Pentalog

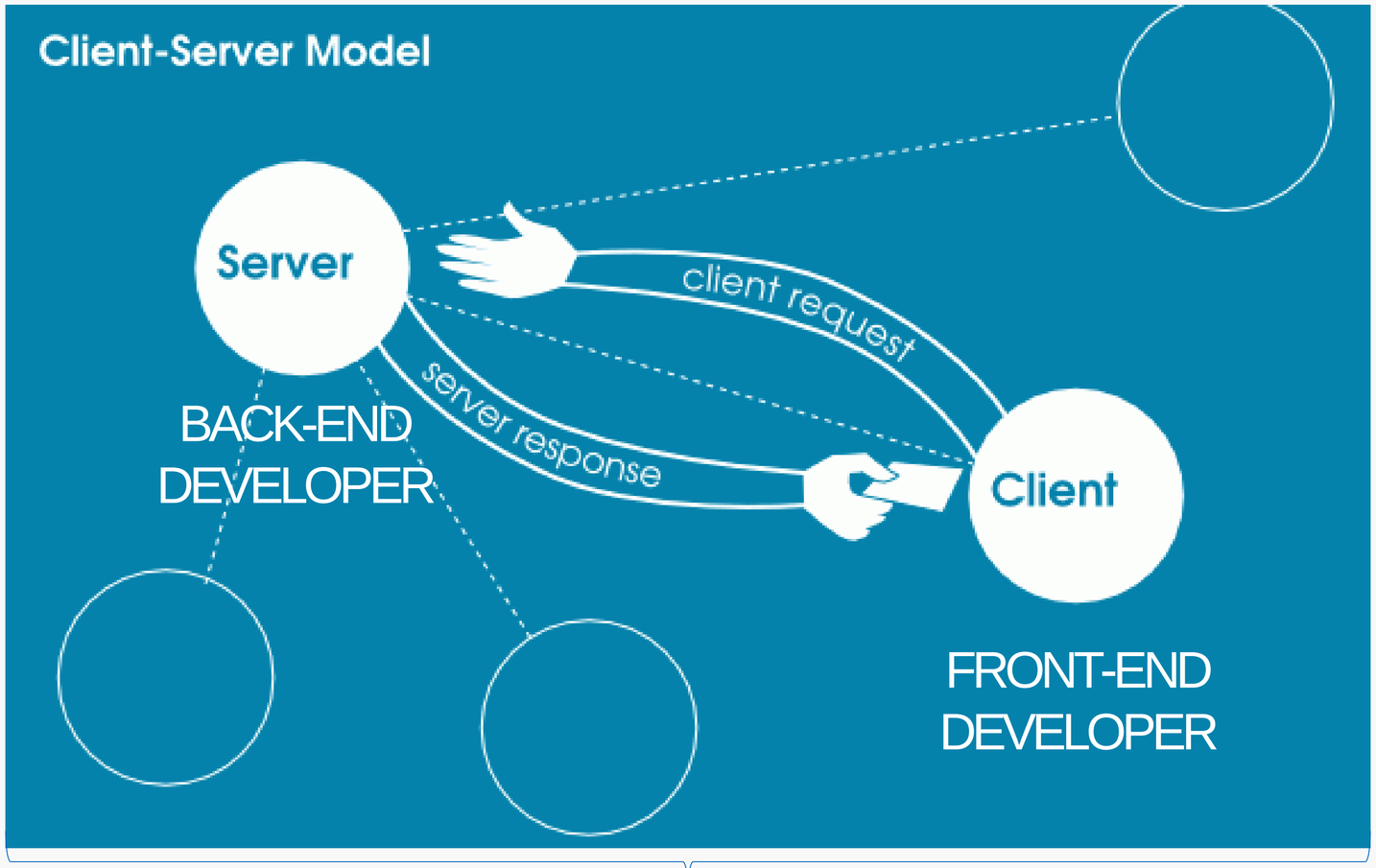
Aplicatii web ASP.NET MVC

PentaStagiu Remote Brașov

November, 2019 – March, 2020

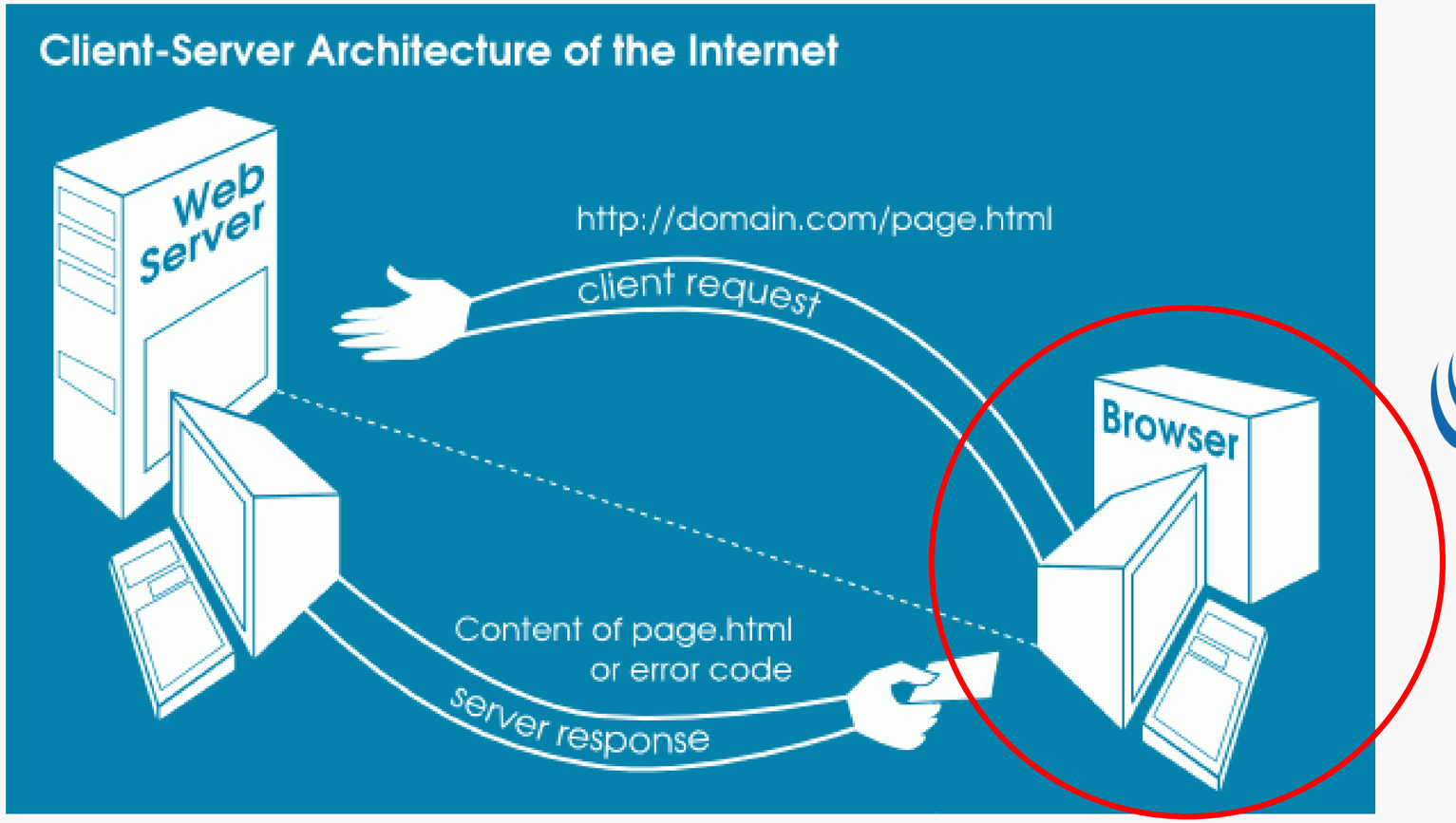
DATA ANALYSIS
UX/UI
FRONT-END
TECHNOLOGY
NEAR/OFFSHORE
AGILITY
BACK-END
SPRINT
KANBAN
CAMPAIGNS
GROWTH HACKING
SCRUM
BACKLOG
DEVOPS
DESIGN
SEO
CONTINUOUS INTEGRATION
MOBILE
QA
AUTOMATION
RESPONSIVE
UNIT TESTING

