

ASP.NET MVC 5

Agenda

- Introduction
- Request Pipeline
- Routing
- Controllers
- Models
- Data Access
- Views
- Bootstrap Framework
- Helpers

ASP.NET MVC 5

Agenda

- Strongly-Typed Views
- HTML Forms
- Action Selectors
- Model Binding
- Partial Views and Child Actions
- ViewModels
- Data Validation
- Ajax
- jQuery UI

ASP.NET MVC 5

Agenda

- Action Filters
- Asynchronous Controller Actions
- Mobile Clients
- Web API
- Security
- Deployment

ASP.NET MVC 5

Introduction

- Goals of Modern Web Development
- ASP.NET MVC
- Model-View-Controller Design Pattern
- Unit Testing
- Convention Over Configuration
- Managing Dependencies
- Case-Study Application
- Lab: Create a New ASP.NET MVC Project

Introduction

Goals of Modern Web Development

- Design based on standards
 - HTML5, CSS3, JavaScript
- Provide a good experience for different client types
 - Desktop, tablet, phone
 - Can be done client-side (responsive) and/or server-side (adaptive)
- Expose clean, consistent, meaningful URLs
- Leverage client-side frameworks
 - jQuery, Bootstrap, Angular, React, ...

Introduction

ASP.NET MVC

- ASP.NET MVC is a framework for building web applications
- Applies the Model-View-Controller (MVC) design pattern to the ASP.NET framework
- Uses the same underlying plumbing as ASP.NET Web Forms
 - Request, Response, Session, Cache, etc.
- Avoids use of System.Web.UI components
 - Server controls, ViewState, etc.
- Open source
 - aspnetwebstack.codeplex.com

Introduction

Model-View-Controller Design Pattern

- MVC provides an elegant means of separating concerns within an application
- Has been used in dozens of frameworks on a variety of platforms

Introduction

Model-View-Controller Design Pattern

- Model
 - Set of classes that describe the data you're working with as well as reusable business logic
- View
 - The application's user interface (UI)
 - Defines how model data is represented to users
- Controller
 - Set of classes that handle communication from the user, overall application flow, and application-specific logic

Introduction

Model-View-Controller Design Pattern

- Each module should focus on what it is good at while maintaining as little knowledge of the other parts as possible
- Model should not know anything about Views and Controllers
- View should not know about the existence of Controllers
- Controller should know about the existence of relevant Views and their individual purpose but avoid going deeper
 - Controller code should provide data to a View but should not be aware of the view's implementation

Introduction

Unit Testing

- Testing your code for accuracy and errors is at the core of good software development
- Unit testing as a concept should be part of all development
- Testability and a loosely-coupled design go hand-in-hand
- Even if not writing tests, keeping testability in mind helps to create more flexible, maintainable software

Introduction

Convention Over Configuration

- ASP.NET MVC applications, by default, rely heavily on conventions
- Some examples...
 - Controller classes are expected to end with Controller and are typically located in a folder named Controllers
 - View template files are stored in \Views\[ControllerName]
- All convention-based defaults can be overridden if desired

Introduction

Managing Dependencies

- Effective unit testing requires stable inputs and predictable outputs
- Requires developers to identify dependencies
- Provide a method for injecting dependencies at runtime
 - Constructor injection
 - Parameter injection
 - Property injection
 - Interfaces
 - Factory objects
 - Mock and Stub objects

Introduction

Managing Dependencies

- Follow the Single Responsibility Principle (SRP) for classes
- Use interfaces to avoid unnecessary assembly dependencies
- Avoid using singletons and static methods as much as possible
 - Static method calls introduce a dependency on a concrete type

Introduction

Managing Dependencies

- An Inversion of Control (IoC) container can be used to more formally support dependency injection
 - Ninject is one popular example

