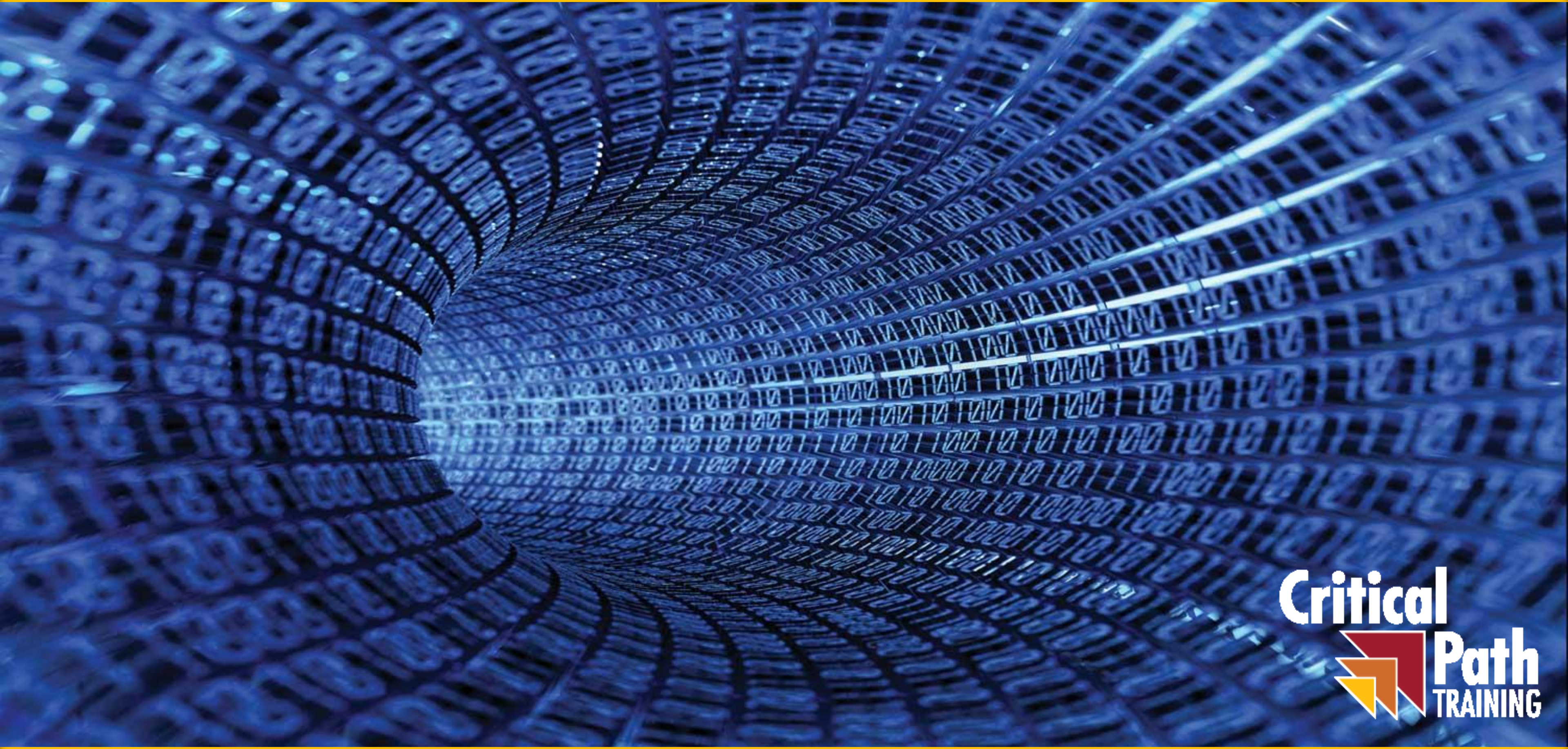
Developing Web Apps using MVC



Agenda

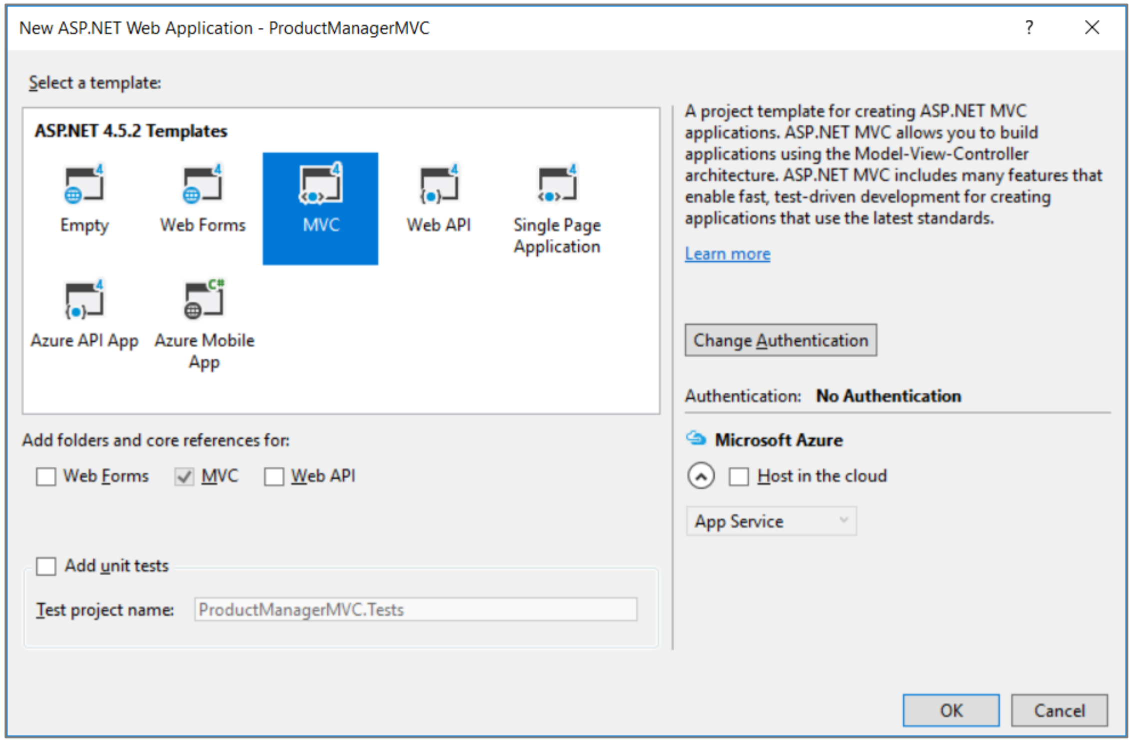
- ASP.NET MVC Fundamentals
- Controllers, Views and Routing
- Designing Strongly-typed Models
- Adding MVC Forms and Validation
- Developing Asynchronous Controller

