ASP.NET MVC

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

About the Tutorial

ASP.NET MVC is an open-source software from Microsoft. Its web development framework combines the features of MVC (Model-View-Controller) architecture, the most up-to-date ideas and techniques from Agile development and the best parts of the existing ASP.NET platform.

This tutorial provides a complete picture of the MVC framework and teaches you how to build an application using this tool.

Audience

This tutorial is designed for all those developers who are keen on developing best-in-class applications using ASP.NET MVC. The tutorial provides a hands-on approach to the subject with step-by-step program examples that will assist you to learn and put the acquired knowledge into practice.

After completing this tutorial, you will have a better understating of Windows apps and learn what you can do with Windows application using XAML and C#.

Prerequisites

To gain advantage of this tutorial, you need to be familiar with programming for Windows. You also need to know the basics of C#.

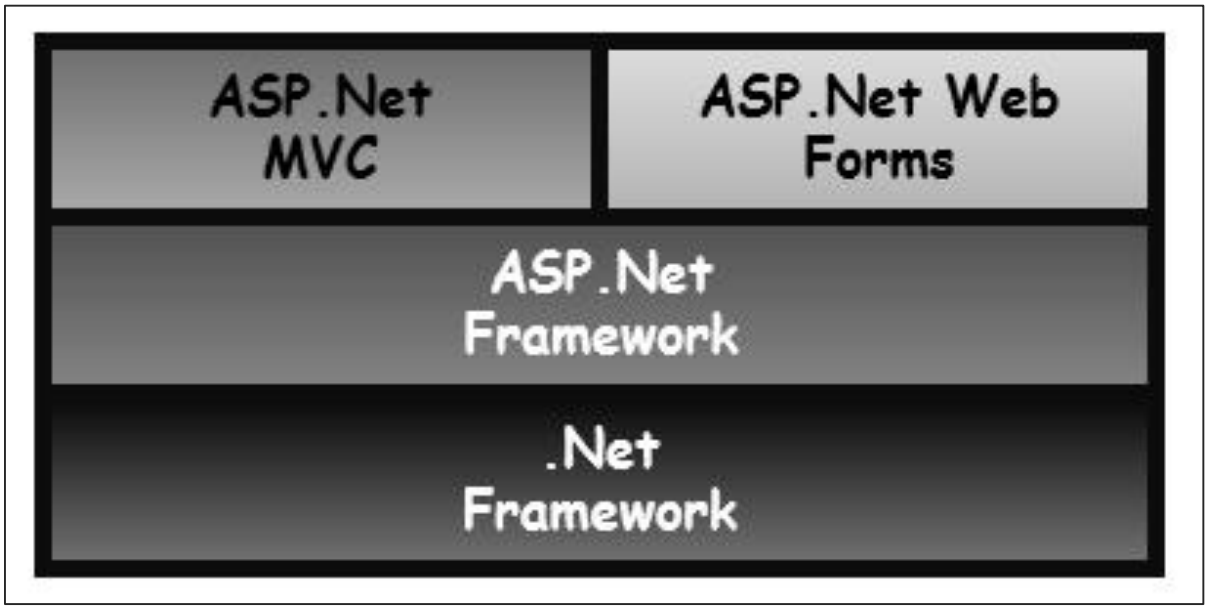
Disclaimer & Copyright

Table of Contents

1. ASP.NET MVC – OVERVIEW

ASP.NET MVC is basically a web development framework from Microsoft, which combines the features of MVC (Model-View-Controller) architecture, the most up-to-date ideas and techniques from Agile development, and the best parts of the existing ASP.NET platform.

ASP.NET MVC is not something, which is built from ground zero. It is a complete alternative to traditional ASP.NET Web Forms. It is built on the top of ASP.NET, so developers enjoy almost all the ASP.NET features while building the MVC application.



History

ASP.NET 1.0 was released on January 5, 2002, as part of .Net Framework version 1.0. At that time, it was easy to think of ASP.NET and Web Forms as one and the same thing. ASP.NET has however always supported two layers of abstraction:

- **System.Web.UI:** The Web Forms layer, comprising server controls, ViewState, and so on.

- **System.Web:** It supplies the basic web stack, including modules, handlers, the HTTP stack, etc.

By the time ASP.NET MVC was announced in 2007, the MVC pattern was becoming one of the most popular ways of building web frameworks.

In April 2009, the ASP.NET MVC source code was released under the Microsoft Public License (MS-PL). "ASP.NET MVC framework is a lightweight, highly testable presentation framework that is integrated with the existing ASP.NET features.

Some of these integrated features are master pages and membership-based authentication. The MVC framework is defined in the System.Web.Mvc assembly.

In March 2012, Microsoft had released part of its web stack (including ASP.NET MVC, Razor and Web API) under an open source license (Apache License 2.0). ASP.NET Web Forms was not included in this initiative.

Why ASP.NET MVC?

Microsoft decided to create their own MVC framework for building web applications. The MVC framework simply builds on top of ASP.NET. When you are building a web application with ASP.NET MVC, there will be no illusions of state, there will not be such a thing as a page load and no page life cycle at all, etc.

Another design goal for ASP.NET MVC was to be extensible throughout all aspects of the framework. So when we talk about views, views have to be rendered by a particular type of view engine. The default view engine is still something that can take an ASPX file. But if you don't like using ASPX files, you can use something else and plug in your own view engine.

There is a component inside the MVC framework that will instantiate your controllers. You might not like the way that the MVC framework instantiates your controller, you might want to handle that job yourself. So, there are lots of places in MVC where you can inject your own custom logic to handle tasks.

The whole idea behind using the Model View Controller design pattern is that you maintain a separation of concerns. Your controller is no longer encumbered with a lot of ties to the ASP.NET runtime or ties to the ASPX page, which is very hard to test. You now just have a class with regular methods on it that you can invoke in unit tests to find out if that controller is going to behave correctly.

Benefits of ASP.NET MVC

Following are the benefits of using ASP.NET MVC:

- Makes it easier to manage complexity by dividing an application into the model, the view, and the controller.
- Enables full control over the rendered HTML and provides a clean separation of concerns.
- Direct control over HTML also means better accessibility for implementing compliance with evolving Web standards.
- Facilitates adding more interactivity and responsiveness to existing apps.

- Provides better support for test-driven development (TDD).
- Works well for Web applications that are supported by large teams of developers and for Web designers who need a high degree of control over the application behavior.

2. ASP.NET MVC – MVC PATTERN

The MVC (Model-View-Controller) design pattern has actually been around for a few decades, and it's been used across many different technologies. Everything from Smalltalk to C++ to Java, and now C Sharp and .NET use this design pattern to build a user interface.

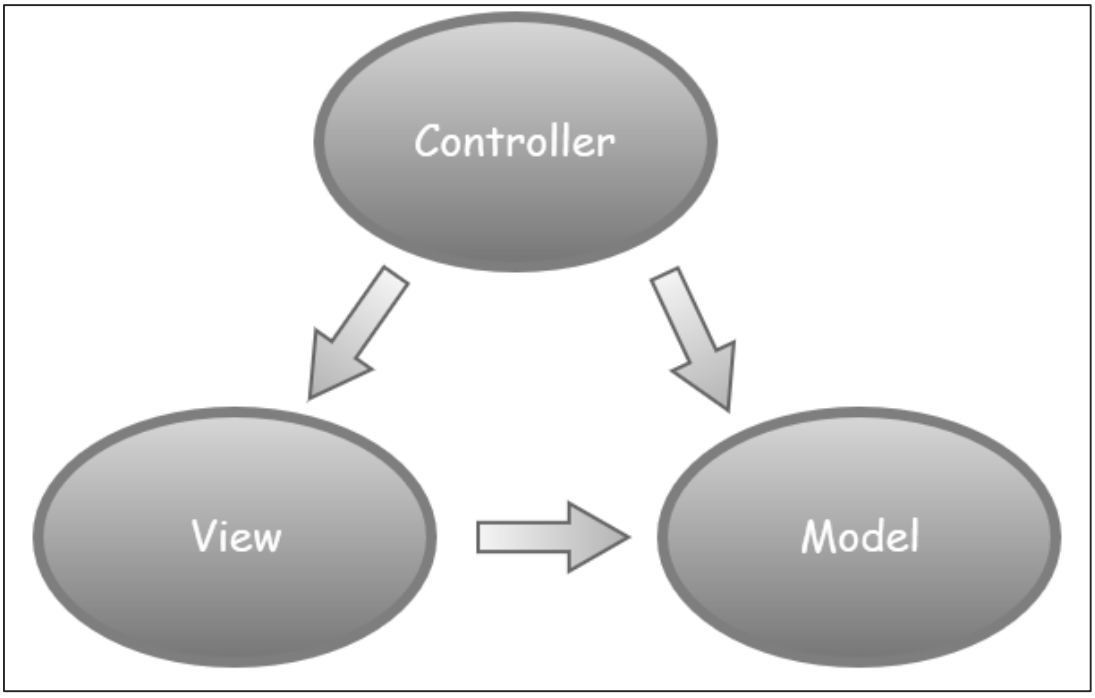
Following are some salient features of the MVC pattern:

- Originally it was named Thing-Model-View-Editor in 1979, and then it was later simplified to Model- View-Controller.

- It is a powerful and elegant means of separating concerns within an application (for example, separating data access logic from display logic) and applies itself extremely well to web applications.

- Its explicit separation of concerns does add a small amount of extra complexity to an application's design, but the extraordinary benefits outweigh the extra effort.

The MVC architectural pattern separates the user interface (UI) of an application into three main parts.



- **The Model**: A set of classes that describes the data you are working with as well as the business logic.

- **The View**: Defines how the application's UI will be displayed. It is a pure HTML, which decides how the UI is going to look like.

- **The Controller**: A set of classes that handles communication from the user, overall application flow, and application-specific logic.

Idea Behind MVC

The idea is that you'll have a component called the view, which is solely responsible for rendering this user interface whether that be HTML or whether it actually be UI widgets on a desktop application.

The view talks to a model, and that model contains all of the data that the view needs to display. Views generally don't have much logic inside of them at all.

In a web application, the view might not have any code associated with it at all. It might just have HTML and then some expressions of where to take pieces of data from the model and plug them into the correct places inside the HTML template that you've built in the view.

The controller that organizes is everything. When an HTTP request arrives for an MVC application, that request gets routed to a controller, and then it's up to the controller to talk to either the database, the file system, or the model.

