Session: 2

Controllers in ASP.NET MVC

Objectives

- Define and describe controllers
- Describe how to work with action methods
- Explain how to invoke action methods
- Explain routing requests
- Describe URL patterns

Working with Controllers 1-2

- A controller, in an ASP.NET MVC application does the following:
  - Manages the flow of the application.
  - Is responsible for intercepting incoming requests and executing the appropriate application code.
  - Communicates with the models of the application and selects the required view to be rendered for the request.
  - Is a C# class that extends the `Controller` class of the `System.Web.Mvc` namespace.
  - Allows separating the business logic of the application from the presentation logic.

Working with Controllers 2-2

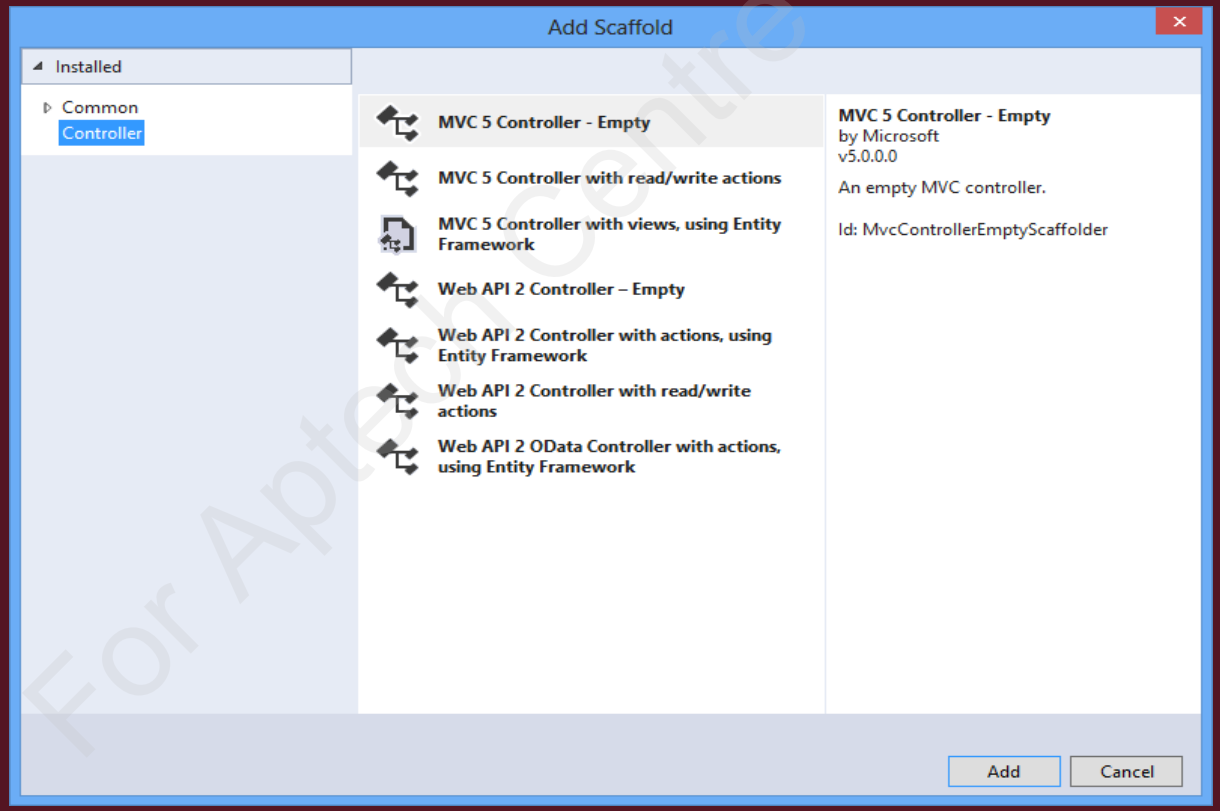
- In an ASP.NET MVC application, a controller is responsible to:
  - Locate the appropriate method to call for an incoming request.
  - Validate the data of the incoming request before invoking the requested method.
  - Retrieve the request data and passing it to requested method as arguments.
  - Handle any exceptions that the requested method throws.
  - Help in rendering the view based on the result of the requested method.