Case-Study Application

Socrates

- Manage information for a fictional training company
 - Departments, Courses, Instructors
- Use the latest and greatest technologies
 - ASP.NET MVC 5
 - Entity Framework 6 (Code First)
 - Bootstrap Framework
 - jQuery, jQuery UI, and Ajax
 - Web API 2

Lab I

Create a New ASP.NET MVC Project

- Create a new ASP.NET MVC project
- Examine the folder structure and starter files

ASP.NET MVC 5

Request Pipeline

- HTTP request received
- Check is done for a physical file that corresponds to the URL
 - If a file exists, it is returned
- If no physical file exists, the routing engine uses the URL and a list of rules to determine a destination controller
- An instance of the controller class is created and an appropriate action method is invoked
- The action method returns a result that is sent to the client

ASP.NET MVC 5

Routing

- Configuration
- RouteValueDictionary
- Route Constraints
- Catch-All Parameter
- Ignoring Routes
- Lab: Routing

Routing

Configuration

- In Global.asax.cs, Application_Start calls RegisterRoutes() of RouteConfig class
- RegisterRoutes is responsible for adding routes to the application's RouteTable
- It is possible to add routes using information obtained from an external source (e.g. separate XML file)
 - Avoid expensive operations (e.g. reading from a database) since this code will run whenever the application is restarted
 - Remember that a failure in creating the RouteTable will prevent the application from starting

Routing

Configuration

- Each route must have a unique name, URL and specify a controller name and an action name
- Route URLs are case-insensitive

```
routes.MapRoute(  
    name: "Default",  
    url: "test/hello",  
    defaults: new { controller = "Home", action = "Hello" }  
);
```

Routing

Configuration

- A route can specify that the controller name and action name should be obtained from the URL
- If the values are not supplied, the specified default values will be used

```
routes.MapRoute(  
    name: "Default",  
    url: "test/{controller}/{action}",  
    defaults: new { controller = "Home", action = "Hello" }  
);
```

Routing

Configuration

- Additional parameter names can be specified as part of the route
- Parameter value will be available when the controller's action is called

```
routes.MapRoute(  
    name: "Default",  
    url: "test/{controller}/{action}/{message}"  
);
```

```
public ActionResult Hello(string message)  
{  
    ...  
}
```


Routing

Configuration

- Route URLs can contain literal values
- Avoid characters that are invalid for a URL or need to be encoded
- Parameters with a default value must be listed after parameters that are required

```
site/{controller}/{action}/{id}
```

```
{language}-{country}/library/{controller}/{action}
```



```
{controller}{action}/{id}
```

Routing

Attribute Routing

- Instead of adding a route in RegisterRoutes, you can add an attribute to the controller action itself
 - Use whichever technique you prefer or use a mixture of both
- If using attribute routing, you must tell MVC to add the attribute-based routes by calling MapMvcAttributeRoutes in RegisterRoutes

```
[Route("product/{id}")]  
public ActionResult Details(int id) { ... }
```

```
routes.MapMvcAttributeRoutes();
```

Routing

Configuration

- Resource Not Found (404) response will be generated when...
 - URL does not match a physical directory or file and no route has a URL pattern that matches the request
 - Class cannot be found for the controller name provided
- Internal Server Error (500) response will be generated when...
 - Controller is instantiated but a method cannot be found for the action name provided
- Forbidden (403) response will be generated when...
 - The request corresponds to a physical directory but there is no default document and directory browsing is disabled

Routing

RouteValueDictionary

- Once a matching route for a request is found, an instance of a RouteValueDictionary is created and populated with information found in the request
- An entry must exist in the RouteValueDictionary for controller and action

```
routes.MapRoute(  
    name: "Default", url: "{controller}/{action}"  
);
```

```
routes.MapRoute(  
    name: "Route2", url: "foo.php",  
    defaults: new { controller = "Simple", action = "Test" }  
);
```

Routing

RouteValueDictionary

- Other URL components and query string values are added to the RouteValueDictionary
- Available to be received as action parameters