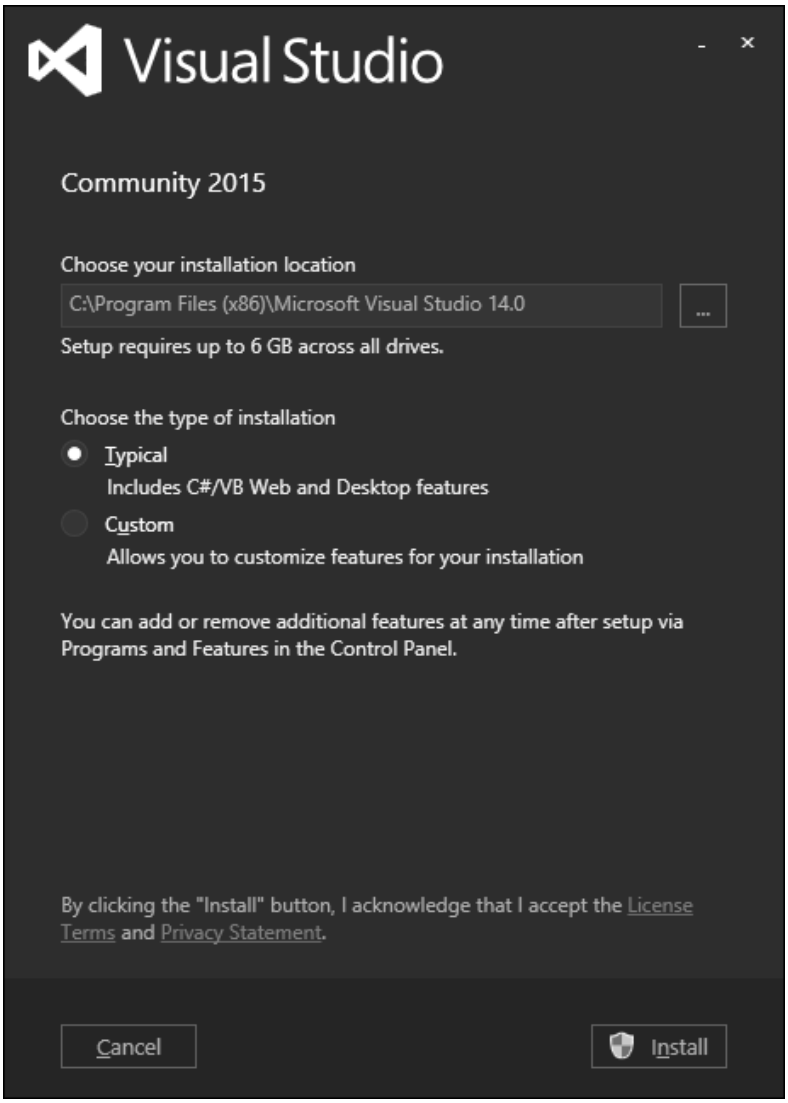
3. ASP.NET MVC – ENVIRONMENT SETUP

MVC development tool is included with Visual Studio 2012 and onwards. It can also be installed on Visual Studio 2010 SP1/Visual Web Developer 2010 Express SP1. If you are using Visual Studio 2010, you can install MVC 4 using the Web Platform Installer http://www.microsoft.com/web/gallery/install.aspx?appid=MVC4VS2010

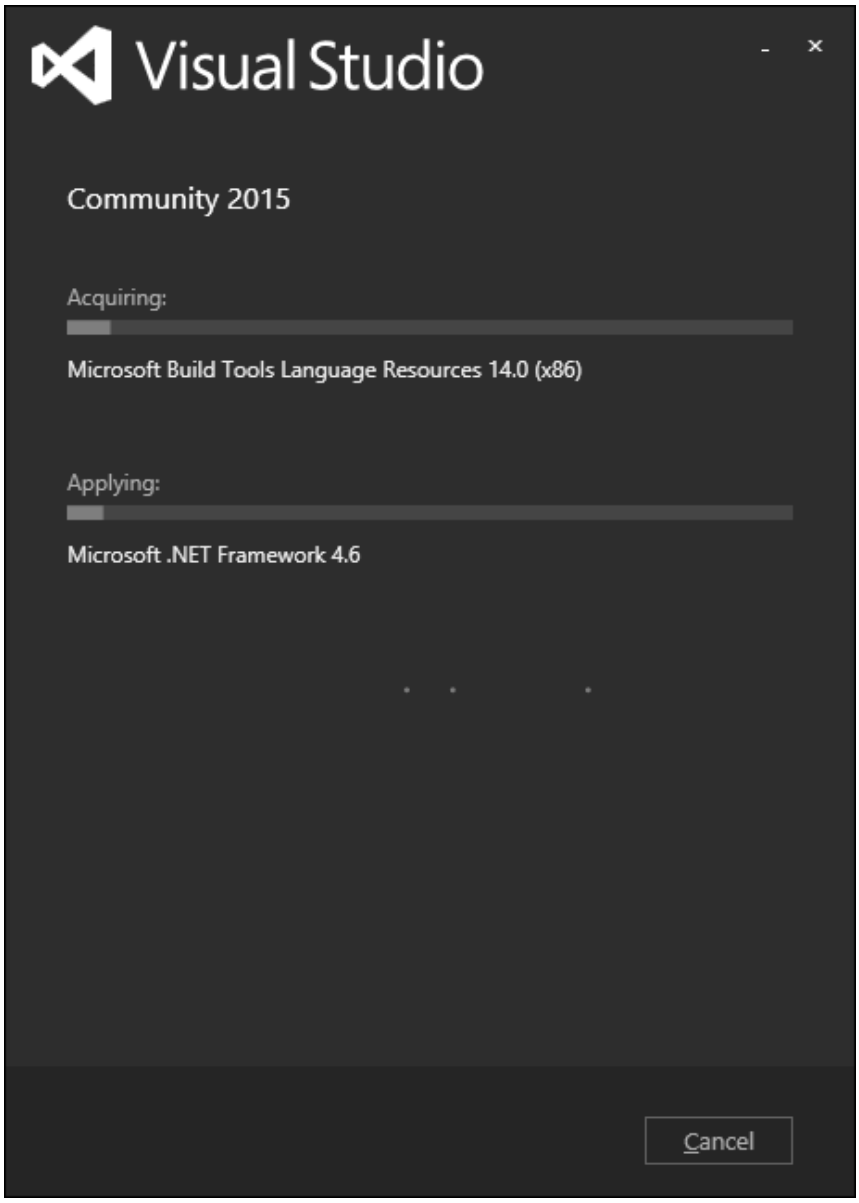
Microsoft provides a free version of Visual Studio, which also contains SQL Server and it can be downloaded from https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx.

Installation

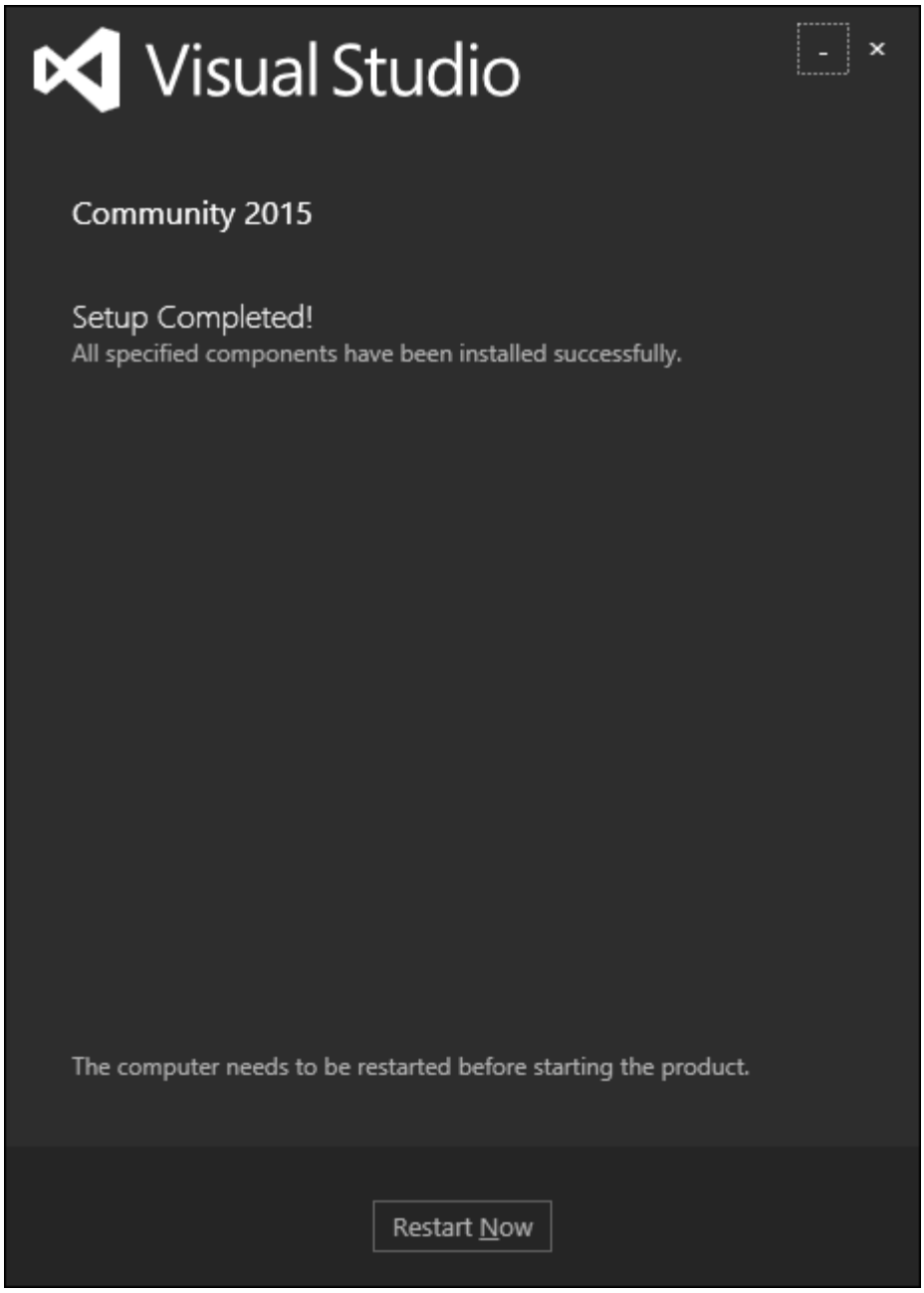
Step (1): Once downloading is complete, run the installer. The following dialog will be displayed.



Step (2): Click the 'Install' button and it will start the installation process.

