What is MVC FRAMEWORK?

ASP.NET MVC is a web application framework developed by Microsoft that implements the model–view–controller pattern.

The MVC architectural pattern has existed for a long time in software engineering. All most all the languages use MVC with slight variation, but conceptually it remains the same.

MVC stands for Model, View, and Controller. MVC separates an application into three components - Model, View, and Controller.

MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic. It emphasizes a separation between the software's business logic and display.

The MVC framework includes the following components:

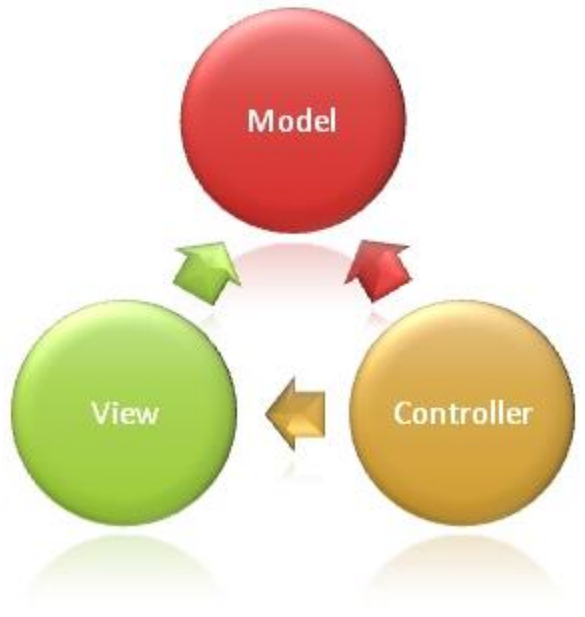


Figure 1: Invoking a controller action that expects a parameter value

- **Models**. Model objects are the parts of the application that implement the logic for the application s data domain. Often, model objects retrieve and store model state in a database. For example, a Product object might retrieve information from a database, operate on it, and then write updated information back to a Products table in SQL Server.

In small applications, the model is often a conceptual separation instead of a physical one. For example, if the application only reads a data set and sends it to the view, the application does not have a physical model layer and associated classes. In that case, the data set takes on the role of a model object.

- **Views**. Views are the components that display the application s user interface (UI). Typically, this UI is created from the model data. An example would be an edit view of a Products table that displays text boxes, drop-down lists, and check boxes based on the current state of a Products object.
- **Controllers**. Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI. In an MVC application, the view only displays information; the controller handles and responds to user input and interaction. For example, the controller handles query-string values, and passes these values to the model, which in turn queries the database by using the values.

The MVC pattern helps you create applications that separate the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements. The pattern specifies where each kind of logic should be located in the application. The UI logic belongs in the view. Input logic belongs in the controller. Business logic belongs in the model. This separation helps you manage complexity when you build an application, because it enables you to focus on one aspect of the implementation at a time. For example, you can focus on the view without depending on the business logic.

The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller. The ASP.NET MVC framework provides an alternative to the ASP.NET Web Forms pattern for creating MVC-based Web applications.

The ASP.NET MVC framework is a lightweight, highly testable presentation framework that (as with Web Forms-based applications) is integrated with existing ASP.NET features, such as master pages and membership-based authentication. The MVC framework is defined in the **System.Web.Mvc** namespace and is a fundamental, supported part of the **System.Web** namespace.

Deciding When to Create an MVC Application

You must consider carefully whether to implement a Web application by using either the ASP.NET MVC framework or the ASP.NET Web Forms model. The MVC framework does not replace the Web Forms model; you can use either framework for Web applications. (If you have existing Web Forms-based applications, these continue to work exactly as they always have.)

Before you decide to use the MVC framework or the Web Forms model for a specific Web site, weigh the advantages of each approach.

Advantages of an MVC-Based Web Application

The ASP.NET MVC framework offers the following advantages:

- It makes it easier to manage complexity by dividing an application into the model, the view, and the controller.
- It does not use view state or server-based forms. This makes the MVC framework ideal for developers who want full control over the behavior of an application.
- It uses a Front Controller pattern that processes Web application requests through a single controller. This enables you to design an application that supports a rich routing infrastructure. For more information, see Front Controller on the MSDN Web site.
- It provides better support for test-driven development (TDD).
- It works well for Web applications that are supported by large teams of developers and Web designers who need a high degree of control over the application behavior.

