****

本科毕业论文（设计）

**英文文献原文**

文献英文题目GettingStartedwithASP.NETMVC5

学 生 姓 名 徐雯俊

学 号 2014053030

所 在 学 院 计算机学院

专 业 数字媒体技术

班 级 数媒2014级1班

指 导 教 师 廖德钦（副教授）

2018 年 1月

成都信息工程大学 计算机学院

# Getting Started with ASP.NET MVC 5

## 1. introduction

This tutorial teaches ASP.NET Core MVC with controllers and views. Razor Pages is a new alternative in ASP.NET Core 2.0, a page-based programming model that makes building web UI easier and more productive. We recommend you try the [Razor Pages](https://docs.microsoft.com/aspnet/core/mvc/razor-pages) tutorial before the MVC version. The Razor Pages tutorial:

* + Is easier to follow.
  + Covers more features.
  + Is the preferred approach for new application development.

This tutorial will teach you the basics of building an ASP.NET MVC 5 web app using [Visual Studio 2017](https://www.visualstudio.com/). Final Source for tutorial located on [GitHub](https://github.com/aspnet/Docs/tree/master/aspnet/mvc/overview/getting-started/introduction/sample/MvcMovie/MvcMovie).

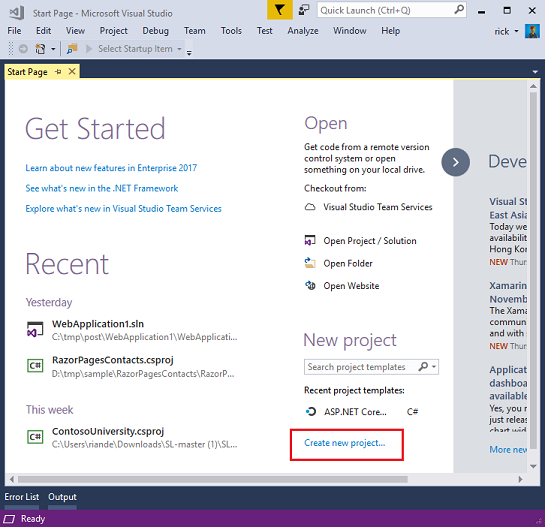
You need an Azure account to deploy this app to Azure:

* + You can [open an Azure account for free](https://azure.microsoft.com/en-us/pricing/free-trial/?WT.mc_id=A443DD604) - You get credits you can use to try out paid Azure services, and even after they're used up you can keep the account and use free Azure services.
  + You can [activate MSDN subscriber benefits](https://azure.microsoft.com/en-us/pricing/member-offers/msdn-benefits-details/?WT.mc_id=A443DD604) - Your MSDN subscription gives you credits every month that you can use for paid Azure services.

## 2.Getting Started

Start by installing and running [Visual Studio 2017](https://www.visualstudio.com/).

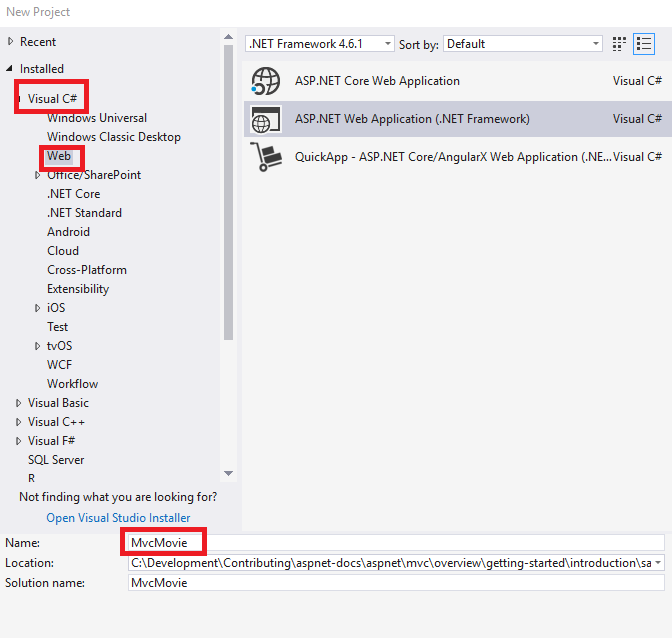
Visual Studio is an IDE, or integrated development environment. Just like you use Microsoft Word to write documents, you'll use an IDE to create applications. In Visual Studio there's a list along the bottom showing various options available to you. There's also a menu that provides another way to perform tasks in the IDE. (For example, instead of selecting New Project from the Start page, you can use the menu and select File > New Project.)