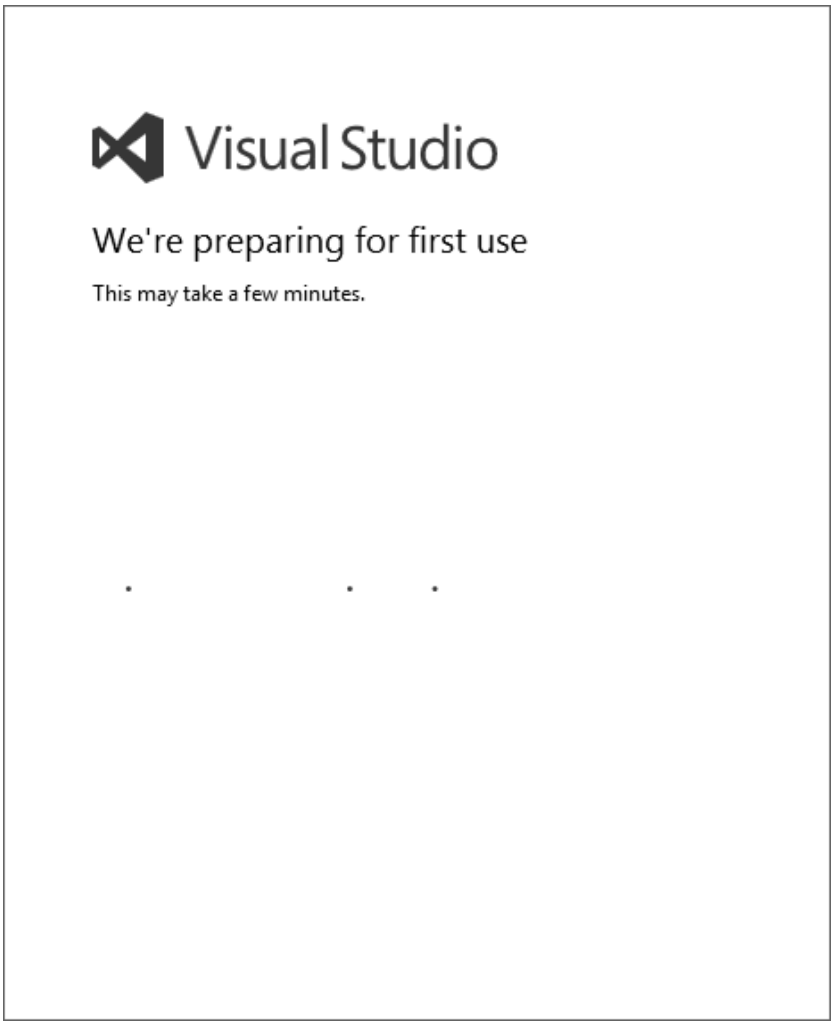


Once the installation process is completed successfully, you will see the following dialog.

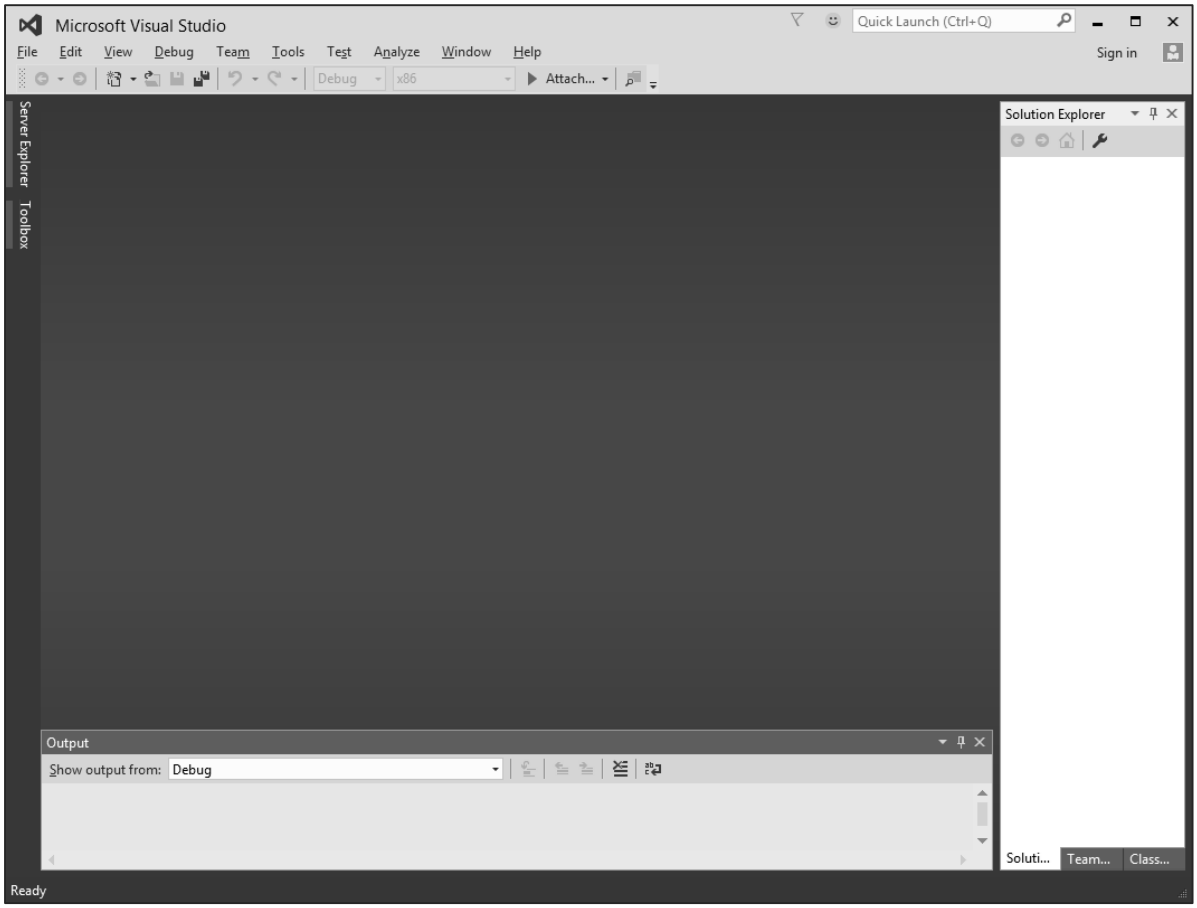


Step (3): Close this dialog and restart your computer if required.

Step (4): Open Visual Studio from the Start Menu, which will open the following dialog. It will take a while for the first time only for preparation.



Once all is done, you will see the main window of Visual Studio as shown in the following screenshot.



You are now ready to start your application.

4. ASP.NET MVC – GETTING STARTED

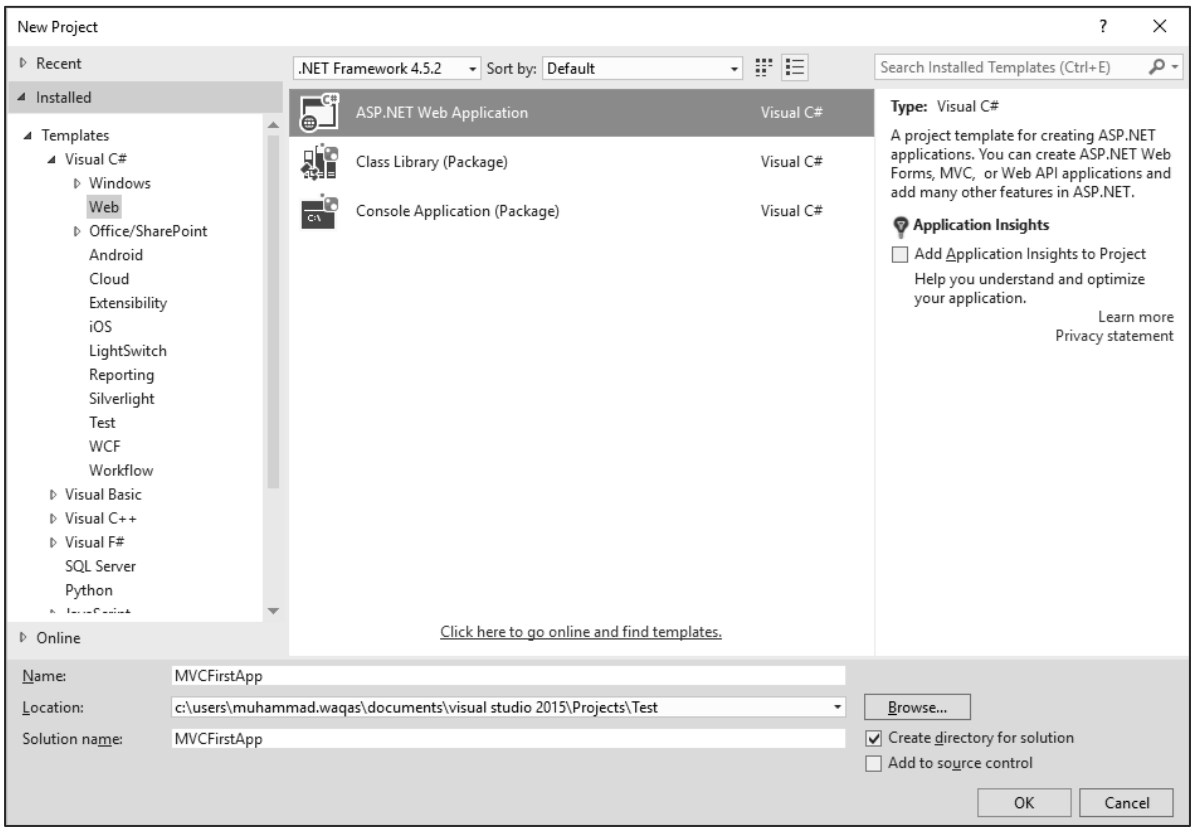
In this chapter, we will look at a simple working example of ASP.NET MVC. We will be building a simple web app here. To create an ASP.NET MVC application, we will use Visual Studio 2015, which contains all of the features you need to create, test, and deploy an MVC Framework application.

Create ASP.Net MVC Application

Following are the steps to create a project using project templates available in Visual Studio.

Step (1): Open the Visual Studio. Click File -> New -> Project menu option.