Advantages of a Web Forms-Based Web Application

The Web Forms-based framework offers the following advantages:

- It supports an event model that preserves state over HTTP, which benefits line-of-business Web application development. The Web Forms-based application provides dozens of events that are supported in hundreds of server controls.
- It uses a Page Controller pattern that adds functionality to individual pages. For more information, see Page Controller on the MSDN Web site.
- It uses view state or server-based forms, which can make managing state information easier.
- It works well for small teams of Web developers and designers who want to take advantage of the large number of components available for rapid application development.
- In general, it is less complex for application development, because the components (the **Page** class, controls, and so on) are tightly integrated and usually require less code than the MVC model.

Features of the ASP.NET MVC Framework

- The ASP.NET MVC framework provides the following features:
- Separation of application tasks (input logic, business logic, and UI logic), testability, and test-driven development (TDD) by default. All core contracts in the MVC framework are interface-based and can be tested by using mock objects, which are simulated objects that imitate the behavior of actual objects in the application. You can unit-test the application without having to run the controllers in an ASP.NET process, which makes unit testing fast and flexible. You can use any unit-testing framework that is compatible with the .NET Framework.
- An extensible and pluggable framework. The components of the ASP.NET MVC framework are designed so that they can be easily replaced or customized. You can plug in your own view engine, URL routing policy, action-method parameter serialization, and other components. The ASP.NET MVC framework also supports the use of Dependency Injection (DI) and Inversion of Control (IOC) container models. DI allows you to inject objects into a class, instead of relying on the class to create the object itself. IOC specifies that if an object requires another object, the first objects should get the second object from an outside source such as a configuration file. This makes testing easier.
- A powerful URL-mapping component that lets you build applications that have comprehensible and searchable URLs. URLs do not have to include file-name extensions, and are designed to support URL naming patterns that work well for search engine optimization (SEO) and representational state transfer (REST) addressing.
- Support for using the markup in existing ASP.NET page (.aspx files), user control (.ascx files), and master page (.master files) markup files as view templates. You can use existing ASP.NET features with the ASP.NET MVC framework, such as nested master pages, in-line expressions (<%= %>), declarative server controls, templates, data-binding, localization, and so on.
- Support for existing ASP.NET features. ASP.NET MVC lets you use features such as forms authentication and Windows authentication, URL authorization, membership and roles, output and data caching, session and profile state management, health monitoring, the configuration system, and the provider architecture.

https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

Routing in MVC

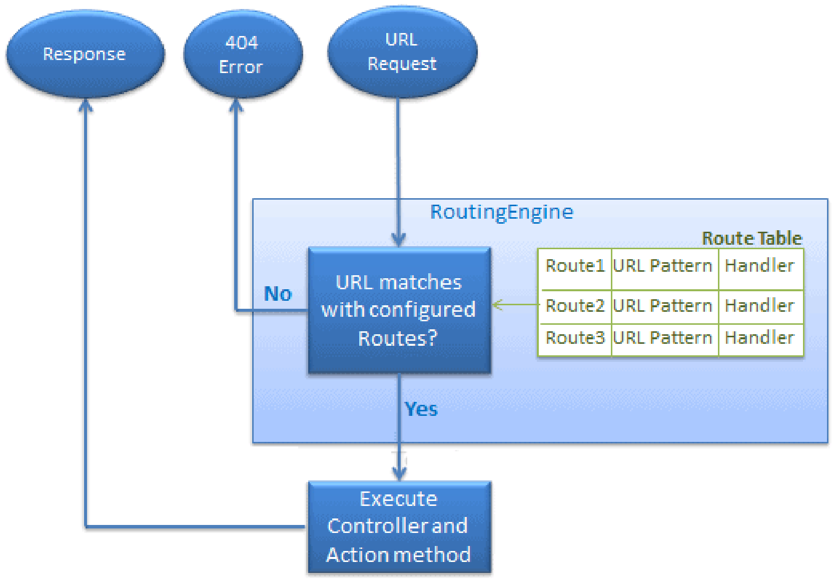
In the ASP.NET Web Forms application, every URL must match with a specific .aspx file. For example, a URL http://domain/studentsinfo.aspx must match with the file studentsinfo.aspx that contains code and markup for rendering a response to the browser.