Web basics



FULL-STACK WEB
DEVELOPER

FRONT-END



HTML

CSS

JavaScript

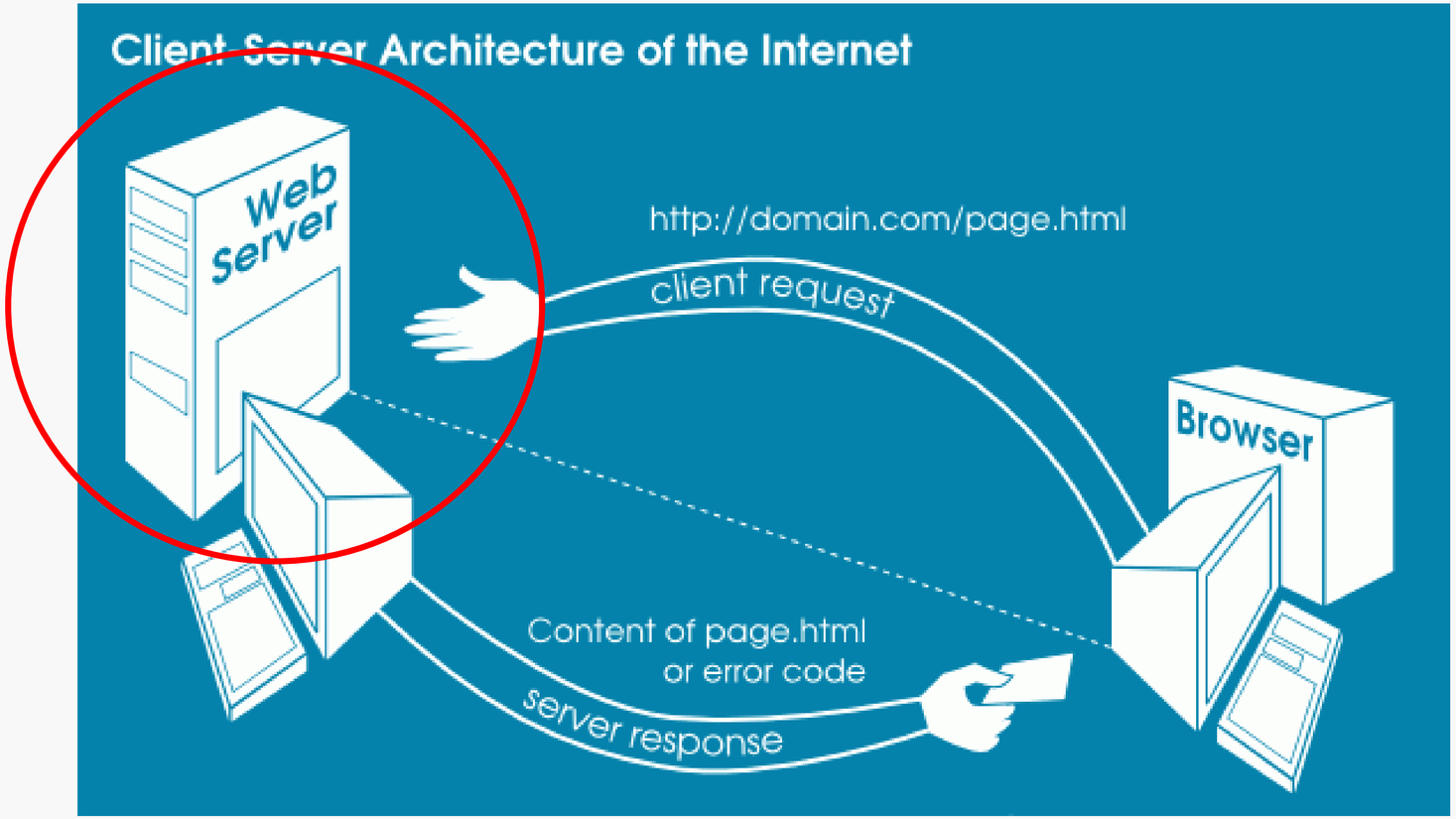
jQuery
write less, do more.

React

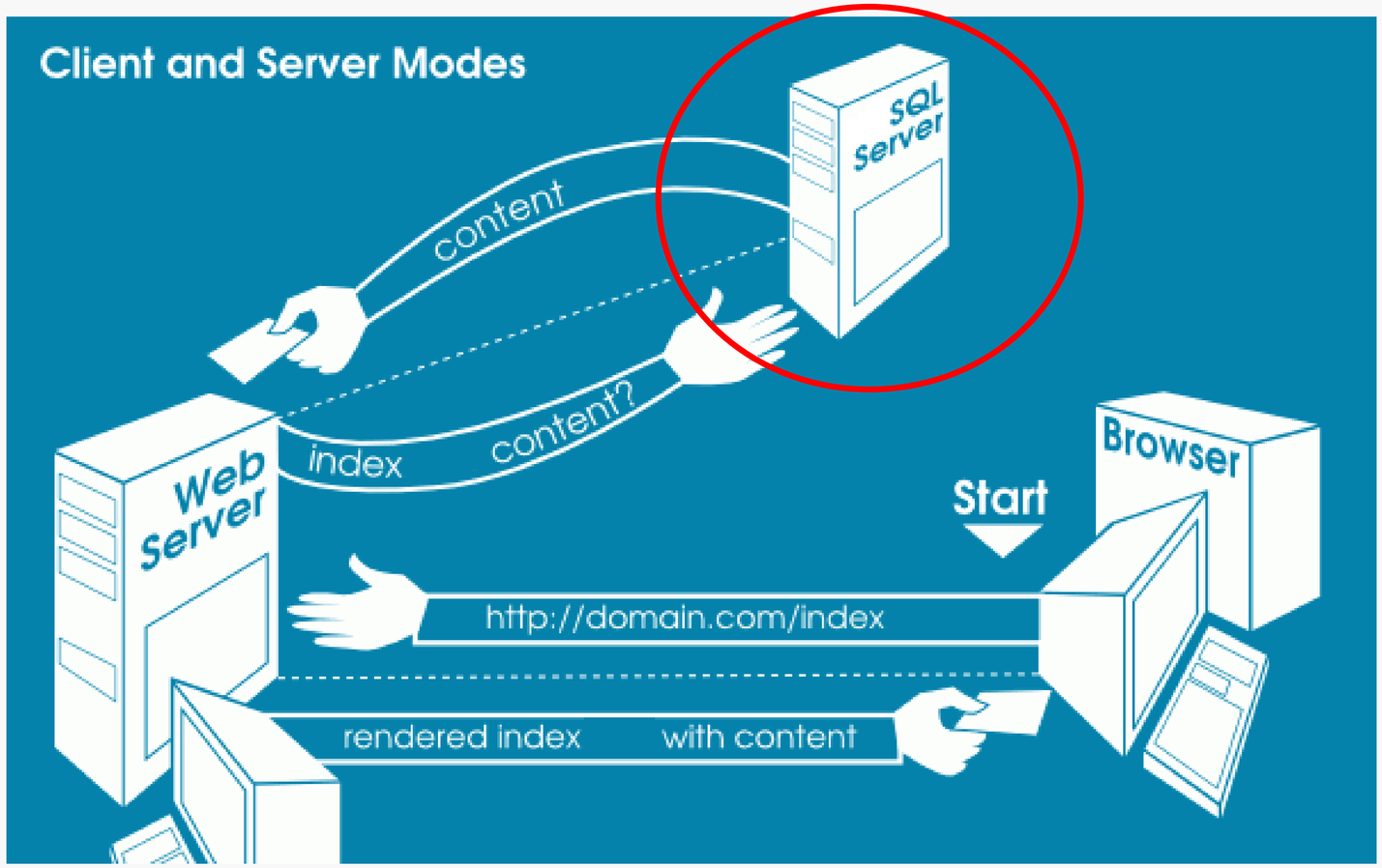
Bootstrap

RAILS

BACK-END


Client Server Architecture of the Internet
Web Server
http://domain.com/page.html
client request
Browser
Content of page.html
or error code
server response