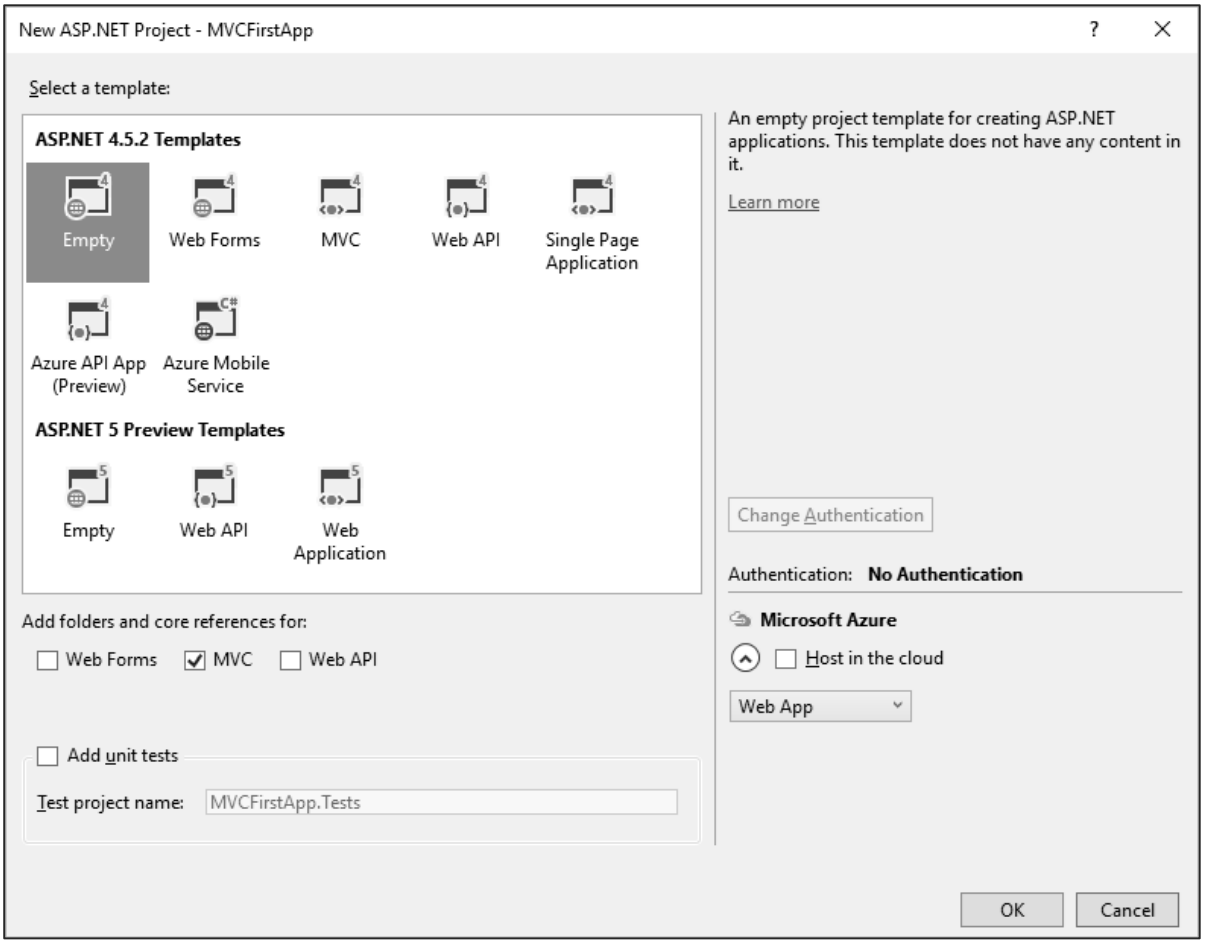
A new Project dialog opens.



Step (2): From the left pane, select Templates -> Visual C# -> Web.

Step (3): In the middle pane, select ASP.NET Web Application.

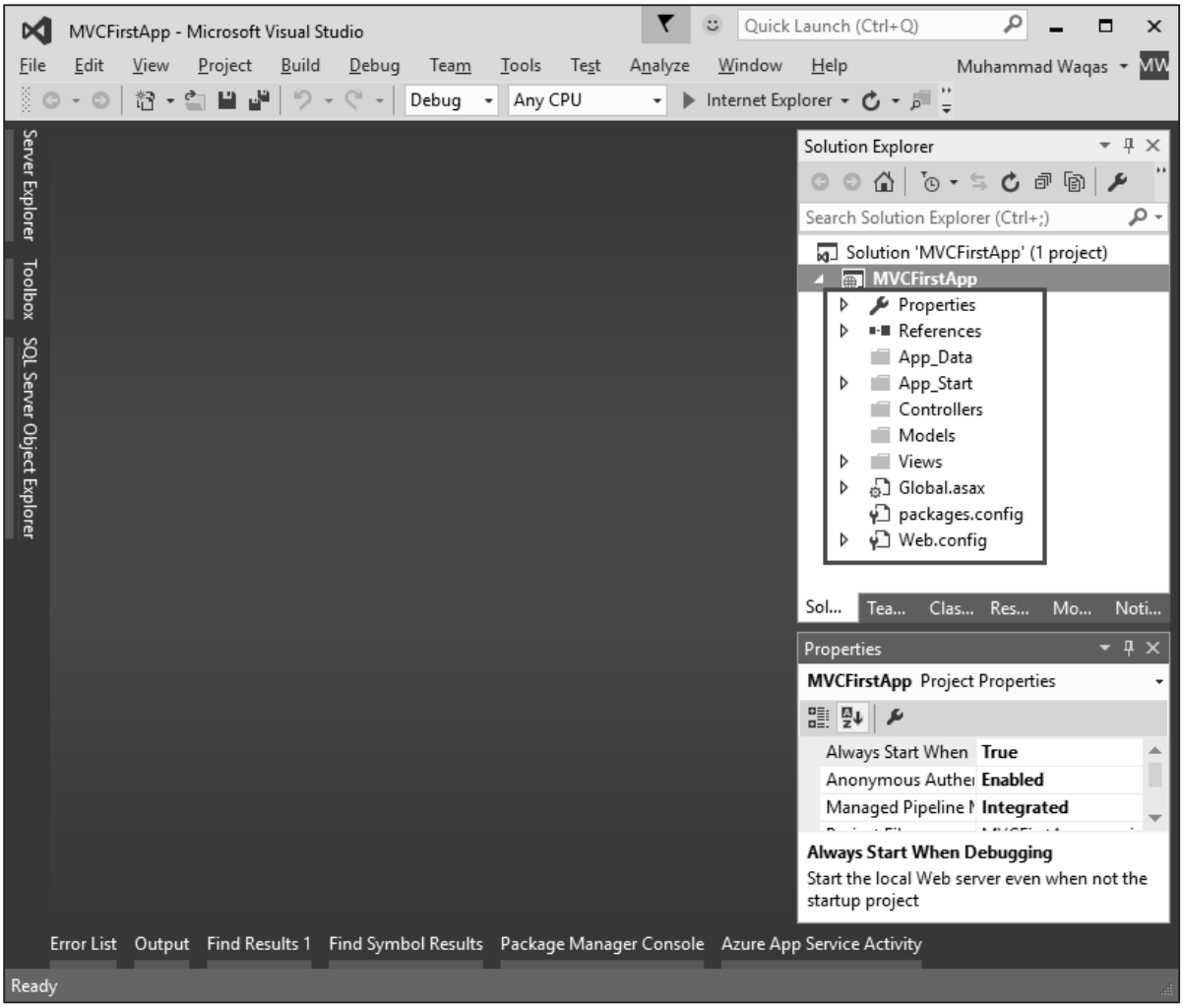
Step (4): Enter the project name, MVCFirstApp, in the Name field and click ok to continue. You will see the following dialog which asks you to set the initial content for the ASP.NET project.



Step (5): To keep things simple, select the 'Empty' option and check the MVC checkbox in the Add folders and core references section. Click Ok.

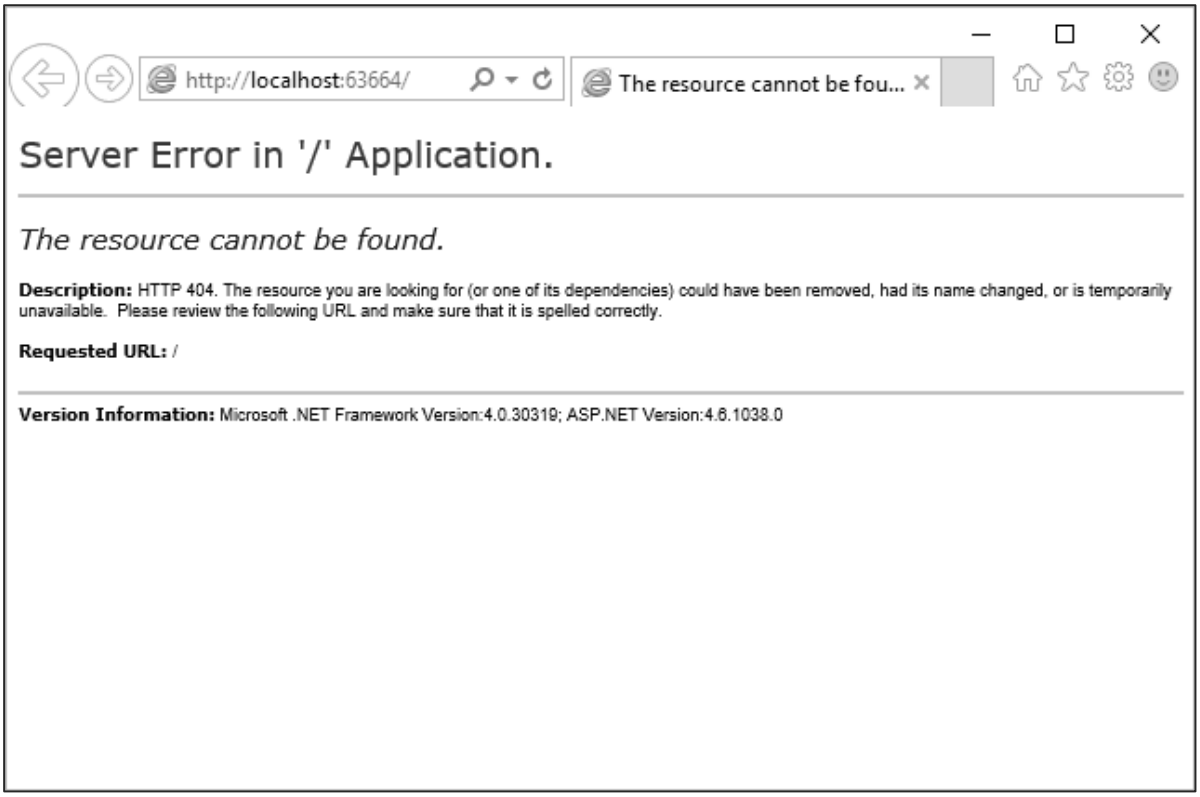
It will create a basic MVC project with minimal predefined content.

Once the project is created by Visual Studio, you will see a number of files and folders displayed in the Solution Explorer window.



As you know that we have created ASP.Net MVC project from an empty project template, so for the moment the application does not contain anything to run.

Step (6): Run this application from Debug -> Start Debugging menu option and you will see a **404 Not Found** Error.

