Pentalog

Aplicatii web ASP.NET MVC - Part 2

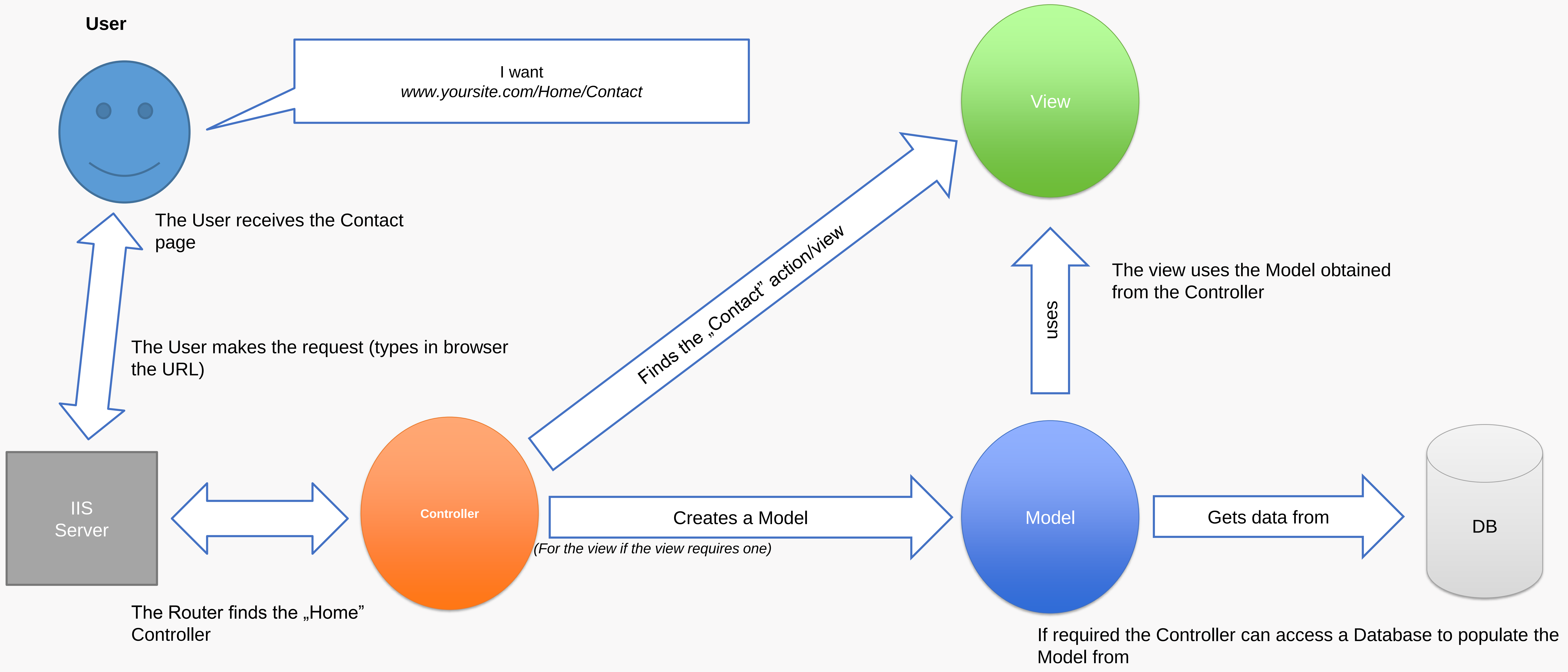
PentaStagiu Remote Brașov

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

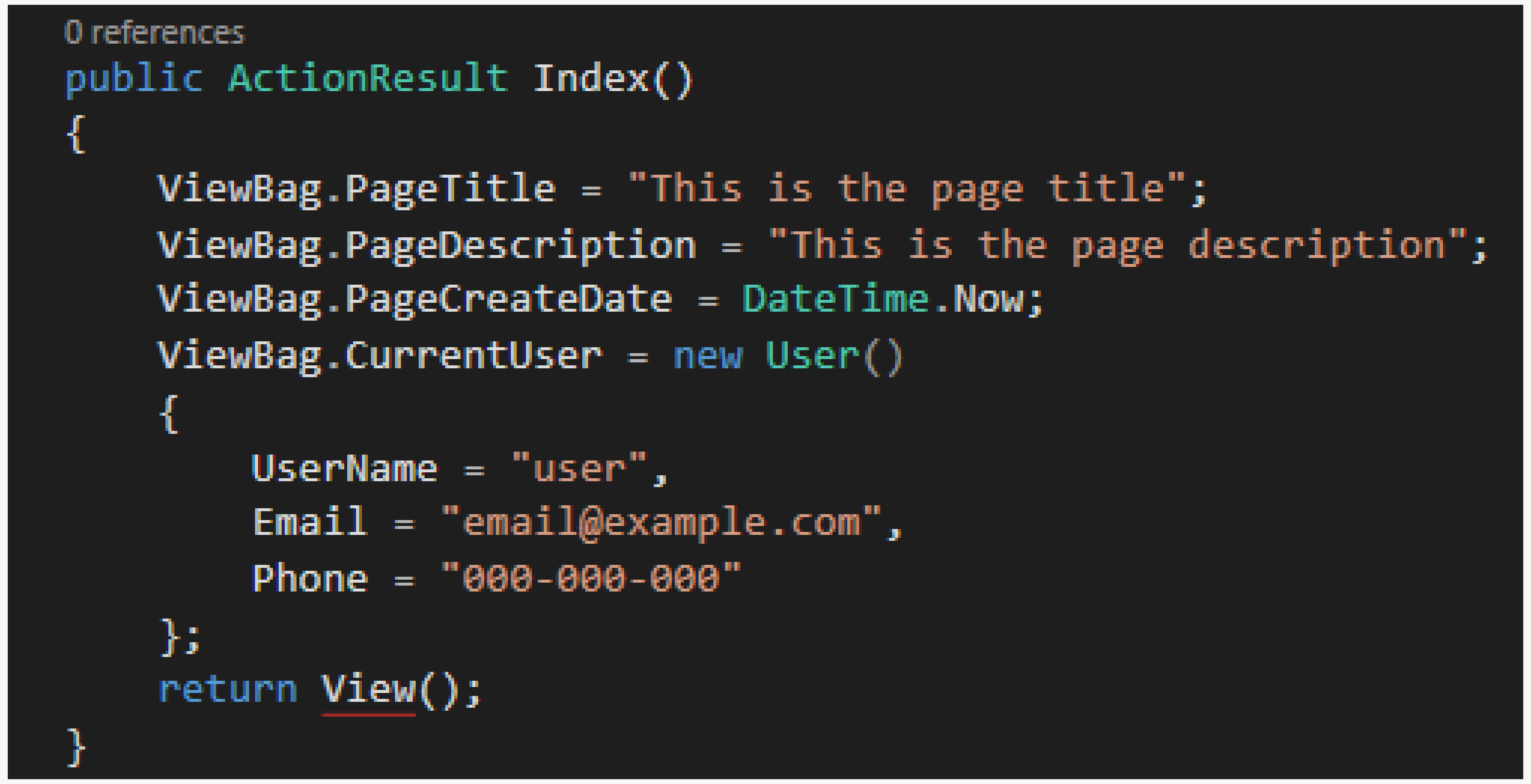
Let’s recap MVC

MVC Recap

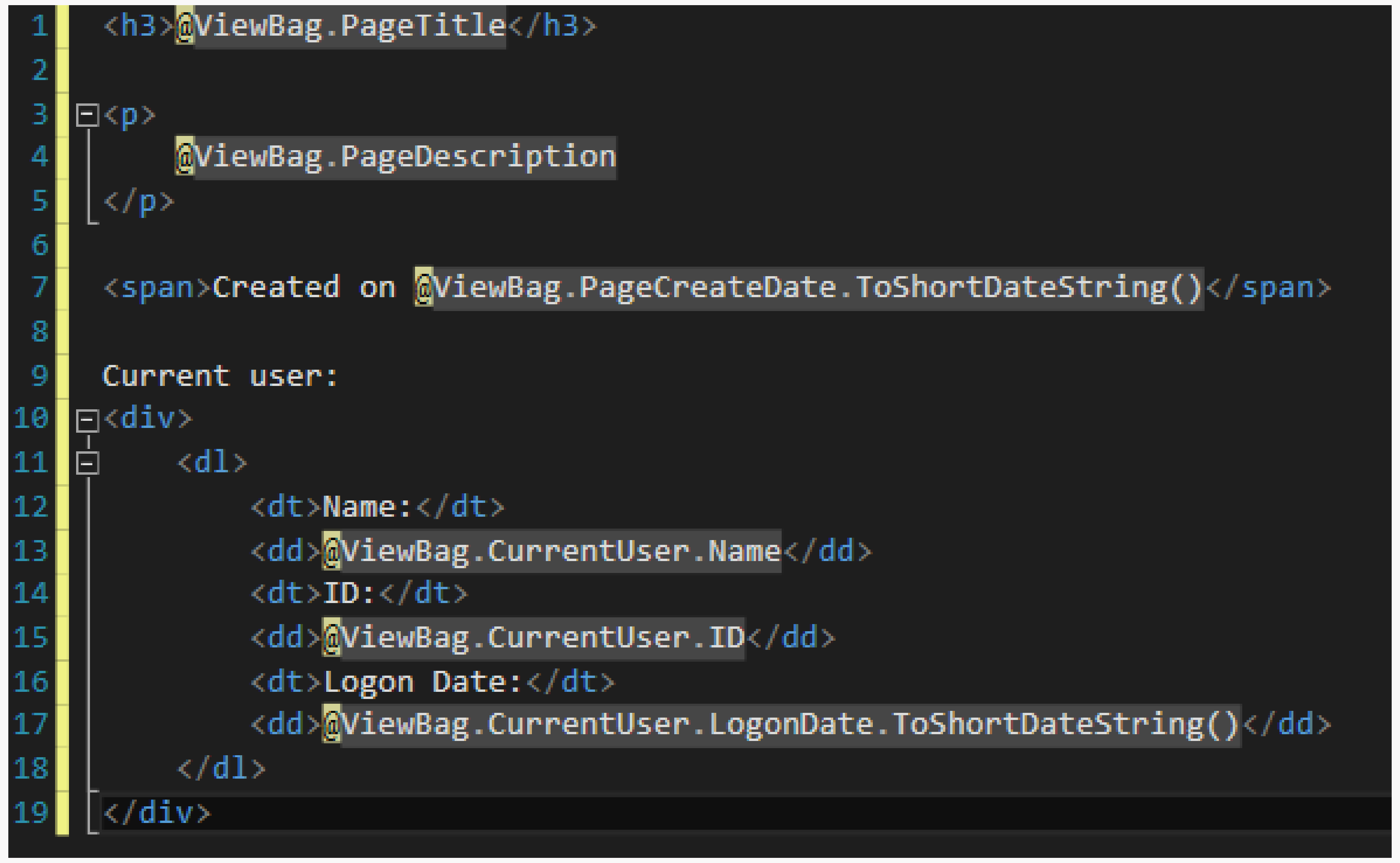

User
I want
www.yoursite.com/Home/Contact
The User receives the Contact page
The User makes the request (types in browser the URL)
IIS Server
The Router finds the „Home” Controller
Controller
Finds the „Contact” action/view
Creates a Model
(For the view if the view requires one)
View
uses
The view uses the Model obtained from the Controller
Model
Gets data from
DB
If required the Controller can access a Database to populate the Model from