Creating a Controller 1-6

- In ASP.NET MVC, the `ControllerBase` class of the `System.Web.Mvc` namespace is the base class for all controllers.
- The `Controller` class extends the `ControllerBase` class to provide a default implementation of a controller.
- To create a controller in an ASP.NET MVC application, you will need to create a C# class that extends the `Controller` class.
- Instead of creating a controller manually, you can use Visual Studio 2013 IDE, which also creates the folder structure for the application automatically.

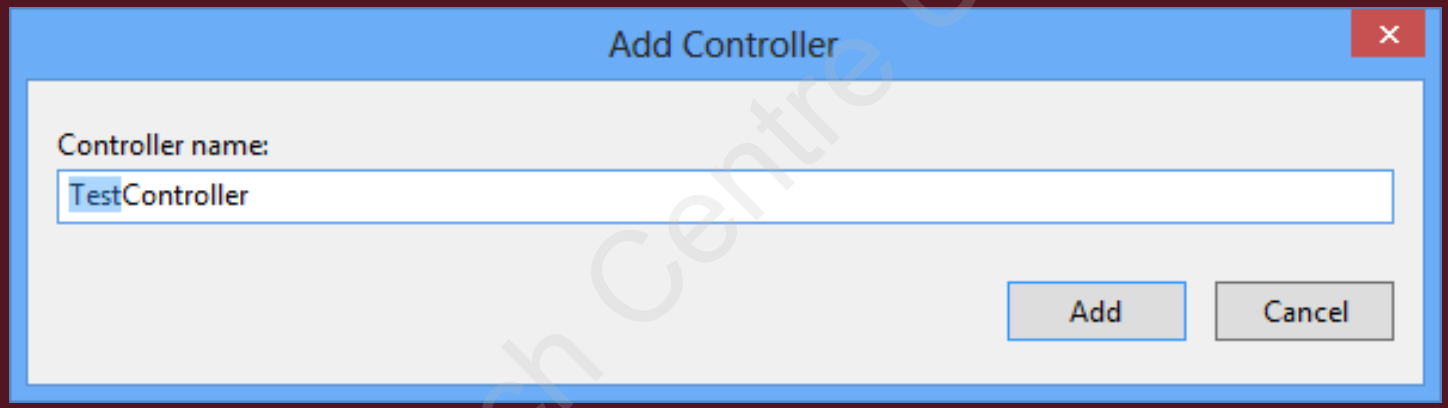
Creating a Controller 2-6

- In Visual Studio 2013 IDE, you can create a controller by performing the following steps:
  1. Right-click the **Controllers** folder in the **Solution Explorer** window.
  2. Select **Add→Controller** from the context menu that appears. The **Add Scaffold** dialog box is displayed, as shown in the following figure:



Creating a Controller 3-6

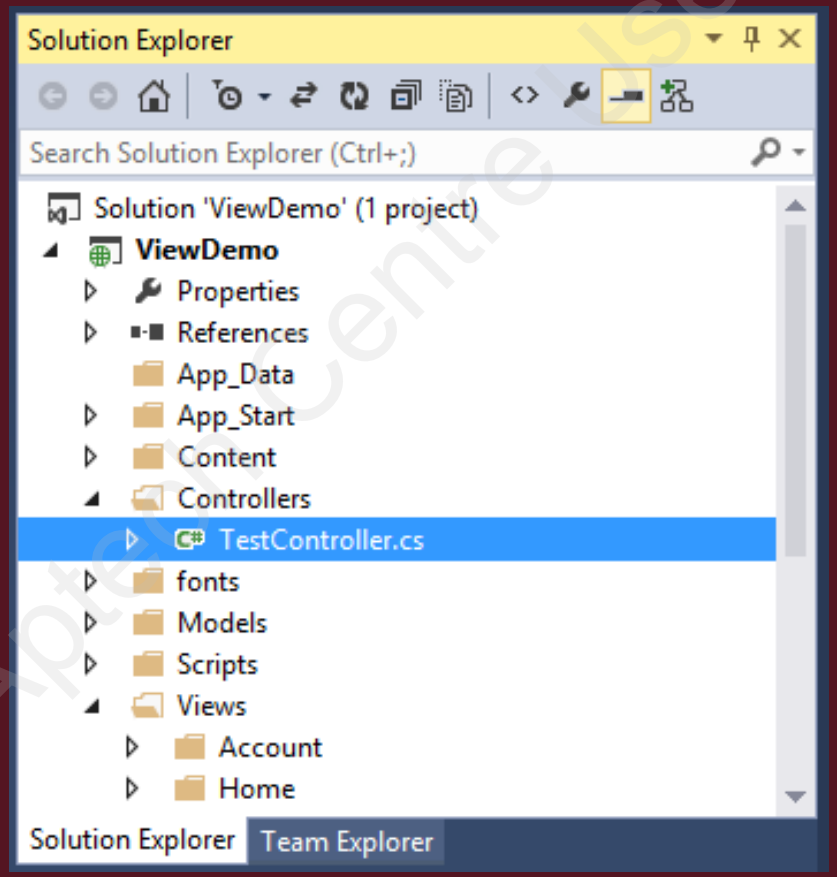
3. Select the **MVC 5 Controller - Empty** in the **Add Scaffold** dialog box.
4. Click **Add**. The **Add Controller** dialog box appears.
5. Type **TestController** in the **Controller name** text field, as shown in the following figure:



6. Click **Add**. The **Solution Explorer** window displays the newly created `TestController` controller under the **Controllers** folder.

Creating a Controller 4-6

- Following figure shows the **Solution Explorer** window that displays the newly created controller under the Controllers folder:



Creating a Controller 5-6

- You can use the following syntax for creating a `Controller` class:

Code Snippet:

```
using System.Web.Mvc;
public class <Controller_Name>Controller:Controller
{
//Some code
}
```

where,

- `Controller_Name`: is a name of the controller.

Creating a Controller 6-6

- Following is the skeleton code of a `Controller` class:

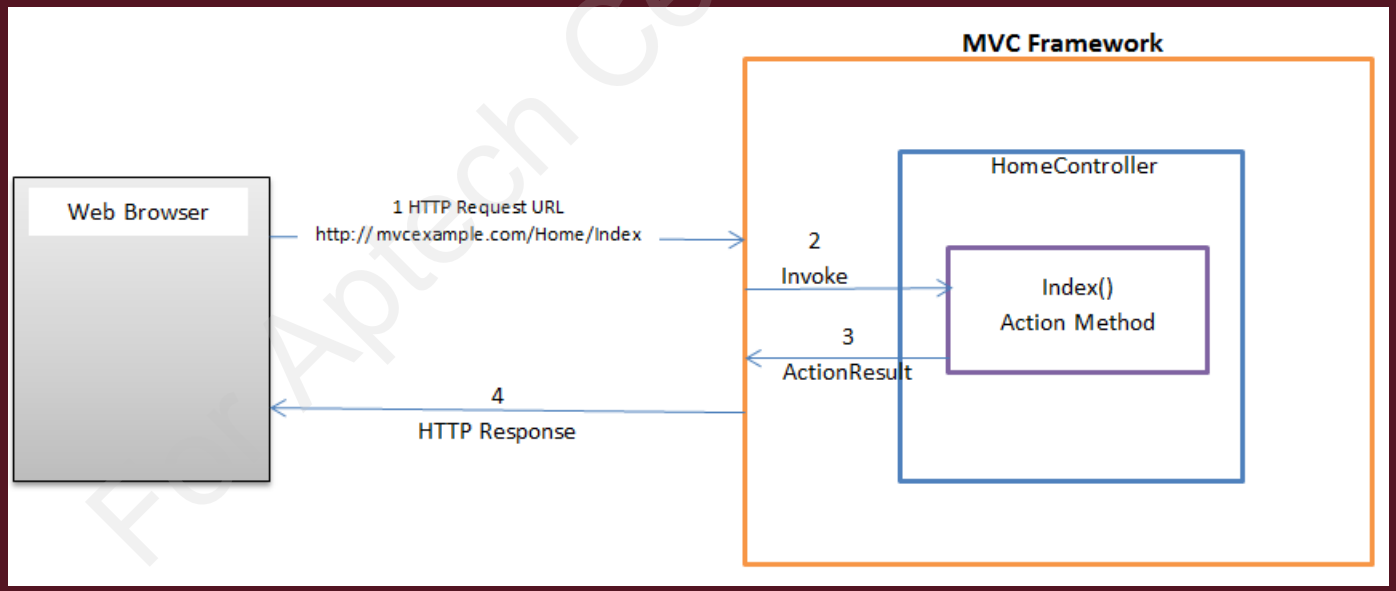
Code Snippet :

```
using System.Web.Mvc
public class TestController : Controller
{
  //Some code
}
```

  - In this code, a controller is created with the name `TestController`.

