Sheridan | Get Creative

Advanced .NET Server Development: ASP.NET MVC

Instructor: Amandeep S. Patti

Objectives

- Introduction to ASP.NET MVC
- Comparison with Web Forms
- Request Routing
- Razor View Engine
- ViewBag
- Passing a Model to a View (@model)
- Model Binding
- HtmlHelpers in Views
- Validation

Introduction to ASP.NET MVC

- Provides another approach (instead of Web Forms) to ASP.NET development with the following benefits:
  - Improved testability
    - ideal for TDD - Test Driven Development
  - Reduced overhead of ViewState
  - Complete control over HTML markup
  - Better separation of concerns
  - Embraces web standards (HTML, JavaScript, CSS)
  - Open Source code available for modifications
- Remember that with the "One ASP.NET" philosophy both approaches are built on the same ASP.NET core and can be mixed into the same project

Video

- http://www.asp.net/mvc/videos/mvc-2/how-do-i/5-minute-introduction-to-aspnet-mvc
-

Sheridan | Get Creative

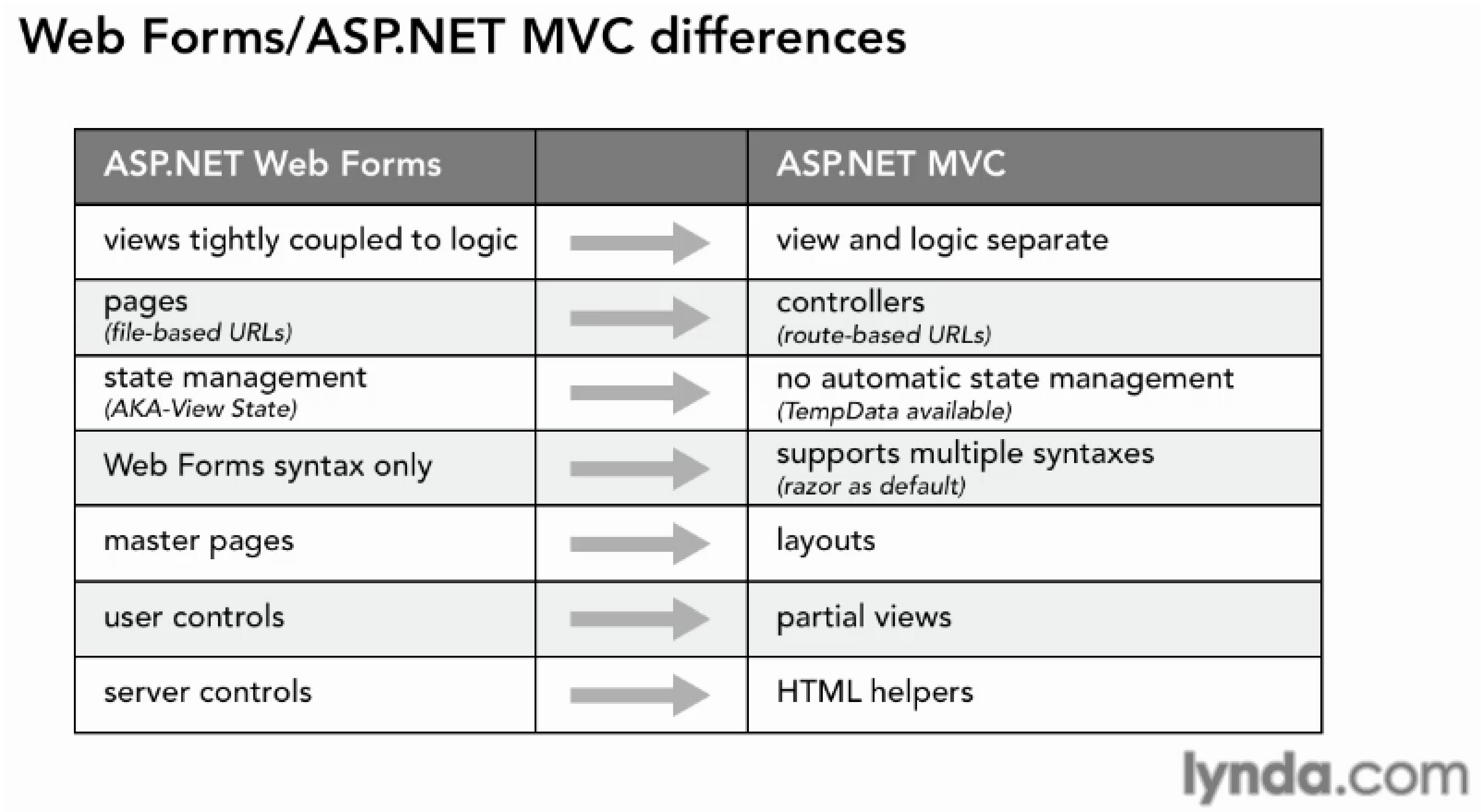
MVC versus Web Forms

- Initially intended by Microsoft for a small subset of ASP.NET applications which
  - needed improved testability and more control over the HTML
- However, ASP.NET MVC has been widely embraced by the ASP.NET community and has great "buzz" around it
- Web Forms has received a major update in Visual Studio 2013 to "catch up" with some of the new features in MVC
- Web Forms will continue to be supported going forward and will still be suitable for those who prefer:
  - Working in the traditional visual (Windows Forms) way
  - Using a Visual Designer and dragging and dropping controls onto web forms
  - Being abstracted from some of the details of the web
  - Using powerful stock GUI controls while sacrificing some control

MVC versus Web Forms

- URLs in MVC do not map to a specific file but instead to a specific controller action
- MVC relies on Conventions rather than Configuration
  - aka Conventions over Configuration to simplify configuration
- They do have a lot in common as well including:
  - Both built on ASP.NET framework
  - Use .NET framework and languages (C#, VB, etc.)
  - Use IIS for hosting
  - Membership and roles
  - Configuration
  - Caching

MVC versus Web Forms



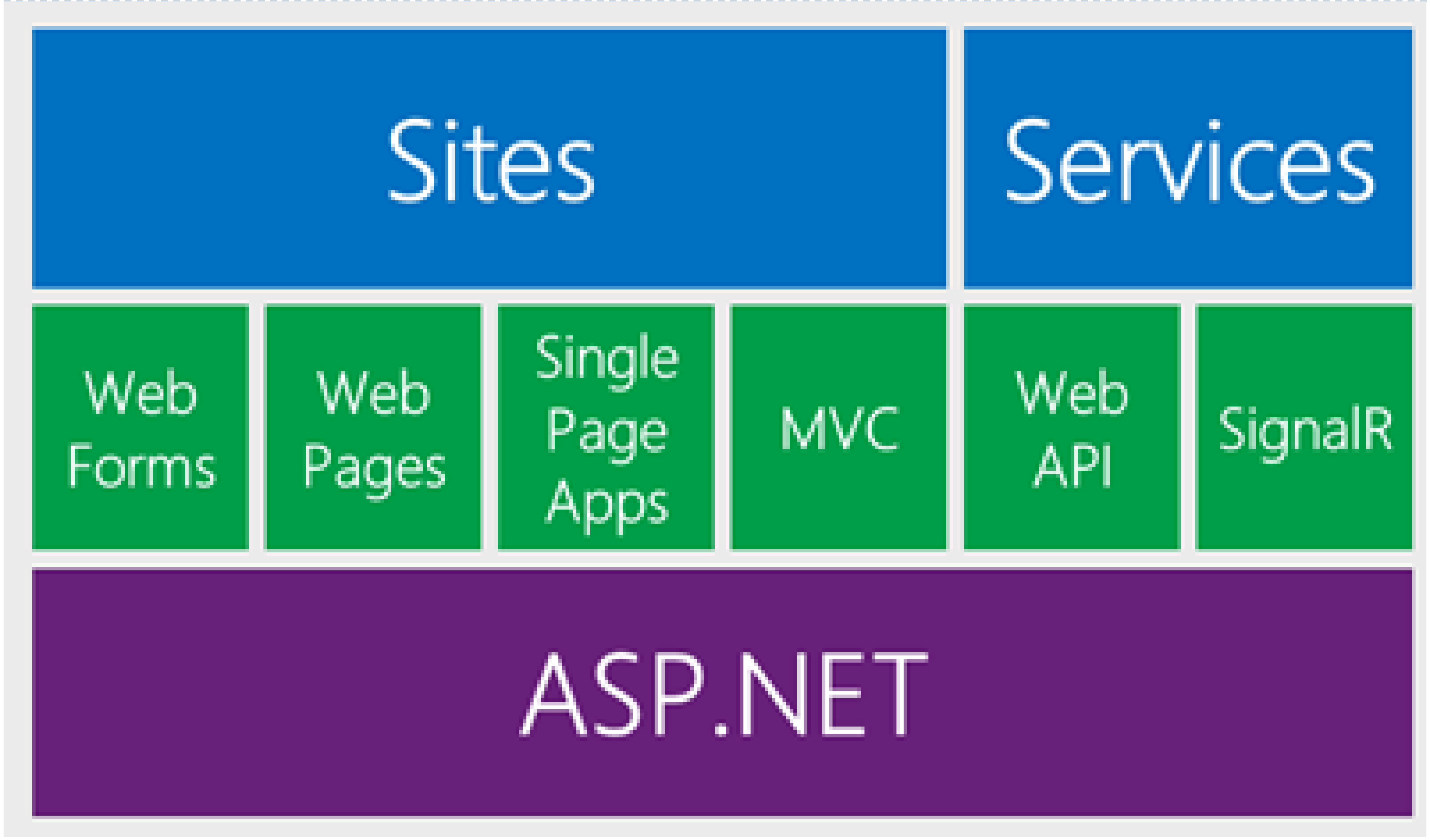
Web Forms/ASP.NET MVC differences

ASP.NET Web Forms		ASP.NET MVC
views tightly coupled to logic	→	view and logic separate
pages *(file-based URLs)*	→	controllers *(route-based URLs)*
state management *(AKA-View State)*	→	no automatic state management *(TempData available)*
Web Forms syntax only	→	supports multiple syntaxes *(razor as default)*
master pages	→	layouts
user controls	→	partial views
server controls	→	HTML helpers

