MCSE 541:Web Computing and Data Mining

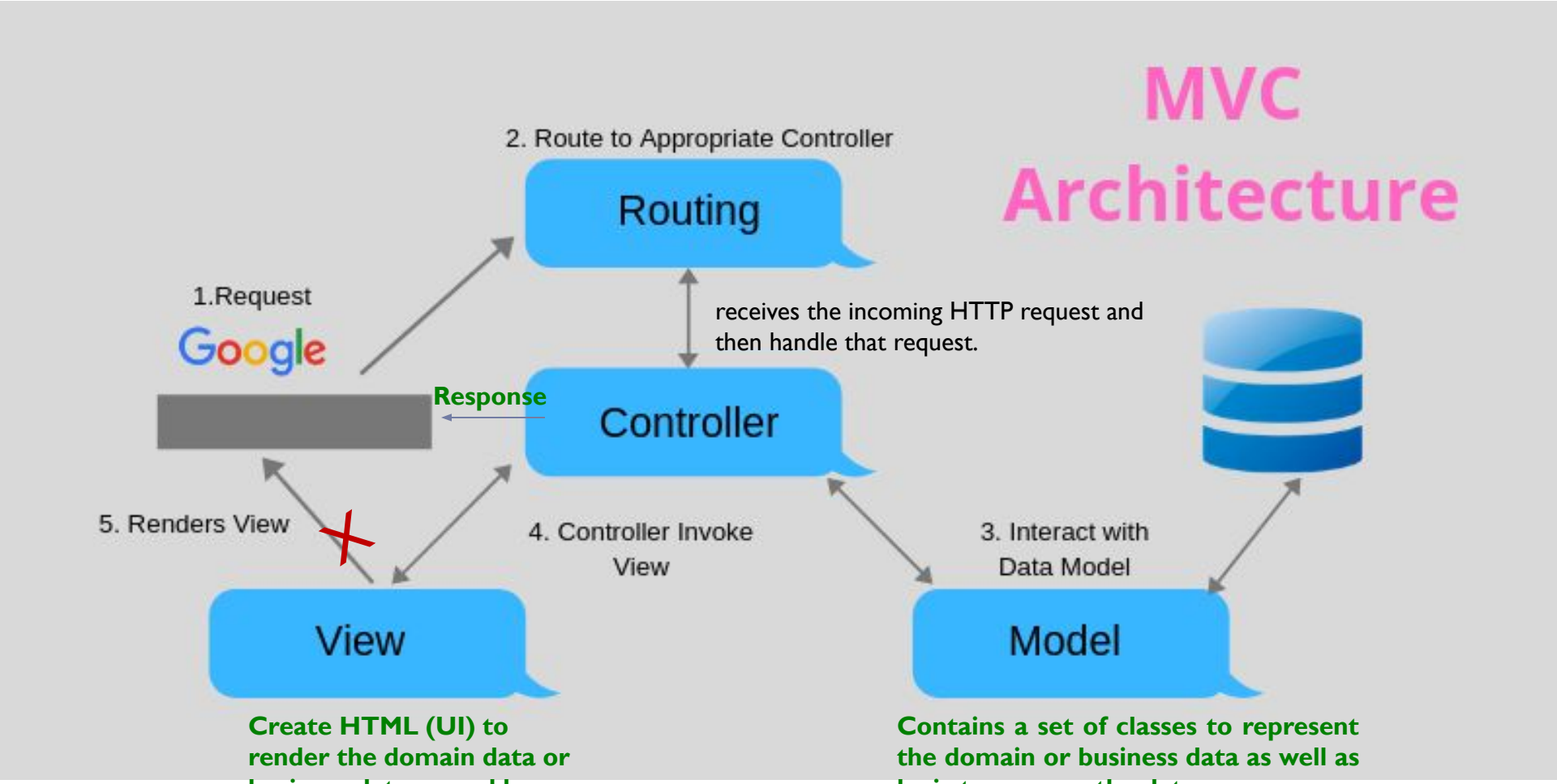
ASP.NET Core-MVC

Prof. Dr. Shamim Akhter

MVC (Model-View-Controller) Architecture

- A concept in software programming
- Divides the program into three parts:
  - Model, View and Controller.
- MVC becomes popular
  - to design web and mobile application
  - famous Frameworks uses MVC like Ruby on rails, Js, Django (Python Framework), Symphony (PHP Framework), CakePHP (PHP), Laravel (PHP framework), CherryPy (Python Framework), react, Angular, and etc.