Figure

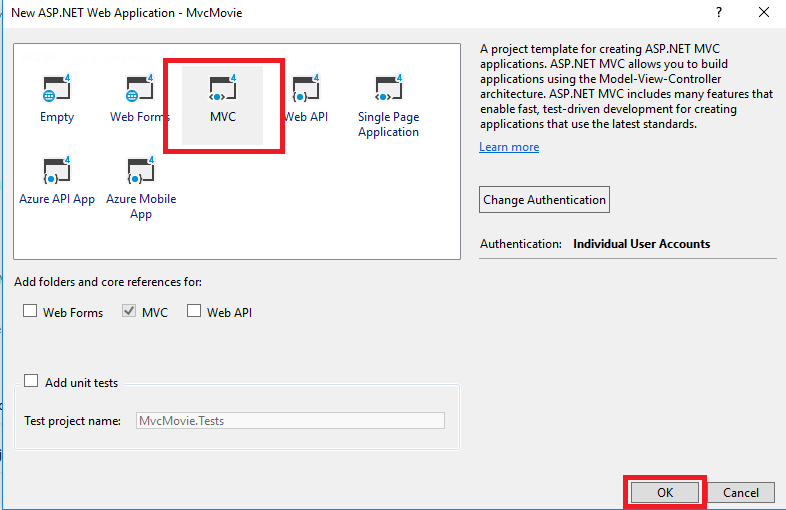
## 3.Creating Your First Application

Click New Project, then select Visual C# on the left, then Web and then select ASP.NET Web Application (.NET Framework). Name your project "MvcMovie" and then click OK.

****

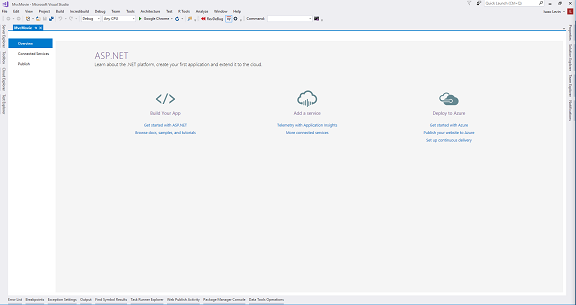
Figure

In the New ASP.NET Project dialog, click MVC and then click OK.



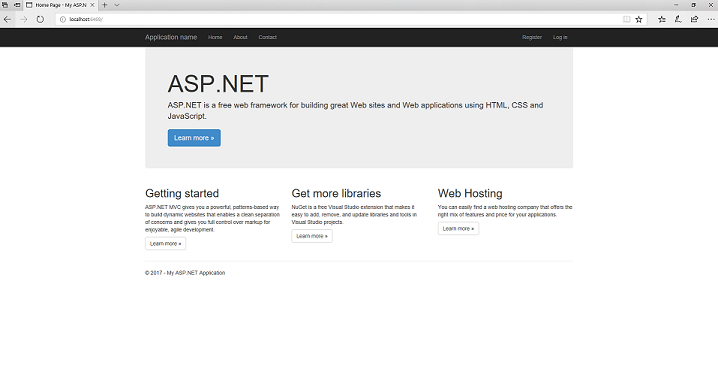
Figure

Visual Studio used a default template for the ASP.NET MVC project you just created, so you have a working application right now without doing anything! This is a simple "Hello World!" project, and it's a good place to start your application.