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

```
http://localhost/Course/Edit/5?name=Bill
```

```
public ActionResult Edit(int id, string name) { ... }
```

Routing

RouteValueDictionary

- RouteValueDictionary can be accessed directly via the RouteData property
- Avoid doing this whenever possible to avoid adding a dependency on RouteData
 - Makes unit testing more difficult

```
public ActionResult Index()  
{  
    string name = RouteData.Values["name"];  
}
```

Routing

Route Constraints

- Sometimes, you need more control over your URLs
- Constraints allow you to specify additional rules that must be met before a route is considered to be valid
 - Regular expression
 - IRouteConstraint

Route Constraints

Regular Expression

- Regular expression can be specified for parameters when adding a route
- Expressions are evaluated in the order they are defined
- If any expression fails, the route is considered invalid and the system moves on to the next route in the RouteTable

```
http://www.myblog.com/2016/02/15  
http://www.myblog.com/microsoft/aspnet/mvc
```

```
routes.MapRoute("BlogByDate", "{year}/{month}/{day}",  
    new { controller = "Blog", action = "ByDate" },  
    new { year = @"\d{4}", month = @"\d{2}", day = @"\d{2}" });  
  
routes.MapRoute("BlogByCategory", "{category}/{subcategory}/{topic}",  
    new { controller = "Blog", action = "ByTopic" });
```

Route Constraints

IRouteConstraint

- You can also implement a custom route constraint that executes code to determine if the parameter is acceptable
- To create a custom route constraint, create a class that implements IRouteConstraint

```
public interface IRouteConstraint
{
    bool Match(HttpContextBase httpContext, Route route,
                string parameterName, RouteValueDictionary values,
                RouteDirection routeDirection)
}
```

Route Constraints

IRouteConstraint

- To use a custom constraint, add it to the dictionary of constraints for a route

```
routes.MapRoute(
    name: "BlogByDate",
    url: "{year}/{month}/{day}",
    defaults: new { controller = "Blog", action = "ByDate" },
    constraints: new { year = @"\d{4}", month = @"\d{2}", day = @"\d{2}",
                      isValidDate = new IsValidDateConstraint() }
);
```


Route Constraints

IRouteConstraint

- If the constraint's Match method returns false for a request, the route is considered invalid

```
public bool Match(HttpContextBase httpContext, ...)
{
    if (routeDirection == RouteDirection.IncomingRequest)
    {
        DateTime date;
        return DateTime.TryParse(String.Format("{0}/{1}/{2}",
            values["day"], values["month"], values["year"]),
            CultureInfo.InvariantCulture, DateTimeStyles.None, out date);
    }
    return true;
}
```

Routing

Catch-All Parameter

- The catch-all parameter (identified using a asterisk) allows for a route to match a URL with an arbitrary number of parameters

```
routes.MapRoute(
    name: "CatchAllRoute",
    url: "query/{category}/{*extrastuff}",
    defaults: new { controller = "Search", Action = "Query" }
);
```

```
http://localhost/query/people/hr/managers
```

```
public ActionResult Query(string category, string extrastuff)
{
    // category -> "people"
    // extrastuff -> "hr/managers"
}
```

Routing

Ignoring Routes

- The IgnoreRoute method can be used to explicitly tell the routing system to avoid attempting to route certain URL patterns
- Can be used to improve the performance of requests made for static resources
- Necessary for requests that will involve dynamic processing outside of the MVC pipeline

```
routes.IgnoreRoute("images/{*pathInfo}");
```

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

Lab 2

Routing

- Add a simple action to HomeController
- Add a simple route
- Pass a parameter to an action
- Use a regular expression route constraint
- Use a custom route constraint

ASP.NET MVC 5

Controllers

- Introduction
- Conventions
- Request Pipeline
- Controller Class Hierarchy
- ActionResult
- Lab: Controller Actions and Return Types

Controllers

Introduction

- Controllers are responsible for responding to user requests
- May need to retrieve or make modifications to model objects
- Determines the appropriate response to return
 - HTML, JSON, XML, redirection, error, etc.