ASP.NET MVC is a **Framework** whereas **MVC** is a **Design Pattern**.



Create HTML (UI) to render the domain data or business data passed by Controller.

Contains a set of classes to represent the domain or business data as well as logic to manage the data.

Example: Student Details Web Page

- Request url:"**http://dotnettutorials.net/student/details/2**"

```
public class StudentController : Controller
{
    public ActionResult Details(int studentId)
    {
        StudentBusinessLayer studentBL = new StudentBusinessLayer();
        Student studentDetail = studentBL.GetById(studentId);
        return View(studentDetail);
    }
}
```

Student ID: 2
Name: James
Gender: Male
Branch: CSE
Section: A2

Model

```
public class Student
{
    public int StudentID { get; set; }
    public string Name { get; set; }
    public string Gender { get; set; }
    public string Branch { get; set; }
    public string Section { get; set; }
}
```

Model to represent the student data

```
public class StudentBusinessLayer
{
    public Student GetById(int StudentID)
    {
        Student student = new Student()
        {
            StudentID = StudentID,
            Name = "James",
            Gender = "Male",
            Branch = "CSE",
            Section = "A2",
        };
        return student;
    }
}
```

Model to manage the Student data

View

```
@model FirstMVCApplication.Models.Student
<html>
<head>
    <title>Student Details</title>
</head>
<body>
    <br />
    <br />
    <table>
        <tr>
            <td>Student ID: </td>
            <td>@Model.StudentID</td>
        </tr>
        <tr>
            <td>Name: </td>
            <td>@Model.Name</td>
        </tr>
        <tr>
            <td>Gender: </td>
            <td>@Model.Gender </td>
        </tr>
        <tr>
            <td>Branch: </td>
            <td>@Model.Branch</td>
        </tr>
        <tr>
            <td>Section: </td>
            <td>@Model.Section </td>
        </tr>
    </table>
</body>
</html>
```

Each developer can work on different parts of the application. For example, one developer may work on the view while the second developer can work on the controller logic and the third developer may work on the business logic.

Look Again

- **"http://dotnettutorials.net/student/details/2"**

```
public class StudentController : Controller
{
    public ActionResult Details(int studentId)
    {
        StudentBusinessLayer studentBL = new StudentBusinessLayer();
        Student studentDetail = studentBL.GetById(studentId);
        return View(studentDetail);
    }
}
```

Something is missing??

Where this mapping is defined?

- Routing-a middleware must be able to route incoming HTTP requests to a controller

  Mapping is defined within the RegisterRoutes() of the RouteConfig class at RouteConfig.cs class within the App_Start Folder.