lynda.com

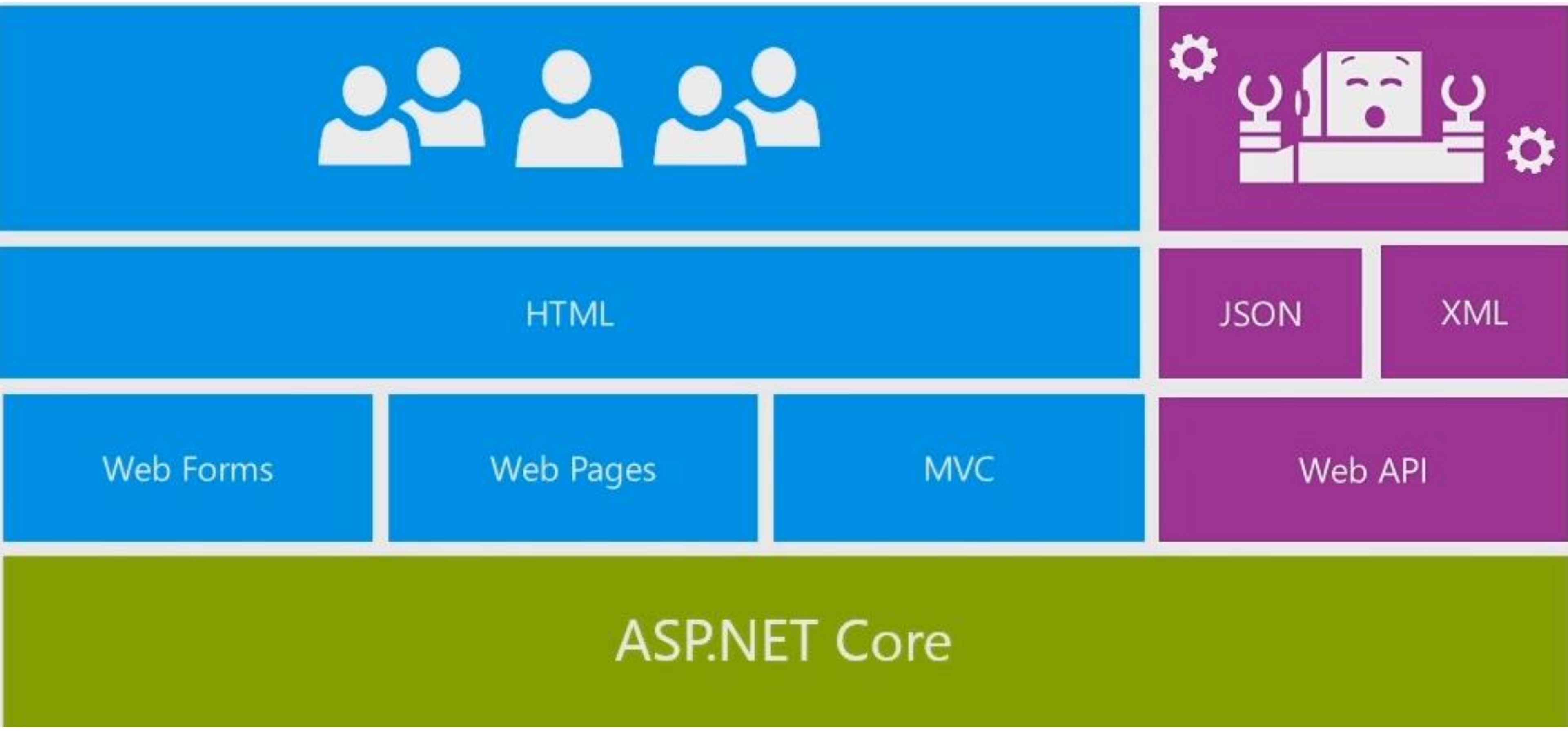
Chadwick, Jess. "ASP.NET MVC for Web Forms developers." ASP.NET MVC 4 Essential Training with Jess Chadwick. lynda.com, 9 Jan. 2013. Web. 3 Jan. 2014. <http://www.lynda.com/ASPNET-tutorials/ASPNET-MVC-Web-Forms-developers/109762/120285-4.html>.

One ASP.NET



"Released: ASP.NET and Web Tools 2012.2 in Context - Scott Hanselman." *Scott Hanselman - Coder, Blogger, Teacher, Speaker, Author*. N.p., n.d. Web. 5 Oct. 2013. <http://www.hanselman.com/blog/ReleasedASPNETAndWebTools20122

One ASP.NET



"Building Web Apps with ASP.NET Jump Start." *ASP.NET Web Apps Development, Create Windows Apps -Microsoft Virtual Academy*. Microsoft Virtual Academy, n.d. Web. 05 Oct. 2013.

Video

- http://www.lynda.com/ASPNET-tutorials/ASPNET-MVC-Web-Forms-developers/109762/120285-4.html

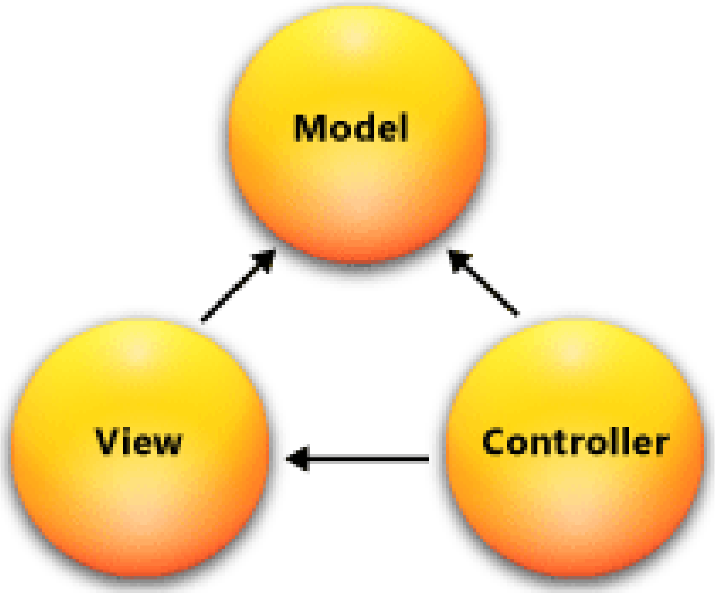
Sheridan | Get Creative

ASP.NET & Capstone Projects

- Each group can choose Web Forms, MVC or a combination of the two for Capstone
- MVC is one important module in this class because several past Capstone groups have had great success using it and have lamented about the lack of coverage in other .NET courses
- Quizzes, exams and labs (outside of your Capstone project) in this class will emphasize MVC over Web Forms since you have already been evaluated on Web Forms in a previous course

Models, Views, and Controllers

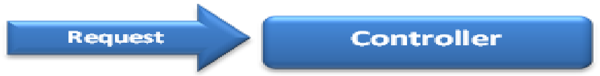
- Models:
  - Holds logic for the application's data domain.
  - Contains domain classes and business logic.
  - Often, deals with DB
- Views:
  - Represents UI components
  - Use HTML/JS/CSS etc.
- Controllers:
  - Handle user interaction
  - Work with the model
  - Select a view to render



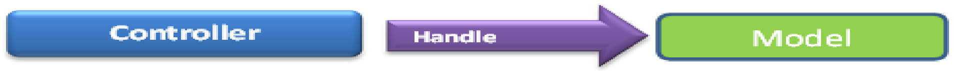
ASP.NET MVC Working

MVC Steps

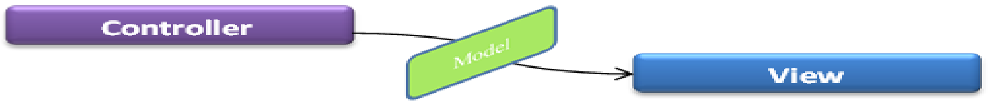
Step 1: Incoming request directed to **Controller.**



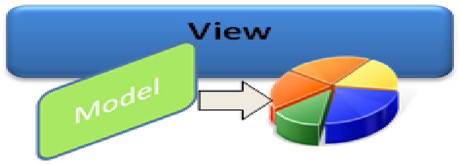
Step 2: **Controller** processes request and forms a data **Model.**



Step 3: **Model** is passed to **View.**



Step 4: **View** transforms **Model** into appropriate output format.