Controllers

Conventions

- Each controller class typically has a name that ends with the word Controller
 - For example, ProductController
- Controller classes are typically located in a project sub-folder (and namespace) named Controllers
- Controller classes typically inherit from a framework class named Controller
- ASP.NET MVC developers typically define one controller class for each type of model object

Controllers

Request Pipeline

- After the routing engine has populated the RouteValueDictionary, the value for the Controller key is used by a class that implements IControllerFactory (DefaultControllerFactory) to locate a class to instantiate
- DefaultControllerFactory requires that a class...
 - Be public and not abstract
 - End with the string Controller (case-insensitive)
 - Implement the IController interface

Controllers

Request Pipeline

- A controller class can exist in any namespace and in any assembly as long as it is visible from the MVC project
- `DefaultControllerFactory` will search the following namespaces (in this order):
 - Namespaces listed as part of the call to `MapRoute`
 - Namespace that corresponds to the MVC project's default Controllers folder
 - Other namespaces in the MVC project assembly
 - Other assemblies referenced by the MVC project

Controllers

Request Pipeline

- Although not common, it is possible to provide ASP.NET MVC with a different implementation of `IControllerFactory`
- Allows you to specify a different controller search algorithm

Controllers

Controller Class Hierarchy

- All controllers must implement the IController interface

```
public interface IController
{
    void Execute(RequestContext requestContext);
}
```

- Once the controller factory has created an instance of the class, the Execute method is invoked

Controller Class Hierarchy

ControllerBase Class

- ControllerBase is an abstract class that implements IController
- Provides properties for sending data to a View
 - ViewData and ViewBag

Controller Class Hierarchy

Controller Class

- Controller class inherits from ControllerBase and adds an implementation of the Execute method
 - Uses an instance of IActionInvoker (ControllerActionInvoker) to find and invoke the appropriate action

Controllers

ActionResult

- Controller actions can return many different types of results
- ActionResult is the base class for many specialized result types

ActionResult Type	Description
EmptyResult	Represents a null or empty response
ContentResult	Writes the specified content directly to the response as text
JsonResult	Serializes the objects it is given into JSON
RedirectResult	Redirects the user to the given URL
HttpNotFoundResult	Redirects to a URL specified via Routing parameters
ViewResult	Calls into a View engine to render a View
PartialViewResult	Renders a partial View to the response
FileResult	Writes a binary response to the stream

ActionResult

Convenience Methods

- The Controller class defines several convenience methods that return ActionResult instances
- Generally named after the action result type
- Redirect, RedirectToAction, RedirectToRoute, View, PartialView, Content, File, Json, JavaScript

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

```
public ActionResult Course(int id)  
{  
    Course c = db.GetCourse(id);  
    return Json(c);  
}
```

ActionResult

Implicit Action Results

- When the return type does not derive from ActionResult, ASP.NET MVC automatically instantiates a ContentResult
 - For non-string types, ToString() is called

```
public int Sum(int x, int y)  
{  
    return x + y;  
}
```

```
public ActionResult Sum(int x, int y)  
{  
    int retVal = x + y;  
    return Content(retVal.ToString());  
}
```


ActionResult

Best Practice

- It is generally considered a best practice to define all of your actions with a return type of ActionResult
- Allows you to easily return a different sub-type when certain conditions are met

```
public ActionResult Details(int id)
{
    Person p = db.People.SingleOrDefault(p => p.Id == id);
    if (p == null) return HttpNotFound();

    return View(p);
}
```

Lab 3

Controllers

- Use ContentResult to return text from an action
- Return HTML from an action
- Return JSON from an action