Web Forms Versus ASP.NET MVC

- ASP.NET provides two different platforms
  - ASP.NET Web Forms (e.g. ASPX files)
  - ASP.NET MVC
- MVC provides better platform for the web
  - More flexible routing
  - Lighter-weight
  - Richer templating
  - Better C# integration
  - Unit testing

ASP.NET MVC Fundamentals

- MVC is a framework provided by ASP.NET
  - Based on Model-View-Controller paradigm
  - Provides alternative to original Web Forms framework


New ASP.NET Web Application - ProductManagerMVC

Select a template:

ASP.NET 4.5.2 Templates

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Authentication: **No Authentication**

Microsoft Azure

☐ Host in the cloud

App Service

Add folders and core references for:

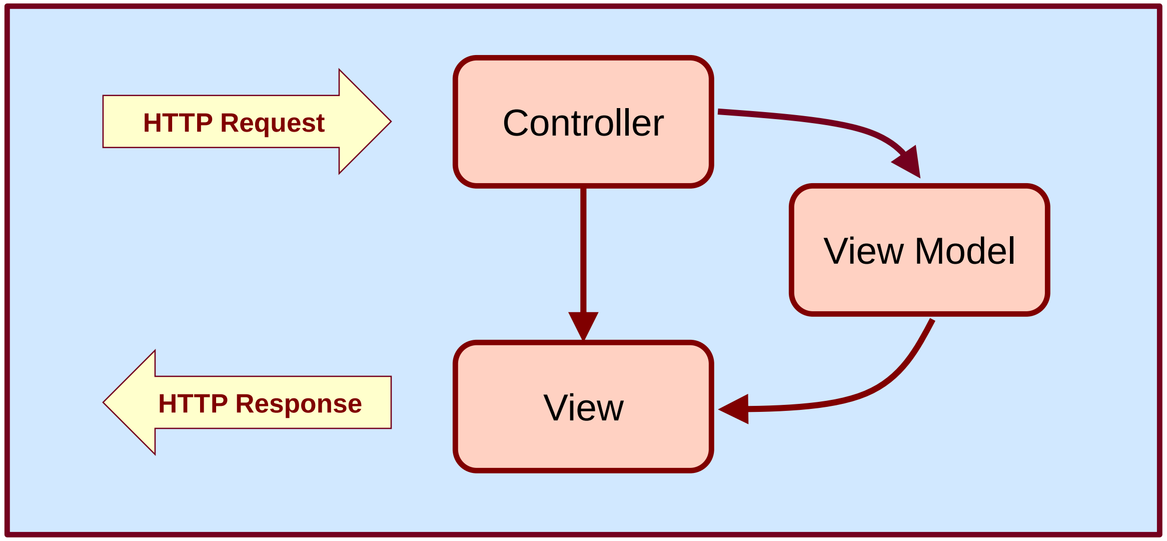
☐ Web Forms ☑ MVC ☐ Web API

☐ Add unit tests

Test project name: ProductManagerMVC.Tests

MVC Request Processing

- How does MVC handle an incoming HTTP request
  - Incoming HTTP request routed to specific Controller
  - Controller creates view model (i.e. presentation object)
  - Controller passes view model to view
  - View renders response as HTTP Response