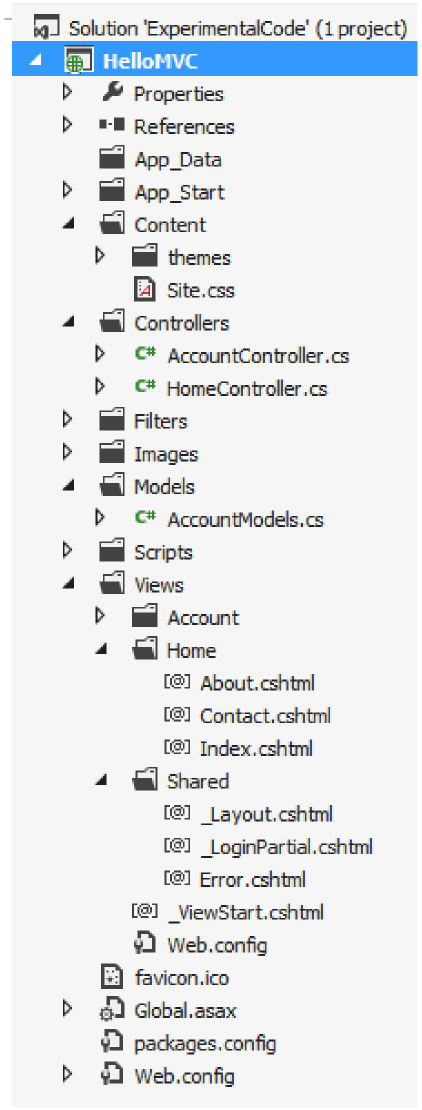


Step 5: Response is rendered.

Sohi, Avtar Singh. "Understanding Basics of UI Design Pattern MVC, MVP and MVVM." *CodeProject*. CodeProject, 20 July 2011. Web. 04 Jan. 2014

Conventions Over Configuration

- Controllers
  - Located in controllers folder
  - Names have a suffix of "Controller" by default
- Views
  - Organized into subfolders in the Views folder
  - Based on the Controller name
  - Names are the same as the corresponding controller action (method) names
- Models
  - Organized into the Models folder
- These conventions can be overridden



```
Solution 'ExperimentalCode' (1 project)
  HelloMVC
    Properties
    References
    App_Data
    App_Start
    Content
      themes
      Site.css
    Controllers
      AccountController.cs
      HomeController.cs
    Filters
    Images
    Models
      AccountModels.cs
    Scripts
    Views
      Account
      Home
        About.cshtml
        Contact.cshtml
        Index.cshtml
      Shared
        _Layout.cshtml
        _LoginPartial.cshtml
        Error.cshtml
      _ViewStart.cshtml
      Web.config
    favicon.ico
    Global.asax
    packages.config
    Web.config
```

Sheridan | Get Creative

Video

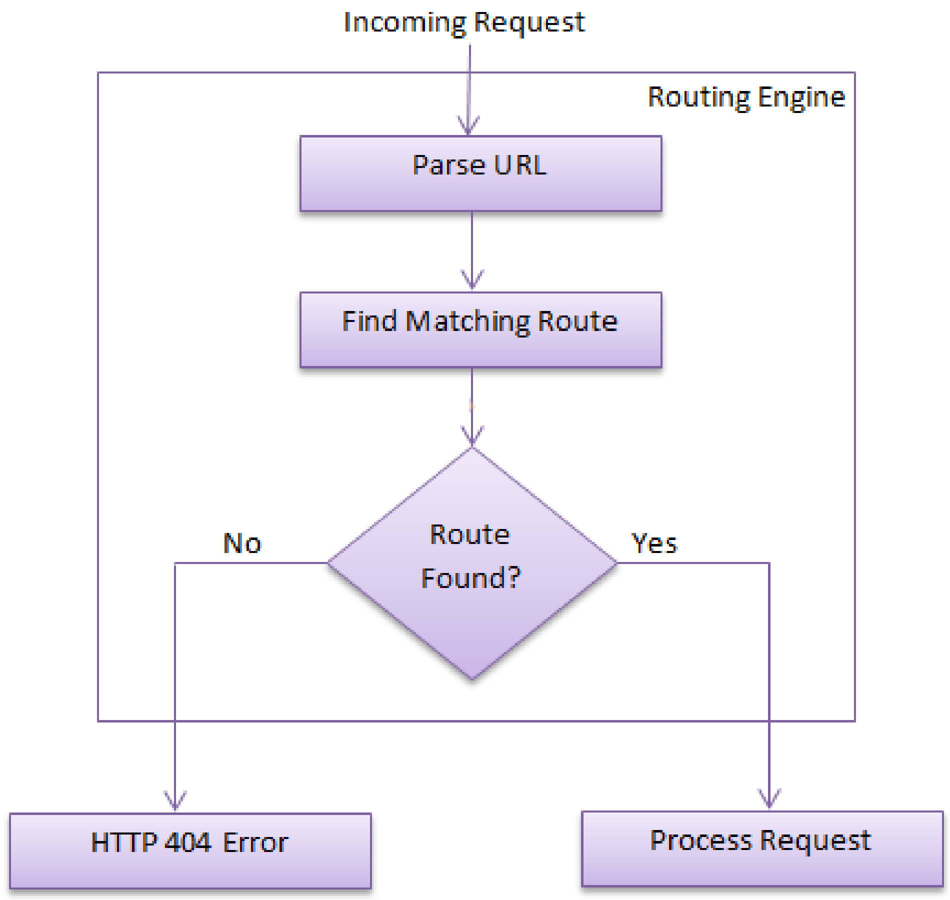
- http://pluralsight.com/training/Player?author=scott-allen&name=mvc4-building-m1-intro&mode=live&clip=0&course=mvc4-building

Sheridan | Get Creative

ASP.NET MVC – Routing Engine

- Routing Engine
  - parses URL
  - looks up the corresponding controller action
  - directs the request
- The action corresponds to a public method in the specified controller
- The Application_Start method in Global.asax.cs calls the RouteConfig.RegisterRoutes method to set up the various routes for an application
- A Map Route provides a name, URL pattern and default values

ASP.NET MVC – Routing & Processing



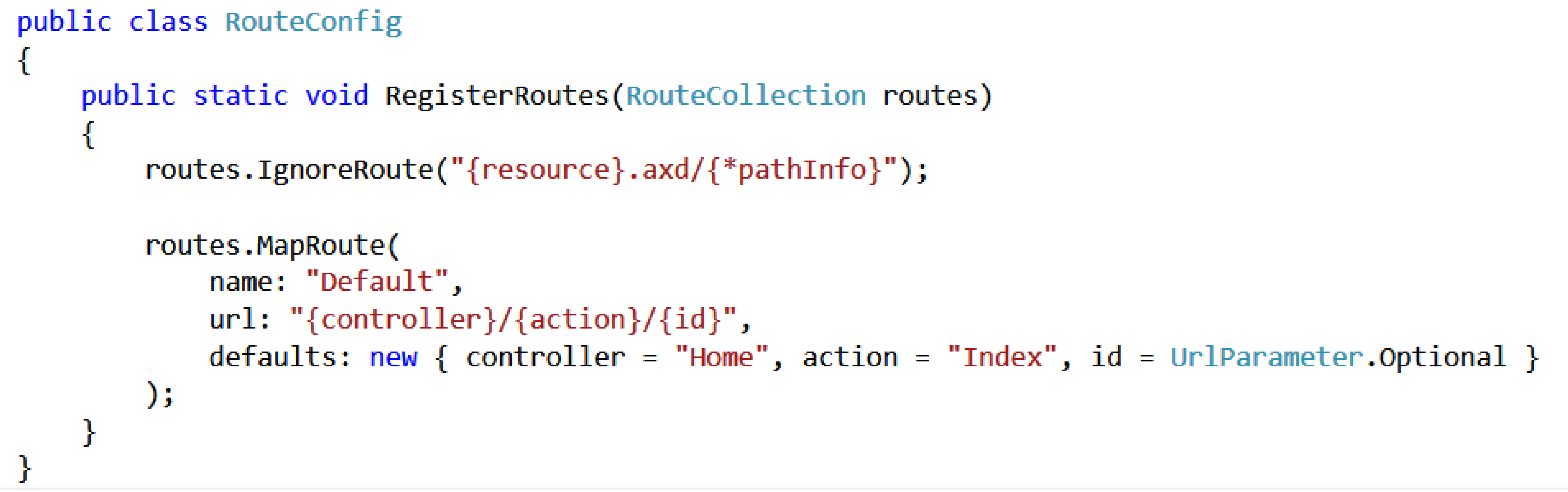
Incoming Request Processing by Routing System

ASP.NET MVC – Routing Engine

- Configuration of the mapping happens in RegisterRoutes call in Global.asax
- Default Route: {controller}/{action}/{id}
- Note that by default URLs map to controller/action/id so http://sheridancollege.ca/Courses/Register/5 would
  - Instantiate an instance of the Courses controller
  - Invoke the Register method of the controller
  - Pass the id parameter set to 5
- Provides clear and simple URLs which is great for Search Engine Optimization (SEO)

ASP.NET MVC – Routing Engine

- Default RegisterRoutes method is shown below



```
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 }
        );
    }
}
```

- We can add additional routes or change the default route if necessary

Video

- http://www.lynda.com/ASPNET-tutorials/How-routing-finds-controller-actions/109762/120297-4.html

Sheridan | Get Creative

ASP.NET MVC - Controller

- Responsible for:
  - Handling user input
  - Building the Model
  - Directing the View to display the Model
- Implements the IController Interface which has only one method: Execute(RequestContext requestContext);