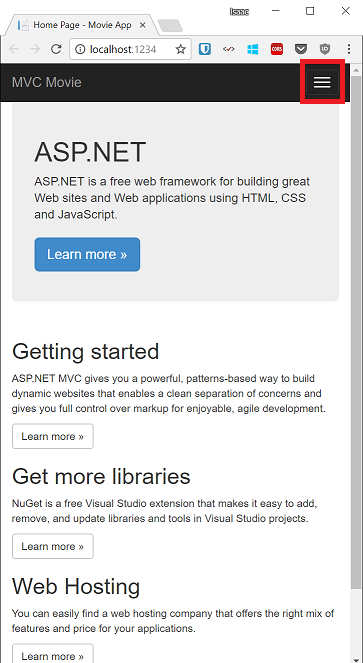
Figure

Click F5 to start debugging. F5 causes Visual Studio to start [IIS Express](https://www.iis.net/learn/extensions/introduction-to-iis-express/iis-express-overview) and run your web app. Visual Studio then launches a browser and opens the application's home page. Notice that the address bar of the browser says localhost:port# and not something like example.com. That's because localhost always points to your own local computer, which in this case is running the application you just built. When Visual Studio runs a web project, a random port is used for the web server. In the image below, the port number is 1234. When you run the application, you'll see a different port number.



Figure

Right out of the box this default template gives you Home, Contact and About pages. The image above doesn't show the Home, About and Contact links. Depending on the size of your browser window, you might need to click the navigation icon to see these links.



Figure

The application also provides support to register and log in. The next step is to change how this application works and learn a little bit about ASP.NET MVC. Close the ASP.NET MVC application and let's change some code.

## 4. Adding a Controller

MVC stands for model-view-controller. MVC is a pattern for developing applications that are well architected, testable and easy to maintain. MVC-based applications contain:

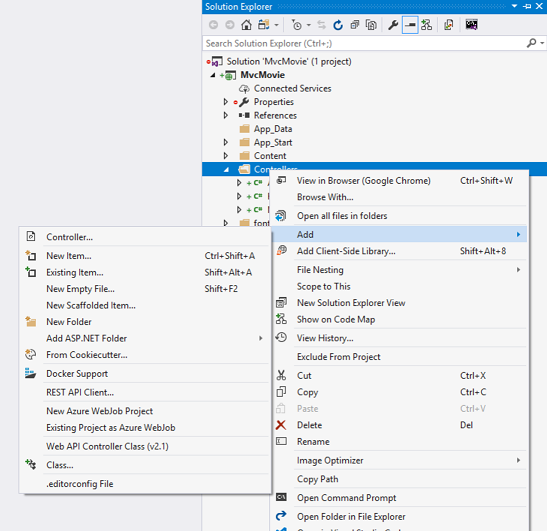
Models: Classes that represent the data of the application and that use validation logic to enforce business rules for that data.

Views: Template files that your application uses to dynamically generate HTML responses.

Controllers: Classes that handle incoming browser requests, retrieve model data, and then specify view templates that return a response to the browser.

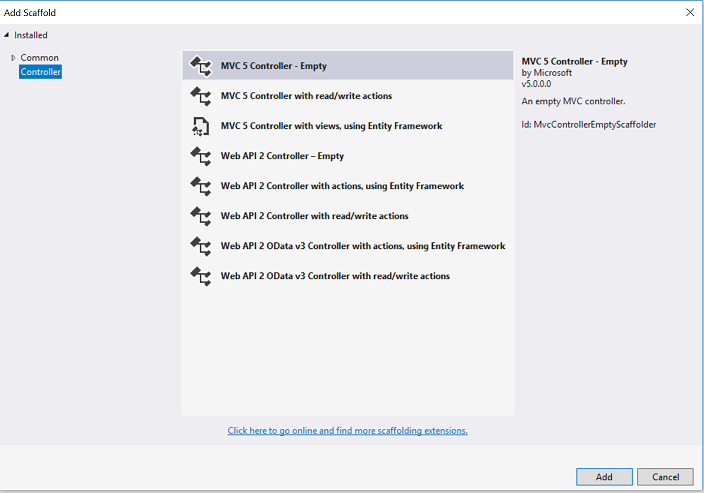
We'll be covering all these concepts in this tutorial series and show you how to use them to build an application.