BACK-END - Databases



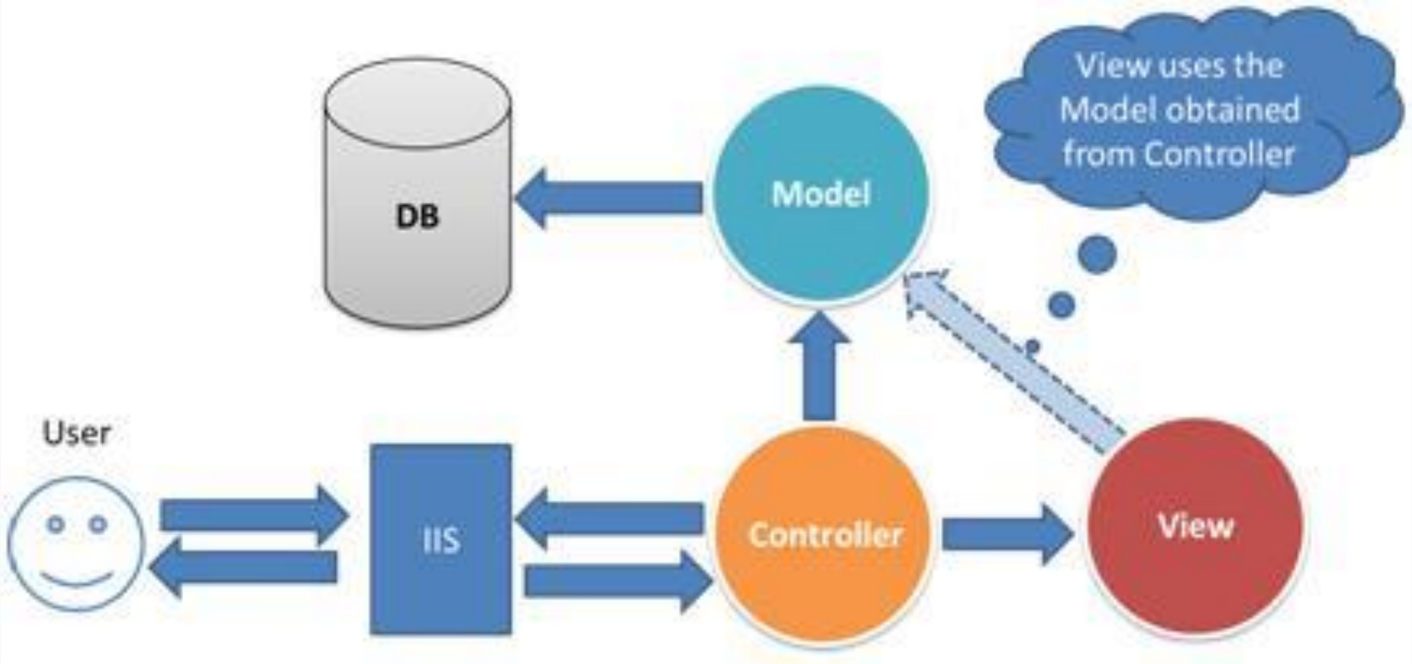
Microsoft SQL Server

Microsoft Entity Framework

ASP.NET MVC

Introduction to ASP.NET MVC

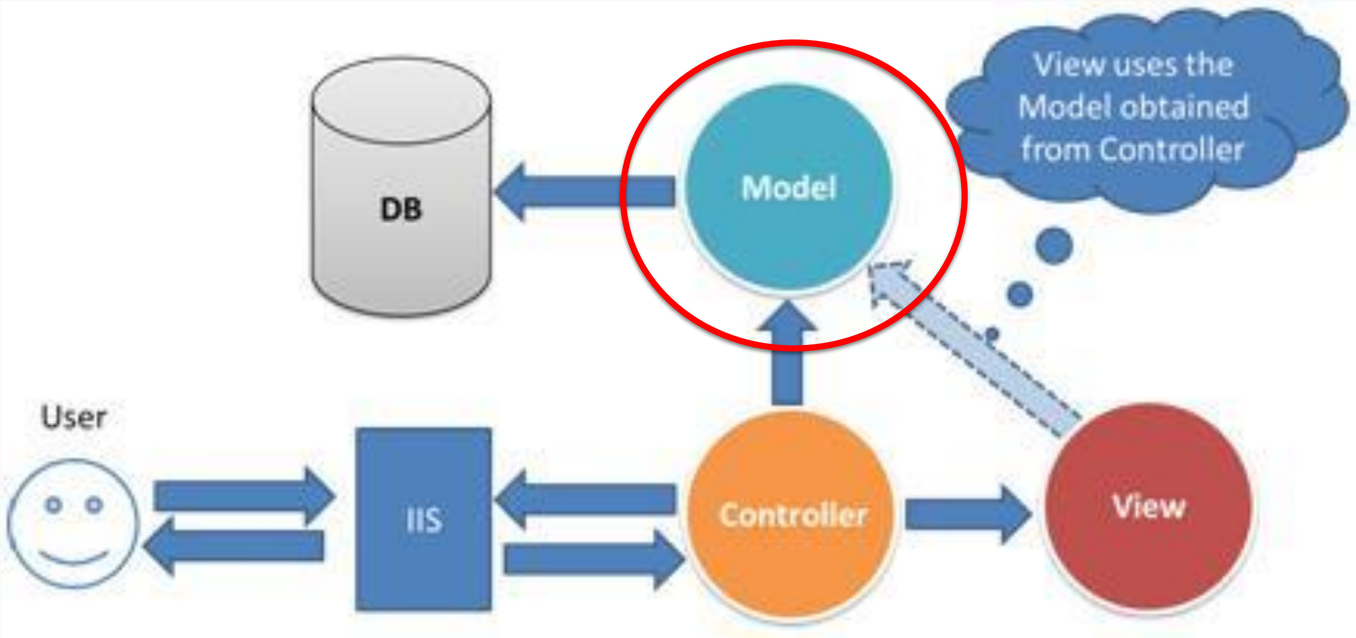
- A web application framework from Microsoft
- It implements the Model-View-Controller pattern
- It uses C# for the server code



ASP.NET MVC - Model

- In ASP.NET MVC a Model is just a simple C# class

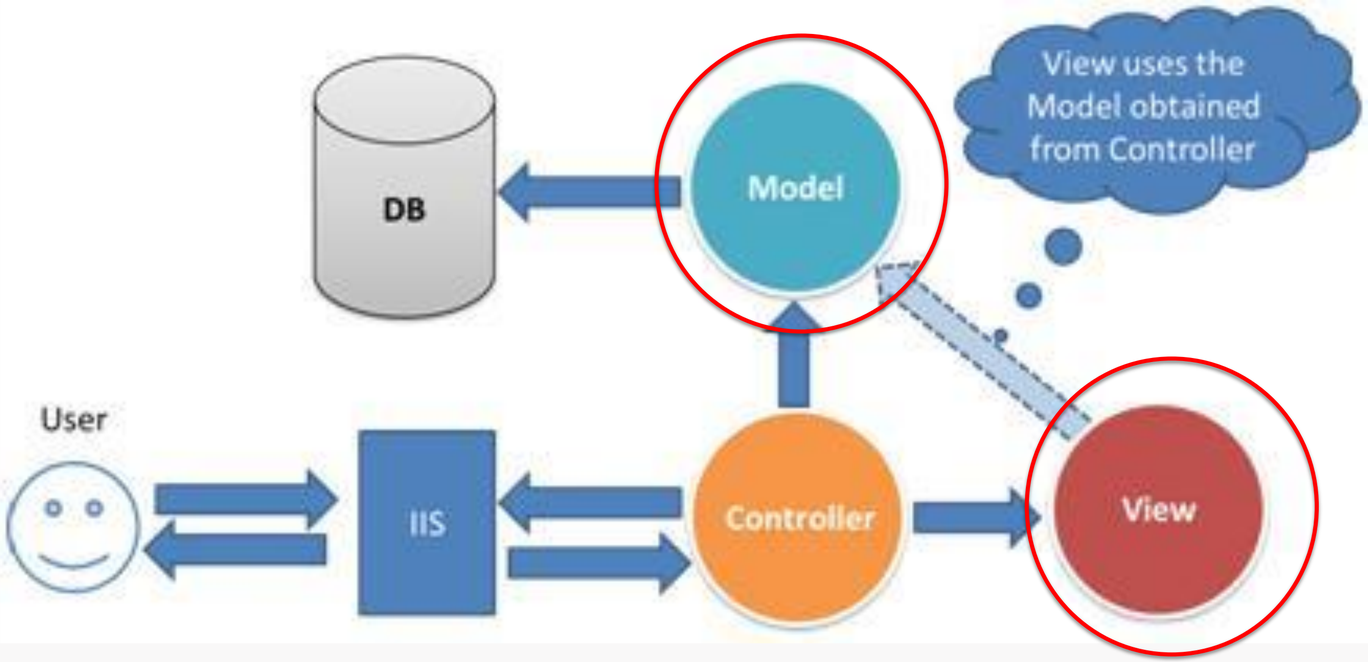
```
namespace MvcCourse.Models
{
    0 references
    public class User
    {
        0 references
        public string UserName { get; set; }
        0 references
        public string Email { get; set; }
        0 references
        public string Phone { get; set; }
    }
}
```



- It is usually the code representation of a table from the database – an entity
- Usually it only has properties (no complex methods)
- Found in the **Models** folder

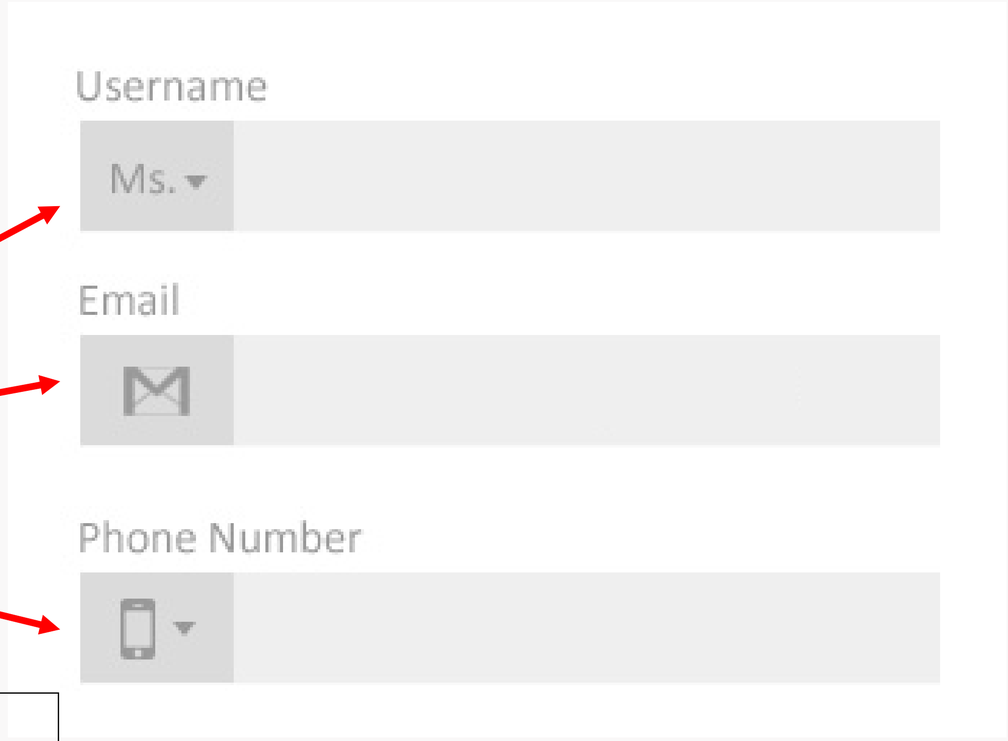
ASP.NET MVC - View

- Simply put, Views are what the user sees (the HTML)
- In ASP.NET MVC a View usually receives a Model (it is strongly typed)
- It is a visual representation of the Model (it displays the data from the Model)
- Only knows about the Model (it doesn't have access to the database)
- Found in the **Views** folder