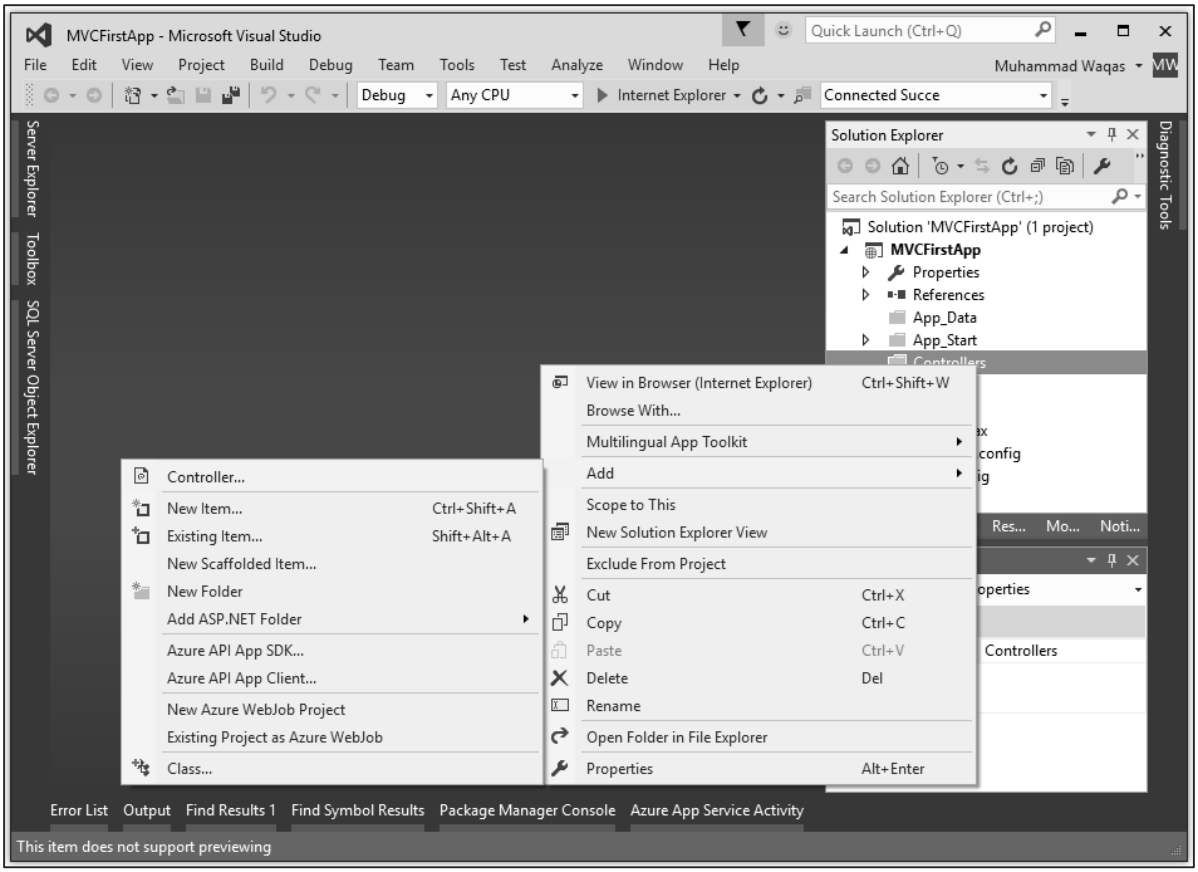


The default browser is, Internet Explorer, but you can select any browser that you have installed from the toolbar.

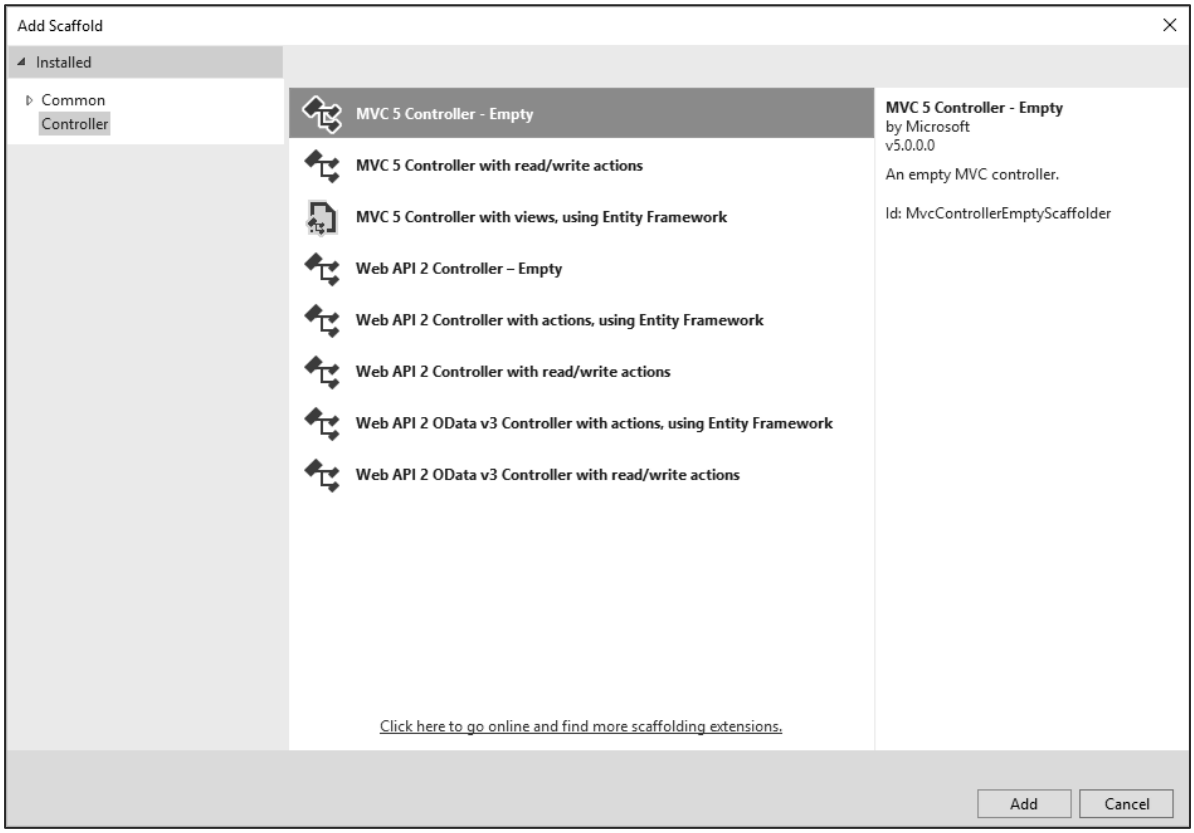
Add Controller

To remove the 404 Not Found error, we need to add a controller, which handles all the incoming requests.

Step (1): To add a controller, right-click on the controller folder in the solution explorer and select Add -> Controller.

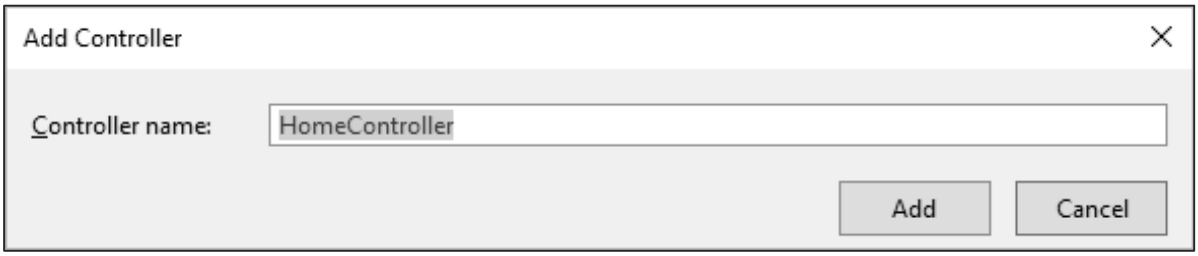


It will display the Add Scaffold dialog.



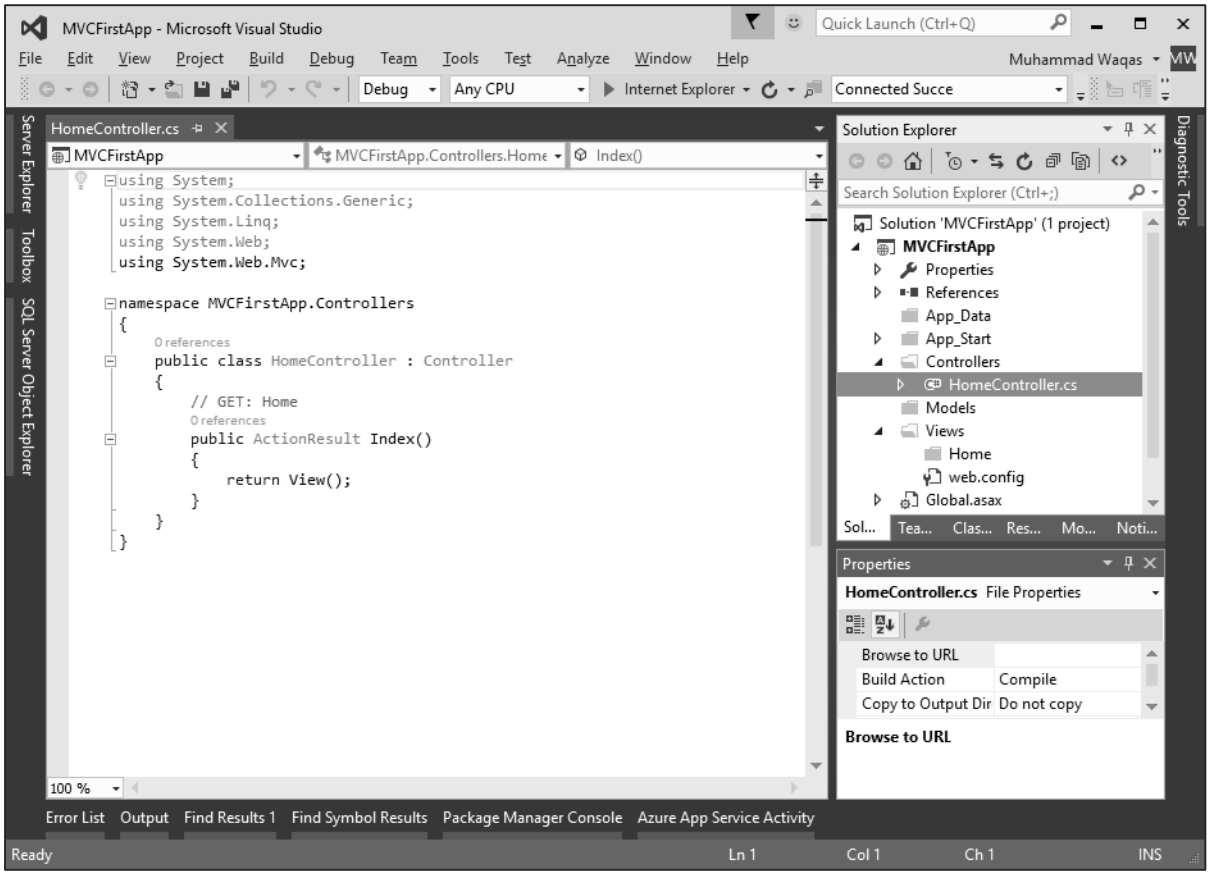
Step (2): Select the MVC 5 Controller – Empty option and click 'Add' button.