ASP.NET introduced Routing to eliminate the needs of mapping each URL with a physical file. Routing enables us to define a URL pattern that maps to the request handler. This request handler can be a file or class. In ASP.NET Webform application, request handler is .aspx file, and in MVC, it is the Controller class and Action method. For example, http://domain/students can be mapped to http://domain/studentsinfo.aspx in ASP.NET Webforms, and the same URL can be mapped to Student Controller and Index action method in MVC.

Route

Route defines the URL pattern and handler information. All the configured routes of an application stored in RouteTable and will be used by the Routing engine to determine appropriate handler class or file for an incoming request.

The following figure illustrates the Routing process.


Response
404 Error
URL Request
RoutingEngine
Route Table
Route1 URL Pattern Handler
Route2 URL Pattern Handler
Route3 URL Pattern Handler
No
URL matches with configured Routes?
Yes
Execute Controller and Action method

Multiple Routes

You can also configure a custom route using the MapRoute extension method. You need to provide at least two parameters in MapRoute, route name, and URL pattern. The Defaults parameter is optional.

You can register multiple custom routes with different names. Consider the following example where we register "Student" route.

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

        routes.MapRoute(
            name: "Student",
            url: "students/{id}",
            defaults: new { controller = "Student", action = "Index"}
        );

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

As shown in the above code, the URL pattern for the Student route is students/{id}, which specifies that any URL that starts with domainName/students, must be handled by the StudentController. Notice that we haven't specified {action} in the URL pattern because we want every URL that starts with students should always use the Index() action of the StudentController class. We have specified the default controller and action to handle any URL request, which starts from domainname/students.

MVC framework evaluates each route in sequence. It starts with the first configured route, and if incoming URL doesn't satisfy the URL pattern of the route, then it will evaluate the second route and so on. In the above example, routing engine will evaluate the Student route first and if incoming URL doesn't start with /students then only it will consider the second route which is the default route.

The following table shows how different URLs will be mapped to the Student route:

URL	Controller	Action	Id
http://localhost/student/123	StudentController	Index	123
http://localhost/student/index/123	StudentController	Index	123
http://localhost/student?Id=123	StudentController	Index	123

Route Constraints

You can also apply restrictions on the value of the parameter by configuring route constraints. For example, the following route applies a limitation on the id parameter that the id's value must be numeric.

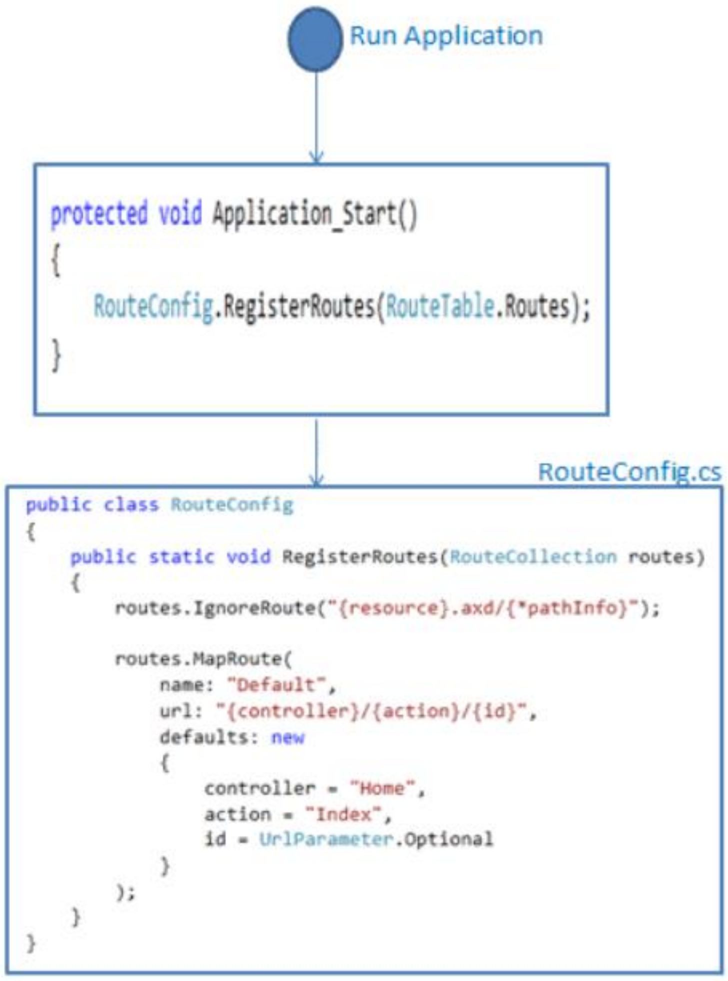
```
routes.MapRoute(
        name: "Student",
        url: "student/{id}/{name}/{standardId}",
        defaults: new { controller = "Student", action = "Index", id = 
UrlParameter.Optional},
        constraints: new { id = @"\d+" }
    );
```