Working with Action Methods 1-5

- A controller class can contains one or more action methods, also known as controller actions.
- Action methods:
  - Are responsible for processing the requests that are sent to the controller.
  - Typically returns an `ActionResult` object that encapsulates the result of executing the method.
- Following figure shows the working of action methods:



Working with Action Methods 2-5

- The steps in the preceding figure are as follows:
  1. The browser sends an HTTP request.
  2. The MVC Framework invokes the controller action method based on the request URL.
  3. The action method executes and returns an `ActionResult` object. This object encapsulates the result of the action method execution.
  4. The MVC Framework converts an `ActionResult` to HTTP response and sends the response back to the browser.

Working with Action Methods 3-5

- Rules that you need to consider while creating an action method are as follows:
  - They must be declared as public.
  - They cannot be declared as static.
  - They cannot have overloaded versions based on parameters.
- Following is the syntax for creating an action method in a `Controller` class:

Code Snippet:

```
public ActionResult <ActionMethod_Name>()
 {
  /*Code to execute logic and return the result as ActionResult*/
 }
```

where,

- `<ActionMethod_Name>`: is a name of the action method.

Working with Action Methods 4-5

- Following code creates two action methods with the name `Index` and `About` in the `HomeController` controller class:

Code Snippet:

```
using System.Web.Mvc;
public class HomeController : Controller
{
 public ActionResult Index()
   {
    /*Code to execute logic and return the result as ActionResult*/
   }
public ActionResult About()
   {
   /*Code to execute logic and return the result as ActionResult*/
   }
}
```

- The code creates two action methods, named `Index` and `About` in the `HomeController` controller class. Both these action methods are declared as `public` and to return `ActionResult` objects.

Working with Action Methods 5-5

- Although, most of the action methods return an `ActionResult` object, an action method can also return other types, such as `String`, `int`, or `bool`, as shown in the following code:

Code Snippet:

```
using System.Web.Mvc;
public class HomeController : Controller
{
 public ActionResult Index()
   {
    /*Code to execute logic and return the result as ActionResult*/
   }
public ActionResult About()
   {
   /*Code to execute logic and return the result as ActionResult*/
   }
}
```

- This code creates two action methods, named `IsValid` that returns a `bool` value and `Contact` that returns a `String` value.

Action Results 1-2

- ActionResult:
  - Is an abstract base class for all implementing classes that provides different types of results.
  - Consists of HTML in combination with server-side and client-side scripts to respond to user actions.
- Following table shows the commonly used classes that extend the `ActionResult` class to provide different implementations of the results of an action method:

Classes	Description
`ViewResult`	Renders a view as an HTML document.
`PartialViewResult`	Renders a partial view, which is a sub-view of the main view.
`EmptyResult`	Returns an empty response.
`RedirectResult`	Redirects a response to another action method.
`JsonResult`	Returns the result as JSON, also known as JavaScript Object Notation (JSON). JSON is an open standard format to store and exchange text information.

Action Results 2-2

Classes	Description
`JavaScriptResult`	Returns JavaScript that executes on the client browser.
`ContentResult`	Returns the content based on a defined content type, such as XML.
`FileContentResult`	Returns the content of a binary file.
`FileStreamResult`	Returns the content of a file using a `Stream` object.
`FilePathResult`	Returns a file as a response.

Invoking Action Methods 1-3

- In an ASP.NET MVC application, you can create multiple action methods in a controller.
- You can invoke an action method by specifying a URL in the Web browser containing the name of the controller and the action method to invoke.
- Following code shows the general syntax to invoke an action method:

Code Snippet:

```
http:// <domain_name> /<controller_name>/<actionmethod_name>
```

where,

- `<domain_name>`: Is the domain name of the application.
- `<controller_name>`: Is the name of the controller without the `Controller` suffix.
- `<actionmethod_name>`: Is the name of the action method to invoke.