Agenda

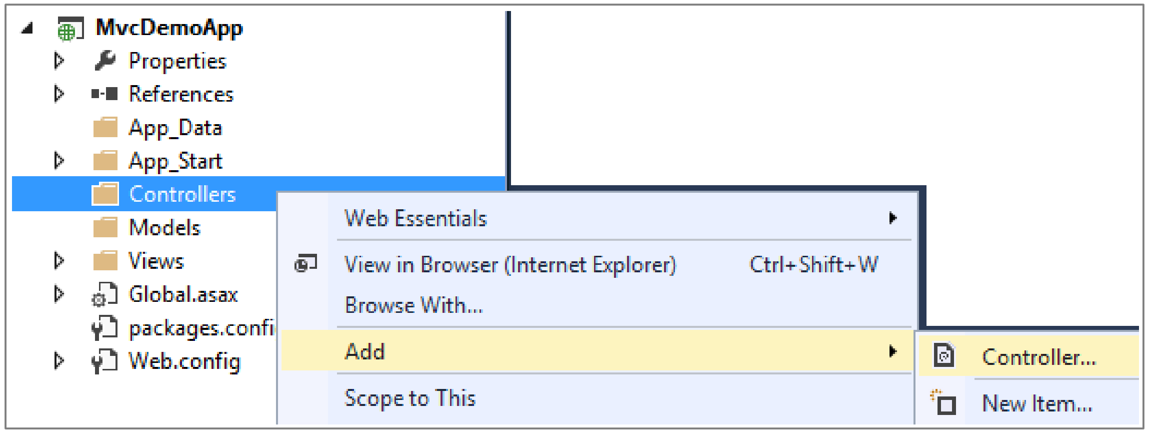
- ✓ ASP.NET MVC Fundamentals
- ➢ Controllers, Views and Routing
- ▪ Designing Strongly-typed Models
- ▪ Adding MVC Forms and Validation
- ▪ Developing Asynchronous Controller

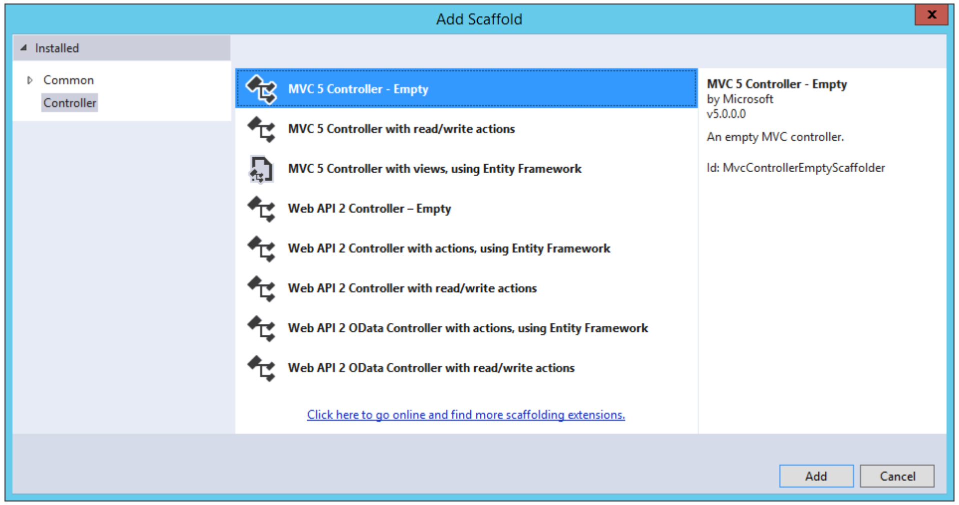
Understanding Controllers

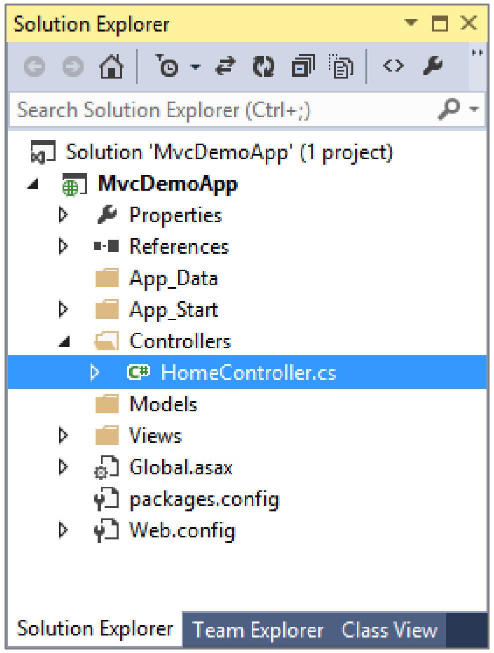
- A set of classes that manage…
  - processing incoming HTTP requests
  - communication to and from user
  - overall application flow and application-specific logic

- Every controller has one or more Actions
  - It’s critical to understand the role of Actions in MVC

Adding a Controller

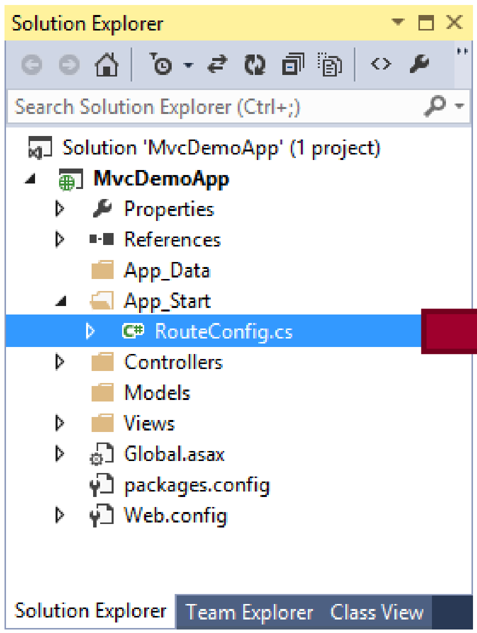



Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData Controller with actions, using Entity Framework
Web API 2 OData Controller with read/write actions
Click here to go online and find more scaffolding extensions.
MVC 5 Controller - Empty
by Microsoft
v5.0.0.0
An empty MVC controller.
Id: MvcControllerEmptyScaffolder
Add
Cancel