ASP.NET MVC 5

Models

- Introduction
- Hands-On Lab Exercise
- Object-Relational Mapping

Models

Introduction

- In an MVC application, the model objects represent the data the user is interacting with
- Typically retrieved by a Controller object and formatted for presentation to the user by a View
- Model objects should be highly reusable and testable
- Often, it is helpful to define model objects in a separate assembly

Lab 4

Models

- Add some model objects to the project

ASP.NET MVC 5

Data Access

- Persistence Ignorance
- Object-Relational Mapping (ORM)
- Entity Framework
- Lab: Data Access

Data Access

Persistence Ignorance

- It is typically a good idea to have clean separation (loose coupling) between your data access code and the rest of your application's code
- Your controllers should not know where the model objects come from or how they are persisted
- This separation allows your data storage mechanism and your web application to vary independently
- The use of interfaces and factories is key to achieving this separation

Data Access

Object-Relational Mapping

- Very often, model data is stored in a relational database
 - SQL Server, Oracle, MySQL, PostgreSQL, etc.
- A necessary task is transforming that relational data into object-oriented data structures and back again
- Doing the work manually using ADO.NET, SQL, and C# can provide for maximum flexibility but can also result in:
 - Lots of code
 - Frequent data type conversions
 - No design-time support when writing SQL

Data Access

ADO.NET and SQL

```
public IEnumerable<Person> FetchAllPeople
{
    var people = new List<Person>();
    Person person;
    var conn = new SqlConnection(connStr);
    var cmd = new SqlCommand("SELECT * From People", conn);
    SqlDataReader rdr;
    try {
        conn.Open();
        using (rdr = cmd.ExecuteReader()) {
            while (rdr.Read()) {
                person = new Person();
                person.FirstName = rdr["FirstName"] As String;
                person.LastName = rdr["LastName"] As String;
                people.Add(person);
            }
            ...
        }
    }
}
```

Data Access

ORM Frameworks

- Entity Framework
 - Version 6 now available
 - Many shortcomings of earlier versions have been addressed
 - Database vendors responsible for creating a provider with EF support (sometimes, not all features supported)
- NHibernate
 - .NET equivalent of the Java Hibernate framework
 - Strong support for all popular database servers
 - Some features not well documented

Data Access

ORM Frameworks

- Regardless of the ORM framework chosen, it is important to understand how the framework does its job
 - It's easy to write inefficient data access code
 - You should have a strategy for handling changes to the model structure in future versions of your application
 - You still have to think about concurrency conflicts and error handling

Data Access

Entity Framework

- Version 1 (VS 2008 SP1)
 - Supported an approach called Database First
- Version 4 (VS 2010)
 - Added Model First approach, POCO support, and default lazy loading
- Version 4.1 - 4.3 (VS 2010)
 - Added the Code First approach

Data Access

Entity Framework

- Version 5 (VS 2012)
 - Performance improvements
 - Support for enums, table value functions, spatial types, batch import of stored procedures, and rich support for ASP.NET MVC
- Version 6 (VS 2013)
 - Asynchronous support for querying and updates
 - Stored procedure support for updates in Code First
 - Now open source
 - entityframework.codeplex.com

Data Access

Entity Framework

- Use NuGet to obtain the most recent version of Entity Framework and add a reference to it
- Use the NuGet Package Manager Console to upgrade an existing reference to Entity Framework

```
PM> Update-Package EntityFramework
```

Data Access

Data Annotations

- Data annotations can be used to annotate model properties
 - Examples include Required, StringLength, Range, and RegularExpression

```
public class Course
{
    [Required]
    public string Number
```

- Other parts of EF and MVC will notice these annotations and perform other actions
 - For example, generate client-side data validation code

Data Access

Entity Framework

- When using EF, application startup time will be longer while your model is initialized
- One way to mitigate this is to use the IIS 8 Application Initialization feature (Application Warm-up module in IIS 7)
 - Allows for startup tasks to be performed prior to the first user request