ASP.Net 4.5 MVC Template

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {

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

Default Route
URL Pattern
Default Controller
Default Action Method

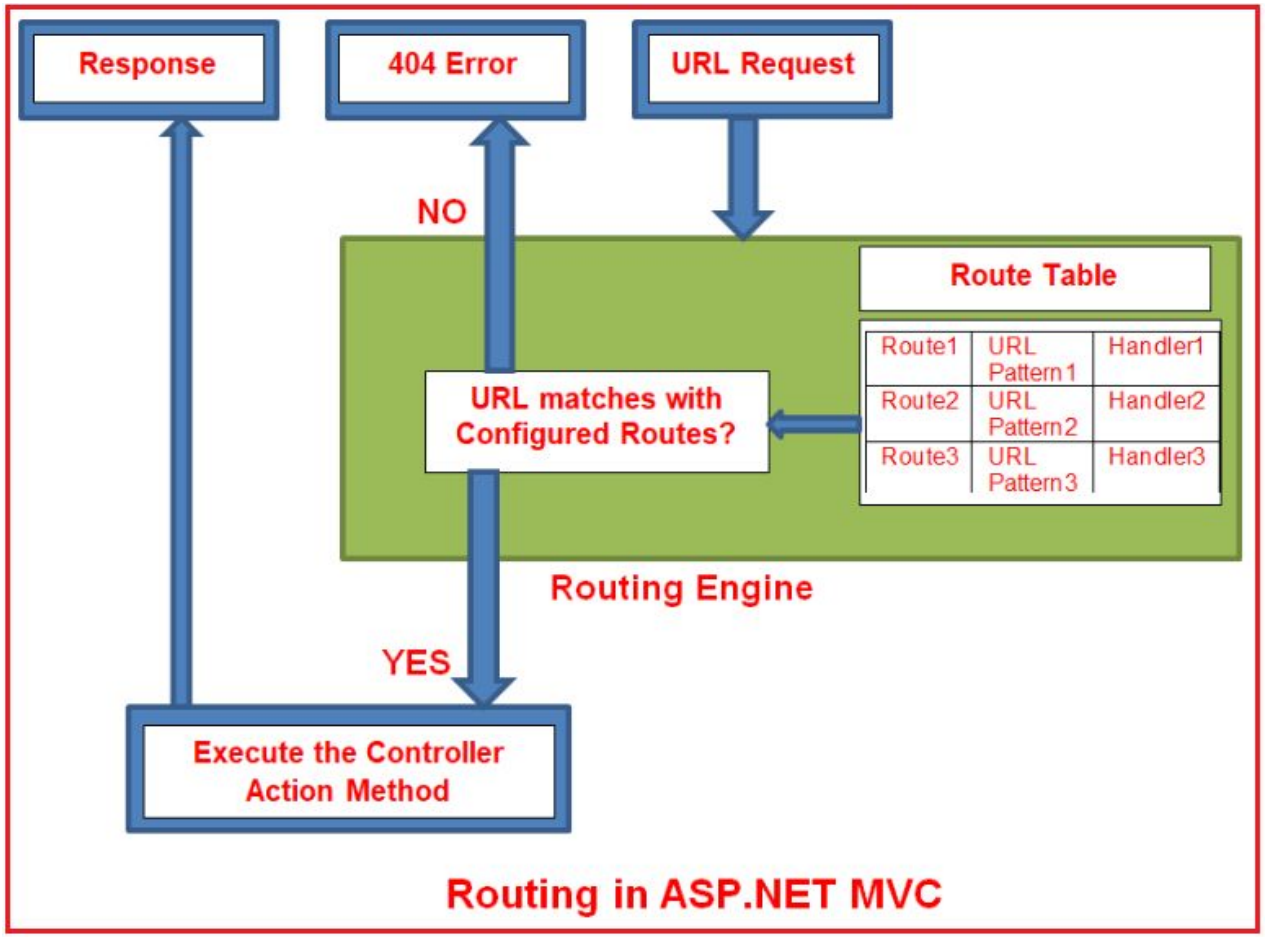
RouteConfig class

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

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

            routes.MapRoute(
                name: "Default", //Route Name
                url: "{controller}/{action}/{id}", //Route Pattern
                defaults: new
                {
                    controller = "Home", //Controller Name
                    action = "Index", //Action method Name
                    id = UrlParameter.Optional //Defaut value for above defined
parameter
                }
            );
        }
    }
}
```

Routing in ASP.NET MVC



Employee example from book.

Routing define in Startup class

/Home/Index/123

ASP.Net Core 2.1 MVC

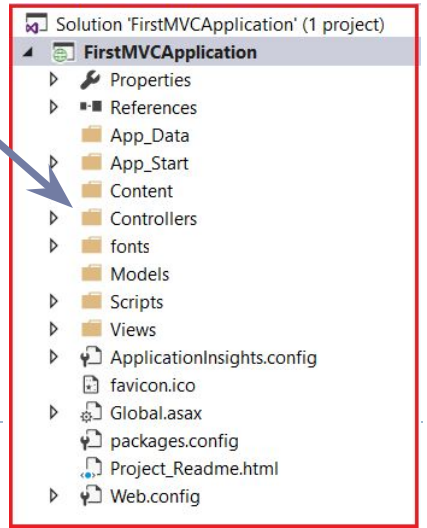
```
app.UseMvc(routes =>
{
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
});
```

Or with an explicit method called from the **UseMvc** method inside the **Configure** method:

```
app.UseMvc(ConfigureRoutes);
```

```
private void ConfigureRoutes(IRouteBuilder routeBuildrer)
{
    routeBuildrer.MapRoute("Default",
        "{controller=Home}/{action=Index}/{Id?}");
}
```

HomeController class, Index method, Id parameter



- **http://localhost:53605/ =>**
  controller = **Home**, action = **Index**, id = **none**, since default value of controller and action are Home and Index respectively.
- **http://localhost:53605/Home =>**
  controller = **Home**, action = **Index**, id = **none**, since default value of action is Index
- **http://localhost:53605/Home/Index =>**
  controller = **Home**, action = **Index**, id=**none**
- **http://localhost:53605/Home/Index/5 =>**
  controller = **Home**, action = **Index**, id = **5**

/Employee/Name/
/Employee/Index
/Employee

```
public class EmployeeController
{
    public string Name()
    {
        return "Jonas";
    }

    public string Country()
    {
        return "Sweden";
    }

    public string Index()
    {
        return "Hello from Employee";
    }
}
```

Types of Routing

- Convention-Based Routing
- Attribute Routing
  - Assign attributes to the controller class and its action methods. The metadata in those attributes tell ASP.NET when to call a specific controller and action.