Wiring Up a Controller

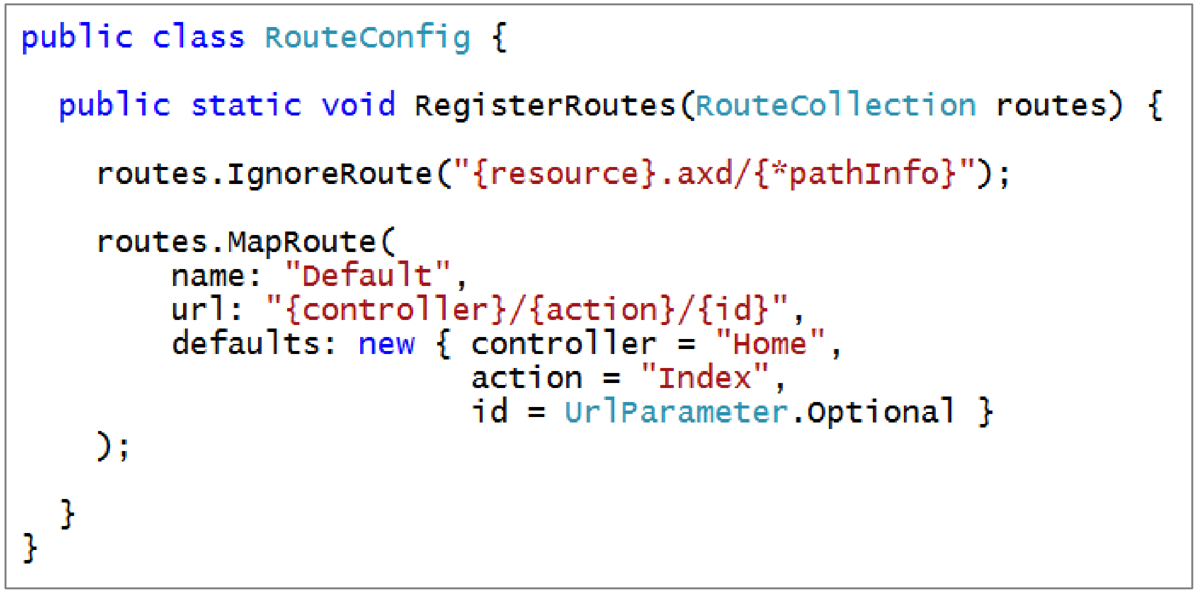
- When you ad a controller…
  - Visual Studio updates **RouteConfig** class
  - Routing scheme defined using standard format
    - **{controller}/{action}/{id}**



Solution Explorer

Search Solution Explorer (Ctrl+;)

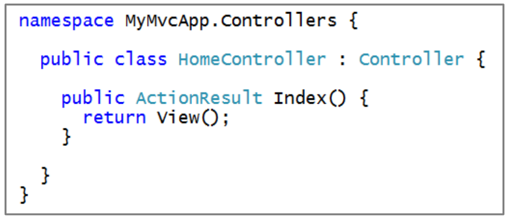
```
Solution 'MvcDemoApp' (1 project)
  MvcDemoApp
    Properties
    References
    App_Data
    App_Start
      RouteConfig.cs
    Controllers
    Models
    Views
    Global.asax
    packages.config
    Web.config
```



```
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 }
    );

  }
}
```

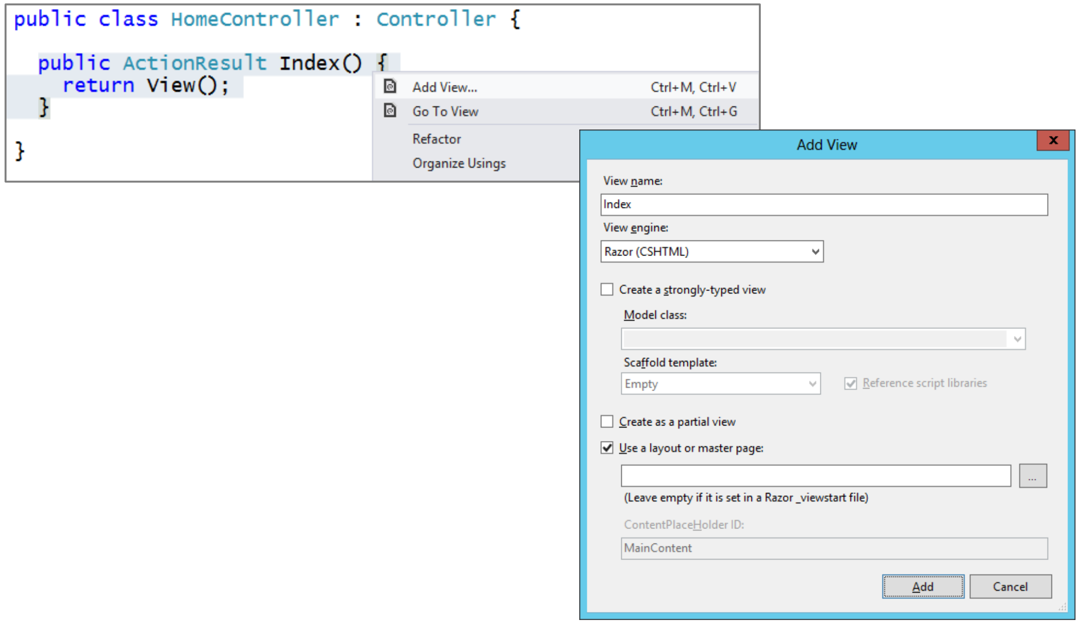
Creating a View from a Controller Action