ASP.NET MVC -> Model - View

```
namespace MvcCourse.Models
{
    0 references
    public class User
    {
        0 references
        public string UserName { get; set; }
        0 references
        public string Email { get; set; }
        0 references
        public string Phone { get; set; }
    }
}
```



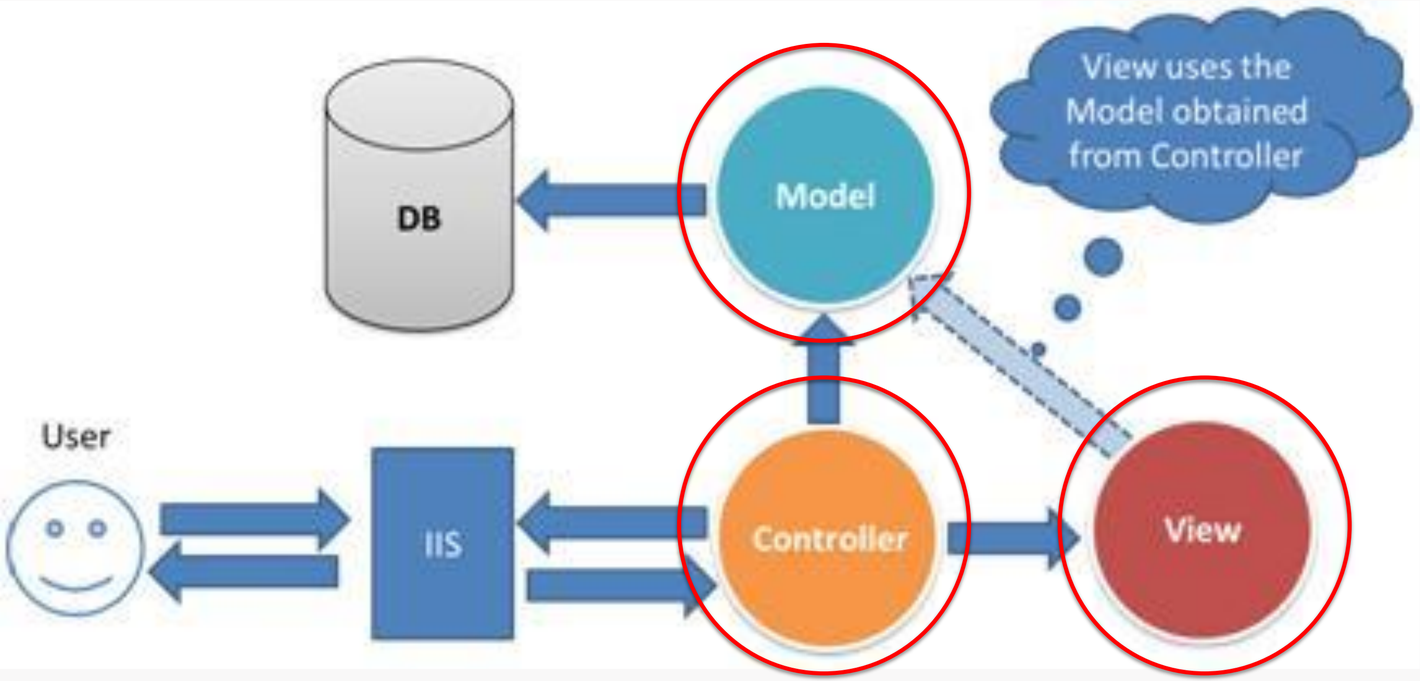
Model → View

View receives a Model

- But how does the View receive the Model?
- Who creates the Model and who decides what data to put in it and where to get it from?

ASP.NET MVC -> Controller

- A controller is responsible for handling requests made to the website
- Found in the **Controllers** folder



ASP.NET MVC -> Controller

```
public class HomeController : Controller
{
    0 references | TFVC Import, 42 days ago | 1 author, 1 change
    public ActionResult Index()
    {
        return View();
    }

    0 references | TFVC Import, 42 days ago | 1 author, 1 change
    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    0 references | TFVC Import, 42 days ago | 1 author, 1 change
    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

It is a C# class too, but it derives from controller

Index is the default action, you can access this by:
www.mysite.com/Home

Or (because Home is the default controller) directly by:
www.mysite.com

You can access this action by:
www.mysite.com/Home/Contact

- The public methods (in this case Index, About, Contact) in a Controller are called **Actions**
- Every **Action** returns a view (in most cases) or a other kinds of results (Action Results)
- You can access an action by: www.site.com/ControllerName/ActionName

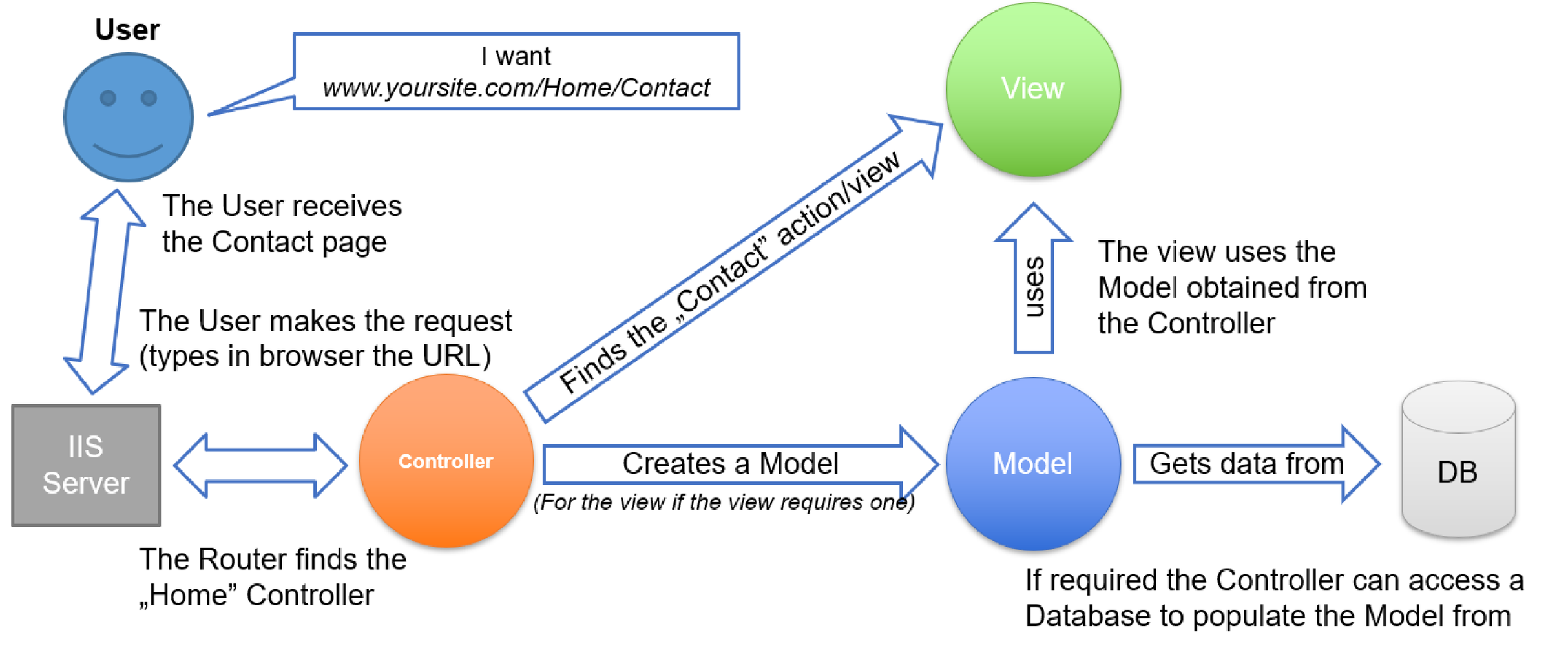
But who decides how you can access an action ?

ASP.NET MVC - Router