Lab 5

Data Access

- Add a reference to Entity Framework
- Add data access code to the application
- Add a connection string to Web.config
- Use a data annotation to identify required fields
- Write a unit test for SocratesContext
- Add some test data to the database

Lab 6

Data Access

- Import some sample data

ASP.NET MVC 5

Views

- Introduction
- Implementation
- View Engines
- Razor Syntax
- View Templates
- Bootstrap
- Layouts
- Lab: Defining Views

Views

Introduction

- In MVC, the View is responsible for providing the user interface to the client
- Transforms model data into a format for presentation to the user
 - Most often HTML but a view can render other formats as well

Views

Implementation

- In ASP.NET MVC, View template files are parsed and converted into a class that implements IView

```
public interface IView
{
    void Render(ViewContext viewContext, TextWriter writer);
}
```

Views

Implementation

- View receives a ViewContext that has properties that provide access to the controller's ViewData and ViewBag properties

```
public ActionResult Index()
{
    ViewData["Title"] = "Cool Page";
    return View();
}
```

```
<h1>@ViewData["Title"]</h1>
```

Views

Implementation

- The default implementation of `IView` in ASP.NET MVC is `WebViewPage`
- Defines properties that provide access to the model, HTTP context, HTML helpers, and Ajax helpers

Views

View Engines

- The component responsible for locating and instantiating a view is called a view engine
- When a view is needed, all registered `IViewEngine` implementations are queried based on the view name
- A view engine returns a `ViewEngineResult` that contains an `IView` instance if a view is found

```
public interface IViewEngine
{
    ViewEngineResult FindPartialView(ControllerContext controllerContext,
        string partialViewName, bool useCache);

    ViewEngineResult FindView(ControllerContext controllerContext,
        string viewName, string masterName, bool useCache);
}
```

View Engines

Razor

- Default view engine in ASP.NET MVC 5 is the Razor view engine
- ASP.NET MVC 5 does still support the Web Forms view engine but VS 2013 (and later) does not provide design-time support
- Several third-party view engines are also available
 - Spark
 - NHaml
 - NVelocity

View Engines

View Naming Convention

- The default view engine behavior is to look for a view with the same name as the action located in a folder under Views with the same name as the controller
- It is possible to specify a different view name or a full path

```
return View("AnotherView");
```

```
return View("~/Views/Stuff/SomeOtherView");
```

View Engines

Custom View Engine

- There are times when creating custom view engine or a subclass of an existing view engine may be desired
- For example, to override the logic used for finding a view (location, name, extension)

```
protected void Application_Start()
{
    ViewEngines.Engines.Clear();
    ViewEngines.Engines.Add(new MyCustomViewEngine());
}
```

Views

Razor Syntax

- Razor provides a streamlined syntax for expressing views
- Minimizes the amount of syntax and extra characters
- Understands the transitions between code and markup

```
<ul>
<% foreach (Course course in Model) { %>
    <li><%= course.Number %> is named <%= course.Title %>.</li>
<% } %>
</ul>
```

```
<ul>
@foreach (Course course in Model) {
    <li>@course.Number is named @course.Title.</li>
}
</ul>
```

Razor Syntax

Code

- The key transition character in Razor is @
- Used to transition from markup to code
- Razor parser uses a look-ahead algorithm to determine the end of a code expression
- Visual Studio editor displays text Razor interprets as code with a darker background color

Razor Syntax

Code Expressions

- Sometimes, Razor needs a little help to identify the end of a code expression
- Use @(to force Razor to interpret all text as code until it encounters the closing parenthesis

```
<span>@course.Price * 0.10</span>
```

```
<span>@(course.Price * 0.10)</span>
```

Razor Syntax

Escaping

- To render @ into the response when Razor thinks it's code, escape the character with a second one

```
<span>Follow @acme on Twitter</span>
```

```
<span>Follow @@acme on Twitter</span>
```