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();
}
```

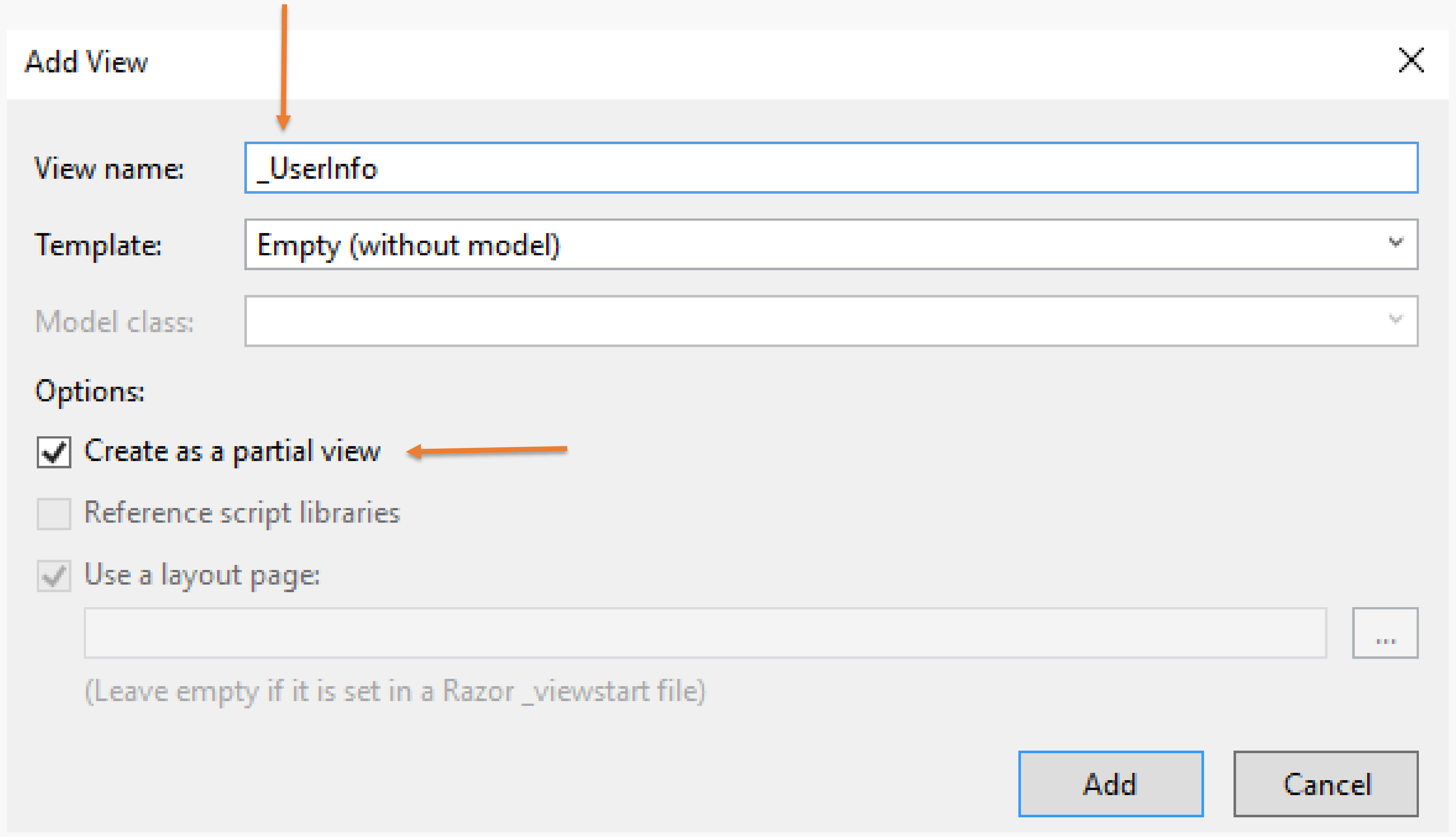


```
<h3>@ViewBag.PageTitle</h3>

<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 starts with "_"
- Have no layout



Add View

View name: _UserInfo

Template: Empty (without model)

Model class:

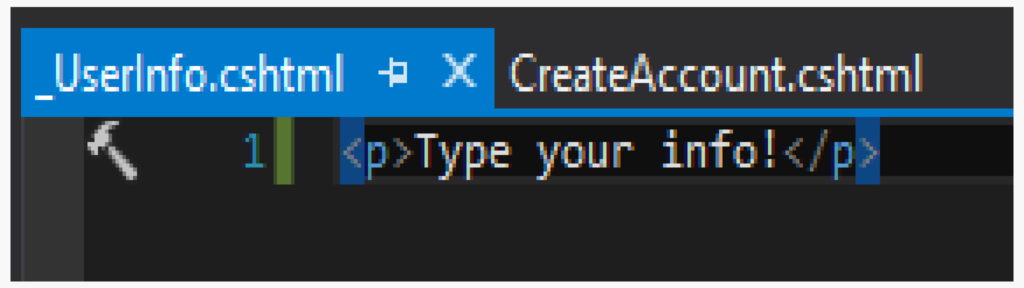
Options:

☑ Create as a partial view

☐ Reference script libraries

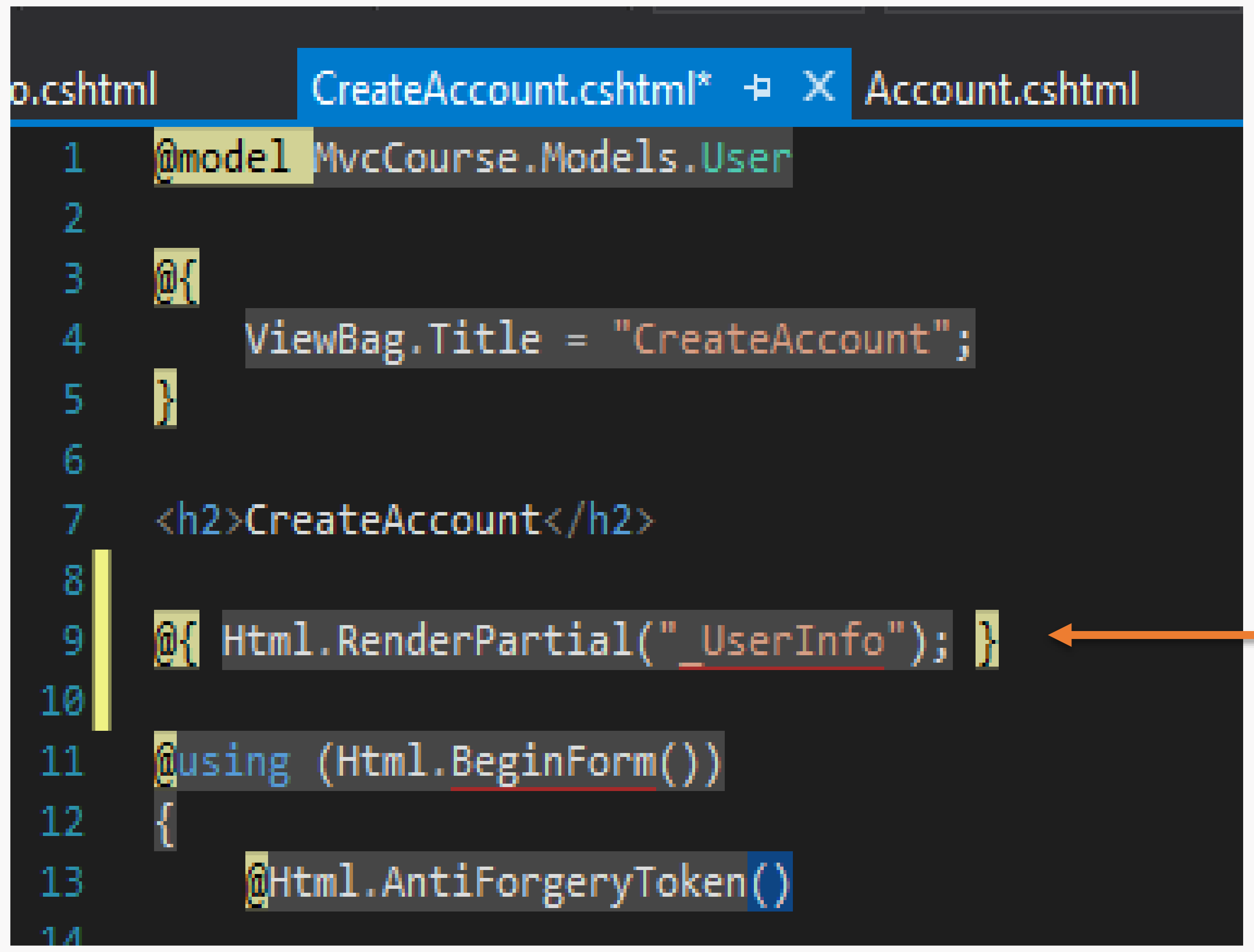
☑ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)