Let's begin by creating a controller class. In Solution Explorer, right-click the Controllers folder and then click Add, then Controller.



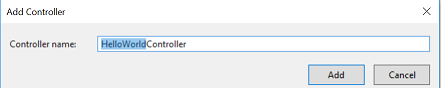
Figure

In the Add Scaffold dialog box, click MVC 5 Controller - Empty, and then click Add.



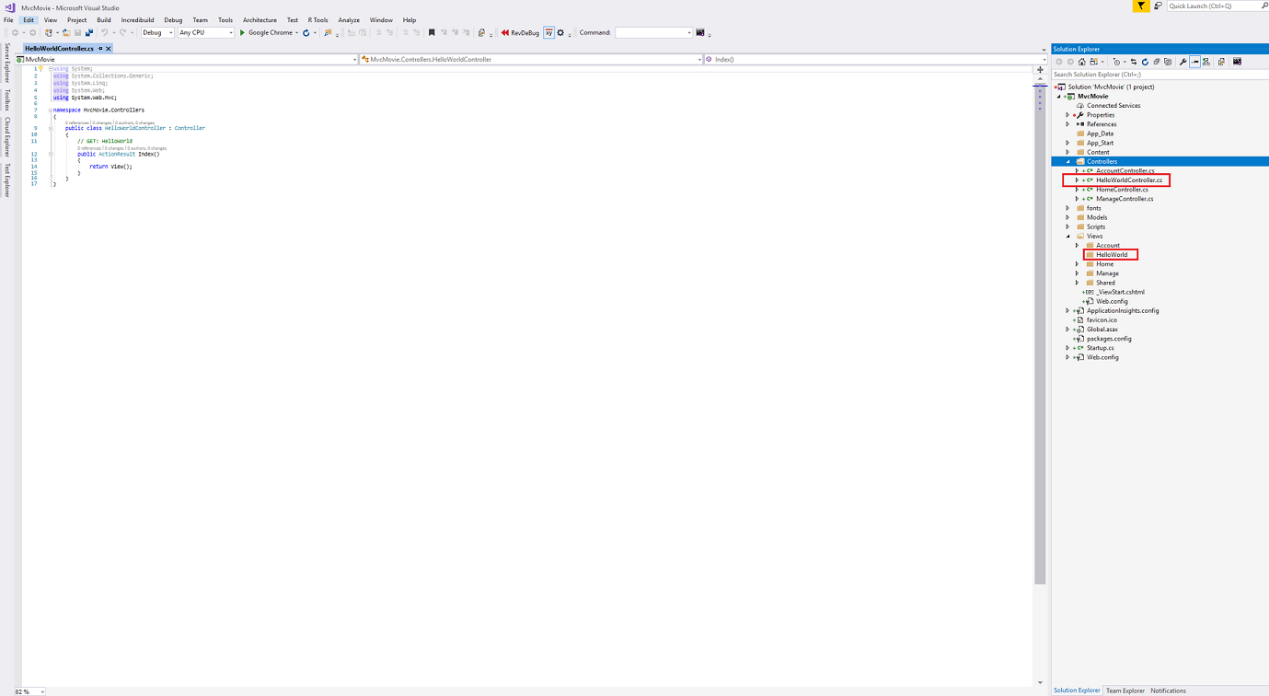
Figure

Name your new controller "HelloWorldController" and click Add.



