▷ Properties
    ▷ References
      App_Data
    ▷ App_Start
      Controllers
      Models
    ▷ Views
    ▷ Global.asax
      packages.config
    ▷ Web.config
```

We created a new Project

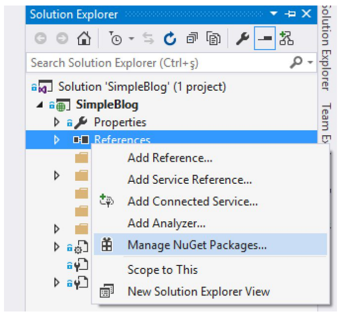


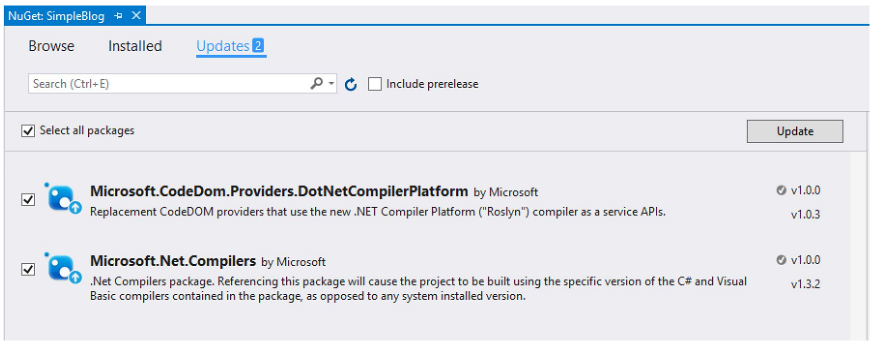
```
◢ References
    Analyzers
    Microsoft.CodeDom.Providers.DotN
    Microsoft.CSharp
    Microsoft.Web.Infrastructure
    System
    System.ComponentModel.DataAnn
    System.Configuration
    System.Core
    System.Data
    System.Data.DataSetExtensions
    System.Drawing
    System.EnterpriseServices
    System.Web
    System.Web.ApplicationServices
    System.Web.DynamicData
    System.Web.Entity
    System.Web.Extensions
    System.Web.Helpers
    System.Web.Mvc
    System.Web.Razor
    System.Web.Services
    System.Web.Webpages
    System.Web.Webpages.Deployment
    System.Web.Webpages.Razor
    System.Xml
    System.Xml.Linq
```

NuGet

After pressing OK Visual Studio is making a lot of stuff. In this stuff, adding references takes a lof of time. We see in references too a lot of things. This list will grow a lot.

We are using Nu-Get Packages to add this references. Nu-Get package is a way to manage packages for C# projects or any type of .net projects. There a way to import external code in to our projects and version them.





☑ **Microsoft.CodeDom.Providers.DotNetCompilerPlatform** by Microsoft — v1.0.0 → v1.0.3
Replacement CodeDOM providers that use the new .NET Compiler Platform ("Roslyn") compiler as a service APIs.

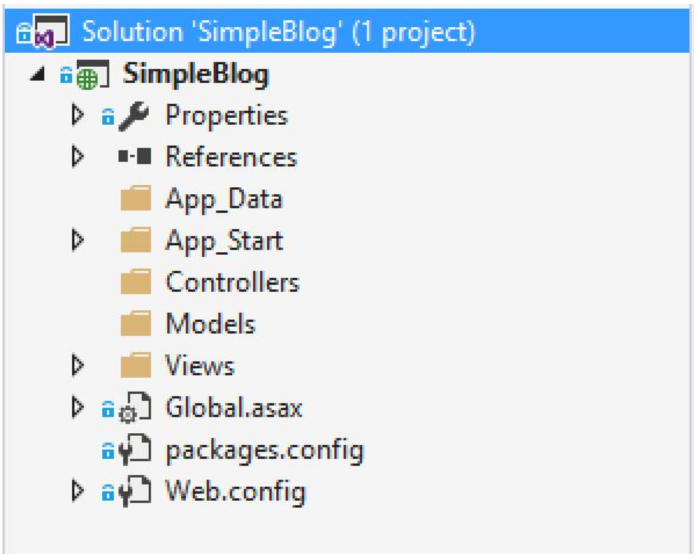
☑ **Microsoft.Net.Compilers** by Microsoft — v1.0.0 → v1.3.2
.Net Compilers package. Referencing this package will cause the project to be built using the specific version of the C# and Visual Basic compilers contained in the package, as opposed to any system installed version.

Our first stuff is to update references

Nu-Get package manager allows us to update references quickly and easy way.

All the time our projects are out of date. So our first stuff is update our references by using Nu-Get.

To update our references, rigth click on references, go to Manage NuGet Packages.



Let’s look at what exctly this template is

Let’s look at what exactly this template is. Here Simple Blog Project is an .NET application. But somehow it is connected to ASP.NET MVC. We talk about how it happens very shortly.

We see that, we have a couple of folders created for us. These folders would be populated if we chosed diffrent template.
We have a App_Data folder, where we can place databases that lives on the file system which we are gone do.

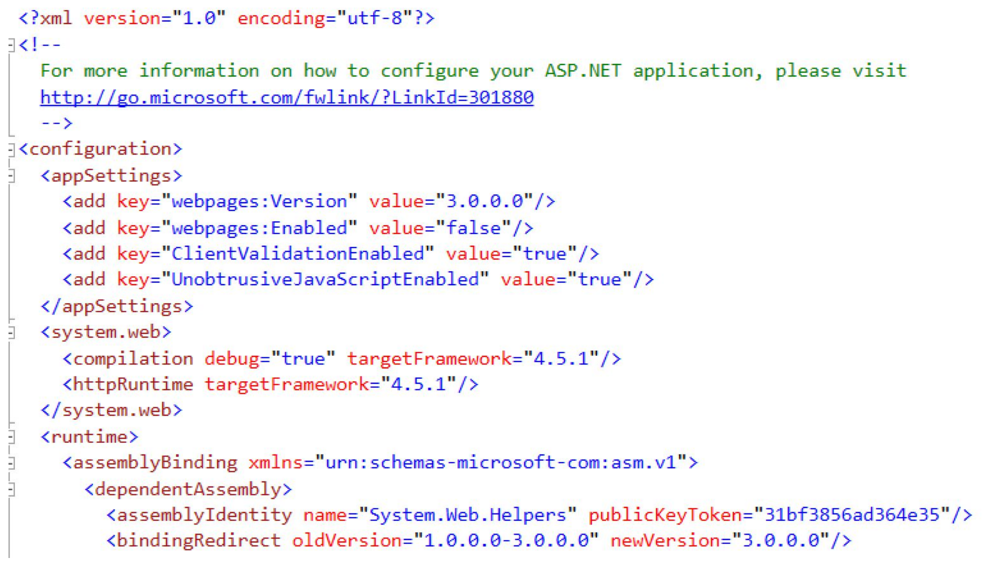
We have a App_Start folder. Our start folder contains a class file. RouteConfig.cs. This class is a basic class and has public static method RegisterRoutes. This is just a convenience to place these methods that perform our configuration inside of static classes. We actually see where these method is invoked. You can also change the name of the class, or the folder of this class but, it is a convention. We will be talk about routes.

Next thing, we have these 3 folders. Controllers, Models and Views. It is MVC. These are our 3 buckets. It is important to understand again these are conventions. Conventions may be overriden. There is actually view folder is a kind of special. But we can override anyway. But for Controllers and Models folder, there is absolutely nothing special with them. These are just folders. This is in contrast to other frameworks, especially frameworks in php, where the directory structure of your project is assumed by the framework. In ASP.NET MVC, that is in the case, we have a lot of flexibility how we organize things, by the fact there are some conventions to follow.

Inside of Views, we have a web.config file. One of our config files. This is important we will talk more about this later. Do not delete it.

Next step, we have two more important files for our project in root folder. Global.asax and Web.Config.
Packages.config files is just a XML file created by Nuget to tell Nuget what packages we have dependencies on. So you can ignore that. That was managed automatically.

But global.asax and web.config is important. In Views folder there is a web.config file too but this file is not same as the file in root folder. The file in Views folder is related with Razor Views.

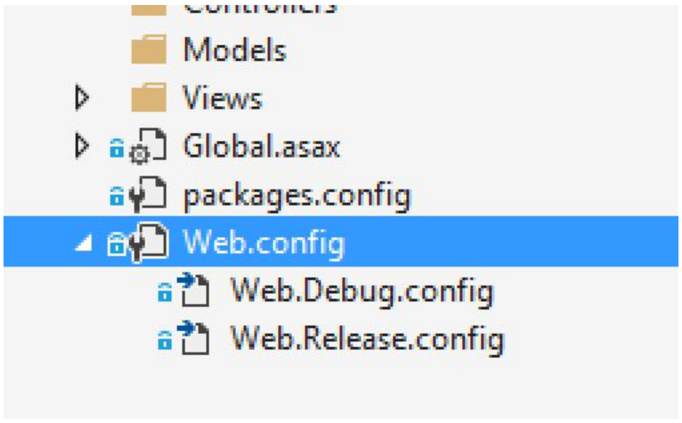