- Controller method often return **ActionResult**



```
namespace MyMvcApp.Controllers {
  public class HomeController : Controller {
    public ActionResult Index() {
      return View();
    }
  }
}
```

- Right-click on Controller method to generate its view



```
public class HomeController : Controller {
  public ActionResult Index() {
    return View();
  }
}
```

Add View

View name: Index

View engine: Razor (CSHTML)

☐ Create a strongly-typed view

Model class:

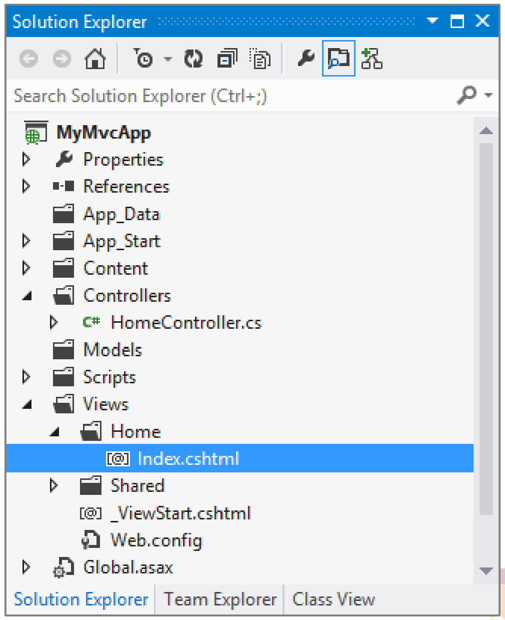
Scaffold template: Empty ☑ Reference script libraries

☐ Create as a partial view

☑ Use a layout or master page:

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

ContentPlaceHolder ID: MainContent



Solution Explorer

Search Solution Explorer (Ctrl+;)

```
MyMvcApp
  Properties
  References
  App_Data
  App_Start
  Content
  Controllers
    HomeController.cs
  Models
  Scripts
  Views
    Home
      Index.cshtml
    Shared
    _ViewStart.cshtml
    Web.config
  Global.asax
```

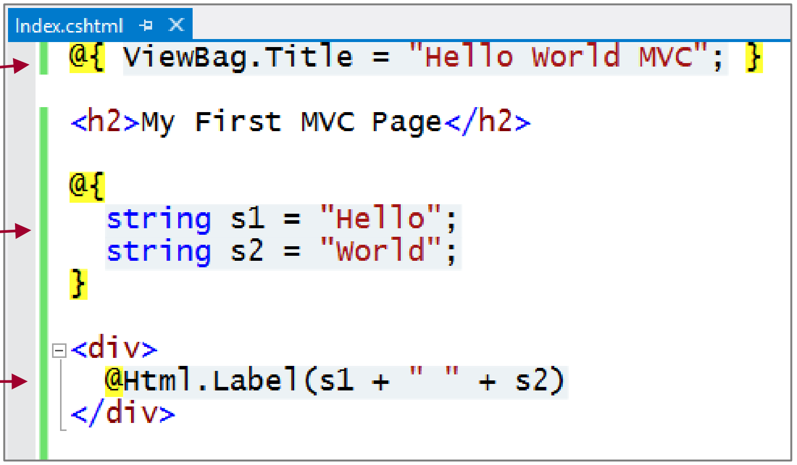
Customizing the View

- Views are creating using the Razor engine
  - Provides a lean and elegant way to create HTML pages

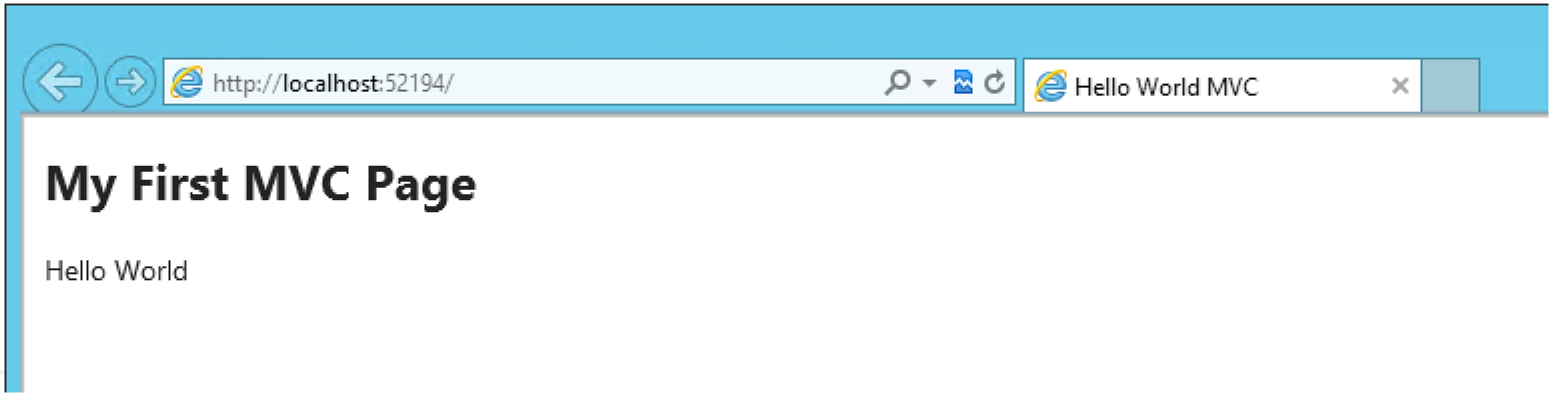
Use data passed from controller

Write in-line C# code

Construct pages using HTML Helpers



```
@{ ViewBag.Title = "Hello World MVC"; }

<h2>My First MVC Page</h2>

@{
    string s1 = "Hello";
    string s2 = "World";
}

<div>
    @Html.Label(s1 + " " + s2)
</div>
```