ASP.NET MVC – Controller Actions

- The controller action corresponds to a public method in the specified controller
- Actions typically return an ActionResult but can return many other ActionResult derived types including ContentResult, ViewResult, FileResult and JsonResult.
- We can return a string from a controller action that returns an ActionResult by calling the Content method
- Alternatively, we can directly return a string as shown below:

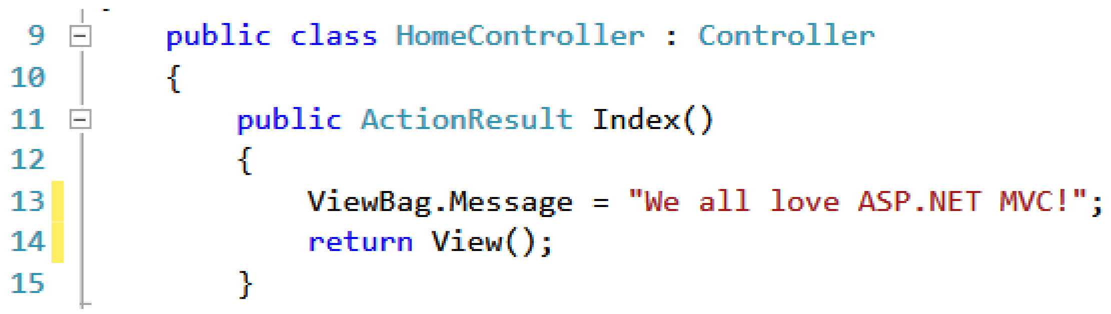
```
public string Index()
{
    return "Hello World!  This is my first ASP.NET MVC App!";
}
```

ASP.NET MVC – Controller Actions

- AcceptVerbs such as the [HttpPost] and [HttpGet] attributes can be used with actions to set the specific type of request (GET, POST, etc.) that the action handles
- Using these attributes we can have two actions with the same name in one controller with different purposes (ex. one to show a form, one that accepts user input)
- The [ActionName] attribute allows us to change the URL required to invoke an action.

ASP.NET MVC – Controller Actions

- To add a controller action
  - Create a public method in the controller class for the action
  - Ensure the method returns an ActionResult derived type
  - Specify any method parameters which the Action requires and will be passed from the user's browser
  - Implement the method's logic
- Basic example:



```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "We all love ASP.NET MVC!";
        return View();
    }
```

Sheridan | Get Creative

Passing Data from Controllers to Views

- ViewData dictionary property
  - Ex.ViewData[“Name”] = “Sheridan”
- ViewBag uses ViewData but replaces the need to cast elements in the ViewData dictionary with a simple property (ex.ViewBag.Name)
  - Ex.ViewBag.Name = “Sheridan”
- TempData dictionary property stored on server for one request
  - Ex.TempData[“Name”] = “Sheridan”
- Strongly typed Views using the Model property (@model Car)
  - Cleaner and clearer
  - Provides IntelliSense support
  - Better for testing
- The Session collection is also available just as in Web Forms but less used because of the other options that are available.
  - Ex. Session[“Name”] = “Sheridan”

Filters

- Action Filters – pre or post processing logic for all controller actions.
- Applied using attributes such as [Authorize], [ValidateAntiForgeryToken].
- Can be applied to individual actions, individual controllers or globally
- Can define our own Action Filters by deriving from ActionFilterAttribute
- By convention, filters are typically placed in the Filters folder of MVC projects
- All global filters are added to the RegisterGlobalFilters method of the FilterConfig class

What are Filters?

- Some requirements cut across logical boundaries are called cross-cutting concerns. Examples include:
  - Authorization
  - Logging
  - Caching
- There are four different types of filters:
  - Authorization filters run before any other filter and before the code in the action method
  - Action filters run before and after the code in the action method
  - Result filters run before and after a result is returned from an action method.
  - Exception filters run only if the action method or another filter throws an exception.

" Developing ASP.NET MVC 4 Web Applications Jump Start." Microsoft Virtual Academy – Free IT Training, Online Learning of Microsoft Technologies. N.p., n.d. Web. 4 Oct. 2013. <http://www.microsoftvirtualacademy.com/training-courses/developing-asp-net-mvc-4-web-applications-jump-start#fbid=F7bplp3CMSW>.

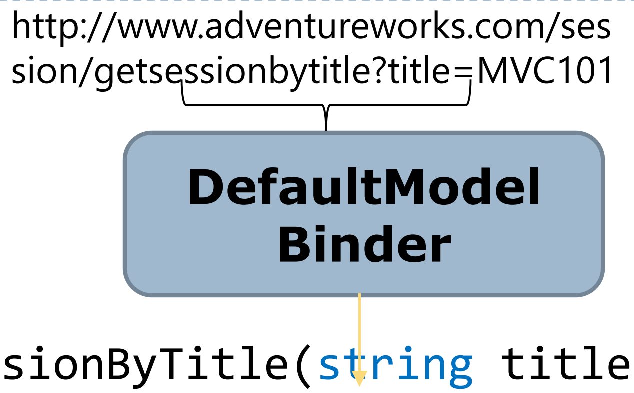
ASP.NET MVC - Model

- Responsible for implementing the domain objects (business logic)
- Normally implemented as POCO classes
- Provide the data that Views need to provide dynamic content to users
- Often created by the Controller during operation and passed to the View by calling the View method and specifying the model object
- Strongly typed in Views by using the @model statement at the start of the View (ex. @model Car)
- CRUD operations for Model objects are normally completed using LINQ to Entities

Controller Binding to Request Data (Model Binding)

- Request fields are mapped to controller action parameters automatically without having to look them up in code!
- Greatly reduces the need to access Request.Form property
- Parameters can be automatically validated too and if any errors are found they are added to the ModelState!
- Sources of model binding values
  - Form post values
  - Route data / URL parameters
  - QueryString parameters
  - User cookies

Using Parameters


http://www.adventureworks.com/session/getsessionbytitle?title=MVC101
DefaultModel Binder

```
public ActionResult GetSessionByTitle(string title){
   var query = from s in context.Sessions
        where s.Title == title
        select s
   Photo session = query.FirstOrDefault();
   return View("Details", session);
}
```

" Developing ASP.NET MVC 4 Web Applications Jump Start." *Microsoft Virtual Academy – Free IT Training, Online Learning of Microsoft Technologies*. N.p., n.d. Web. 4 Oct. 2013. <http://www.microsoftvirtualacademy.com/training-courses/developing-asp-net-mvc-4-web-applications-jump-start#fbid=F7bplp3CMSW>.

Controller Binding to Request Data (Model Binding)

- Bind(Exclude) attribute can be used to ignore fields from the action parameter list. Good security measure but using a whitelist via the Include attribute is even better!
- You can provide a custom Model Binder by deriving from DefaultModelBinder but this is rarely done
- Example of Model Binding to action parameter
  - public ActionResult Create([Bind(Exclude="ID")] Models.Car car)
- Model Binding can also be explicitly invoked by calling the UpdateModel (throws exceptions for invalid data) or TryUpdateModel (adds ModelState error for invalid data) methods in a controller actions. This is seldom necessary.

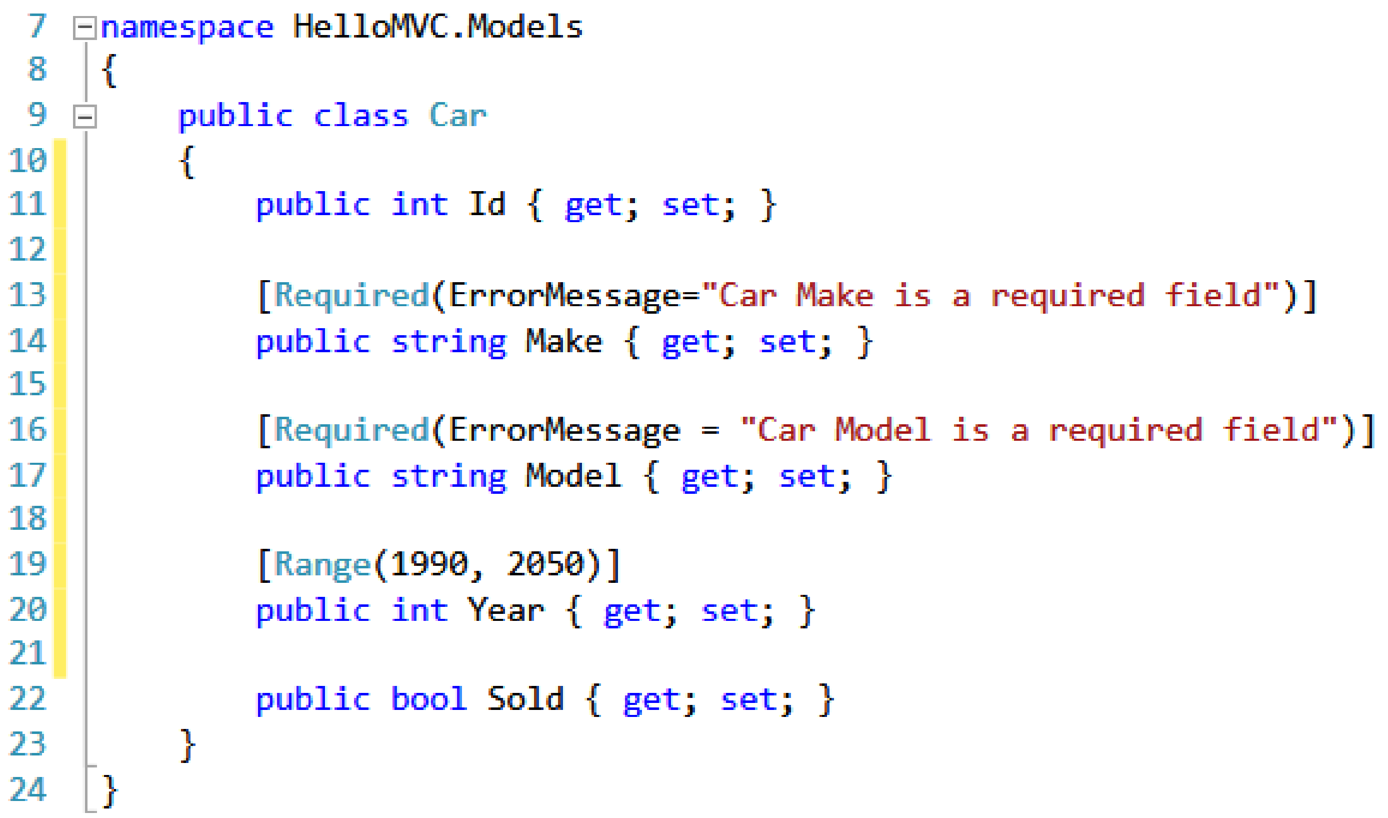
Model Validation

- Can validate manually in each controller action and call ModelState.AddModelError for validation errors
- Manual validation can be tedious and may require similar code to be added to different controllers so ASP.NET provides better ways…