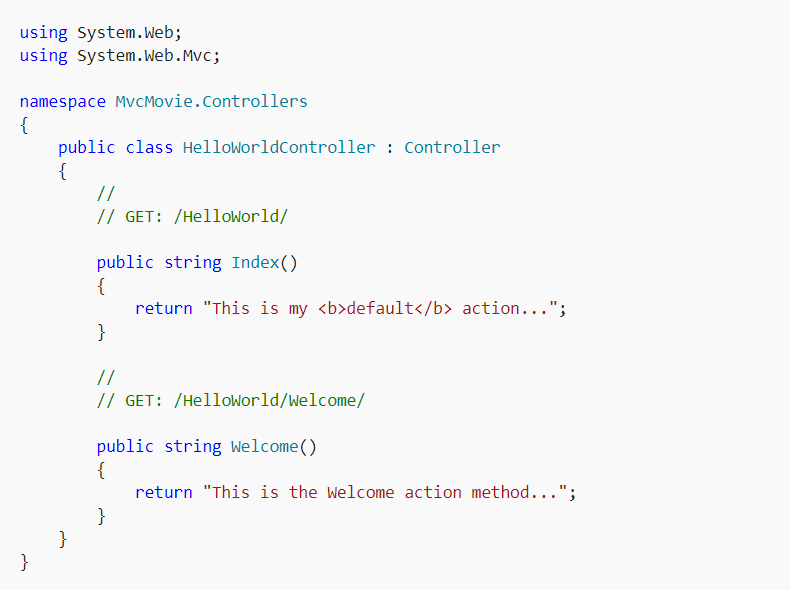
Figure

Notice in Solution Explorer that a new file has been created named HelloWorldController.cs and a new folder Views\HelloWorld. The controller is open in the IDE.



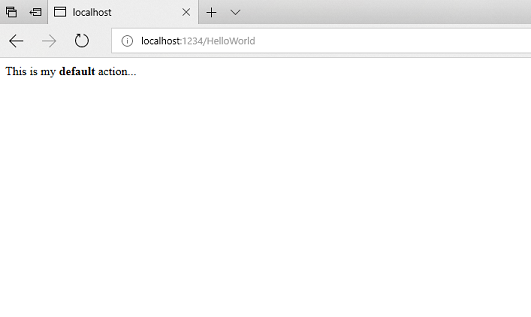
Figure

Replace the contents of the file with the following code.



Figure

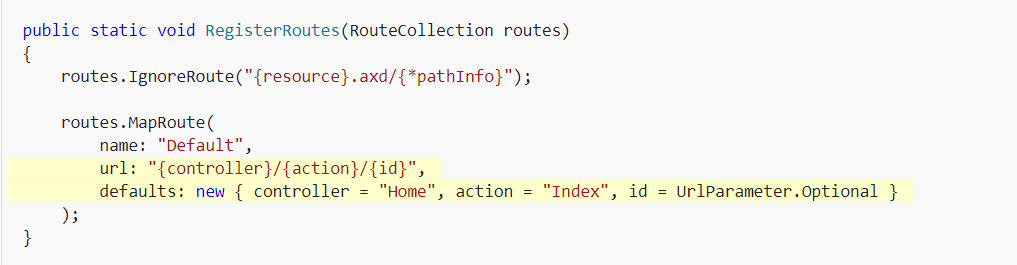
The controller methods will return a string of HTML as an example. The controller is named HelloWorldController and the first method is named Index. Let's invoke it from a browser. Run the application (press F5 or Ctrl+F5). In the browser, append "HelloWorld" to the path in the address bar. (For example, in the illustration below, it's http://localhost:1234/HelloWorld.) The page in the browser will look like the following screenshot. In the method above, the code returned a string directly. You told the system to just return some HTML, and it did!



Figure

ASP.NET MVC invokes different controller classes (and different action methods within them) depending on the incoming URL. The default URL routing logic used by ASP.NET MVC uses a format like this to determine what code to invoke:/[Controller]/[ActionName]/[Parameters].

You set the format for routing in the App\_Start/RouteConfig.cs file.

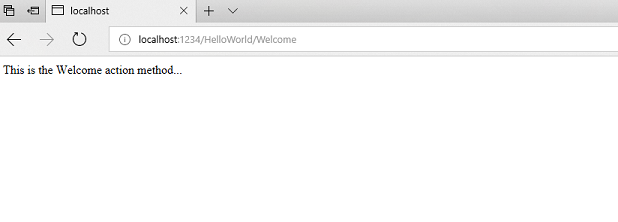


Figure

When you run the application and don't supply any URL segments, it defaults to the "Home" controller and the "Index" action method specified in the defaults section of the code above.