```
[Route("employee")]
public class EmployeeController
```

Ambiguous Action Exception

To solve this, you can specify the **Route** attribute for each of the action methods, and use an empty string for the default action. Let's make the **Index** action the default action, and name the routes for the other action methods the same as the methods.

```
[Route("")]
public string Index()
{
    return "Hello from Employee";
}

[Route("name")]
public string Name()
{
    return "Jonas";
}

[Route("country")]
public string Country()
{
    return "Sweden";
}
```

Attribute Routing

Let's clean up the controller and make its route more reusable. Instead of using a hardcoded value for the controller's route, you can use the **[controller]** token that represents the name of the controller class (*Employee* in this case). This makes it easier if you need to rename the controller for some reason.

```
[Route("[controller]")]
public class EmployeeController
```

You can do the same for the action methods, but use the **[action]** token instead. ASP.NET will then replace the token with the action's name.

```
[Route("[action]")]
public string Name()
{
    return "Jonas";
}
```

```
[Route("")]
[Route("[action]")]
public string Index()
{
    return "Hello from Employee";
}
```

Attribute Routing

```
[Route("company/[controller]/[action]")]
public class EmployeeController
{
    public string Name()
    {
        return "Jonas";
    }

    public string Country()
    {
        return "Sweden";
    }

    public string Index()
    {
        return "Hello from Employee";
    }
}
```

No need to place on top of every action

IActionResult

The controller actions that you have seen so far have all returned strings. When working with actions, you rarely return strings. Most of the time you use the **IActionResult** return type, which can return many types of data, such as objects and views. To gain access to **IActionResult** or derivations thereof, the controller class must inherit the **Controller** class.

There are more specific implementations of that interface, for instance the **ContentResult** class, which can be used to return simple content such as strings. Using a more specific return type can be beneficial when unit testing, because you get a specific data type to test against.

Another return type is **ObjectType**, which often is used in Web API applications because it turns the result into an object that can be sent over HTTP. JSON is the default return type, making the result easy to use from JavaScript on the client. The data carrier can be configured to deliver the data in other formats, such as XML.

Implementing ContentResult

Let's change the **Name** action to return a **ContentResult**.

1. Open the **EmployeeController** class.
2. Have the **EmployeeController** class inherit the **Controller** class.
   ```
   public class EmployeeController : Controller
   ```
3. Change the **Name** action's return type to **ContentResult**.
   ```
   public ContentResult Name()
   ```
4. Change the **return** statement to return a content object by calling the **Content** method, and pass in the string to it.
   ```
   public ContentResult Name()
   {
       return Content("Jonas");
   }
   ```
5. Save all files, open the browser, and navigate to the *Company/Employees/Name* URL.
6. Your name should be returned to the browser, same as before.

ObjectResult

Video Model Class

```
public class Video
{
    public int Id { get; set; }
    public string Title { get; set; }
}
```

Controller Class

```
public class HomeController : Controller
{
    public ObjectResult Index()
    {
        var model = new Video { Id = 1, Title = "Shreck" };
        return new ObjectResult(model);
    }
}
```

Introduction to Views

The most popular way to render a view from a ASP.NET Core MVC application is to use the Razor view engine. To render the view, a **ViewResult** is returned from the controller action using the **View** method. It carries with it the name of the view in the filesystem, and a model object if needed.

```
public ViewResult Index()
{
    var model = new List<Video>
    {
        new Video { Id = 1, Title = "Shreck" },
        new Video { Id = 2, Title = "Despicable Me" },
        new Video { Id = 3, Title = "Megamind" }
    };
    return View(model);
}
```

```
@model IEnumerable<AspNetCoreVideo.Models.Video>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Video</title>
</head>
<body>
    <table>
        @foreach (var video in Model)
        {
        <tr>
            <td>@video.Id</td>
            <td>@video.Title</td>
        </tr>
        }
    </table>
</body>
</html>
```