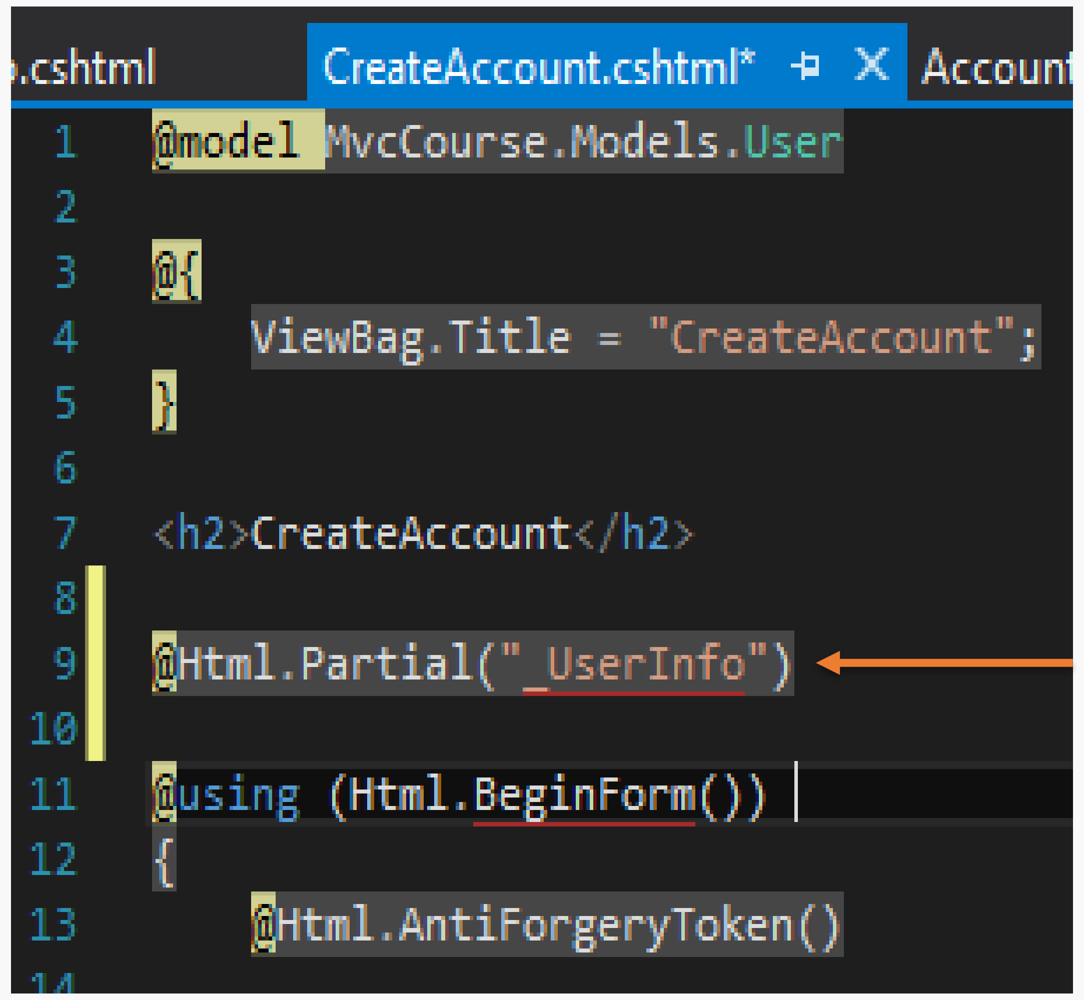
```
<p>Type your info!</p>
```

To use a partial view:



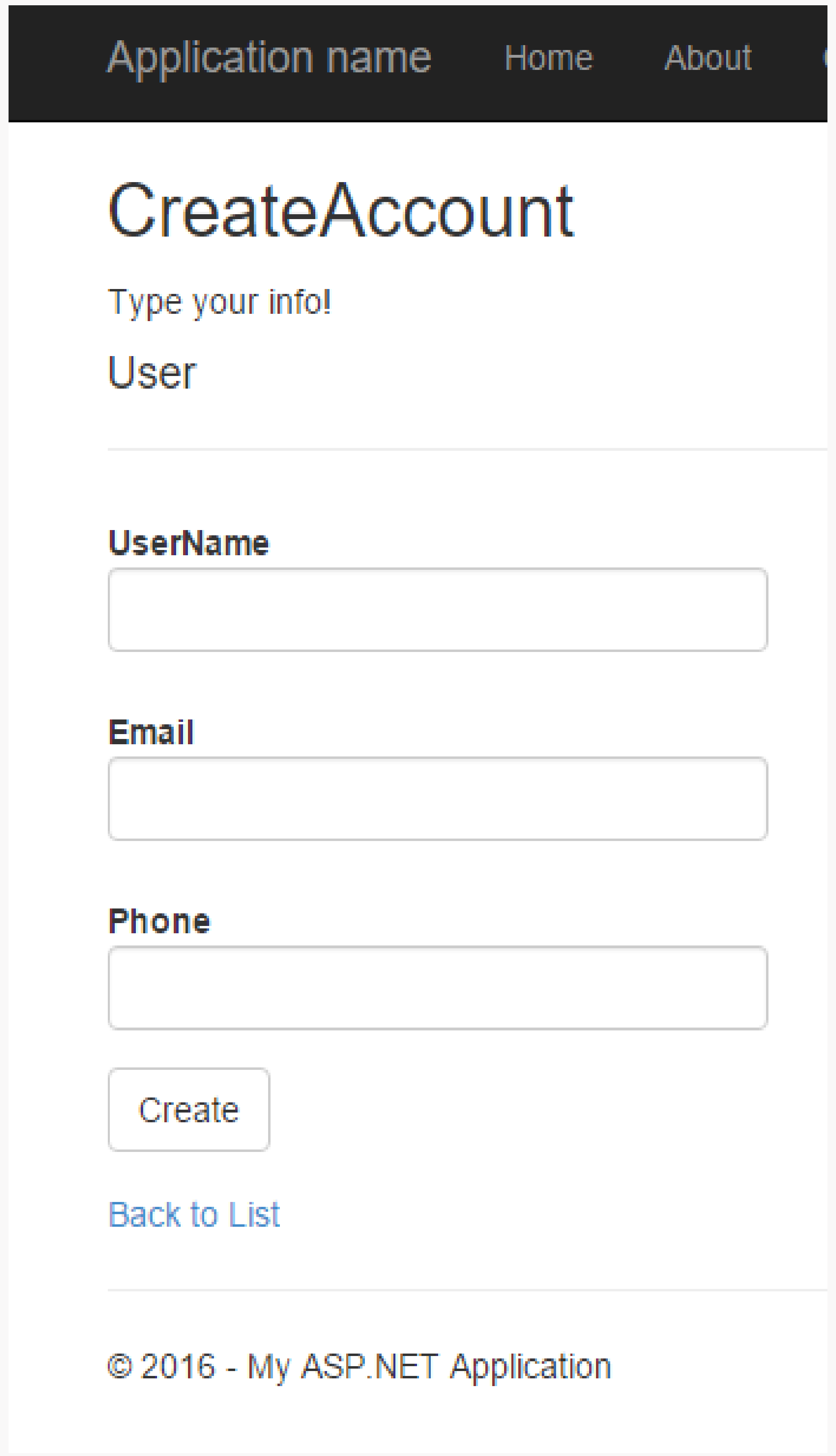
```
@model MvcCourse.Models.User

@{
    ViewBag.Title = "CreateAccount";
}

<h2>CreateAccount</h2>

@{ Html.RenderPartial("_UserInfo"); }

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

or



```
@model MvcCourse.Models.User

@{
    ViewBag.Title = "CreateAccount";
}

<h2>CreateAccount</h2>

@Html.Partial("_UserInfo")

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

`Html.Partial` is easier to use



Application name Home About

CreateAccount

Type your info!

User

UserName

Email

Phone

Create

Back to List

© 2016 - My ASP.NET Application

You can also pass a model to partial views

```
@Html.Partial("_UserInfo", Model)
```