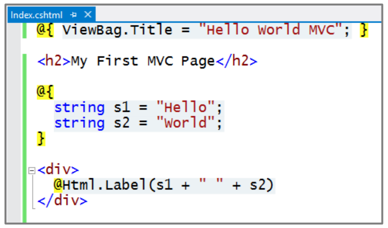


My First MVC Page

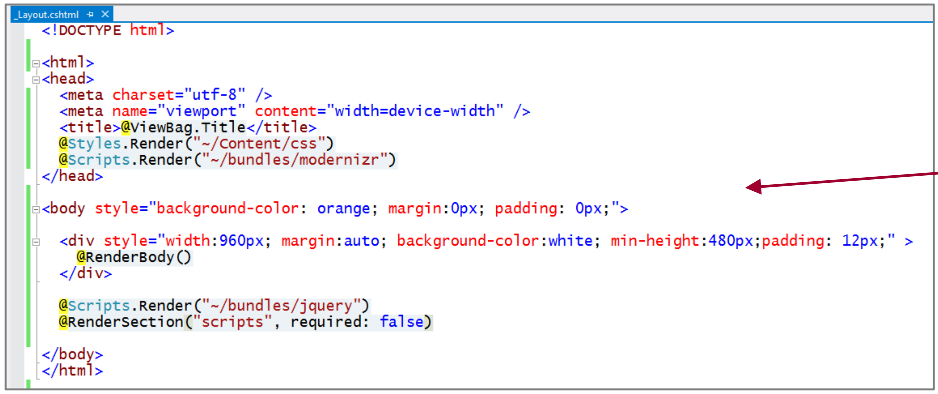
Hello World

Customizing the Shared View

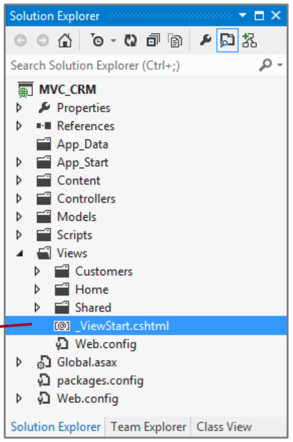
- MVC provides Shared Views
  - Provides same purpose as master pages in ASP.NET web forms
  - Default MVC shared view is named **_ViewStart.cshtml**


Index.cshtml

```
@{ ViewBag.Title = "Hello World MVC"; }

<h2>My First MVC Page</h2>

@{
    string s1 = "Hello";
    string s2 = "World";
}

<div>
    @Html.Label(s1 + " " + s2)
</div>
```


_Layout.cshtml

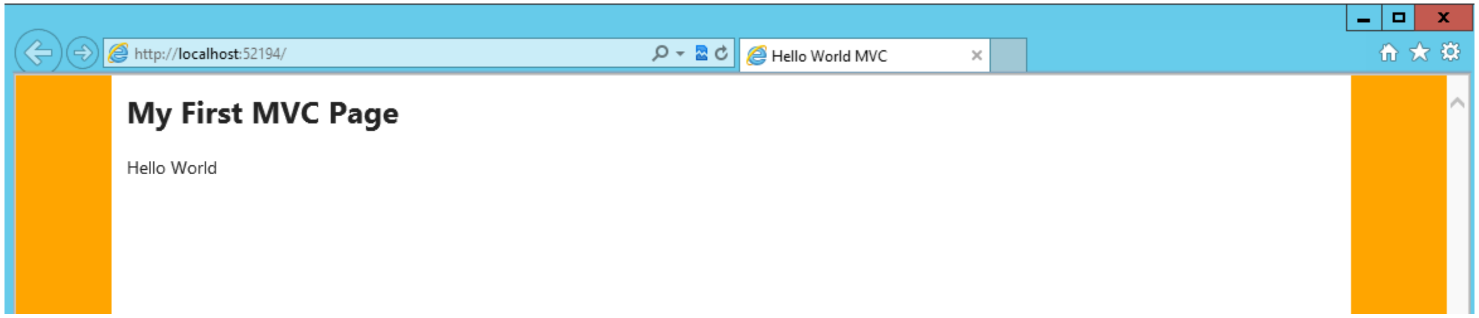
```
<!DOCTYPE html>

<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>

<body style="background-color: orange; margin:0px; padding: 0px;">

    <div style="width:960px; margin:auto; background-color:white; min-height:480px;padding: 12px;" >
      @RenderBody()
    </div>

    @Scripts.Render("~/bundles/jquery")
    @RenderSection("scripts", required: false)

</body>
</html>
```