Invoking Action Methods 2-3

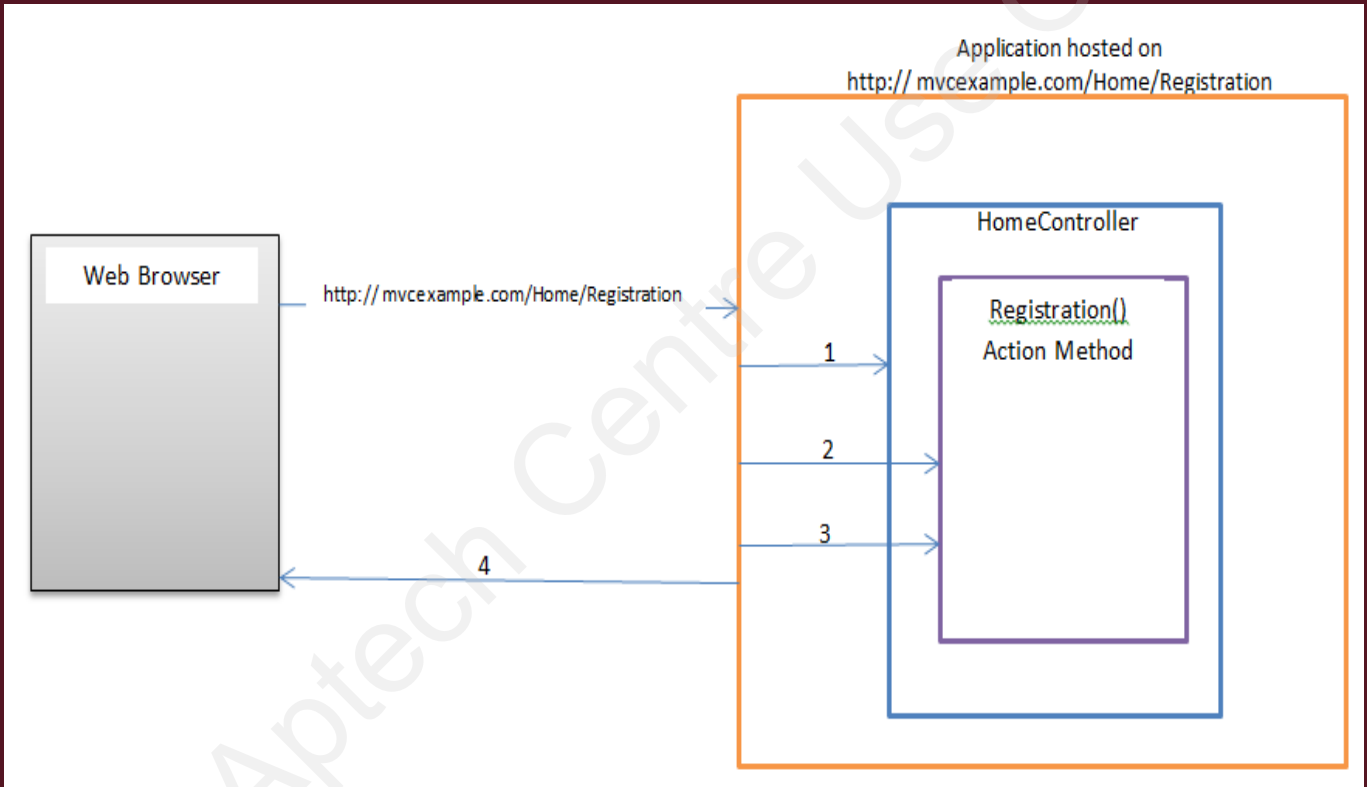
- Consider the following URL:

```
http:// mvcexample.com/Home/Registration
```

- When this URL is sent to the application through a Web browser, the MVC Framework performs the following tasks:
  - Searches for the `HomeController` controller class.
  - Searches for the `Registration()` action method in the `HomeController` controller class.
  - Executes the `Registration()` action method.
  - Returns the response back to the browser.