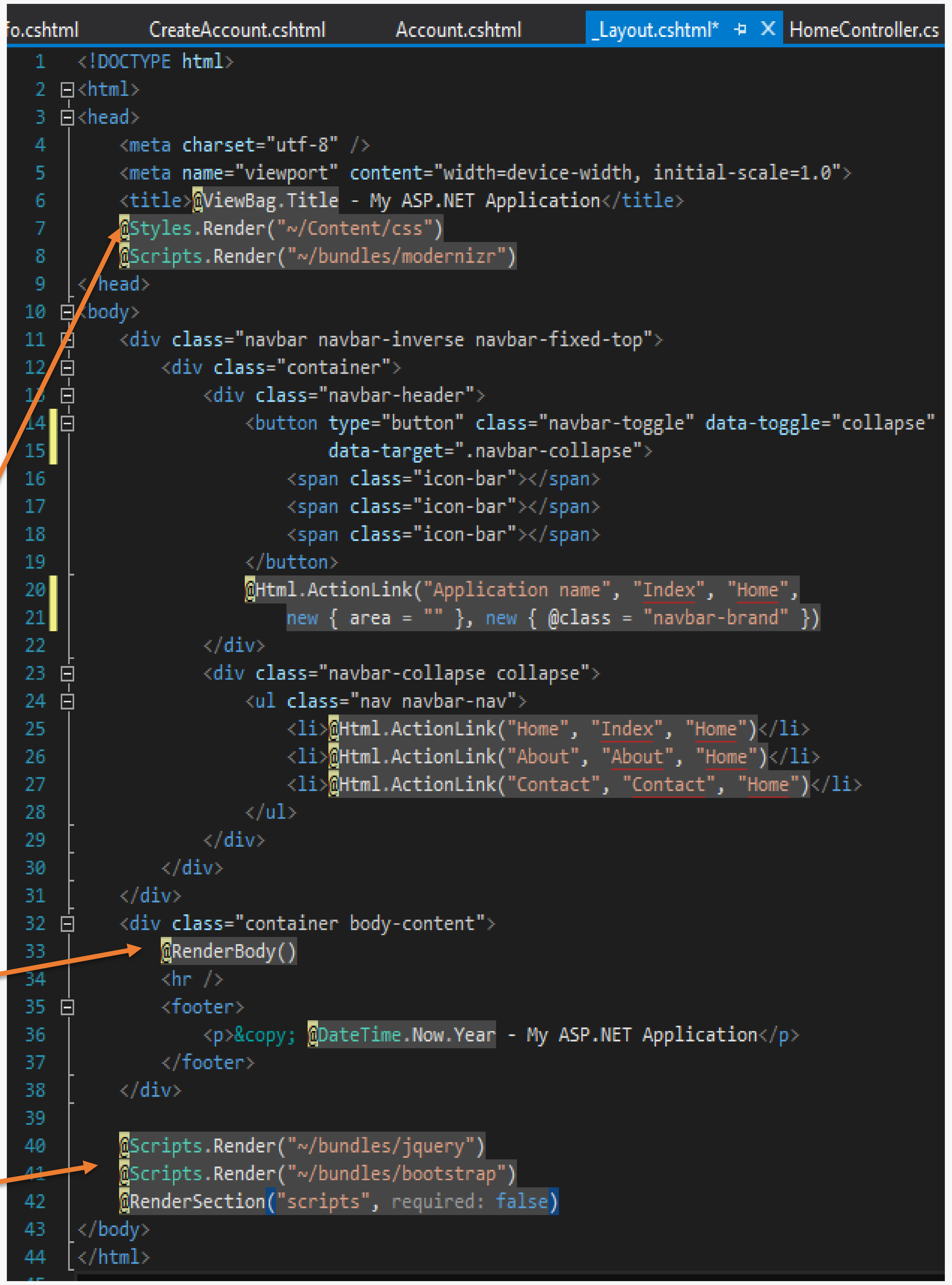
Layouts

- 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

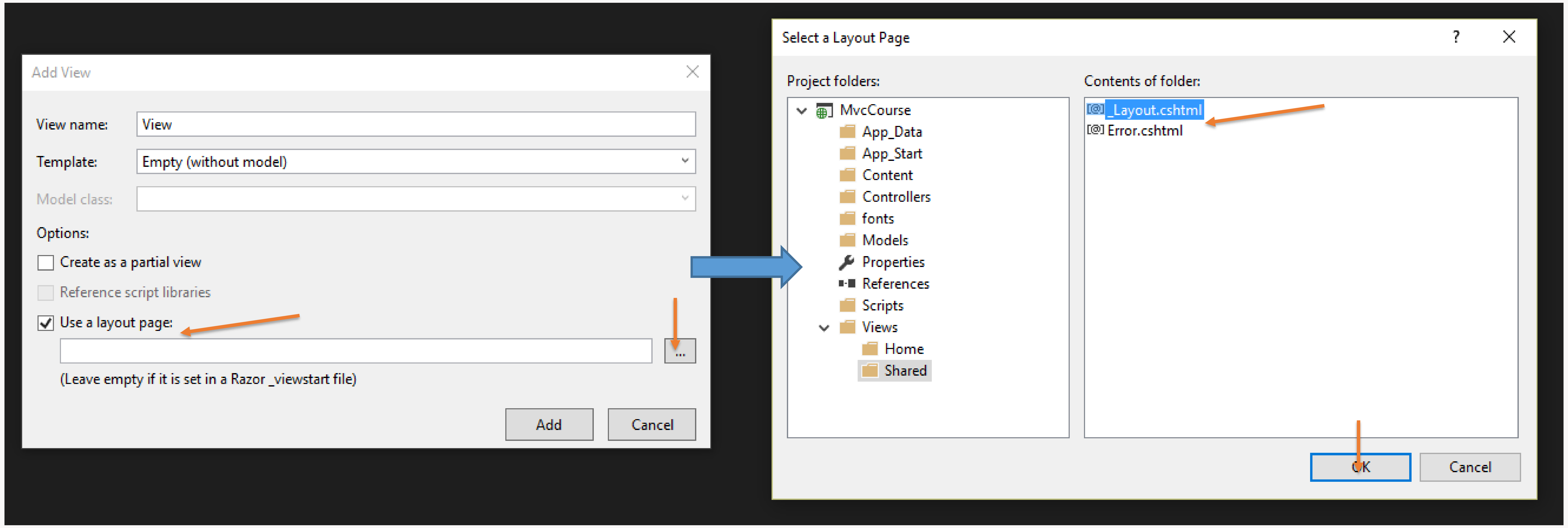
This is where the content of each view that uses this layout is rendered

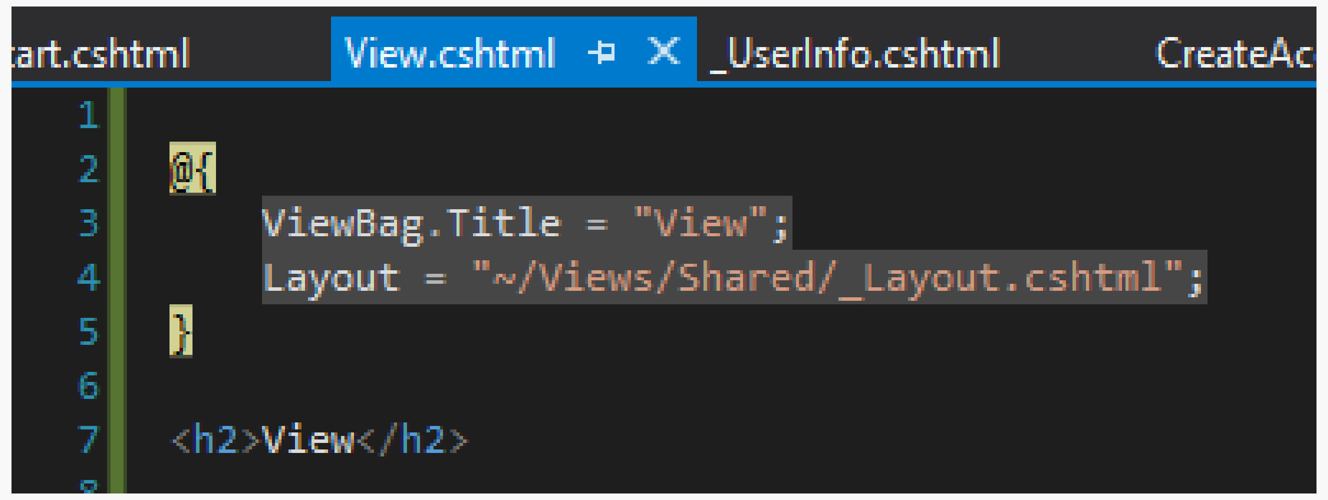
This is where the JavaScript files are added



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse"
                        data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home",
                    new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

When creating a view you have the possibility to overwrite the default layout.