Validation Attributes

- Some Data Annotation Attributes in model class are associated with validation such as [Required], [Range], [StringLength], and [RegularExpression]
- Other Data Annotation Attributes in model class are associated with presentation such as [DisplayName], [DisplayFormat], [DataType]
- Can specify a DataType of EmailAddress and have the HTML5 email control automatically used
- Can define custom attributes by deriving from ValidationAttribute
- DRY (Don't Repeat Yourself) means that these annotations are automatically used elsewhere (HTML Helpers in Views, etc.)!
- Check ModelState.IsValid in controller actions before handling any update requests

Validation Attributes Sample



```
namespace HelloMVC.Models
{
    public class Car
    {
        public int Id { get; set; }

        [Required(ErrorMessage="Car Make is a required field")]
        public string Make { get; set; }

        [Required(ErrorMessage = "Car Model is a required field")]
        public string Model { get; set; }

        [Range(1990, 2050)]
        public int Year { get; set; }

        public bool Sold { get; set; }
    }
}
```

Model Binding Validation

- Other approaches to validation:
  - Client side validation is automatically included in most default projects through the JQuery Validation Library and JQuery Unobtrusive Validation Library which helps to keep things DRY
  - Model class can implement the Validate method of the IValidatableObject interface which is automatically invoked during Model Binding

Video

- http://www.lynda.com/ASPNET-tutorials/Automatically-binding-data-request/109762/120301-4.html

Sheridan | Get Creative

ASP.NET MVC - View

- Responsible for displaying the model to the user
- cshtml files are used and can be considered templates with static html and C# code for dynamically inserting the model's properties
- Use Razor View Engine by default for displaying the View
- Easy way to create a new view - right click inside a controller action and select "Add View"
- In paths, "~" represents the root folder of the application

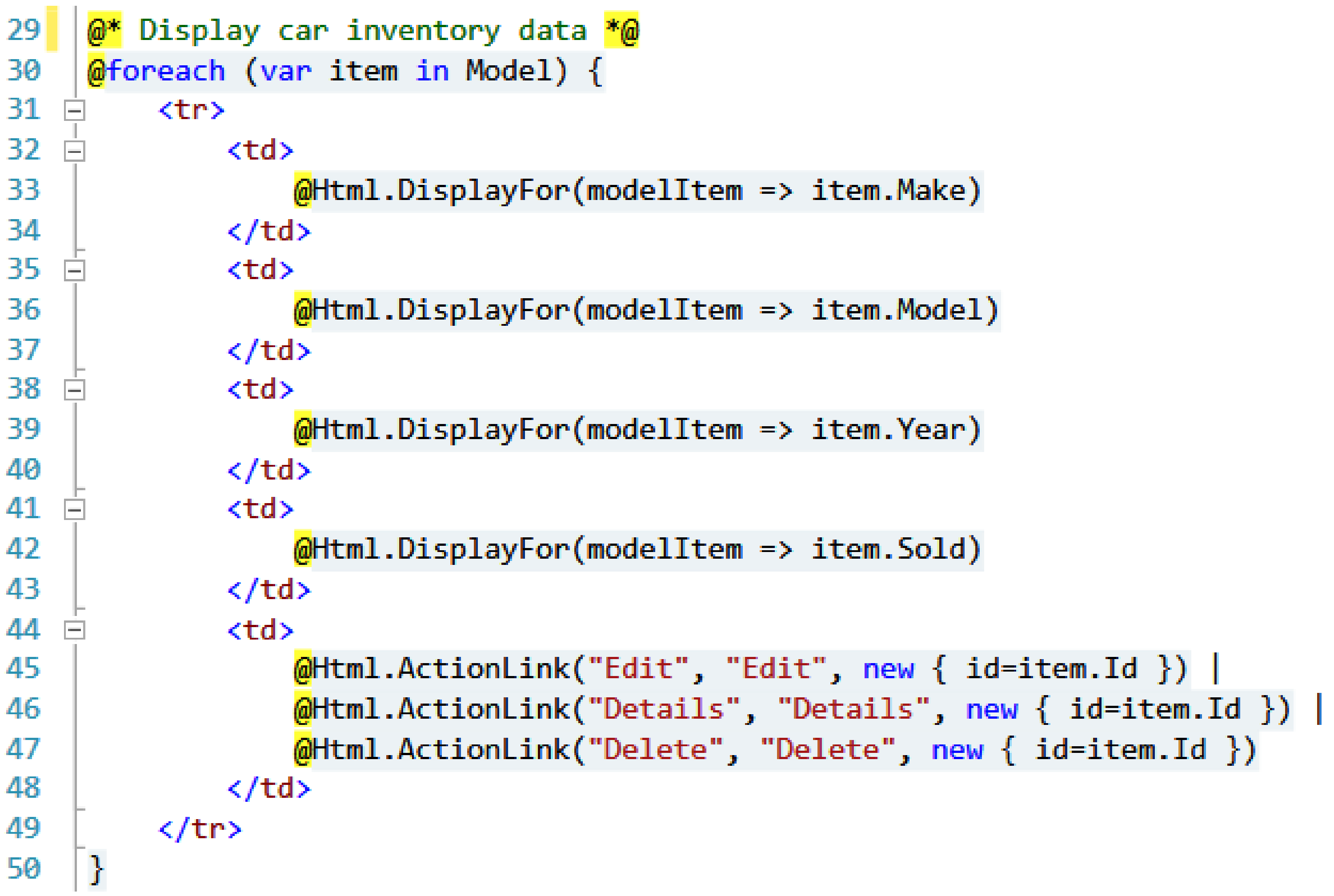
MVC Razor View Engine

- New View Engine which is the default for ASP.NET MVC
- Uses a clean and simple syntax for mixing HTML markup with C# code
- Supports strongly typed Views using the Model property (@model Car)
- The Web Forms (aspx) View Engine is still available but is rarely used (legacy projects, etc.)
- Responsive design for different device sizes is automatically built into the default "Internet Application" and other projects using Bootstrap

MVC Razor Syntax

- C# code is delimited by
  - @ (ex. @ViewBag.Title)
  - @() (Explicit code expression)
  - @{} (Code block)
- Markup can be delimited in code blocks by the "@:" prefix or using a <text> block. However, Razor will often switch naturally between code and markup without these delimiters
- @@ to escape the @ symbol
- Comment Syntax: @* comment *@
- Razor expressions (ex. @ViewBag.Name) are automatically HTML encoded to help prevent scripting attacks. Can be overridden by the Html.Raw() helper
- See the following link for more Razor syntax tips

Razor Sample Code



```
@* Display car inventory data *@
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Make)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Model)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Year)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Sold)
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
            @Html.ActionLink("Details", "Details", new { id=item.Id }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.Id })
        </td>
    </tr>
}
```

Sheridan | Get Creative

Video

- http://www.lynda.com/ASPNET-tutorials/Introduction-Razor-AKA-symbol/109762/120287-4.html

View (HTML and URL) Helpers

- Simplifies HTML coding in Views by using metadata and annotations
- @Url.Action("Action", "Controller") – action URL, no anchor tag
- @Html.ActionLink("Text", "Action", "Controller") – action link with anchor tag
- @Html.LabelFor - HTML label for the field name
- @Html.DisplayNameFor – plain text to display the read-only field name
- @Html.DisplayFor – figures out the best HTML5 for read-only display of the actual field value
- @Html.EditorFor - figures out best HTML5 control to edit a given field and names it accordingly (ex. email address)
- @Html.EditorForModel – figures out the best HTML5 controls for all properties of the model
- @Html.ValidationMessageFor – validation message for individual field
- @Html.ValidationSummary – lists all validation errors for the view
- @Html.Partial – renders a partial view in another view
- @Html.BeginForm – to start an HTML form
- @Html.AntiForgeryToken – to prevent spoofing by other servers. Works with [ValidateAntiForgeryToken] attribute in the controller.