The Add Controller dialog will appear.



Step (3): Set the name to HomeController and click the Add button.

You will see a new C# file HomeController.cs in the Controllers folder, which is open for editing in Visual Studio as well.

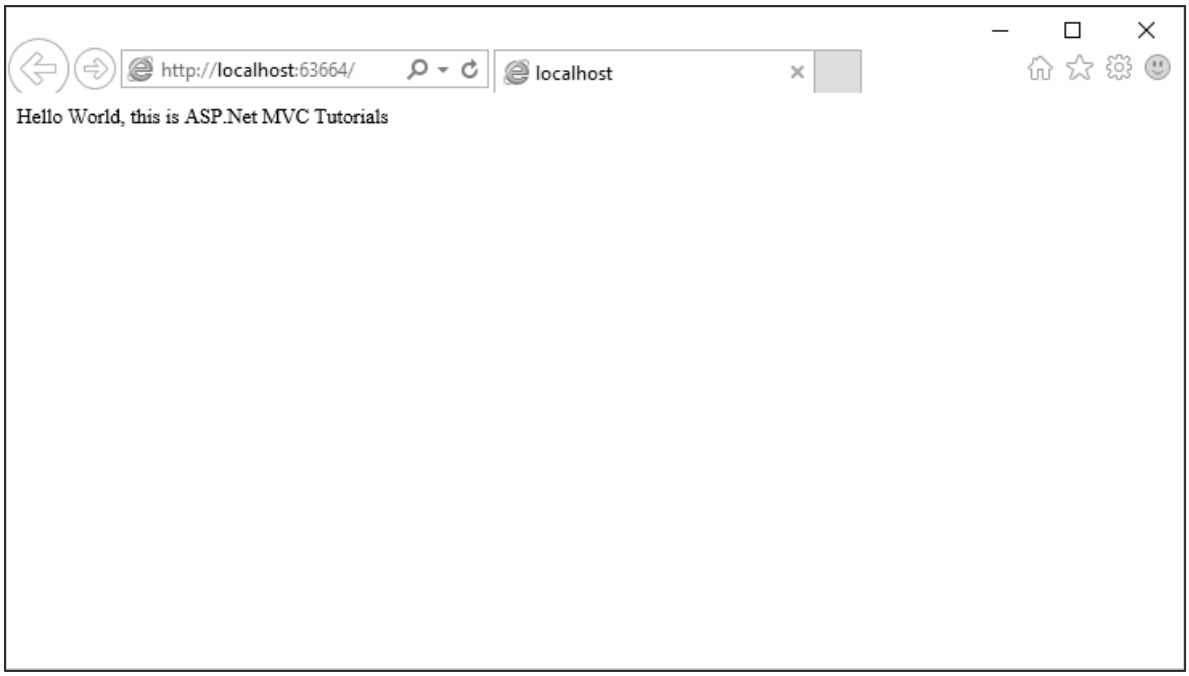


Step (4): To make this a working example, let's modify the controller class by changing the action method called **Index** using the following code.

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

namespace MVCFirstApp.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public string Index()
        {
```

```
            return "Hello World, this is ASP.Net MVC Tutorials";
        }
    }
}
```

Step (5): Run this application and you will see that the browser is displaying the result of the Index action method.



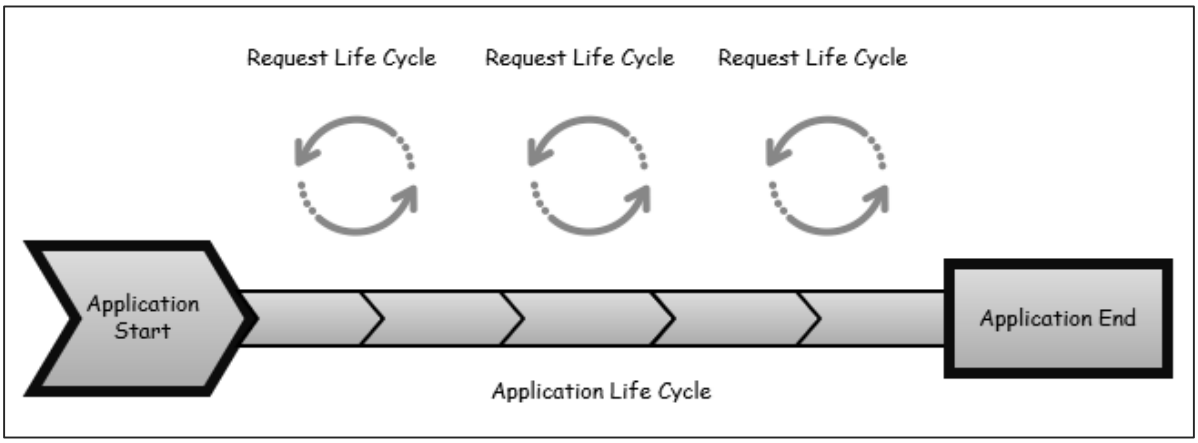
5. ASP.NET MVC – LIFE CYCLE

In this chapter, we will discuss the overall MVC pipeline and the life of an HTTP request as it travels through the MVC framework in ASP.NET. At a high level, a life cycle is simply a series of steps or events used to handle some type of request or to change an application state. You may already be familiar with various framework life cycles, the concept is not unique to MVC.

For example, the ASP.NET webforms platform features a complex page life cycle. Other .NET platforms, like Windows phone apps, have their own application life cycles. One thing that is true for all these platforms regardless of the technology is that understanding the processing pipeline can help you better leverage the features available and MVC is no different.

MVC has two life cycles:

- The application life cycle
- The request life cycle



The Application Life Cycle

The application life cycle refers to the time at which the application process actually begins running IIS until the time it stops. This is marked by the application start and end events in the startup file of your application.

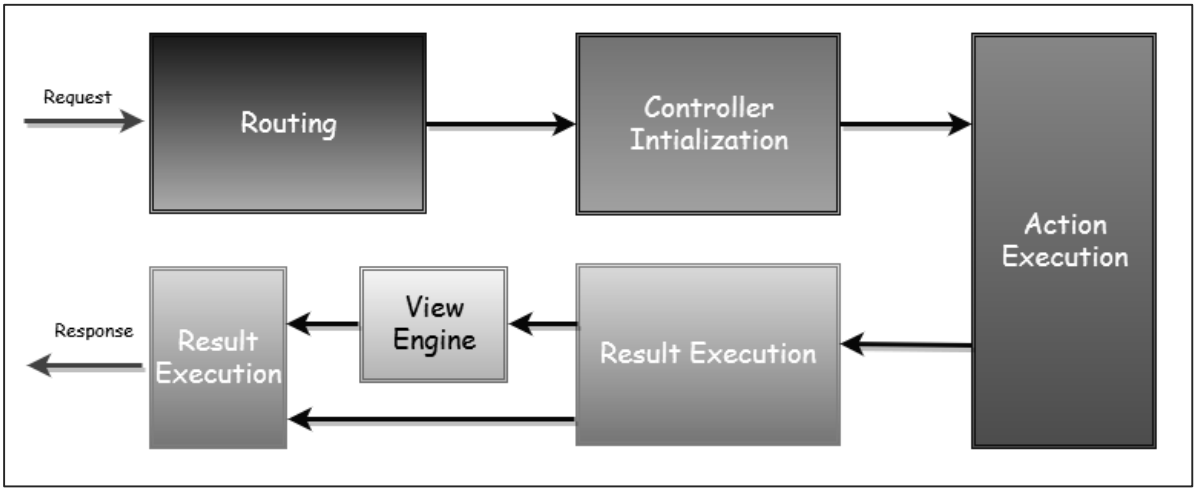
The Request Life Cycle

It is the sequence of events that happen every time an HTTP request is handled by our application.

The entry point for every MVC application begins with routing. After the ASP.NET platform has received a request, it figures out how it should be handled through the URL Routing Module.

Modules are .NET components that can hook into the application life cycle and add functionality. The routing module is responsible for matching the incoming URL to routes that we define in our application.

All routes have an associated route handler with them and this is the entry point to the MVC framework.



The MVC framework handles converting the route data into a concrete controller that can handle requests. After the controller has been created, the next major step is **Action Execution**. A component called the **action invoker** finds and selects an appropriate Action method to invoke the controller.

After our action result has been prepared, the next stage triggers, which is **Result Execution**. MVC separates declaring the result from executing the result. If the result is a view type, the View Engine will be called and it's responsible for finding and rending our view.

If the result is not a view, the action result will execute on its own. This Result Execution is what generates an actual response to the original HTTP request.

6. ASP.NET MVC – ROUTING

Routing is the process of directing an HTTP request to a controller and the functionality of this processing is implemented in System.Web.Routing. This assembly is not part of ASP.NET MVC. It is actually part of the ASP.NET runtime, and it was officially released with the ASP.NET as a .NET 3.5 SP1.

System.Web.Routing is used by the MVC framework, but it's also used by ASP.NET Dynamic Data. The MVC framework leverages routing to direct a request to a controller. The Global.asax file is that part of your application, where you will define the route for your application.

This is the code from the application start event in Global.asax from the MVC App which we created in the previous chapter.

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

namespace MVCFirstApp
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Following is the implementation of RouteConfig class, which contains one method RegisterRoutes.

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

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

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

You will define the routes and those routes will map URLs to a specific controller action. An action is just a method on the controller. It can also pick parameters out of that URL and pass them as parameters into the method.

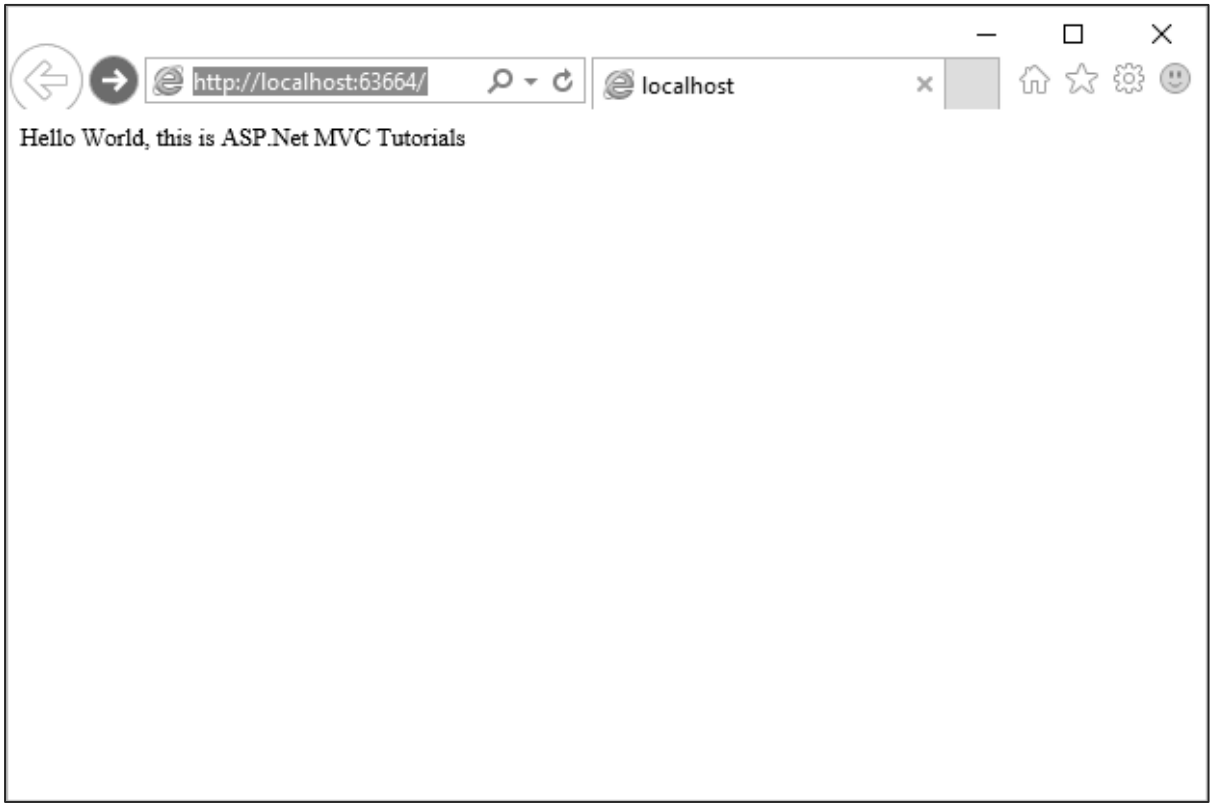
So this route that is defined in the application is the default route. As seen in the above code, when you see a URL arrive in the form of (something)/(something)/(something), then the

first piece is the controller name, second piece is the action name, and the third piece is an ID parameter.

Understanding Routes

MVC applications use the ASP.NET routing system, which decides how URLs map to controllers and actions.

When Visual Studio creates the MVC project, it adds some default routes to get us started. When you run your application, you will see that Visual Studio has directed the browser to port 63664. You will almost certainly see a different port number in the URL that your browser requests because Visual Studio allocates a random port when the project is created.



In the last example, we have added a HomeController, so you can also request any of the following URLs, and they will be directed to the Index action on the HomeController.

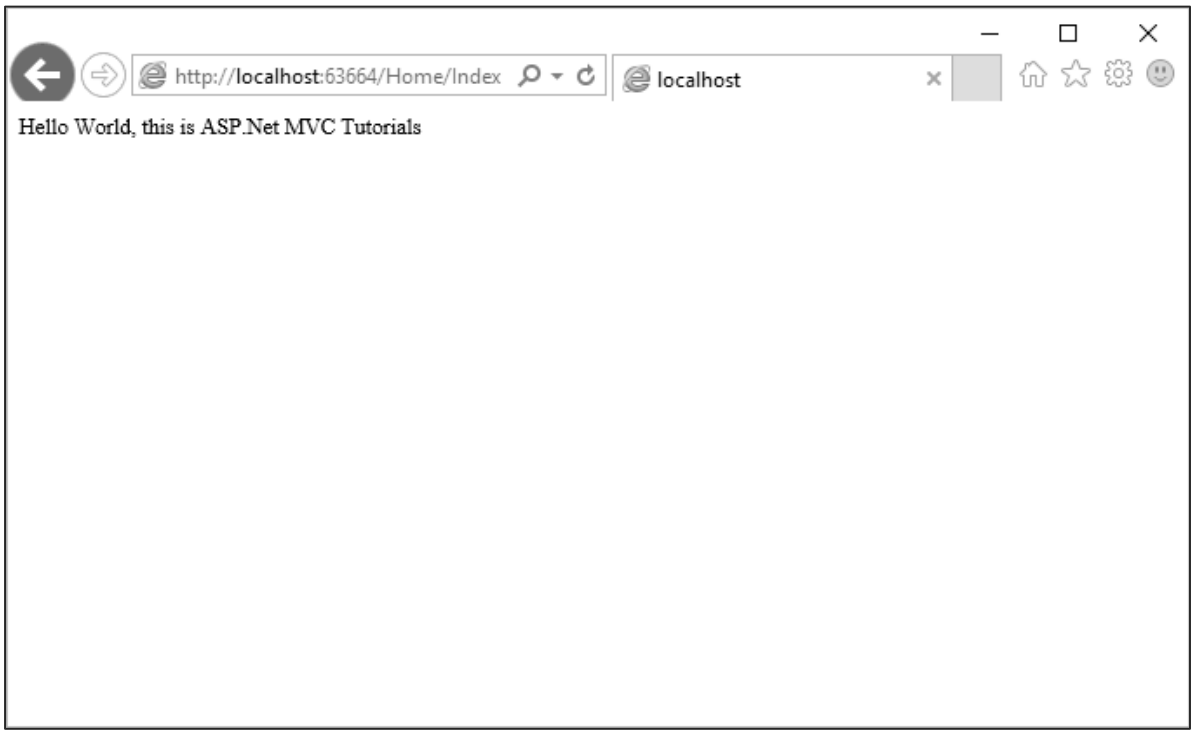
http://localhost:63664/Home/

http://localhost:63664/Home/Index

When a browser requests http://mysite/ or http://mysite/Home, it gets back the output from HomeController's Index method.

You can try this as well by changing the URL in the browser. In this example, it is http://localhost:63664/, except that the port might be different.

If you append /Home or /Home/Index to the URL and press 'Enter' button, you will see the same result from the MVC application.



As you can see in this case, the convention is that we have a controller called HomeController and this HomeController will be the starting point for our MVC application.

The default routes that Visual Studio creates for a new project assumes that you will follow this convention. But if you want to follow your own convention then you would need to modify the routes.

Custom Convention

You can certainly add your own routes. If you don't like these action names, if you have different ID parameters or if you just in general have a different URL structure for your site, then you can add your own route entries.

Let's take a look at a simple example. Consider we have a page that contains the list of processes. Following is the code, which will route to the process page.