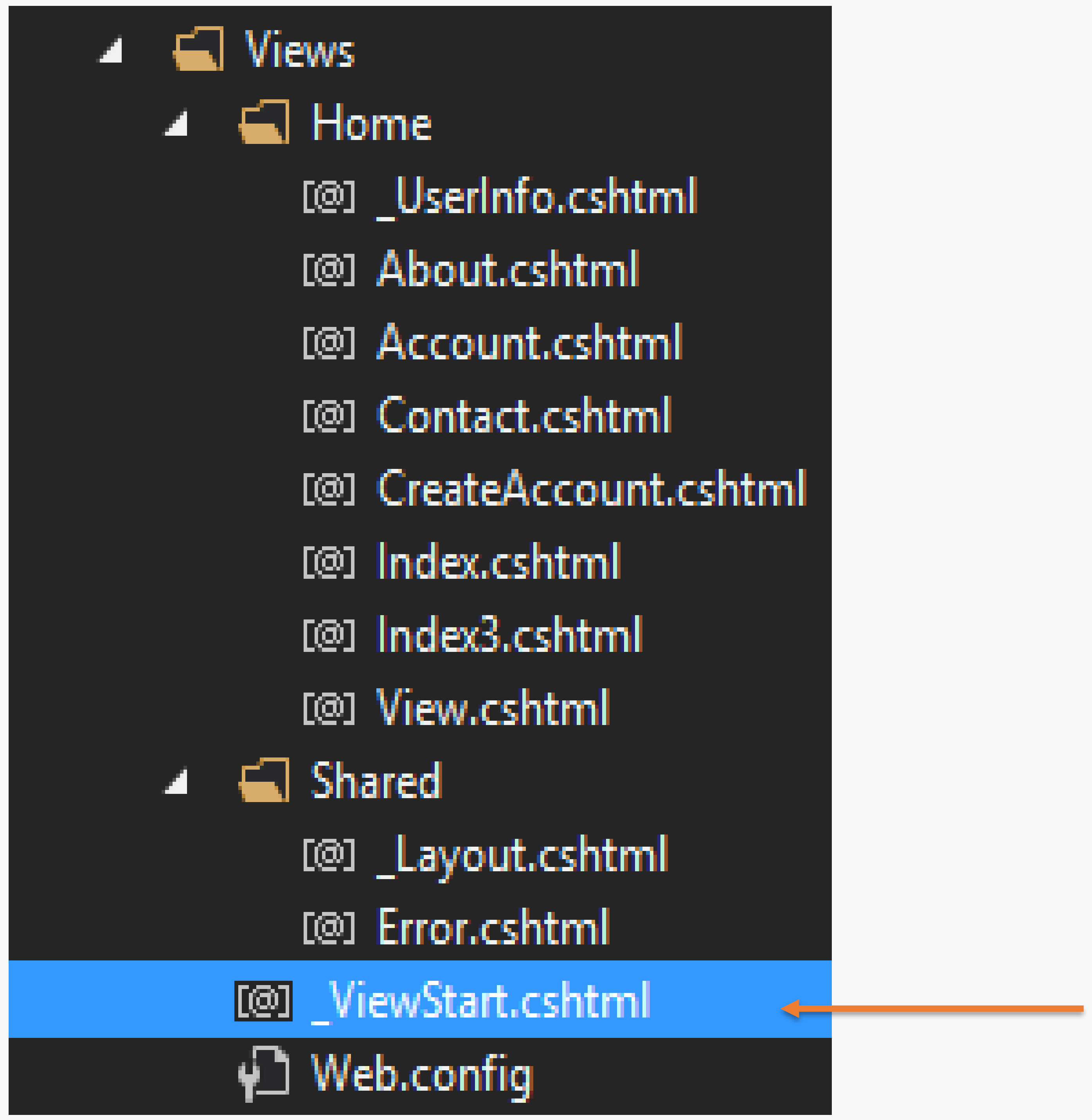


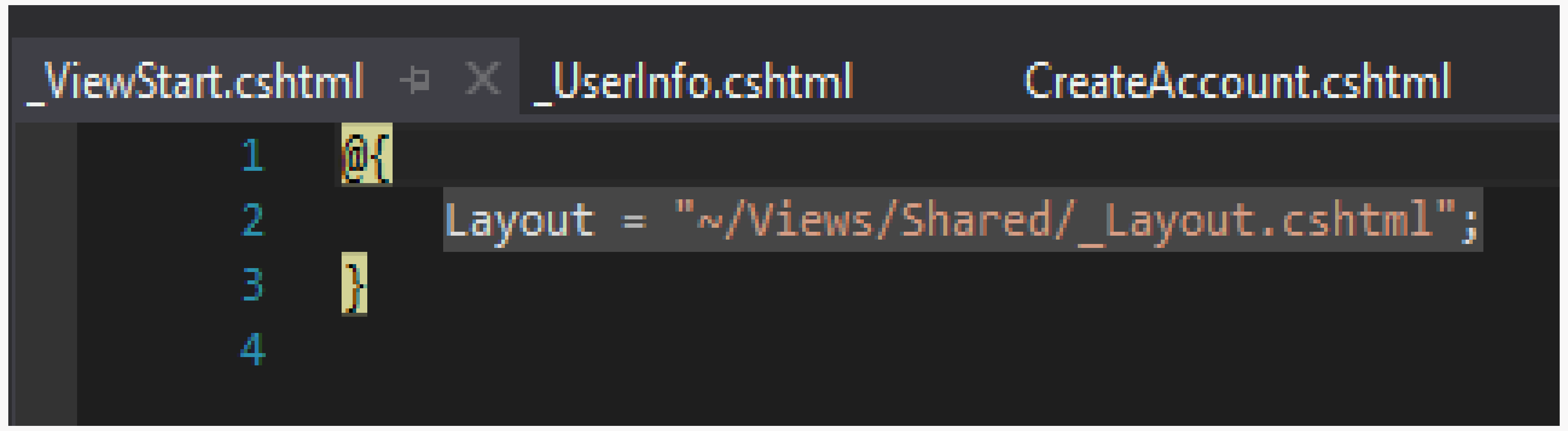


```
@{
    ViewBag.Title = "View";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>View</h2>
```