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
MVC_CRM
  Properties
  References
  App_Data
  App_Start
  Content
  Controllers
  Models
  Scripts
  Views
    Customers
    Home
    Shared
    _ViewStart.cshtml
    Web.config
  Global.asax
  packages.config
  Web.config
```

Solution Explorer | Team Explorer | Class View


http://localhost:52194/

Hello World MVC

My First MVC Page

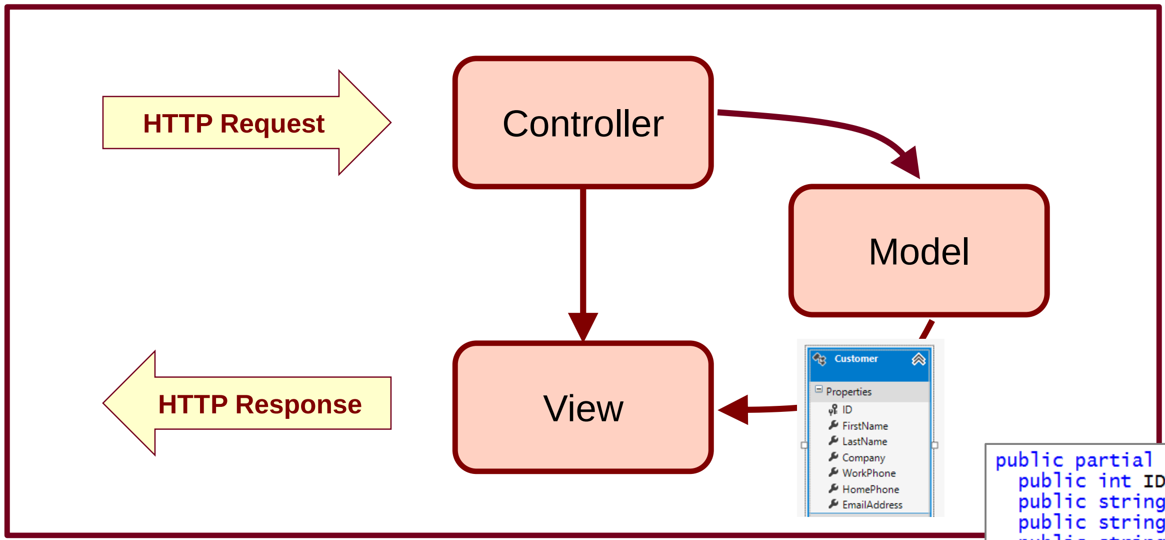
Hello World

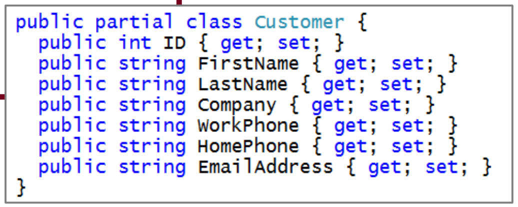
Agenda

- ✓ ASP.NET MVC Fundamentals
- ✓ Controllers, Views and Routing
- ➢ Designing Strongly-typed Models
- ▪ Adding MVC Forms and Validation
- ▪ Developing Asynchronous Controller

Motivation for Strongly-typed Models

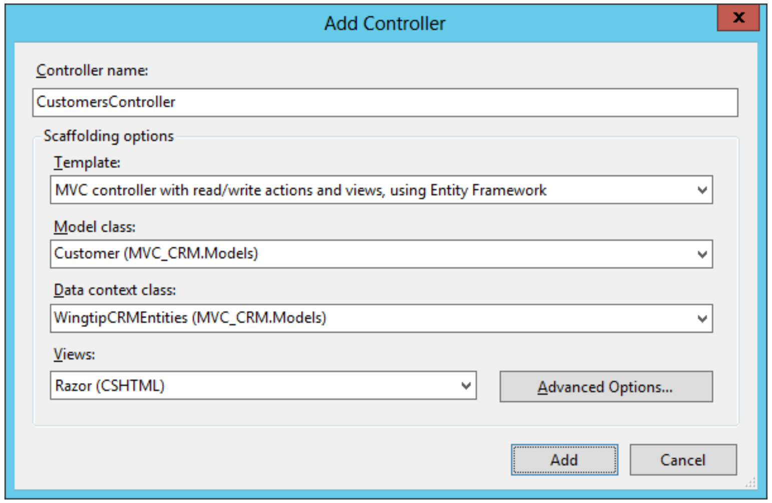
- MVC designed based on strongly-typed models
  - Controller creates model object and passes it to view
  - Razor view engine supplies IntelliSense for model behind view
  - HTML helpers make it easy to create views and forms