```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0"/>
    <add key="webpages:Enabled" value="false"/>
    <add key="ClientValidationEnabled" value="true"/>
    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.5.1"/>
    <httpRuntime targetFramework="4.5.1"/>
  </system.web>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35"/>
        <bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0"/>
```

Web.config file - It describes web server how our projects is going to be run

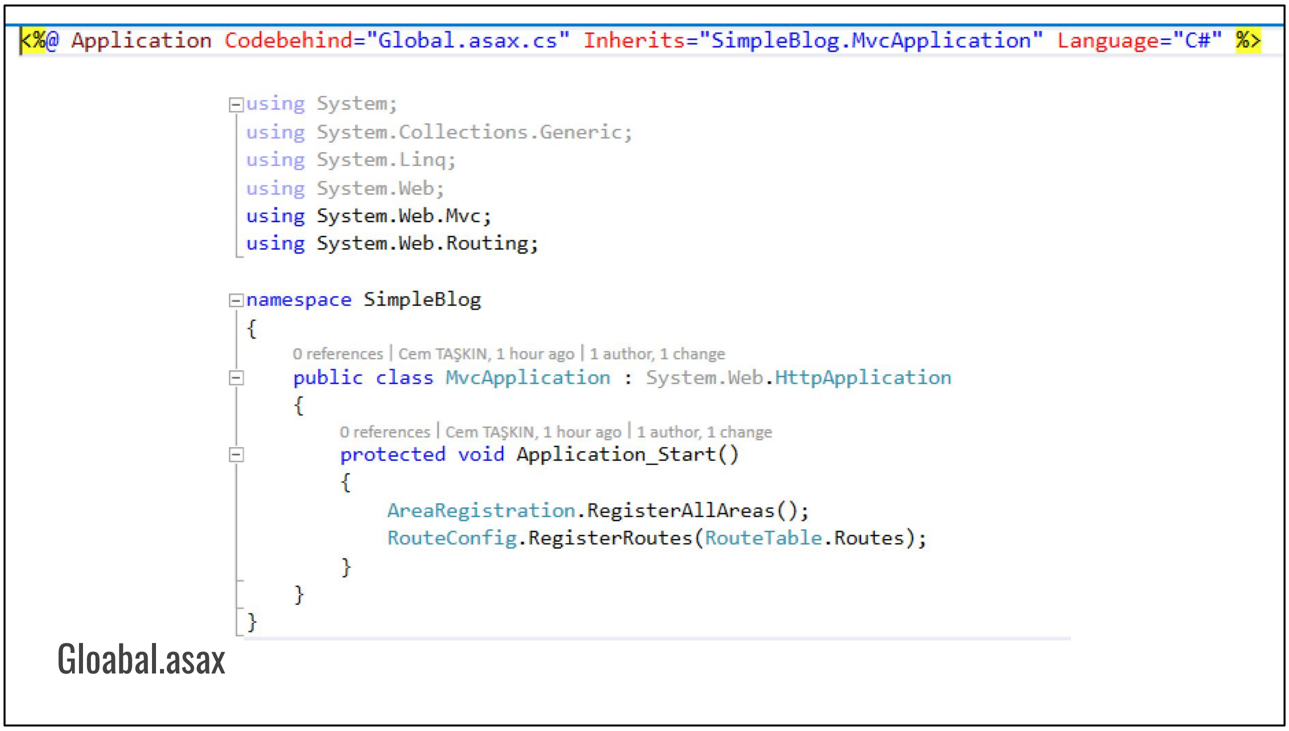
On the other hand, web.config placed in root folder is global configuration for entire project.
We have app settings. We have things like system.web here, and our handlers and so one and so one. We will be using this file quite a bit in order to add things like database connection and for other things.
Web.config is important because it describes the web server how our project is going to be run.

For example we see that we tell it to use the 4.5.1 Framework. We tell it to be in debug mode.



web.Debug.confif and web.Release.config

It is also possible to expand web.config to Web.Debug.config and Web.Release.config. We will talk about these files in the last section of courses. They have to do with deployment.