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

The default layout is set in the _ViewStart.cshtml file



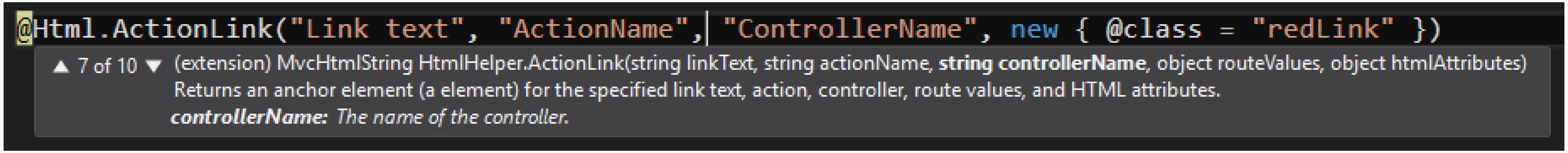


```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

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:
<a href="/ControllerName/ActionName" class="redLink">Link text</a>

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

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/

EF & DataBases Teaser + Explaining Identity

Authentication

Authentication is the process of recognizing a user’s identity. It is the mechanism of associating an incoming request with a set of identifying credentials.

Factors / Classes:

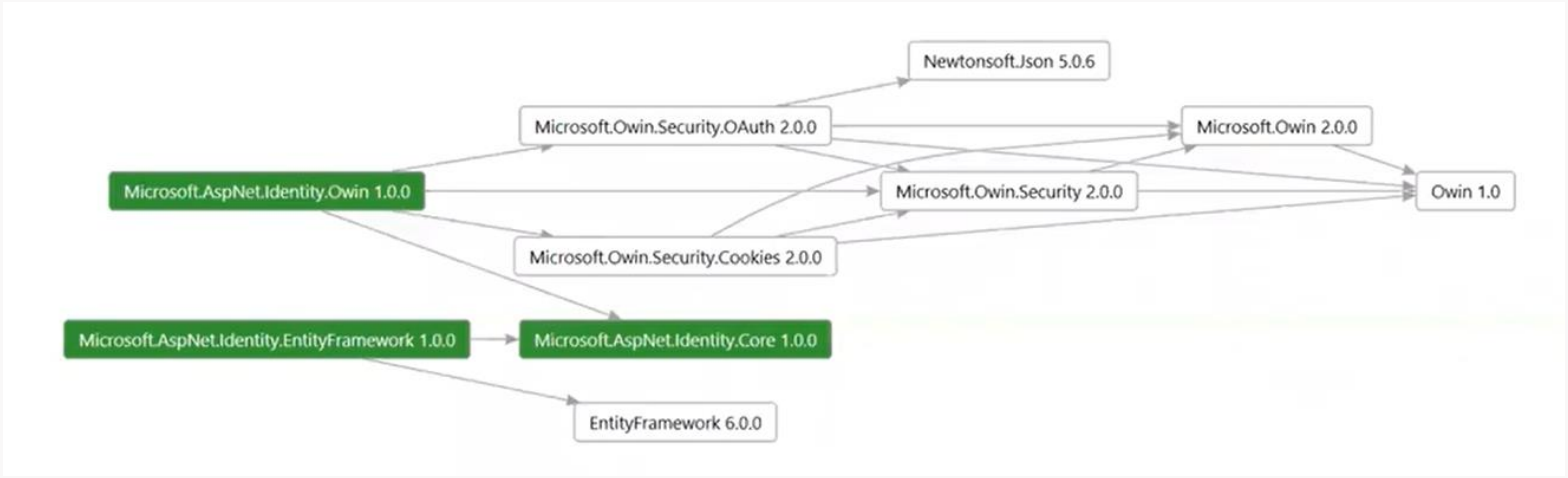
- the **knowledge factors:** Something the user **knows**
- The **ownership factors:** Something the user **has**
- The **inherence factors:** Something the user **is** or **does**

- Single-factor authentication
- Two-factor authentication
- Multi-factor authentication
- Strong authentication
- Continuous Authentication

ASP.NET Identity

- A modern membership system which enables redirection-based log-ins to authentication providers such as Facebook, Twitter, and others
- One ASP.NET Identity system
- Ease of plugging in profice data about the user
- Persistence control
- Unit testability
- Role provider
- Claims Based
- Social Login Providers
- OWIN Integration
- NuGet package

ASP.NET Identity Components


Newtonsoft.Json 5.0.6
Microsoft.Owin.Security.OAuth 2.0.0
Microsoft.Owin 2.0.0
Microsoft.AspNet.Identity.Owin 1.0.0
Microsoft.Owin.Security 2.0.0
Owin 1.0
Microsoft.Owin.Security.Cookies 2.0.0
Microsoft.AspNet.Identity.EntityFramework 1.0.0
Microsoft.AspNet.Identity.Core 1.0.0
EntityFramework 6.0.0

ASP.NET Identity Components

- **Microsoft.AspNet.Identity.EntityFramework**

This package has the Entity Framework implementation of ASP.NET Identity which will persist in the ASP.NET Identity data and schema to SQL Server.

- **Microsoft.AspNet.Identity.Core**

This package has the core interfaces for ASP.NET Identity. This package can be used to write an implementation for ASP.NET Identity that targets different persistence stores such as Azure Table Storage, NoSql databases, etc.

- **Microsoft.AspNet.Identity.OWIN**

This package contains functionality that is used to plug in OWIN authentication with ASP.NET Identity in ASP.NET application. This is used when you add sign in functionality to your application and call into OWIN Cookie Authentication middleware to generate a cookie.

ASP.NET Identity Components

- **Microsoft.Owin.Security.Cookies**

Middleware that enables an application to use cookie based authentication, similar to ASP.NET’s Forms Authentication.

- **EntityFramework**

Entity Framework is Microsoft’s recommended data access technology for relational databases.

More details about ASP.NET Identity:
https://docs.microsoft.com/en-us/aspnet/identity/

Authorization

- Authorization refers to the process that determines what a user is able to do.

- For example, an administrative user is allowed to create a document library, add documents, edit documents, and delete them. A non-administrative user working with the library is only authorized to read the documents.

- Authorization is orthogonal and independent from authentication.

However, authorization requires n authentication mechanism.

More details about Authorization in ASP.NET:
https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction?view=aspnetcore-3.1

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
    1. 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
    5. Edit Action (Both the [HttpGet] and [HttpPost] actions – you can add other field validations if you want)
    6. Delete Action (can be done directly with [HttpGet])
4. Use the ViewBag to send the current DateTime from the controller to the Post Index view
5. Create a partial view containing the instructions for creating a post (a text explaining that the post must be no longer than 255 characters, etc) and include it in the Create and in the Edit views.
6. Modify the layout page: in the menubar add an URL to the Index action from your controller. Also in the layout add your name in the footer section of the application.

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