Register Routes

So if you give non-numeric value for id parameter, then that request will be handled by another route or, if there are no matching routes, then "The resource could not be found" error will be thrown.

Now, after configuring all the routes in the RouteConfig class, you need to register it in the Application_Start() event in the Global.asax so that it includes all your routes into the RouteTable.

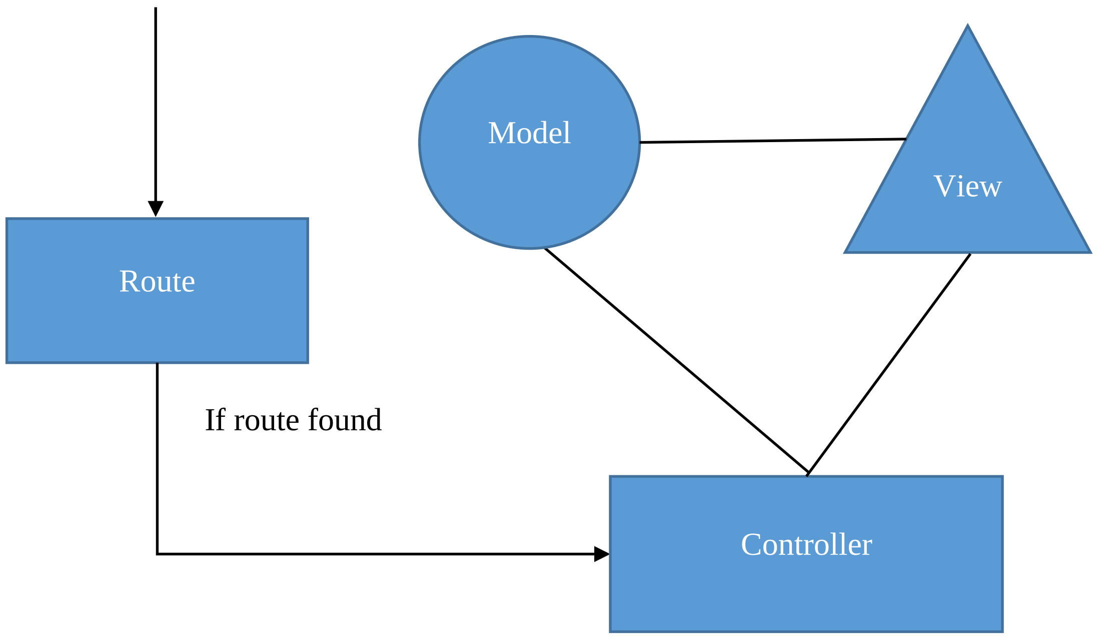
```
public class MvcApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
        RouteConfig.RegisterRoutes(RouteTable.Routes);
    }
}
```



```
protected void Application_Start()
{
    RouteConfig.RegisterRoutes(RouteTable.Routes);
}
```

```
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
            }
        );
    }
}
```

Asp.Net MVC Route Pattern



Controllers in ASP.NET MVC

In this section, you will learn about the Controller in ASP.NET MVC.

The Controller in MVC architecture handles any incoming URL request. The Controller is a class, derived from the base class System.Web.Mvc.Controller. Controller class contains public methods called **Action** methods. Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.