Invoking Action Methods 3-3

- Following figure shows the preceding steps:



Passing Parameters 1-2

- Sometimes you may need to provide input other than the Web page name while requesting for a Web page.
- Consider the following URL:
  - `http://www.mvcexample.com/student/details?Id=006`
  - The preceding URL will invoke the Details action method of the `StudentController` controller class.
  - The URL also contains an `Id` parameter with the value `006`.
- The `Details` action method must accept an `Id` parameter of type `string` in order to return student records based on the `Id` value.

Passing Parameters 2-2

- Following code shows the `Details` action that accepts an `Id` parameter:

Code Snippet:

```
public ActionResult Details(string Id)
{
  /*Return student records based on the Id parameter as an
   ActionResultobject*/

}
```

Routing Requests

- MVC Framework introduces routing that allows you to define URL patterns with placeholders that maps to request URLs pattern.
- In an ASP.NET MVC application, routing:
  - Defines how the application will process and respond to incoming HTTP request.
  - Properly describes the controller action to which the requested needs to be routed.

Uses of Routing

- Routing is a process that maps incoming requests to specified controller actions.
- Two main functions of routing are as follows:
  - Mapping incoming requests to controller action.
  - Constructing outgoing URLs corresponding to controller actions.
- Routing is achieved by configuring route patterns in the application, that includes:
  - Creating the route patterns
  - Registering the patterns with the route table of the MVC Framework
- At the time of creating an ASP.NET MVC application the application can register multiple routing patterns with the MVC Framework's route table.
- Route tables provides the information on how the routing engine process requests that matches those patterns.

The Default Route 1-2

- An MVC application requires a route to handle user requests.
- When you create an ASP.NET MVC application in Visual Studio 2013, a route is automatically configured in the `RouteConfig.cs` file.
- Following code shows the `MapRoute()` method:

Code Snippet:

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

The Default Route 2-2

- In the code:
  - The routes is of type `System.Web.Routing.RouteCollection` represents a collection of routes for the application.
  - The `MapRoute()` method defines a route named Default, a URL pattern, and a default route.
  - The default route is used if the request URL does not match with the defined URL pattern defined in the `MapRoute()` method. For example, if a request URL does not contain the name of a controller and an action, the request will be routed to the `Index` action of the `Home` controller.

Registering Default Route 1-3

- In an ASP.NET MVC application, the Global.asax file:
  - Initializes the application with the features of the MVC Framework when the application starts.
  - Contains a `MVCApplication` class with the `Application_Start()` method where you need to register the default route so that the Framework uses the route when a request starts coming to the application.

Registering Default Route 2-3

- Following code shows the `MVCApplication` class of the `Global.asax` file:

Code Snippet:

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

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

/*Code for registering other MVC components*/
}
}
}
```

Registering Default Route 3-3

- In this code:
  - The `MVCApplication` class extends the `System.Web.HttpApplication` class that defines the common features required by the application objects in an ASP.NET application.
  - The `Application_Start()` method calls the `RouteConfig.RegisterRoutes()` method to register the default route to be used in the application.

URL Patterns 1-2

- URL pattern:
  - Is required to be defined when you create a route.
  - Is compared with the URL of a request by the route engine of the MVC Framework.
  - Contains literal values and placeholders separated by the slash (/) character. Following is an example of the URL Pattern: `"{controller}/{action}/{id}"`
- URL that will match the preceding pattern:

```
http://www.mvcexample.com/student/records/36
```

- When the routing engine matches the preceding URL with the URL pattern it performs the following actions:
  - Assign `student` as the value of the `{controller}` placeholder
  - Assigns records as the value of the `{action}` placeholder
  - Assign `36` as the value of the `{id}` placeholder

URL Patterns 2-2

- A URL parameter can also have a combination of literal values and placeholders.