```
1 reference | TFVC Import, 42 days ago | 1 author, 1 change
public class RouteConfig
{
    1 reference | TFVC Import, 42 days ago | 1 author, 1 change
    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 }
        );
    }
}
```

- The route config defines the default route that your MVC app uses
- You can find it in *App_Start/RouteConfig.cs*
- The default route has ***Home*** as the default controller and ***Index*** as the default action

Let's recap MVC



Let’s recap MVC

Type	Helper Method
ViewResult	View()
PartialViewResult	PartialView()
ContentResult	Content()
RedirectResult	Redirect()
RedirectToRouteResult	RedirectToAction()
JsonResult	Json()
FileResult	File()
HttpNotFoundResult	HttpNotFound()
EmptyResult	

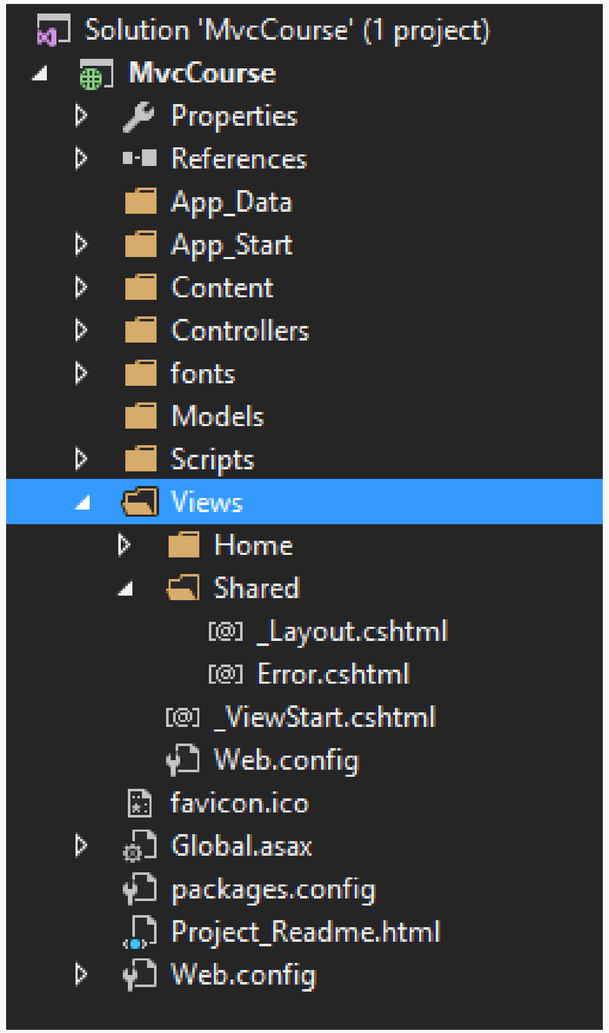
- Besides views, controller actions can return more types (called Action Results)

DEMO

The ASP.NET MVC Request Workflow

Back to Views

- In ASP.NET MVC views have the extension .cshtml (C# + HTML)
- The views are grouped in folders by controller
- The views from the Shared folder are accessible from any controller



Views & Razor

- Views can contain plain HTML for static pages, or HTML combined with C#
- ASP.NET MVC provides a new syntax called Razor which allows mixing HTML with C# server code
- Views that use Razor syntax have a special file extension .cshtml (Razor using C#) or .vbhtml (Razor using VB)
- The 2 main rules of Razor syntax:

    1. C# code blocks are enclosed in @{ … }

    2. Inline expressions (variables and functions) start with @

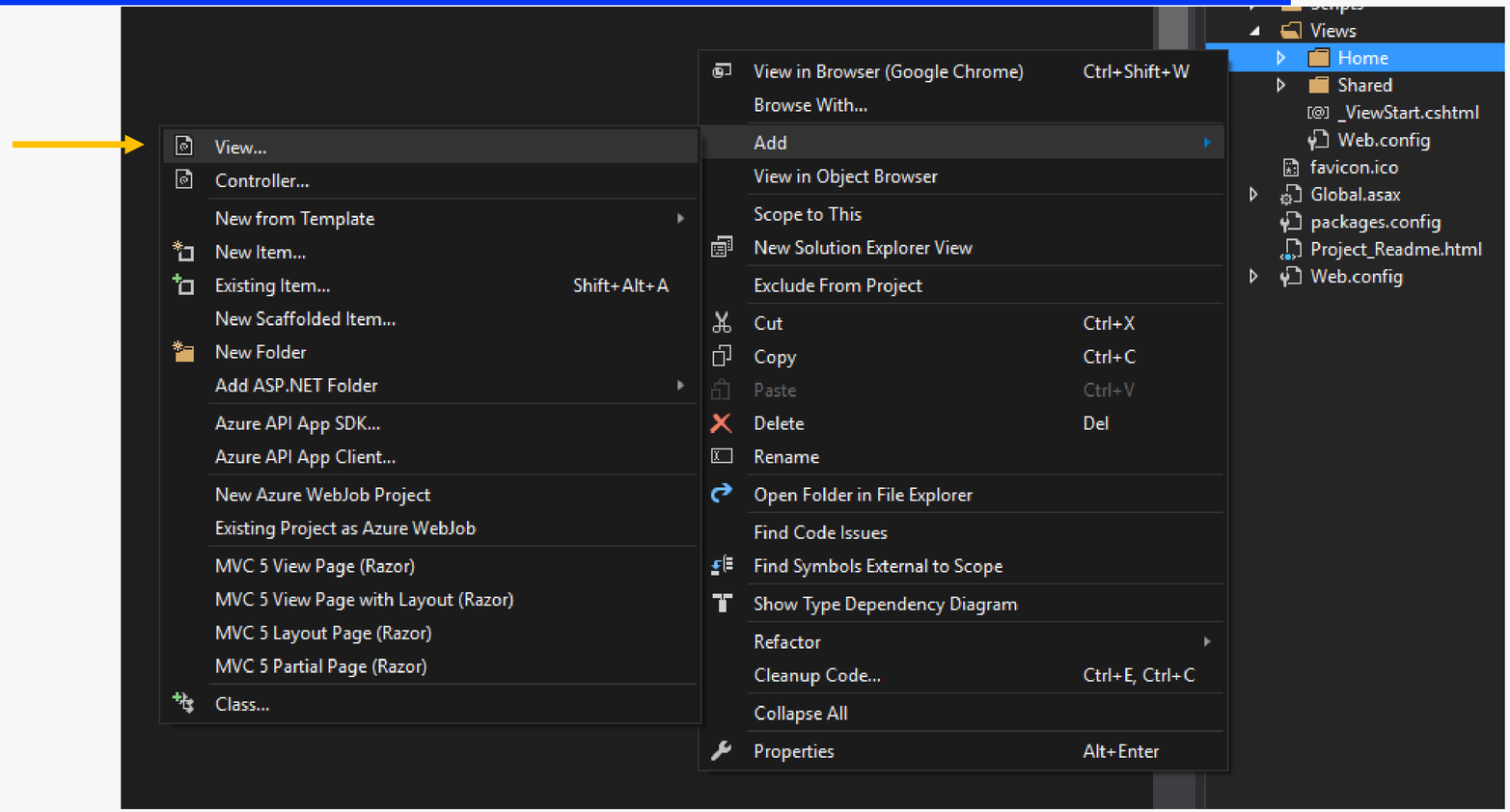
Razor syntax

```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Multi-statement block -->
@{
    var greeting = "";
    if (DateTime.Now.Hour > 12)
    {
        greeting = "Good Evening";
    }
    else
    {
        greeting = "Good Morning";
    }

    var fullName = "";
    if (IsPost) // Checks if the current request is POST
    {
        // Request[] Gets a value from the posted values
        // It can also be used for GET requests, in this case it will look in the query string
        var firstName = Request["firstname"];
        var lasttName = Request["lastname"];
        fullName = firstName + " " + lasttName;
    }

    var weekDay = DateTime.Now.DayOfWeek;
    var greetingMessage = string.Concat(greeting, " ", fullName, " Here it is: " + weekDay);
}