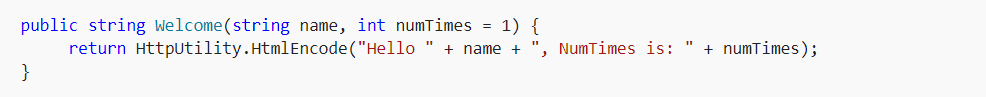
The first part of the URL determines the controller class to execute. So /HelloWorld maps to the HelloWorldController class. The second part of the URL determines the action method on the class to execute. So /HelloWorld/Index would cause the Index method of the HelloWorldController class to execute. Notice that we only had to browse to /HelloWorld and the Index method was used by default. This is because a method named Index is the default method that will be called on a controller if one is not explicitly specified. The third part of the URL segment ( Parameters) is for route data. We'll see route data later on in this tutorial.

Browse to http://localhost:xxxx/HelloWorld/Welcome. The Welcome method runs and returns the string "This is the Welcome action method...". The default MVC mapping is /[Controller]/[ActionName]/[Parameters]. For this URL, the controller is HelloWorld and Welcome is the action method. You haven't used the [Parameters] part of the URL yet.



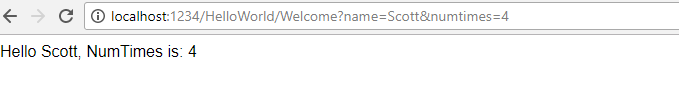
Figure

Let's modify the example slightly so that you can pass some parameter information from the URL to the controller (for example, */HelloWorld/Welcome?name=Scott&numtimes=4*). Change your Welcomemethod to include two parameters as shown below. Note that the code uses the C# optional-parameter feature to indicate that the numTimes parameter should default to 1 if no value is passed for that parameter.



Figure

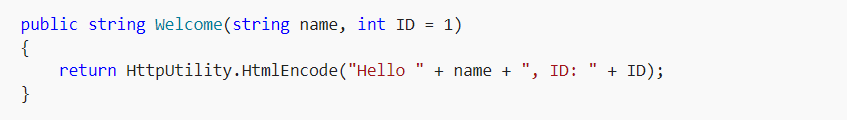
Run your application and browse to the example URL (http://localhost:xxxx/HelloWorld/Welcome?name=Scott&numtimes=4). You can try different values for name and numtimes in the URL. The [ASP.NET MVC model binding system](http://odetocode.com/Blogs/scott/archive/2009/04/27/6-tips-for-asp-net-mvc-model-binding.aspx) automatically maps the named parameters from the query string in the address bar to parameters in your method.



Figure

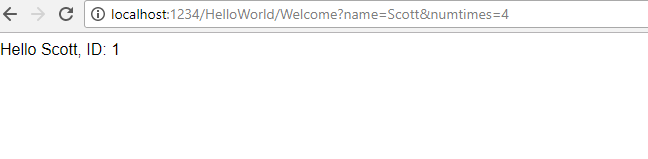
In the sample above, the URL segment ( Parameters) is not used, the name and numTimesparameters are passed as [query strings](http://en.wikipedia.org/wiki/Query_string). The ? (question mark) in the above URL is a separator, and the query strings follow. The & character separates query strings.