```
routes.MapRoute(
                "Process",
                "Process/{action}/{id}",
```

```
                defaults: new { controller = "Process", action = "List ", id =
UrlParameter.Optional }
);
```

When someone comes in and looks for a URL with Process/Action/Id, they will go to the Process Controller. We can make the action a little bit different, the default action, we can make that a List instead of Index.

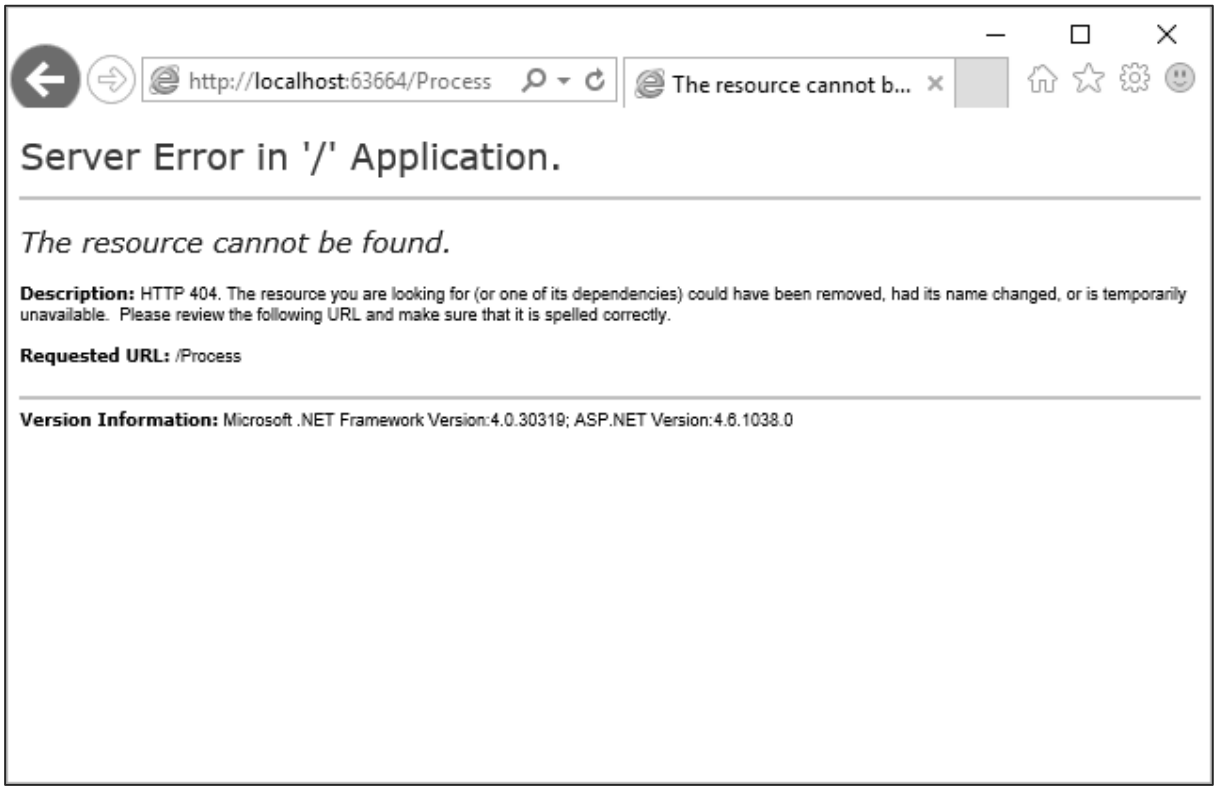
Now a request that arrives looks like localhosts/process. The routing engine will use this routing configuration to pass that along, so it's going to use a default action of List.

Following is the complete class implementation.

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

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

            routes.MapRoute(
                "Process",
                "Process/{action}/{id}",
                defaults: new { controller = " Process", action = "List ", id =
UrlParameter.Optional }
            );

            routes.MapRoute(
```