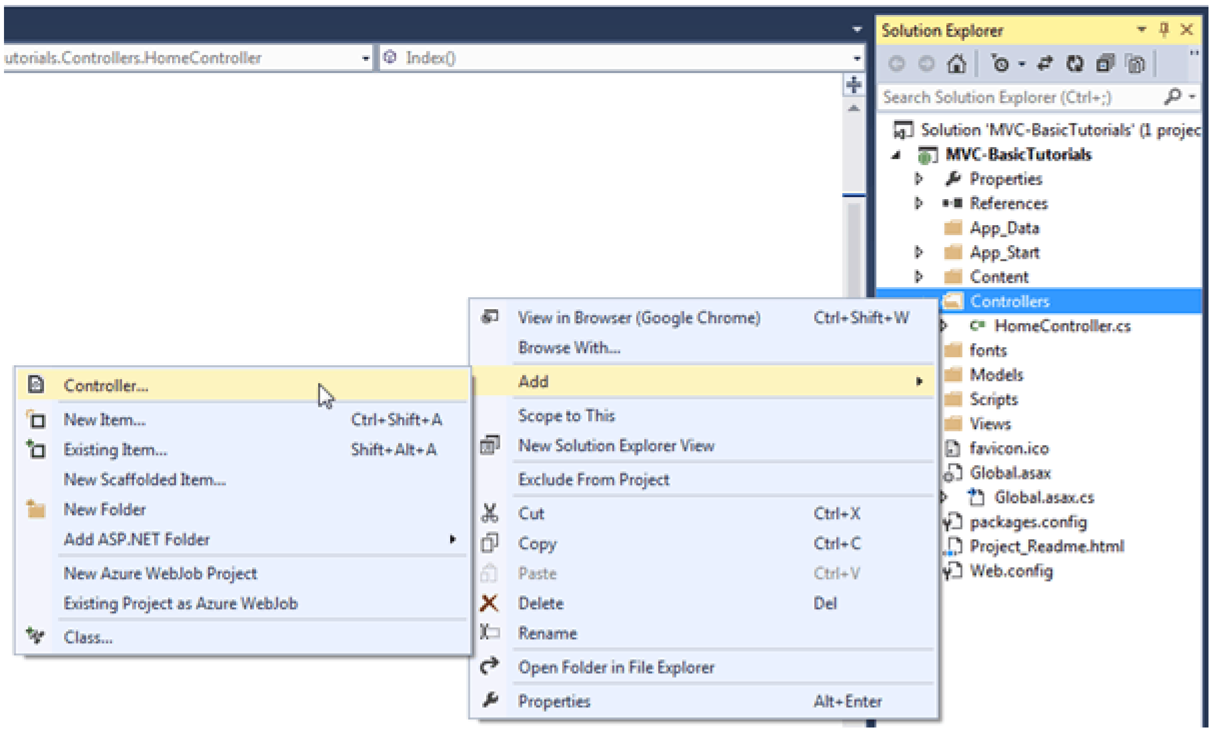
In ASP.NET MVC, every controller class name must end with a word "Controller". For example, the home page controller name must be HomeController, and for the student page, it must be the StudentController. Also, every controller class must be located in the Controller folder of the MVC folder structure.

Adding a New Controller

Now, let's add a new empty controller in our MVC application in Visual Studio.