Replace the Welcome method with the following code:



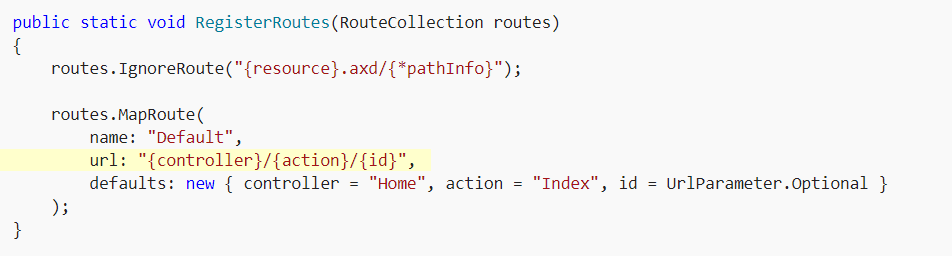
Figure

Run the application and enter the following URL: <http://localhost:xxx/HelloWorld/Welcome/1?name=Scott>



Figure

This time the third URL segment matched the route parameter ID. The Welcome action method contains a parameter (ID) that matched the URL specification in the RegisterRoutes method.



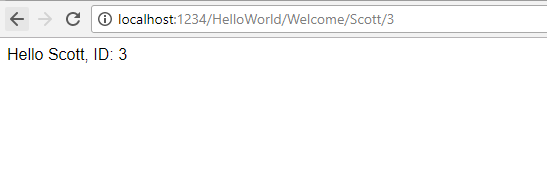
Figure

In ASP.NET MVC applications, it's more typical to pass in parameters as route data (like we did with ID above) than passing them as query strings. You could also add a route to pass both the name and numtimes in parameters as route data in the URL. In the App\_Start\RouteConfig.cs file, add the "Hello" route:



Figure

Run the application and browse to /localhost:XXX/HelloWorld/Welcome/Scott/3.



Figure

For many MVC applications, the default route works fine. You'll learn later in this tutorial to pass data using the model binder, and you won't have to modify the default route for that.

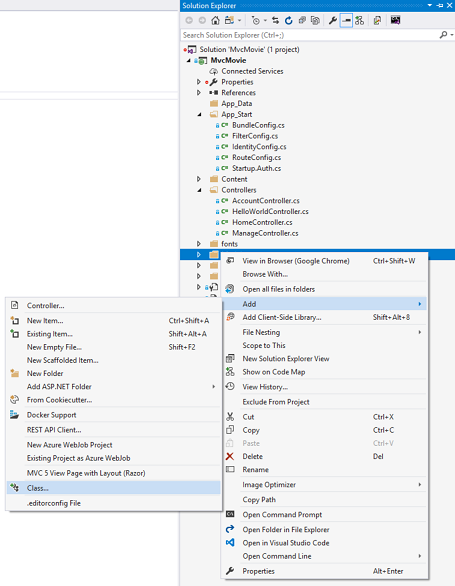
In these examples the controller has been doing the "VC" portion of MVC — that is, the view and controller work. The controller is returning HTML directly. Ordinarily you don't want controllers returning HTML directly, since that becomes very cumbersome to code. Instead we'll typically use a separate view template file to help generate the HTML response. Let's look next at how we can do this.

## 5.Adding a Model

In this section you'll add some classes for managing movies in a database. These classes will be the "model" part of the ASP.NET MVC app.

You'll use a .NET Framework data-access technology known as the [Entity Framework](https://docs.microsoft.com/ef/) to define and work with these model classes. The Entity Framework (often referred to as EF) supports a development paradigm called *Code First*. Code First allows you to create model objects by writing simple classes. (These are also known as POCO classes, from "plain-old CLR objects.") You can then have the database created on the fly from your classes, which enables a very clean and rapid development workflow. If you are required to create the database first, you can still follow this tutorial to learn about MVC and EF app development. You can then follow Tom Fizmakens [ASP.NET Scaffolding](https://docs.microsoft.com/en-us/aspnet/visual-studio/overview/2013/aspnet-scaffolding-overview) tutorial, which covers the database first approach.