```
public partial class Customer {
  public int ID { get; set; }
  public string FirstName { get; set; }
  public string LastName { get; set; }
  public string Company { get; set; }
  public string WorkPhone { get; set; }
  public string HomePhone { get; set; }
  public string EmailAddress { get; set; }
}
```

Creating a Strongly-typed Controller Class


Add Controller

Controller name: CustomersController

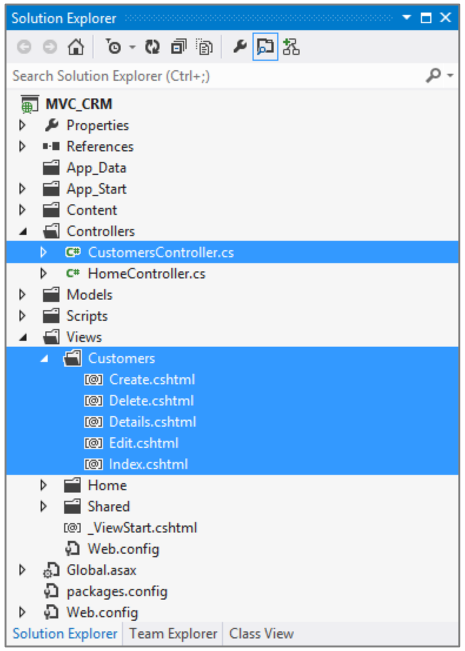
Scaffolding options

Template: MVC controller with read/write actions and views, using Entity Framework

Model class: Customer (MVC_CRM.Models)

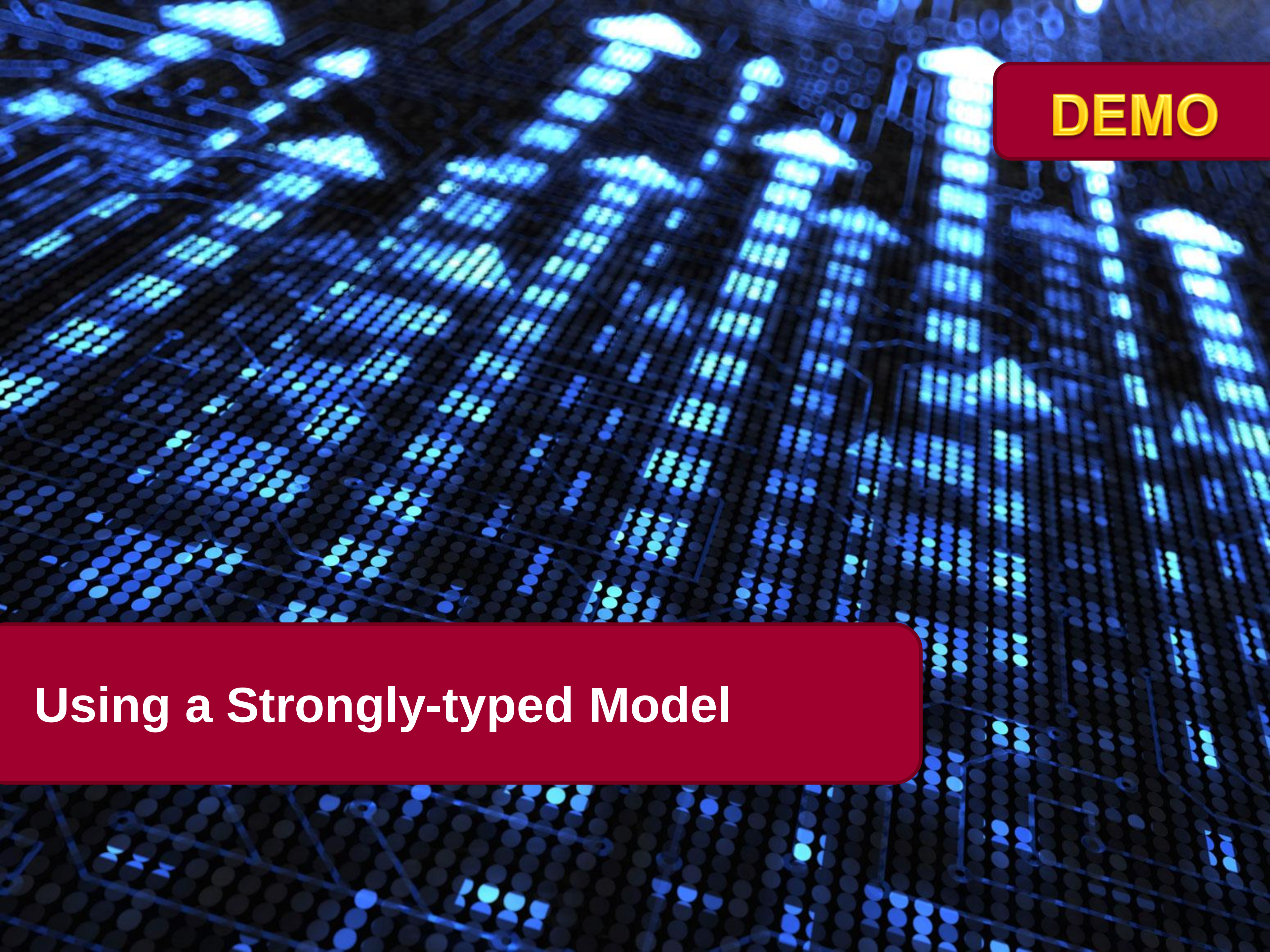
Data context class: WingtipCRMEntities (MVC_CRM.Models)

Views: Razor (CSHTML)


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
MVC_CRM
  Properties
  References
  App_Data
  App_Start
  Content
  Controllers
    CustomersController.cs
    HomeController.cs
  Models
  Scripts
  Views
    Customers
      Create.cshtml
      Delete.cshtml
      Details.cshtml
      Edit.cshtml
      Index.cshtml
    Home
    Shared
    _ViewStart.cshtml
    Web.config
  Global.asax
  packages.config
  Web.config
```



DEMO

Using a Strongly-typed Model

Agenda

- ✓ ASP.NET MVC Fundamentals
- ✓ Controllers, Views and Routing
- ✓ Designing Strongly-typed Models
- ➢ Adding MVC Forms and Validation
- ▪ Developing Asynchronous Controller

DEMO

Adding MVC Form Validation Logic

Agenda

- ✓ ASP.NET MVC Fundamentals
- ✓ Controllers, Views and Routing
- ✓ Designing Strongly-typed Models
- ✓ Adding MVC Forms and Validation
- ➢ Developing Asynchronous Controllers

DEMO

Creating Asynchronous Controllers

Summary

- ✓ ASP.NET MVC Fundamentals
- ✓ Controllers, Views and Routing
- ✓ Designing Strongly-typed Models
- ✓ Adding MVC Forms and Validation
- ✓ Developing Asynchronous Controller