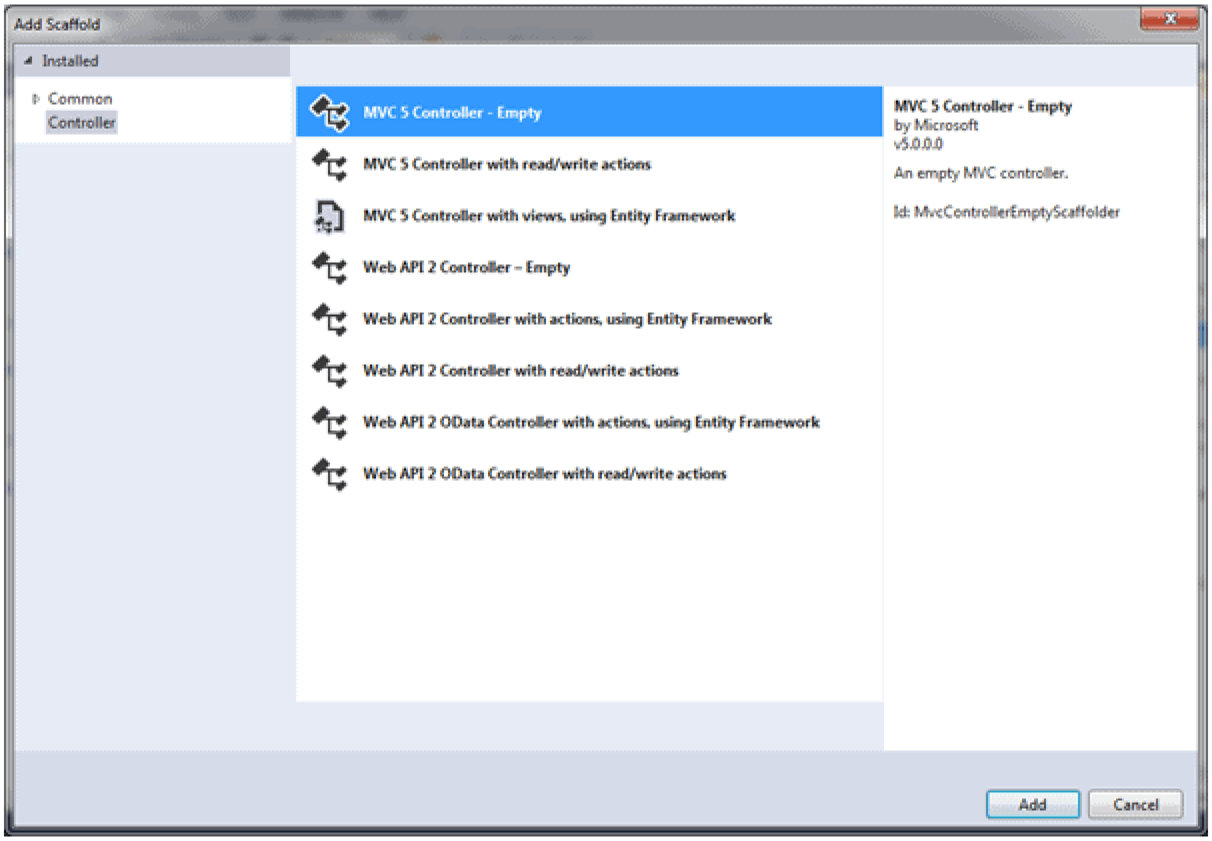
In the previous section, we learned how to create our first MVC application, which created a default HomeController. Here, we will create new StudentController class.

In the Visual Studio, right click on the Controller folder -> select **Add** -> click on **Controller..**



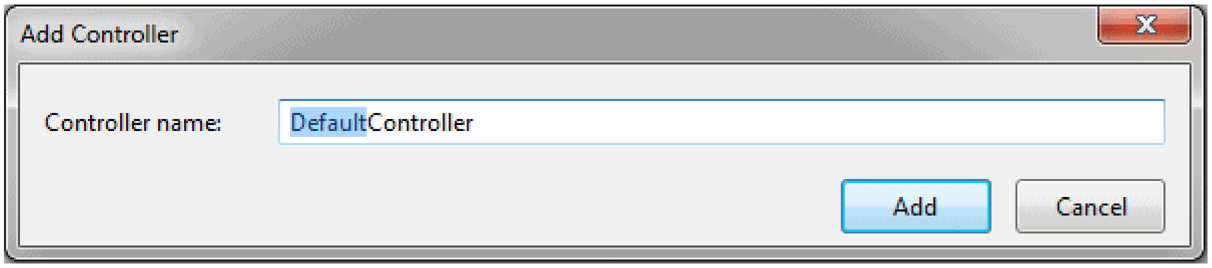
This opens Add Scaffold dialog, as shown below.

(Scaffolding is an automatic code generation framework for ASP.NET web applications. Scaffolding reduces the time taken to develop a controller, view, etc. in the MVC framework. You can develop a customized scaffolding template using T4 templates as per your architecture and coding standards.)