HTML Helpers and Lambda Expressions

- Many HTML Helpers have two versions
  - @Html.xxxx
  - @Html.xxxxFor
  - (ex. Html.Display("Make") and Html.DisplayFor(model => model.Make)
- The @Html.xxxxFor form is preferable since:
  - The Designer can assist and provide Intellisense for the Lambda Expressions
  - The Compiler can provide type checking at compile time
- Note that the Lambda Operator is "=>"
- In the Lambda Expression above (model => model.Make) the compiler infers that the model is a Car type and accesses the Make property of the supplied model object.
- For more information on Lambda Expressions
  - http://msdn.microsoft.com/en-us/library/vstudio/bb397687.aspx

Using Editor Helpers

Html.LabelFor()

```
@Html.LabelFor(model => model.ContactMe)
```

```
<label for="ContactMe">
  Contact Me
</label>
```

Html.EditorFor()

```
@Html.EditorFor(model => model.ContactMe)
```

```
<input type="checkbox"
  name="Description">
```

" Developing ASP.NET MVC 4 Web Applications Jump Start." *Microsoft Virtual Academy – Free IT Training, Online Learning of Microsoft Technologies*. N.p., n.d. Web. 4 Oct. 2013. <http://www.microsoftvirtualacademy.com/training-courses/developing-asp-net-mvc-4-web-applications-jump-start#fbid=F7bplp3CMSW>.

Html.BeginForm Helper

- Use to create an HTML Input Form for a page
- To ensure proper clean up of resources:
  - Remember to prefix the Html.BeginForm Helper with the using keyword (@using Html.BeginForm). This is done automatically by the Designer when using Scaffolding.
  - OR use the Html.EndForm Helper at the end of the form

View Validation

- Model's annotations are automatically validated when using the Html Helpers (ex. Html.EditorFor)
- @Html.ValidationSummary helper can be used to provide a validation summary for the View
- @Html.ValidationMessage can be used to provide a validation message for a specific control in a View

Video

- http://www.lynda.com/ASPNET-tutorials/Speeding-up-view-development-HTML-URL-helpers/109762/120289-4.html

Video

- http://channel9.msdn.com/Events/aspConf/aspConf/Getting-Started-With-ASP-NET-MVC
- http://www.microsoftvirtualacademy.com/training-courses/developing-asp-net-mvc-4-web-applications-jump-start#fbid=F7bplp3CMSW

View Layout Pages

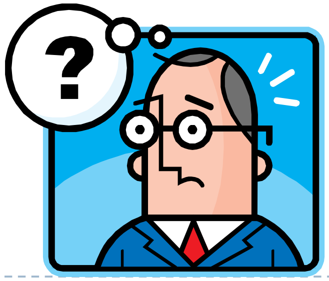
- Designed to provide a consistent look and feel across a site
- Layouts are analogous to Master pages in Web Forms
- _ViewStart.cshtml is used to organize common code for Views in the current folder and all child folders
- By default _ViewStart.cshtml includes the Views/Shared/_Layout.cshtml file in all Views of our site. The @Layout code can be used in a View to override this.
- @RenderBody and @RenderSection in the layout file are used to insert markup from the content views
- @section is used to mark sections in Views
  - Ex. @section featured

Error View

- To avoid having users see the dreaded ASP.NET "Yellow Screen of Death" page in their browsers a default Error.cshtml view is defined in the Shared folder
- Error.cshtml provides a customizable View for application errors. Remember to customize this View as necessary!

Scaffolding

- MVC Scaffolding uses a Wizard that can automatically generate much of the standard code for our Controllers and Views
- This is a really cool feature that amazed previous Capstone groups and convinced many of them that MVC was the way to go!
- Scaffolding is being added to Web Forms in VS 2013 too!
- VS Scaffolding and the Entity Framework Designer are two clear examples of Code Generation tools.

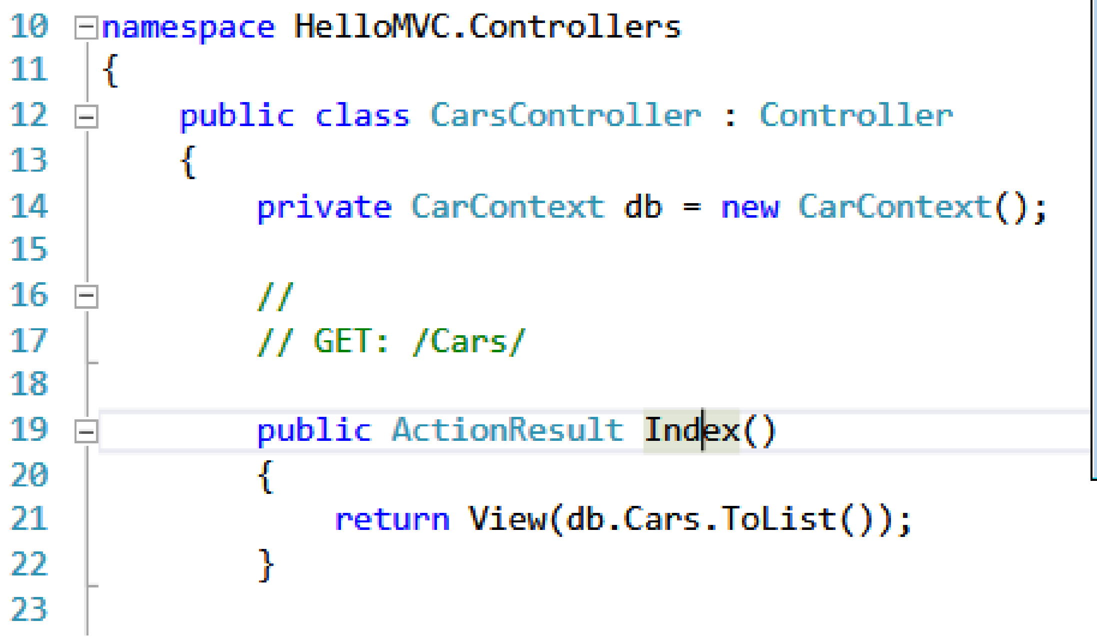


Reflect on the implication of Code Generation tools

- How do they impact the role of Developers?
- How can you deliver the most value to your employers and customers?

Scaffolding – Adding a View for a Controller Action

1. Right click in the Controller Action
2. Select "Add View…"
3. Configure the Dialog
4. Click "Add"

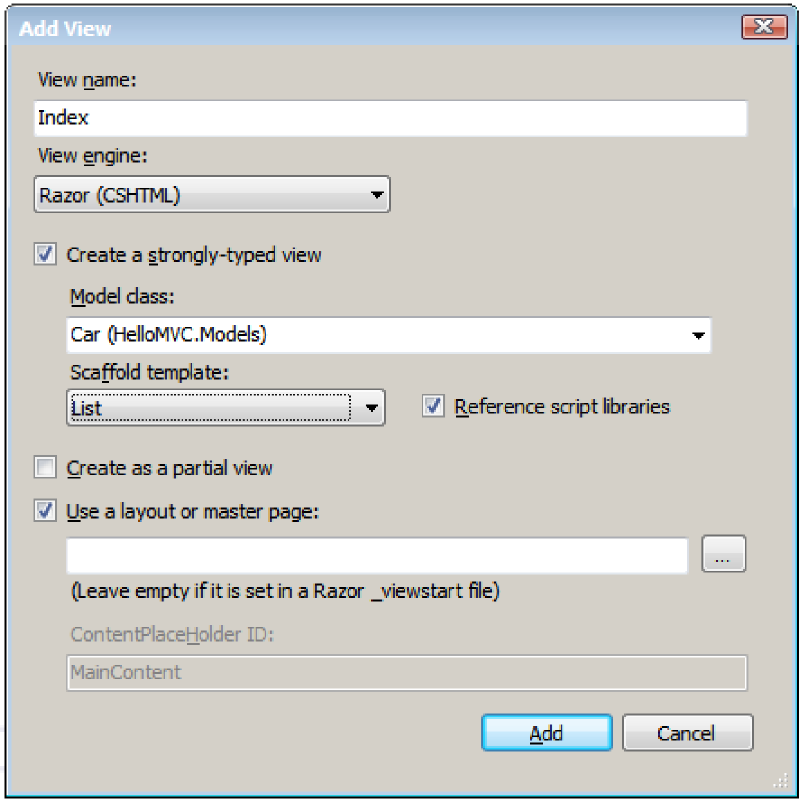


```
namespace HelloMVC.Controllers
{
    public class CarsController : Controller
    {
        private CarContext db = new CarContext();

        //
        // GET: /Cars/

        public ActionResult Index()
        {
            return View(db.Cars.ToList());
        }

```



Add View

View name:
Index

View engine:
Razor (CSHTML)

☑ Create a strongly-typed view

Model class:
Car (HelloMVC.Models)

Scaffold template:
List

☑ Reference script libraries

☐ Create as a partial view

☑ Use a layout or master page:

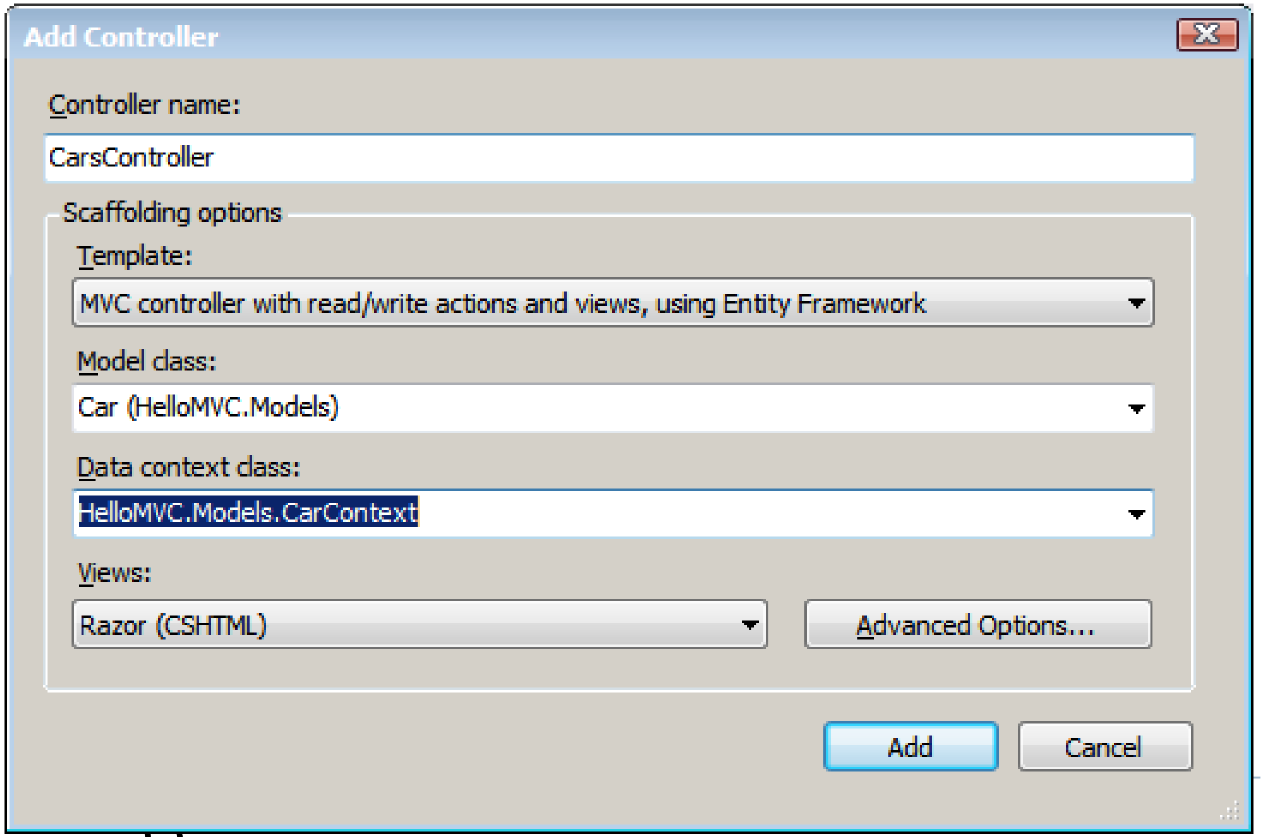
(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Sheridan | Get Creative

Scaffolding – Adding Controller and Views based on a Model

1. Right click on the Controllers folder
2. Select "Add Controller…"
3. Follow the Wizard…



Add Controller

Controller name: CarsController

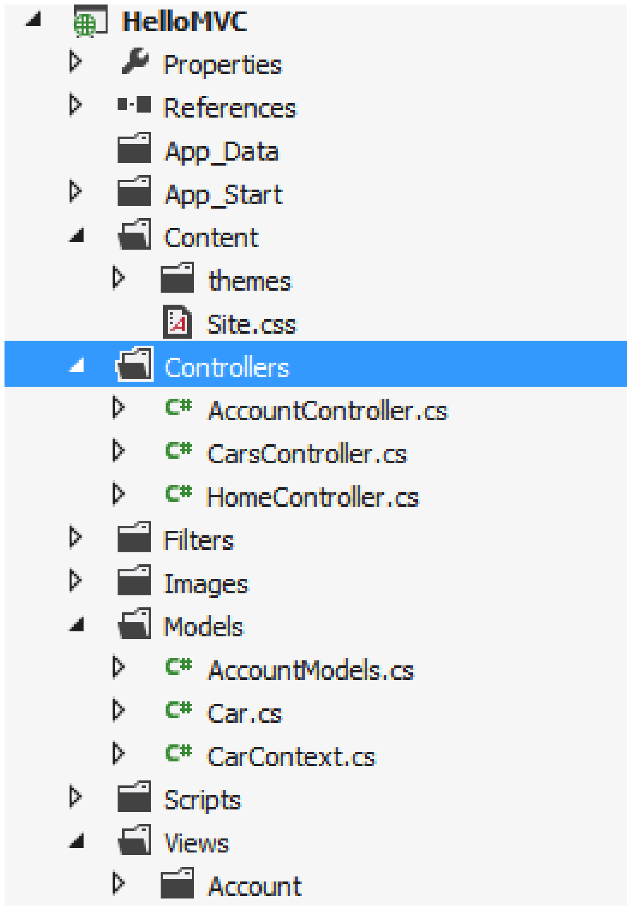
Scaffolding options

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

Model class: Car (HelloMVC.Models)

Data context class: HelloMVC.Models.CarContext

Views: Razor (CSHTML)



```
HelloMVC
  Properties
  References
  App_Data
  App_Start
  Content
    themes
    Site.css
  Controllers
    AccountController.cs
    CarsController.cs
    HomeController.cs
  Filters
  Images
  Models
    AccountModels.cs
    Car.cs
    CarContext.cs
  Scripts
  Views
    Account
```

Sheridan | Get Creative

Partial Views

- Helps with reuse of markup and C# code in views
- Renders a portion of the view
- Reduces complexity of views
- By convention Partial Views are named with a leading underscore ("_")
- Use @Html.Partial("_MyPartialView", myPartialModel) in Content View
- Alternatively, @Html.Action can be used to invoke a controller action that can return a PartialView in order to render a partial view within a view
- Child actions [ChildActionOnly] can help to simplify both controllers and views and are commonly used in layout files. They can not be invoked directly using the URL.

Video

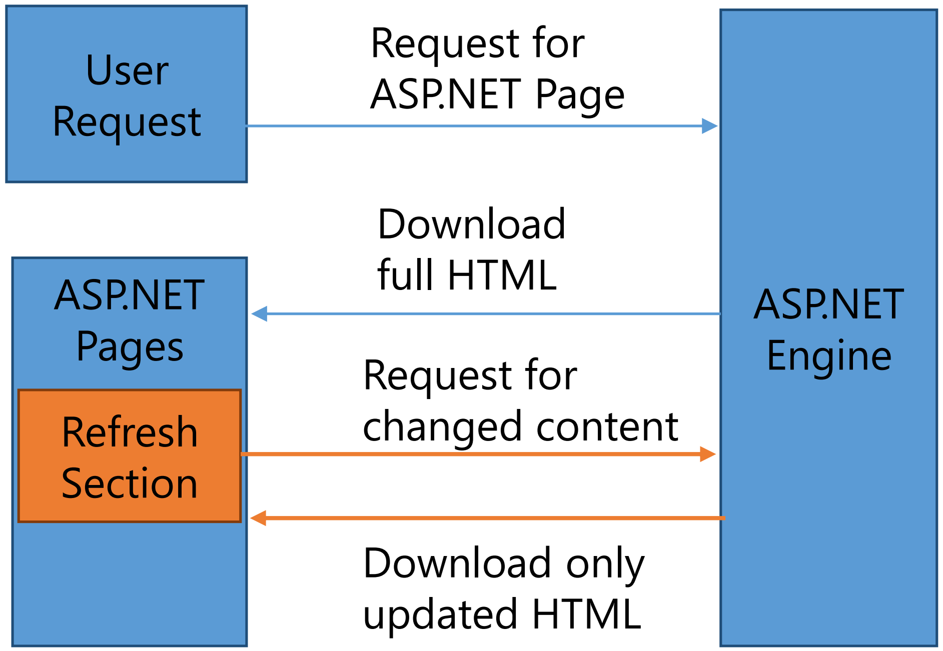
- http://www.lynda.com/ASPNET-tutorials/Reusing-logic-managing-complexity-partial-views/109762/120290-4.html

Sheridan | Get Creative

Why Use Partial Page Updates?