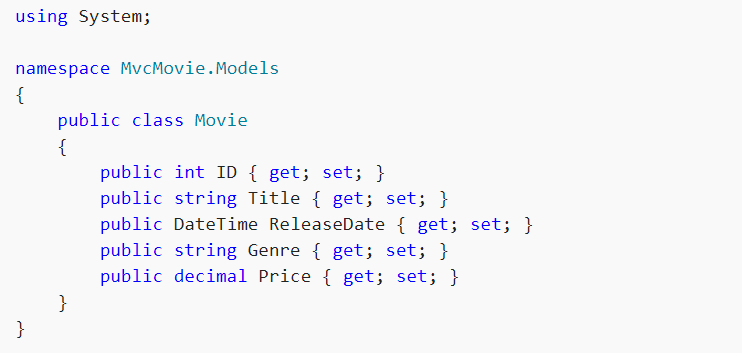
In Solution Explorer, right click the Models folder, select Add, and then select Class.



Figure

Enter the class name "Movie".

Add the following five properties to the Movie class:

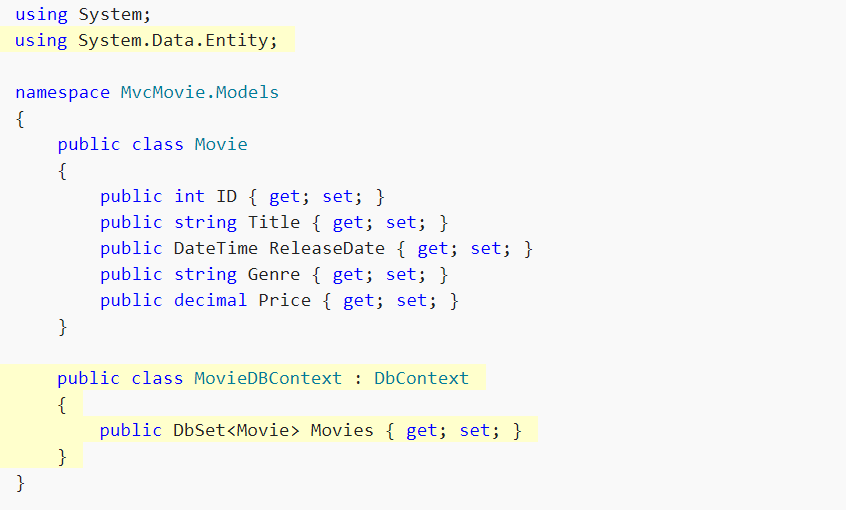


Figure

We'll use the Movie class to represent movies in a database. Each instance of a Movie object will correspond to a row within a database table, and each property of the Movie class will map to a column in the table.

Note: In order to use System.Data.Entity, and the related class, you need to install the [Entity Framework NuGet Package](https://www.nuget.org/packages/EntityFramework/). Follow the link for further instructions.

In the same file, add the following MovieDBContext class:



Figure

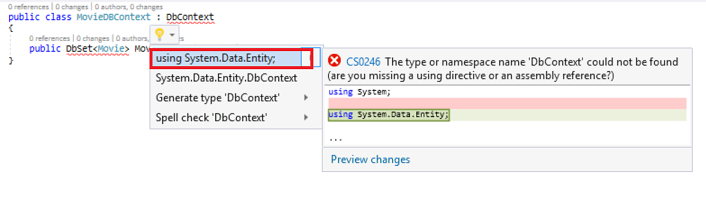
The MovieDBContext class represents the Entity Framework movie database context, which handles fetching, storing, and updating Movie class instances in a database. The MovieDBContext derives from the DbContext base class provided by the Entity Framework.

In order to be able to reference DbContext and DbSet, you need to add the following usingstatement at the top of the file:



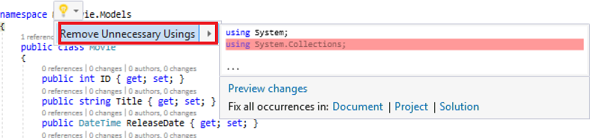
Figure

You can do this by manually adding the using statement, or you can hover over the red squiggly lines, click Show potential fixes and click using System.Data.Entity;



Figure

Note: Several unused using statements have been removed. Visual Studio will show unused dependencies as gray. You can remove unnused dependencies by hovering over the gray dependencies, click Show potential fixes and click Remove Unused Usings.



Figure

We've finally added a model (the M in MVC).