<!-- Inline expression or variable -->
<p>Message: @greetingMessage</p>
```

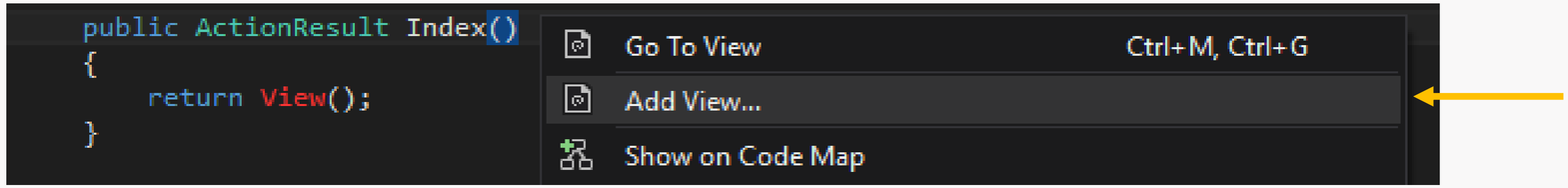
- Although Razor allows writing complex C# code in views, you should keep the logic inside them to the minimum. All the hard work must be done inside the controller.

- The views are compiled dynamically as opposed to the rest of the application (Controllers, Models)

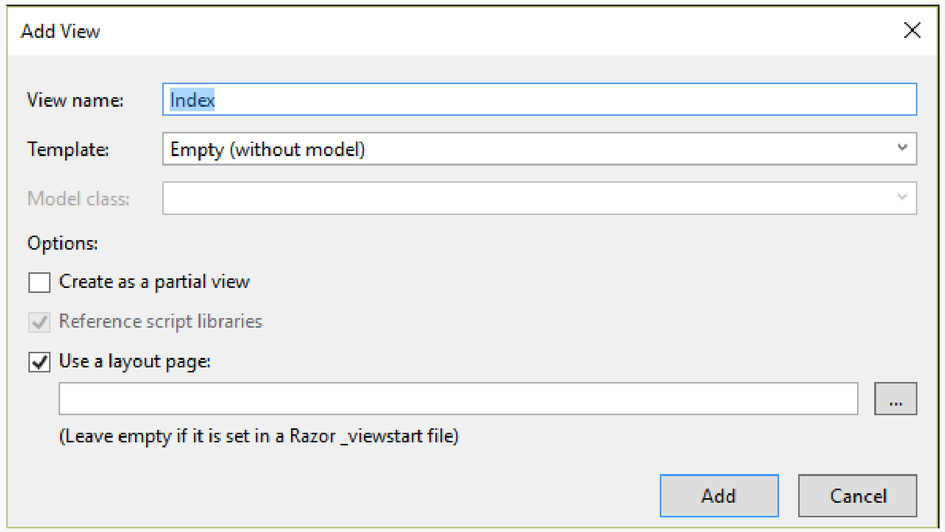
Adding a view

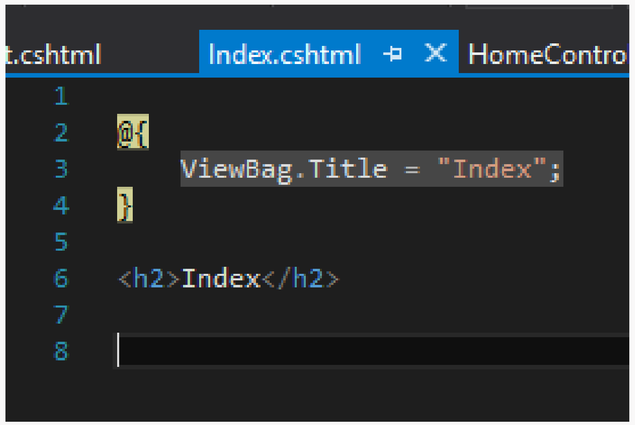


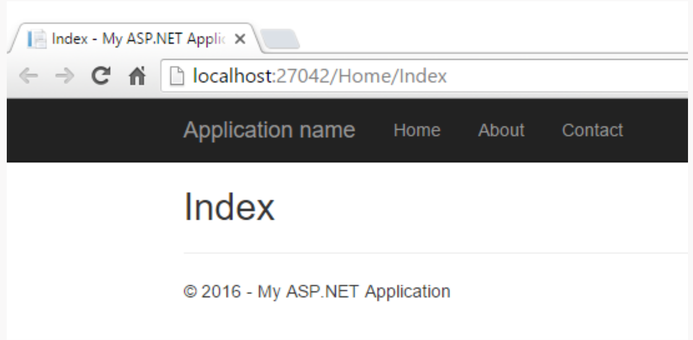
or



Adding a view (2)







Returning views

```
public ActionResult Index()
{
    return View();
}
```

```
public ActionResult Index()
{
    return View("About");
}
```

```
public ActionResult Index()
{
    return View("~/Views/Cars/Cars.cshtml");
}
```

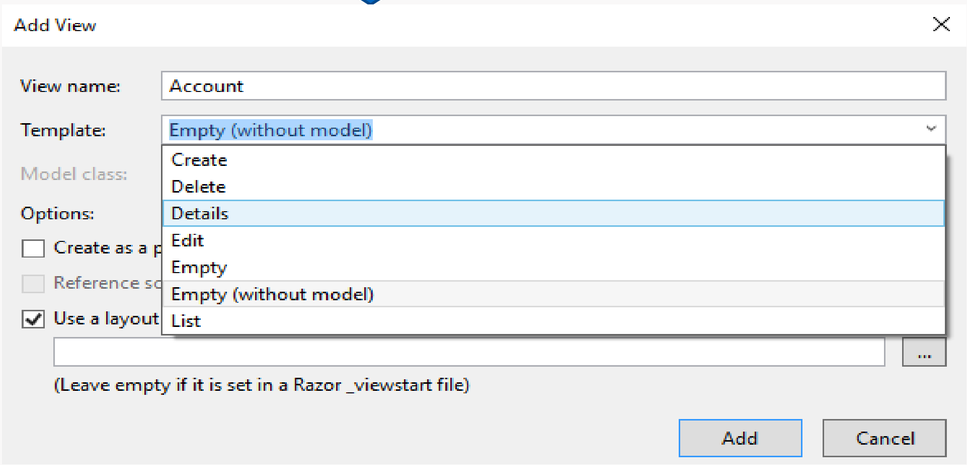
- A view can only be accessed through an action
- If the view has the same name as the action it’s not necessary to specify the name but it should be in the folder with the same name as the controller or in the Shared folder
- If the view has another name it should be specified
- If the view is from another folder the entire path should be specified

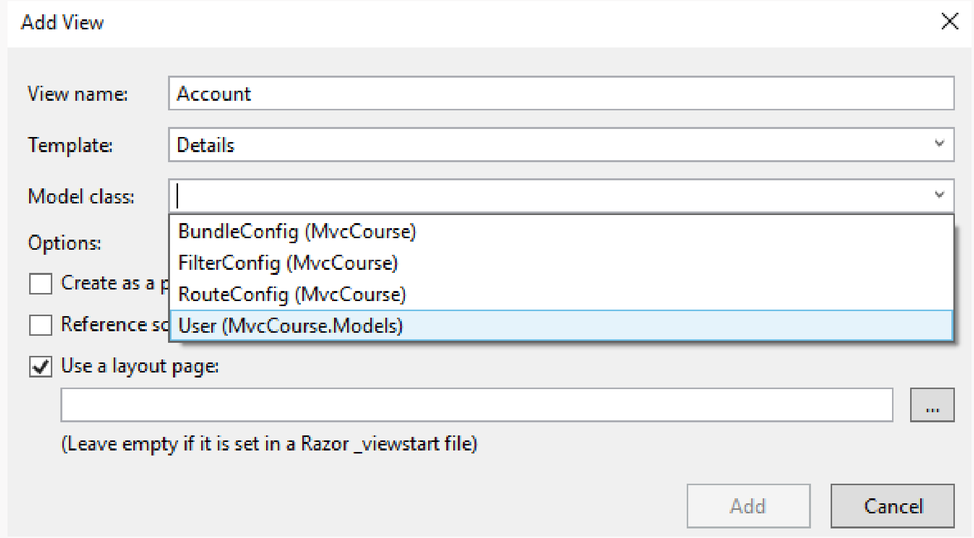
Strongly typed views

- Views can be restricted to using a type (model)

```
public ActionResult Account()
{
    User user = new User
    {
        UserName = "user",
        Email = "example@email.com",
        Phone = "000-000-000"
    };

    return View("Account", user);
}
```

Go To View Ctrl+M, Ctrl+G
Add View...
Show on Code Map
Find All References on Code Map
Show Related Items on Code Map

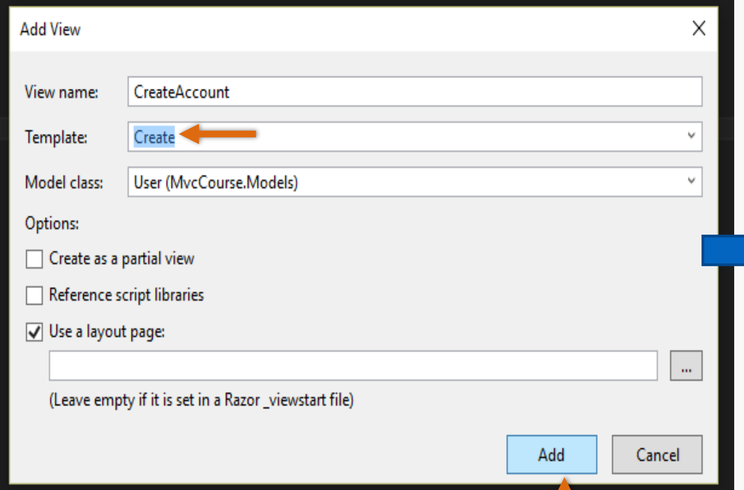


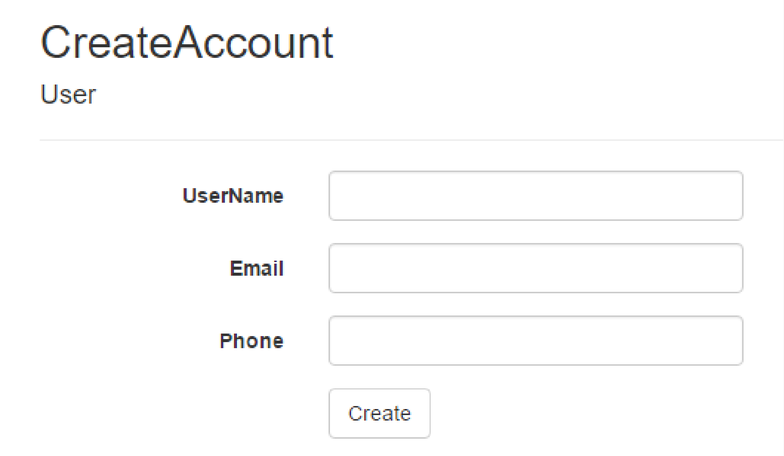


When selecting a template Visual Studio uses scaffolding to automatically generate the view's basic code for the chosen CRUD operation.

Returning views with model - POST