Partial page updates:

- Allow updates of individual sections of a webpage, during postback
- Increase the responsiveness of a web application



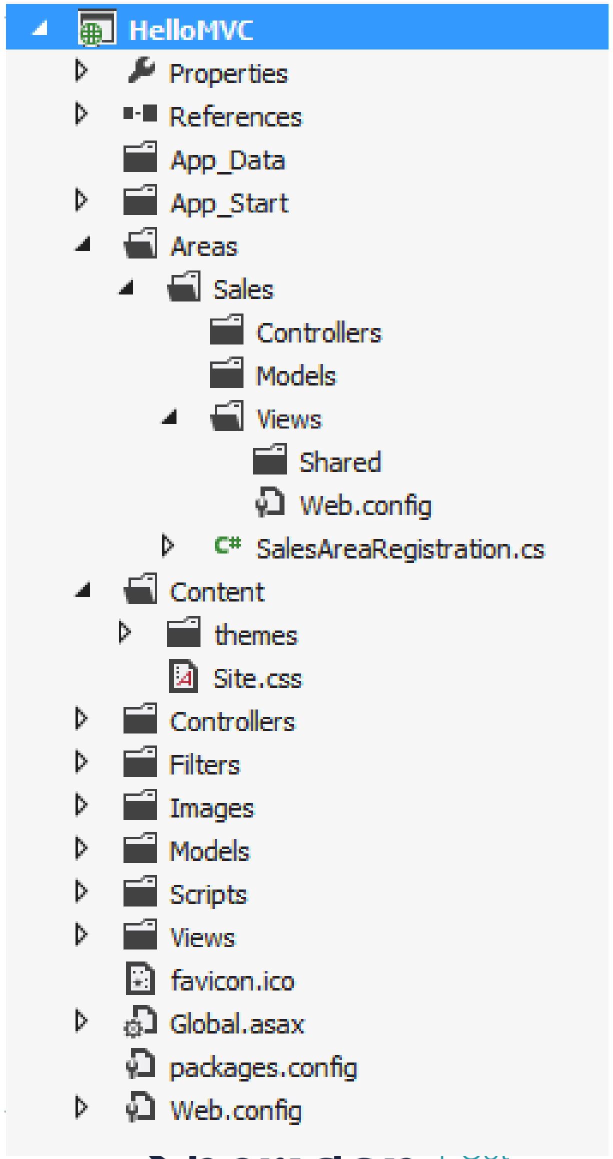
" Developing ASP.NET MVC 4 Web Applications Jump Start." *Microsoft Virtual Academy – Free IT Training, Online Learning of Microsoft Technologies*. N.p., n.d. Web. 4 Oct. 2013. <http://www.microsoftvirtualacademy.com/training-courses/developing-asp-net-mvc-4-web-applications-jump-start#fbid=F7bplp3CMSW>.

Partial Page Updates using AJAX

- Controller action should:
  - Specify a return type of PartialViewResult
  - Return the result of a call to the PartialView method
- Views should leverage Helpers as follows:
  - Define a partial view for AJAX portion of View
  - Call Html.Action to invoke the partial view from the "main" view
  - Call Ajax.BeginForm instead of Html.BeginForm from "main" view
  - (Optional) Ajax.ActionLink can be used from the View to asynchronously invoke a controller action
- Many JavaScript libraries are also provided for additional AJAX support including: JQuery, JQuery UI, JQuery Validate, Knockout, Modernizr

Organizing Solutions

- When the MVC folder structure gets too complicated, portions of the site can be organized into Areas.
  - Each area by default is organized into Models, Views and Controllers subfolders
  - By default a new route is added for each new area
- Solutions can also be split into Projects. One common approach is to move the Domain Model into a separate Class Library Project.

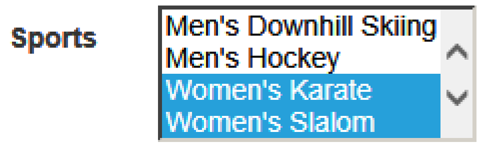


Common Design Patterns

- ViewModel – separates the data that the view needs to present. It is the Presentation Model instead of the Domain Model. The AutoMapper library is often used to simplify the mapping between the two models. LINQ projections can also be used.
- Repository – move data access (LINQ) code out from controller and into a special layer that provides a collection like interface
- Inversion of Control – depend on interfaces instead of concrete classes. Great for TDD!
- Dependency Injection – pass the dependency (ex. interface) into the class. Commonly used to implement Inversion of Control. Great for TDD!

Views: Editing Many-To-Many Relationships

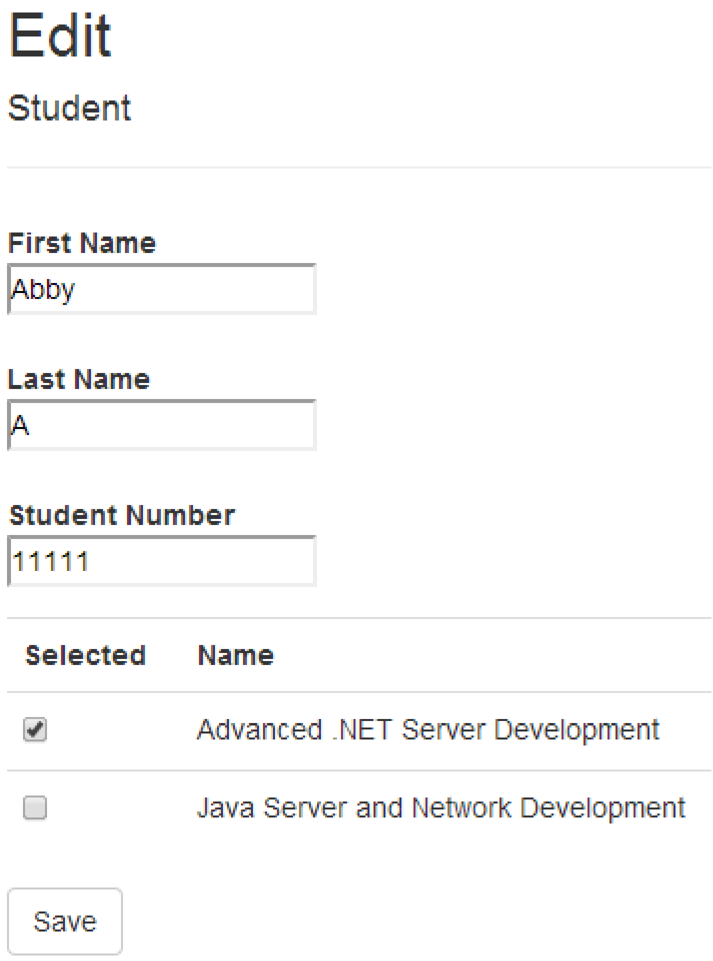
- When editing an entity which is part of a Many-To-Many relationship how do you change the list of related entities?
- Unfortunately, scaffolding on its own will normally not do it!
- One approach relies on using a ListBox in the View to display a MultiSelectList built by the Controller and placed in the ViewBag



Sheridan | Get Creative

Views: Editing Many-To-Many Relationships

- Another approach relies on the Controller building a custom ViewModel with a collection of possible selections
- When updating the View, remember that:
  - The View will normally need to use indexes in a for loop order to allow binding to multiple elements:
    - @for (int i = 0; i < Model.Courses.Count(); i++)
    - @Html.EditorFor(m => m.Courses[i].Selected)
  - The View will often represent the associations with CheckBoxes



Edit

Student

First Name: Abby

Last Name: A

Student Number: 11111

Selected	Name
✓	Advanced .NET Server Development
☐	Java Server and Network Development

MVC Testing

- MVC is very well suited to testing since Controller Actions are public methods that can be easily invoked
- Basic MVC testing focusses on testing Controllers.
- Generally to test an MVC Controller action:
  - Create the controller
  - Call the controller action under test
  - Check (Assert) the expected result(s)
- Visual Studio has great support for building and running automated unit tests
- We'll be covering testing in more detail later in this course!

Caching

- Caching can be used to improve the performance of an ASP.NET Application
- ASP.NET MVC Caching allows us to:
  - Cache the entire HTML from a Controller Action using the [OutputCache] action filter to avoid executing the same action repeatedly
  - Cache the HTML from a Controller Child Action (invoked from a View using @Html.Action or @Html.RenderAction) to cache a portion of the entire HTML for the page
  - Cache the data used in an action by using the System.Web.Caching.Cache collection (ex. Cache[“cars”] = DB.GetCars())
- Caching can be configured in many different ways for optimization purposes

Video

- http://pluralsight.com/training/Player?author=scott-allen&name=mvc4-building-m1-intro&mode=live&clip=0&course=mvc4-building - 8.2

Sheridan | Get Creative

Localization & Resources

- Thread.CurrentCulture property controls formatting of strings by setting the culture for the current thread
- Thread.CurrentUICulture property controls which resources the Resource Manager loads at run time
- To make localization happen automatically based on the browser's language preference:
  - In Web.Config set the globalization attributes of culture and uiCulture to auto
- Resx files are used for localized text and assets
  - String.resx for default
  - Strings.es.resx for Spanish
- Can use @Resources.Name in the View to get the localized text

MVC Best Practices I

- Use Data Annotations liberally to benefit from the built in Model Binding validation
- Use the HTML Helpers to simplify the coding of your views and leverage the model's Data Annotations
- Prefer the templated HTML helpers (suffix of "For") over the non-templated and non-type safe counterparts
- Stick with the conventions in ASP.NET (naming, folders, etc.) unless you have a good reason to deviate
- Consider providing a filter and/or investigating the predefined filters for common code in actions
- Use layouts to standardize the look and feel of your Views
- Use Partial Views to eliminate repetitive portions of Views

MVC Best Practices II

- Use strongly typed views (@model) as much as possible
- Prefer the ViewBag over the ViewData dictionary if you are not using a strongly typed view
- Consider including only specific fields in Model Binding to prevent some security issues
- Check the ModelState.IsValid property in controller actions before processing the user's data. If invalid then reload the view with the model data to display the error:
  - return View(MyModel);
- Consider the Caching options if and when performance issues arise
- Consider using some of the Design Patterns described previously (ViewModel, Repository, etc.)

MVC Best Practices III

- Use Projects and Areas to organize your solution
- Don't be afraid to mix in a bit of Web Forms in your MVC projects
- Plan to use Automated Unit Testing in Visual Studio. MVC was built with testability in mind
- Review scaffolded code and views to ensure they are optimized for your application
- Consider using something to log exceptions, errors, etc. such as:
  - ELMAH (Error Logging Modules and Handlers) to provide remote access to ASP.NET errors.
  - Tracing using the HttpContext.Trace class and the trace.axd file
  - ASP.NET Health Monitoring