```
"student/{action}/{id}"
```

- Some of the URLs that will match with the preceding URL pattern are:
  - `http://www.mvcexample.com/student/records/36`
  - `http://www.mvcexample.com/student/delete/21`
  - `http://www.mvcexample.com/student/view/23`

Ordering Routes 1-2

- Sometimes you may need to register multiple routes in an ASP.NET MVC application.
- For that you can configure the sequence in which the routes will execute.
- A route engine starts matching a request URL with a URL pattern starting from the first registered route.
- When a matching route is encountered the route engine stops the matching process.

Ordering Routes 2-2

- Following code snippet demonstrates two routes:

Code Snippet:

```
routes.MapRoute(
 name: "general",
  url: "{controller}/{action}",
  defaults: new { controller = "Home", action = "Index"});
    routes.MapRoute(
    name: "manager",
    url: "Manager/{action}",
    defaults: new { controller = "Manager", action = "Browse"}
   );
```

- This code:
  - Contains two placeholders and sets the default value of the controller parameter to `Home` and the action parameter to Index.
  - Second route contains a literal, `Manager`, and a placeholder, and sets the default value of the controller parameter to `Manager` and the action parameter to Browse.

Constraining Route 1-4

- In an ASP.NET MVC application, the routing engine enables you to apply constraints around the placeholder values.
- You can use constraints when an application have identical route URLs but based on the application requirement the route engine should resolve the URLs to different controllers or actions.
- Following code shows a route with a route constraint:

Code Snippet:

```
routes.MapRoute(
"Product",
"{controller}/{action}/{id}",
  new { controller = "Product", action = "Browse", id =
UrlParameter.Optional },
  new { id = "(|Jewellery|Jeans|Mobile)" }
  );
```

- In this code, a constraint is applied to the route, so that id placeholder can have only one of the `Jewellery`, `Jeans`, and `Mobile` values.

Constraining Route 2-4

- Following table describes whether the routing engine will match different URLs based on the routing constraints:

URL	Matching Results
`http://www.mvcexample.com`	Yes. The default values of the controller and action are specified as Product and Browse respectively.
`http://www.mvcexample.com/Product`	Yes. The default values of the action is specified as Browse.
`http://www.mvcexample.com/Product/Browse`	Yes. The id specified as an optional parameter.
`http://www.mvcexample.com/Product/Browse/Jewellery`	Yes. Jewellery specified in the URL is present in the list containing valid values for id parameter.
`http://www.mvcexample.com/Product/Browse/Jeans`	Yes. Jeans specified in the URL is present in the list containing valid values for id parameter.

Constraining Route 3-4

URL	Matching Results
`http://www.mvcexample.com/Product/Browse/Mobile`	Yes. Mobile specified in the URL is present in the list containing valid values for id parameter.
`http://www.mvcexample.com/Product/Browse/Laptop`	No. Laptop specified in the URL is not present in the list containing valid values for id parameter.
`http://www.mvcexample.com/Product/Browse/Glasses`	No. Glasses specified in the URL is not present in the list containing valid values for id parameter.

Constraining Route 4-4

- Following code shows a route with a route constraint which specifies that the `id` placeholder can match only with an `integer` value:

Code Snippet:

```
routes.MapRoute(
name: "Product",
url: "{controller}/{action}/{id}",
defaults: new { controller = "Product", action = "Browse",
id=UrlParameter.
Optional},
constraints: new { id = @"\d*" }
);
```

- In this code, a route with three placeholders, `controller`, `action`, and `id`. In this route, the default value for the controller placeholder is set to `Product`, the action placeholder is set to `Browse`.
- In addition, a constraint is applied to an `id` optional parameter. This constraint uses a regular expression to specify that the id parameter can match only with an `integer` value.

Ignoring a Route

- The MVC Framework provides you the flexibility to ignore routes.
- You can use the `IgnoreRoute()` method of the `RoutesTable` class to specify a route that the MVC routing engine should ignore.
- Following code shows how to ignore a route:

Code Snippet:

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

- In this code, the `routes.IgnoreRoute()` method specifies that resources with the `.axd` extension should be ignored by the route engine and served directly as response.

Summary

- A controller is responsible for intercepting incoming requests and executing the appropriate application code.
- To create a controller in an ASP.NET MVC application, you will need to create a C# class that extends the Controller class.
- A controller class can contains one or more action methods, also known as controller actions.
- Although, most action methods return an ActionResult object, an action method can also return other types, such as String, int, or bool.
- Routing is a process that maps incoming requests to specified controller actions.
- When you create a route, you need to define a URL pattern that can contain literal values and placeholders separated by the slash (/) character for the route.
- The routing engine allows you to apply constraints around the placeholder values and also ignore routes.