```
[HttpGet]
0 references
public ActionResult Account()
{
    return View("CreateAccount");
}

[HttpPost]
0 references
public ActionResult Account(User model)
{
    if (!ModelState.IsValid)
    {
        return View("Error");
    }

    return View("Account", model);
}
```






```
@model MvcCourse.Models.User

@{
    ViewBag.Title = "CreateAccount";
}

<h2>CreateAccount</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>User</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.UserName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.UserName, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.UserName, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Email, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Email, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Email, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Phone, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Phone, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Phone, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

DEMO

ASP.NET MVC
- Use Scaffolding to auto-generate views based on models
- Model binding

ViewBag

ViewBag enables you to dynamically pass data from the controller to the view. ViewBag is of type dynamic.

```
0 references
public ActionResult Index()
{
    ViewBag.PageTitle = "This is the page title";
    ViewBag.PageDescription = "This is the page description";
    ViewBag.PageCreateDate = DateTime.Now;
    ViewBag.CurrentUser = new User()
    {
        UserName = "user",
        Email = "email@example.com",
        Phone = "000-000-000"
    };
    return View();
}
```

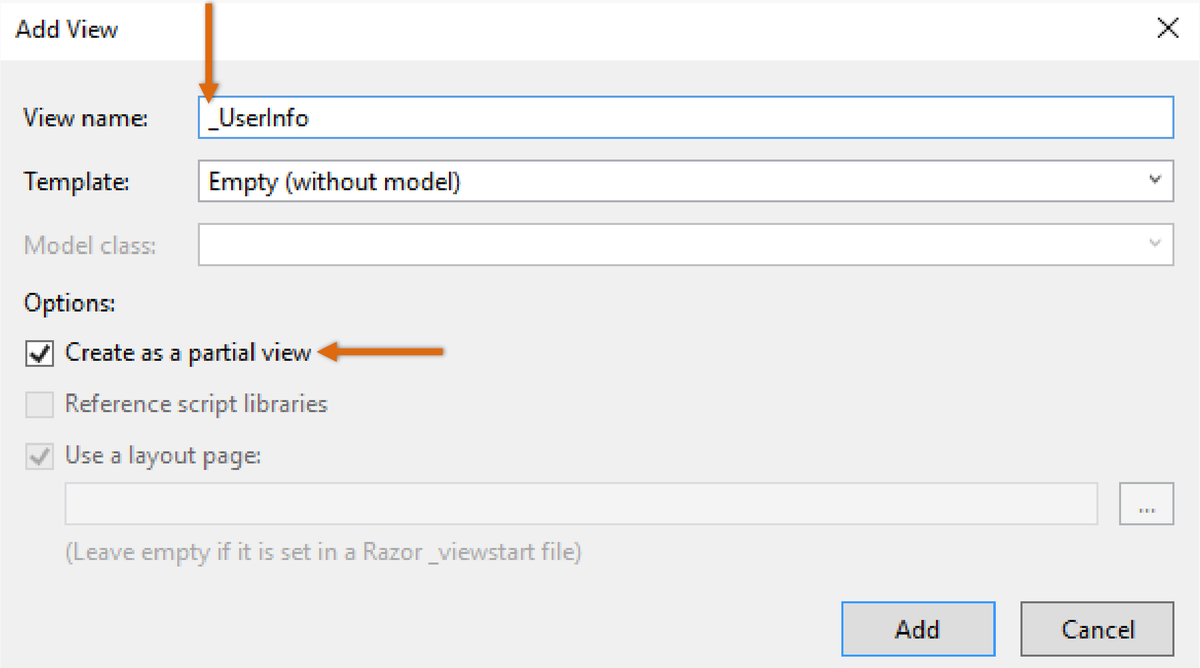
```
<h3>@ViewBag.PageTitle</h3>

<p>
    @ViewBag.PageDescription
</p>

<span>Created on @ViewBag.PageCreateDate.ToShortDateString()</span>

Current user:
<div>
    <dl>
        <dt>Name:</dt>
        <dd>@ViewBag.CurrentUser.Name</dd>
        <dt>ID:</dt>
        <dd>@ViewBag.CurrentUser.ID</dd>
        <dt>Logon Date:</dt>
        <dd>@ViewBag.CurrentUser.LogonDate.ToShortDateString()</dd>
    </dl>
</div>
```

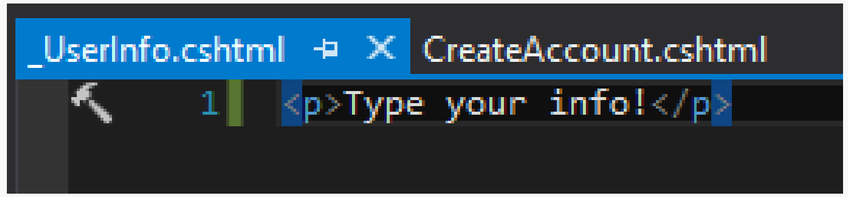
More about ViewBag and also ViewData, TempData

- http://rachelappel.com/when-to-use-viewbag-viewdata-or-tempdata-in-asp-net-mvc-3-applications/

Partial views

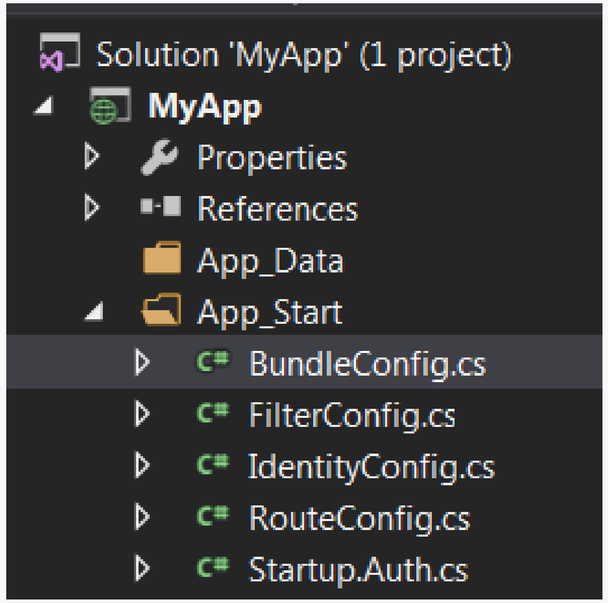
- Allow you to define a view that will be rendered inside a parent view
- Are a way to reuse code
- Usually their name start with “_”
- Have no layout





Bundle Config

The BundleConfig defines the CSS/JS bundles that are defined like below




```
public class BundleConfig
{
    1 reference | 0 exceptions
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                    "~/Scripts/jquery-{version}.js"));

        bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
                  "~/Scripts/bootstrap.js",
                  "~/Scripts/respond.js"));

        bundles.Add(new StyleBundle("~/Content/css").Include(
                  "~/Content/bootstrap.css",
                  "~/Content/site.css"));
    }
}
```

A bundle contains more CSS/JS files that are combined in a single file and minified

Layouts

- A layout is a way to set the general structure of a website or a specific area of a website
- The general layout is a good place to render the majority of the website's CSS and JS

This is where the CSS is Added

This is where the content of each view that uses this layout is rendered

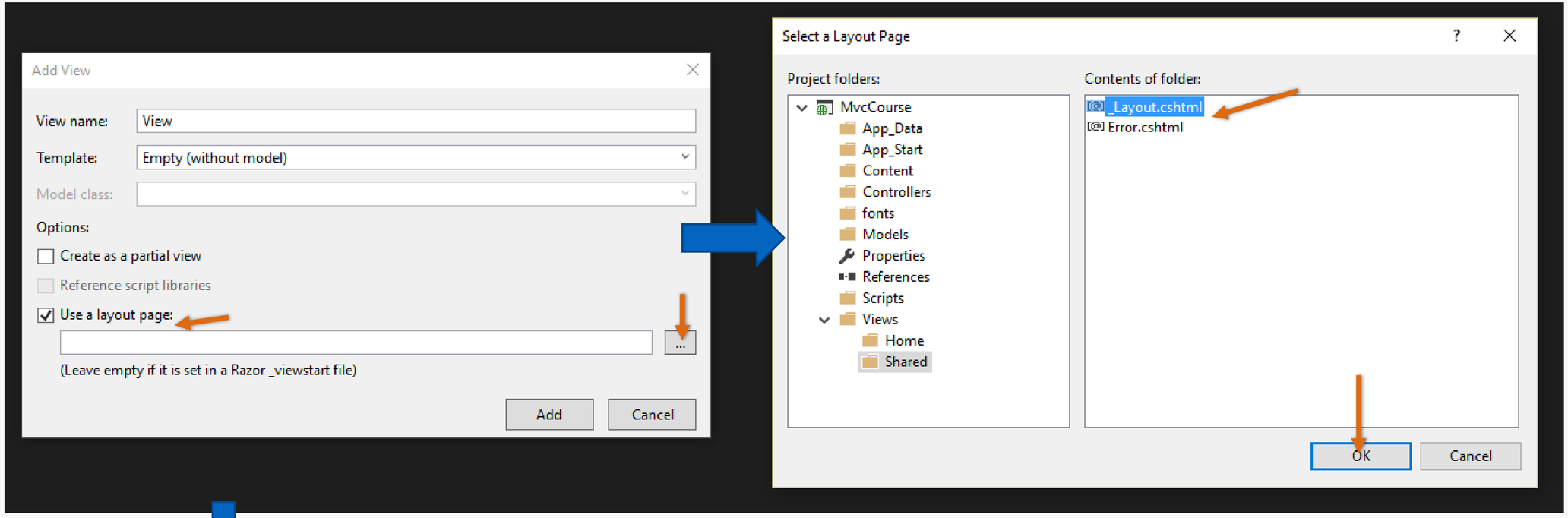
This is where the JavaScript files are added