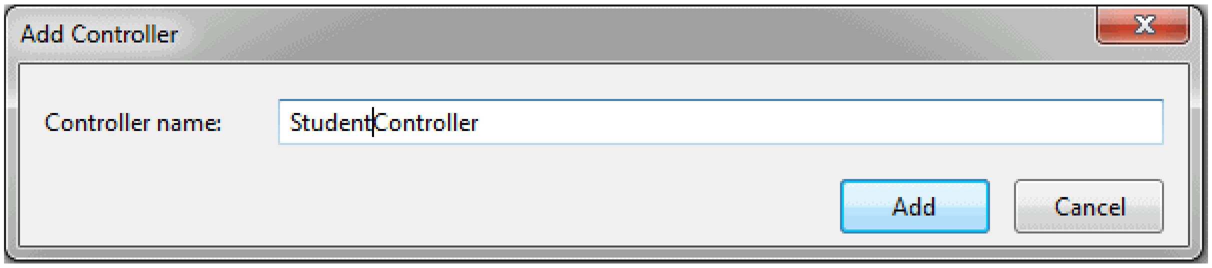


Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData Controller with actions, using Entity Framework
Web API 2 OData Controller with read/write actions
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Add
Cancel

Add Scaffold dialog contains different templates to create a new controller. We will learn about other templates later. For now, select "MVC 5 Controller - Empty" and click Add. It will open the Add Controller dialog, as shown below



In the Add Controller dialog, enter the name of the controller. Remember, the controller name must end with Controller. Write StudentController and click **Add.**



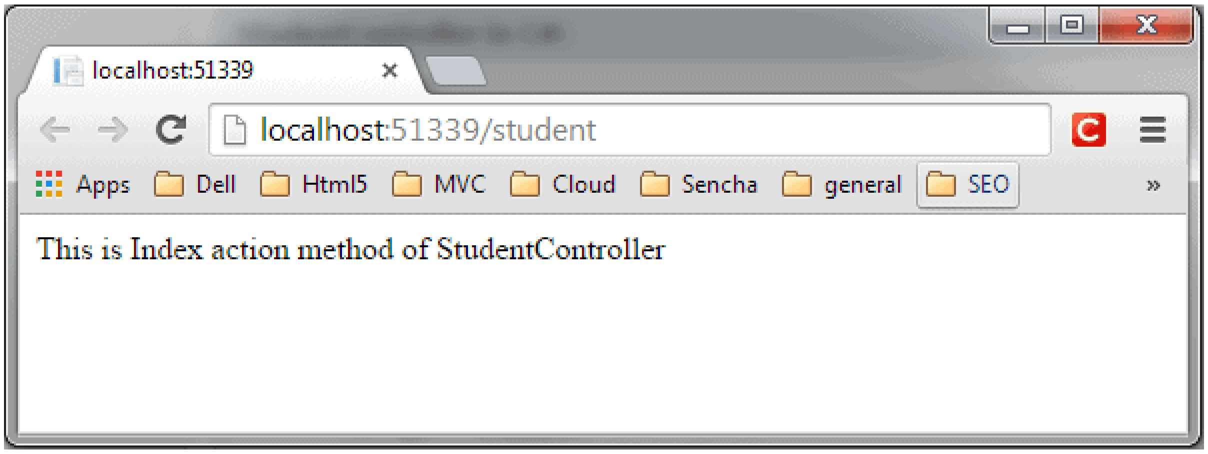
This will create the StudentController class with the Index() method in StudentController.cs file under the Controllers folder, as shown below.

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

namespace MVC_BasicTutorials.Controllers
{
    public class StudentController : Controller
    {
        // GET: Student
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

As you can see above, the StudentController class is derived from the Controller class. Every controller in MVC must be derived from this abstract Controller class. This base Controller class contains helper methods that can be used for various purposes. Now, we will return a dummy string from the Index action method of above the StudentController. Changing the return type of Index method from ActionResult to string and returning dummy string is shown below. You will learn about the ActionResult in the next section.

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

namespace MVC_BasicTutorials.Controllers
{
    public class StudentController : Controller
    {
        // GET: Student
        public string Index()
        {
                return    "This    is    Index    action    method    of
StudentController";
        }
    }
}
```

We have already seen in the routing section that the URL request **http://localhost/student** or **http://localhost/student/**index is handled by the Index() method of the StudentController class, as shown above. So let's invoke it from the browser and you will see the following page in the browser.



```
This is Index action method of StudentController
```