```
<%@ Application Codebehind="Global.asax.cs" Inherits="SimpleBlog.MvcApplication" Language="C#" %>
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace SimpleBlog
{
    0 references | Cem TAŞKIN, 1 hour ago | 1 author, 1 change
    public class MvcApplication : System.Web.HttpApplication
    {
        0 references | Cem TAŞKIN, 1 hour ago | 1 author, 1 change
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Gloabal.asax

Global.asax is actually two files. But Visual Studio combines them. One for us Visual Purposes and when we double click on it we get Global.asax.cs. And let's look at on the file system,and show look at other file. We have global.asax and global.asax.cs files.

And if you want actually to see global.asax file, right click on Global.asax and click View Mark-Up.

Global.asax is the entiry point into web application that are read top on ASP.NET. The global.asax file or class is used to handle things like Application Start Up, Application Errors, Request start, Request end and all that fun stuff. Which means if you need to do anything when your application starts, you put in the global.asax. As you can see here, we talked about RouteConfig file which is located at App_Start. You see that all the Application Start work is to invoke RouteConfig static method.

If I take all the code in RouteConfig into global.asax file you will have the same effect. But using separate files and conventions organizes better. So remember RouteConfig static method invoked on Application start.

So another thing in Global.asax file. This is MvcApplication class. It is a class inherited from HttpApplication.
What happens is ASP.Net tries to load our program and looks as global.asax file. Looks at the application declaration. It sees that we have a codebehind. Codebehind term is commonly used in ASP.NET Web Forms. The important thing that we see is

that it inherits which is our class that is in our global.asax.cs file.

So that means, ASP.NET or IIS loads up this file it knows to launch this file which then does the things in Application_Start() method.

So here a lot of things to remember. First, Application_Start method is not overriden but has a magic name that is loaded by ASP.NET. Because has the name Application_Start it would be magically invoked by ASP.NET. Now, I did say, application start over be invoked only once.