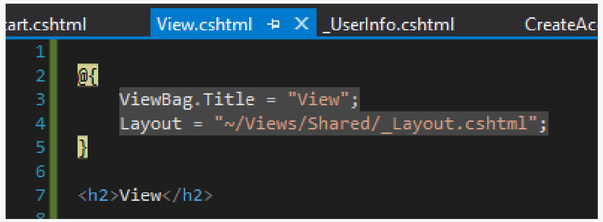
fo.cshtml | CreateAccount.cshtml | Account.cshtml | _Layout.cshtml* | HomeController.cs

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>@ViewBag.Title - My ASP.NET Application</title>
7      @Styles.Render("~/Content/css")
8      @Scripts.Render("~/bundles/modernizr")
9  </head>
10 <body>
11     <div class="navbar navbar-inverse navbar-fixed-top">
12         <div class="container">
13             <div class="navbar-header">
14                 <button type="button" class="navbar-toggle" data-toggle="collapse"
15                         data-target=".navbar-collapse">
16                     <span class="icon-bar"></span>
17                     <span class="icon-bar"></span>
18                     <span class="icon-bar"></span>
19                 </button>
20                 @Html.ActionLink("Application name", "Index", "Home",
21                     new { area = "" }, new { @class = "navbar-brand" })
22             </div>
23             <div class="navbar-collapse collapse">
24                 <ul class="nav navbar-nav">
25                     <li>@Html.ActionLink("Home", "Index", "Home")</li>
26                     <li>@Html.ActionLink("About", "About", "Home")</li>
27                     <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
28                 </ul>
29             </div>
30         </div>
31     </div>
32     <div class="container body-content">
33         @RenderBody()
34         <hr />
35         <footer>
36             <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
37         </footer>
38     </div>
39
40     @Scripts.Render("~/bundles/jquery")
41     @Scripts.Render("~/bundles/bootstrap")
42     @RenderSection("scripts", required: false)
43 </body>
44 </html>
```

Layouts

When createing a view you have the possibility to overwrite the default layout

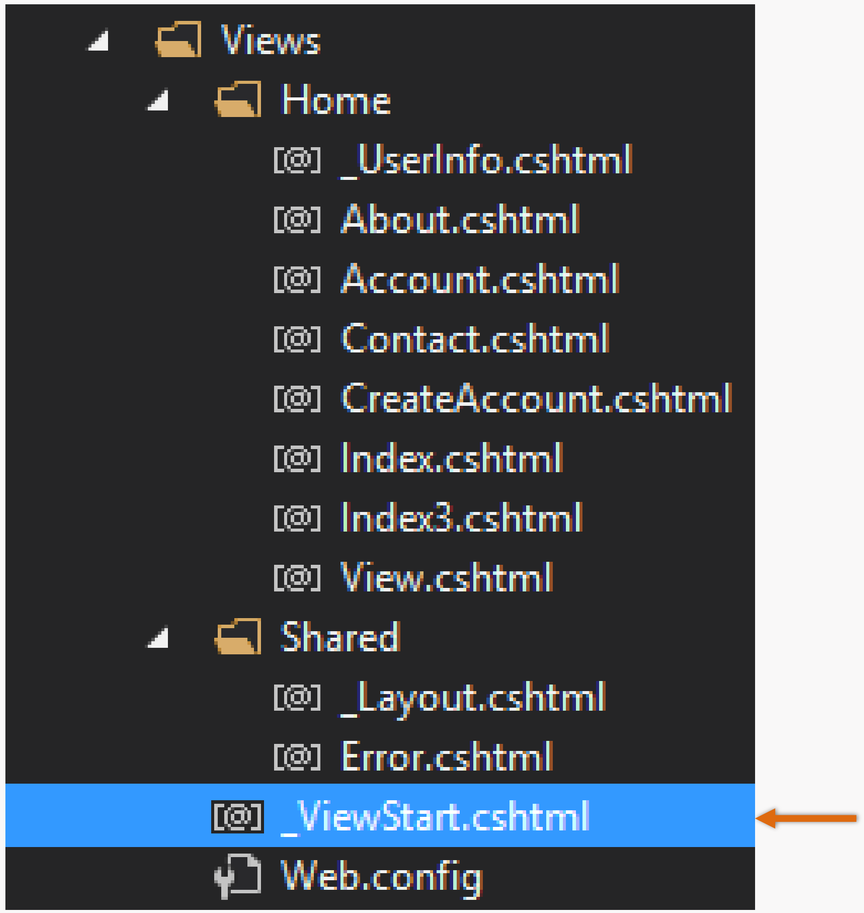


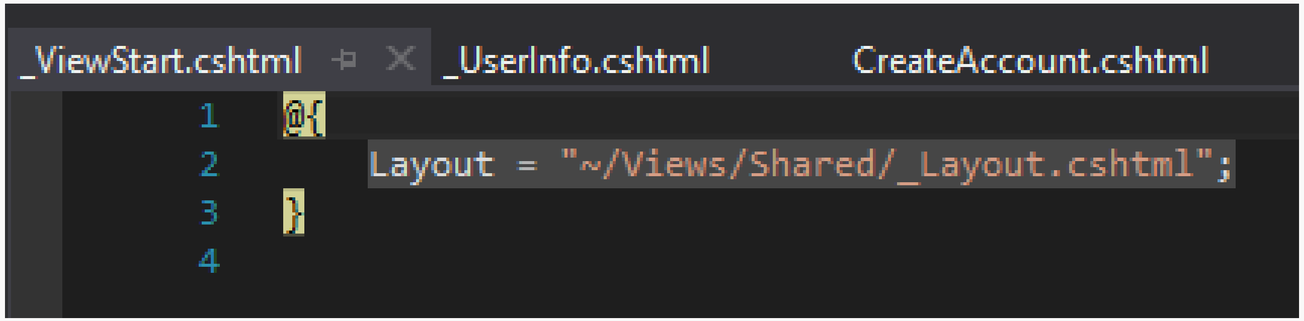


The layout is set to explicitly; if you want a view to have no layout set the Layout = null

Layouts

The default layout is set in the _ViewStart.cshtml file





HTML Helpers

- HTML Helpers are used to modify HTML output
- In most cases, an HTML helper is just a method that returns a string.
- With MVC, you can create your own helpers but it also includes standard helpers for the most common types of HTML elements, like HTML links and HTML form elements

Example: Rendering a link:

```
@Html.ActionLink("Link text", "ActionName", "ControllerName", new { @class = "redLink" })
```

▲ 7 of 10 ▼ (extension) MvcHtmlString HtmlHelper.ActionLink(string linkText, string actionName, **string controllerName**, object routeValues, object htmlAttributes)
Returns an anchor element (a element) for the specified link text, action, controller, route values, and HTML attributes.
controllerName: *The name of the controller.*

Generates the following HTML:

Link text

TIP! Use ***Ctrl+Shift+Space*** in Visual Studio to see information about a helper's parameters

HTML Helpers

- Built-In Form Html Helpers

BeginForm()	RadioButton()
EndForm()	ListBox()
TextArea()	DropDownList()
TextBox()	Hidden()
CheckBox()	Password()

- More about Built-In and custom HTML helpers:
- http://www.dotnet-tricks.com/Tutorial/mvc/N50P050314-Understanding-HTML-Helpers-in-ASP.NET-MVC.html
- http://www.c-sharpcorner.com/UploadFile/d98ae4/creating-custom-html-helpers-in-mvc5/

Homework

1. Create a new ASP.NET MVC 5 application with UserAuthentication
2. In the new app, add your entity (Post/Message) in the models folder
   - The Post class should have: Id (int), UserId(int), TimeOfPosting(DateTime), Message(string), PostType(Enum with 2 values: Text and Photo), IsSticky(bool), Priority (int – optional with values ranging from 1 to 5)
3. Create a Controller - PostsController with the following actions:
   1. Index Action (Create also a view for it with the List template)
   2. Details Action (Create also a view for it with the Details template)
   3. Create Action (With the [HttpGet] attribute, create also a view for it with the Create template)
   4. Create Action (With the [HttpPost] attribute)
      - The create POST action will receive the Post class as a parameter
      - If the post Message is null or empty return 404
      - If the post Message is not null or empty return the Details view with the post object as a parameter

Pentalog

Thank you!

DATA ANALYSIS
UX/UI
FRONT-END
TECHNOLOGY
NEAR/OFFSHORE
AGILITY
BACK-END
SPRINT
KANBAN
CAMPAIGNS
GROWTH HACKING
SCRUM
BACKLOG
DEVOPS
DESIGN
SEO
CONTINUOUS INTEGRATION
MOBILE
QA
AUTOMATION
RESPONSIVE
UNIT TESTING