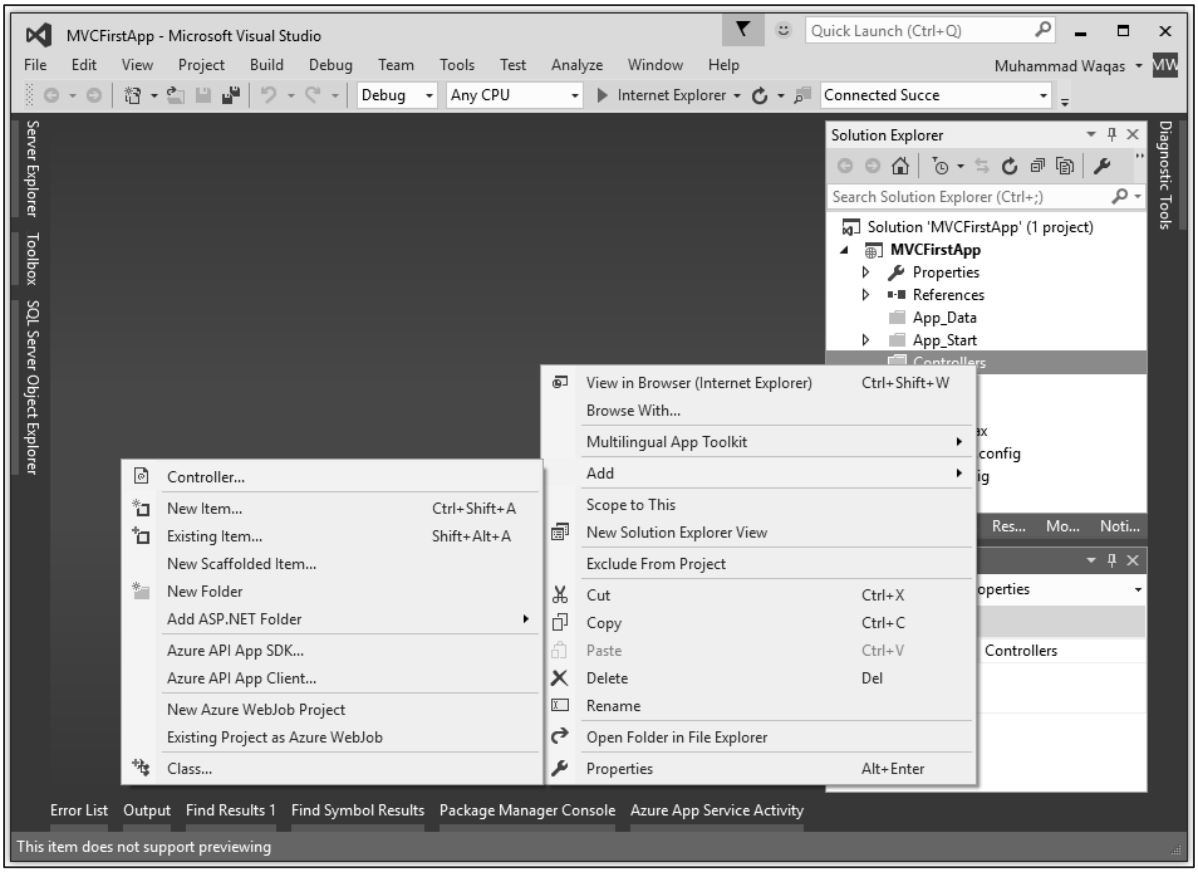
```
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
            );
        }
    }
}
```

Step (1): Run this and request for a process page with the following URL http://localhost:63664/Process

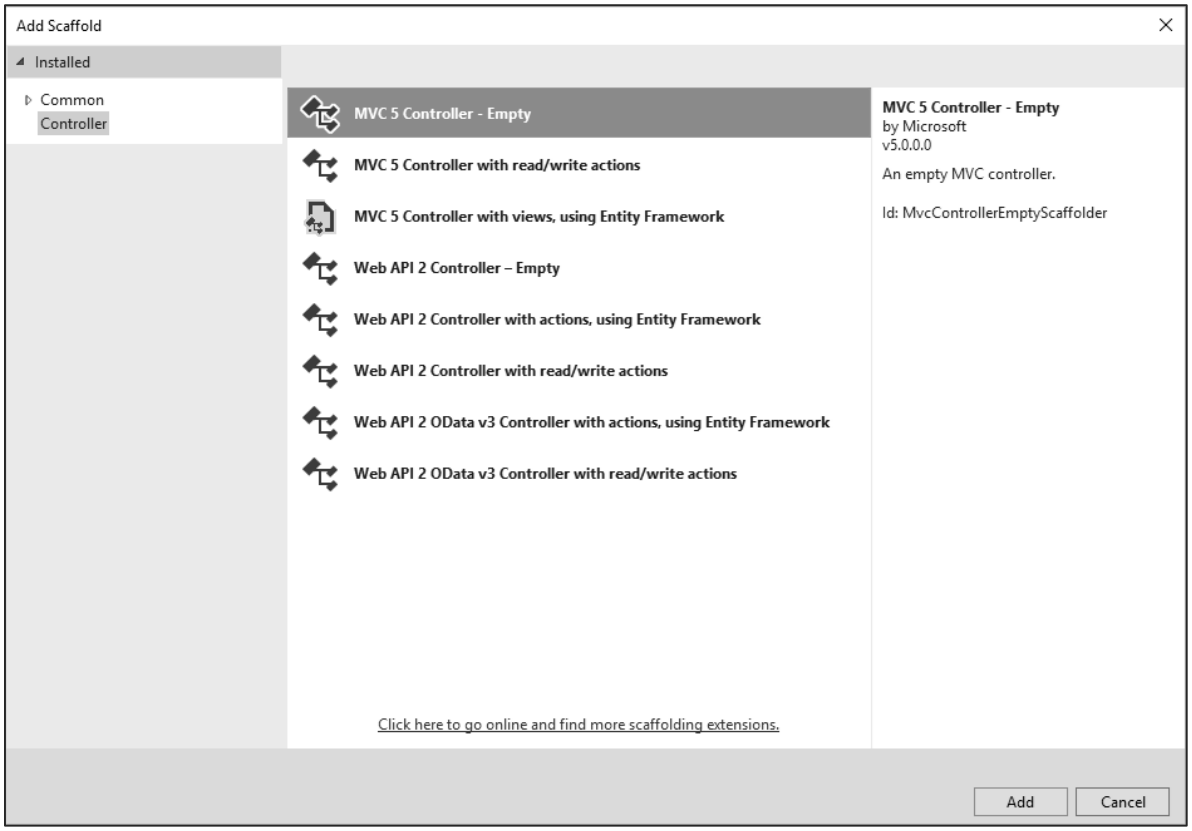


You will see an HTTP 404, because the routing engine is looking for ProcessController, which is not available.

Step (2): Create ProcessController by right-clicking on Controllers folder in the solution explorer and select Add -> Controller.

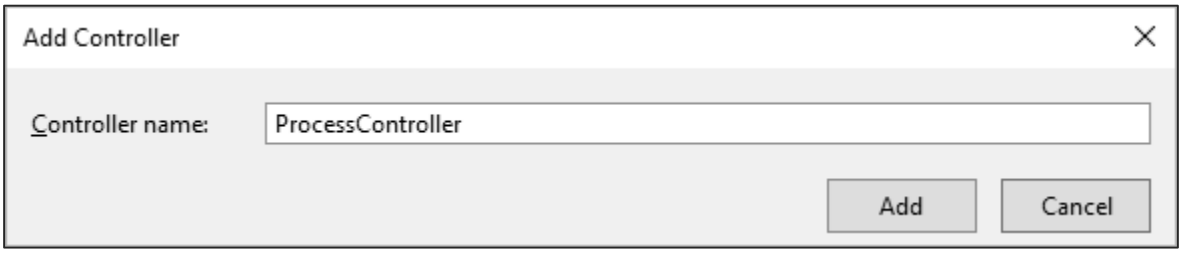


It will display the Add Scaffold dialog.



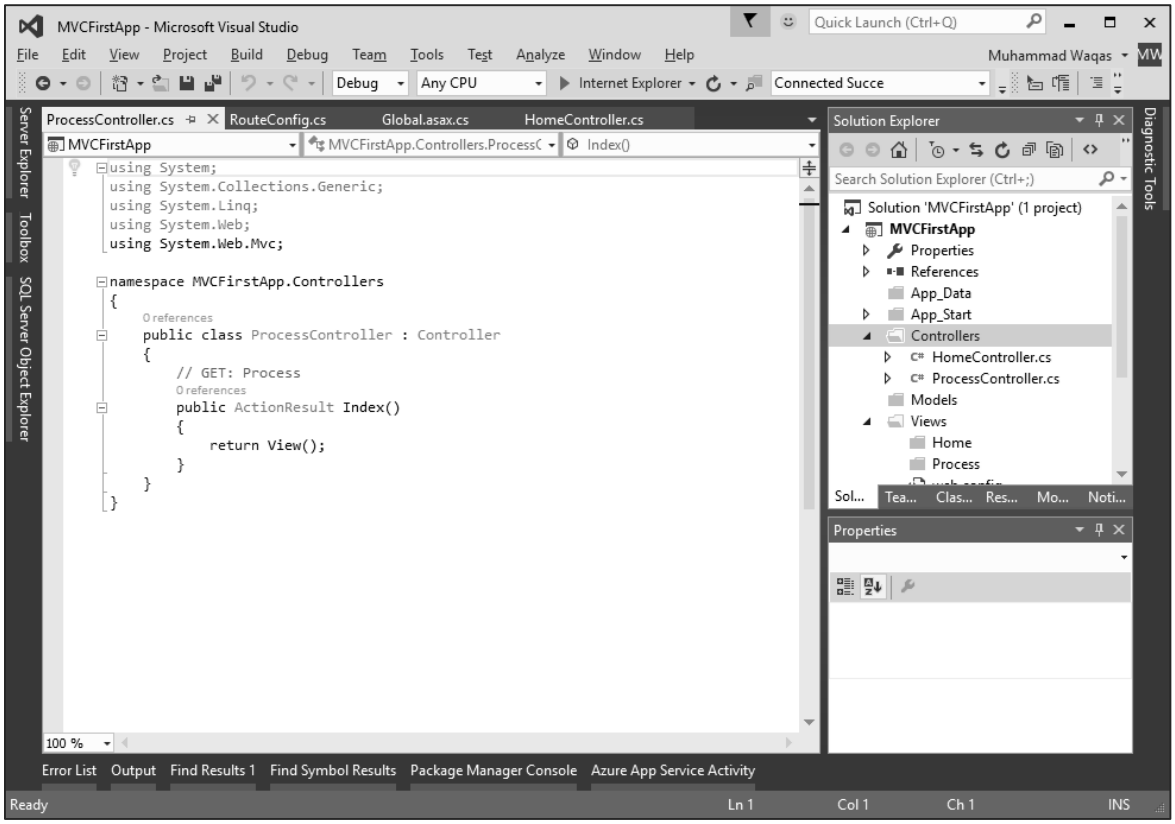
Step (3): Select the MVC 5 Controller – Empty option and click 'Add' button.

The Add Controller dialog will appear.



Step (4): Set the name to ProcessController and click 'Add' button.

Now you will see a new C# file ProcessController.cs in the Controllers folder, which is open for editing in Visual Studio as well.



Now our default action is going to be List, so we want to have a List action here instead of Index.

Step (5): Change the return type from ActionResult to string and also return some string from this action method using the following code.

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

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**