Razor Syntax

HTML Encoding

- The output from a Razor expression is automatically HTML encoded
- Disable using `Html.Raw()`

```
<span>@Html.Raw(course.Description)</span>
```


Razor Syntax

Code Blocks

- A stand-alone code block can be specified using { }
- Statements within a stand-alone block must end with a semi-colon

```
@{  
    int i = 5;  
    i++;  
}
```

Views

View Templates

- Visual Studio provides a selection of scaffold templates to choose from when creating a view
 - Uses the Visual Studio T4 templating system to generate a view based on a selected model type
- The option to reference script libraries refers to the jQuery validation and unobtrusive libraries
 - Useful for views that include a form

Views

View Templates

- Default templates provided by T4 templates
 - %VS_DIR%\Common7\IDE\ItemTemplates\CSharp\Web
- Visual Studio will also look for a CodeTemplates folder in the root of your project
- Other tools and resources exist for helping you create your own custom templates

Views

Bootstrap

- Starting with ASP.NET MVC 5, the built-in view templates use the Bootstrap framework
- Created by a a designer and a developer at Twitter in 2010 and released to the public in August 2011, Bootstrap is now an independent open source project
 - getbootstrap.com
- Bootstrap has quickly become one of the most popular front-end frameworks and open source projects in the world

Bootstrap

Components

- Bootstrap is composed of CSS and JavaScript
 - bootstrap.css and bootstrap.js
 - Minimized versions and CDN also available
- Includes a grid system that uses CSS media queries and allows you to easily create a responsive page design
- Includes a collection of components for drop downs, navigation bars, alerts, progress bars, panels, and more
- Includes jQuery plug-ins for modal dialogs, tabs, popovers, carousels, and more

Bootstrap

Grid System

- Bootstrap's grid system is used for creating page layouts through a series of rows and columns that house your content
- Rows must be placed within a .container for proper alignment and padding
- Columns are created by specifying the number of twelve available columns you wish to span
 - Different column span values can be provided for different container widths

Bootstrap

Grid System

	Extra small (phone)	Small (tablet)	Medium (desktop)	Large (desktop)
Container width	< 768 px	>= 768 px	>= 992 px	>= 1200 px
Class prefix	.col-xs-	.col-sm-	.col-md-	.col-lg-

Bootstrap

Grid System

```
<div class="container">
  <div class="row">
    <div class="col-md-4">
      <h2>Getting started</h2>
      <p>ASP.NET MVC gives you a powerful, ...</p>
    </div>
    <div class="col-md-4">
      <h2>Get more libraries</h2>
      <p>NuGet is a free Visual Studio extension ...</p>
    </div>
    <div class="col-md-4">
      <h2>Web Hosting</h2>
      <p>You can easily find a web hosting company ...</p>
    </div>
  </div>
</div>
```

Views

Layouts

- Layouts help maintain a consistent look and feel across multiple views
- Defines a common template for some or all of your views
- Call to `RenderBody()` in a layout marks the location where the content of the view will be rendered

```
<div class="container body-content">  
  @RenderBody()  
  <hr />  
  <footer>  
    <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>  
  </footer>  
</div>
```

Layouts

Specifying

- A layout for a view can be specified in the view itself or can be specified for a collection of views using a `_ViewStart` file
- Code within a `_ViewStart` file is executed before the code in any view within the same folder

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

Layouts

Sections

- A layout can define other sections
- Views based on the layout define the content for a section
- Can be required or optional

```
<html>
<head>
  <title>@ViewBag.Title</title>
</head>
<body>
  @RenderBody()
  @RenderSection("Footer")
</body>
</html>
```

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

<p>This is the main content</p>

@section Footer {
    <span>The footer!</span>
}
```

Lab 7

Views

- Modify the homepage for the application

ASP.NET MVC 5

Helpers

- Introduction
- HTML Helpers
- `Html.ActionLink`
- `Url.Action`
- Strongly-Typed Helpers
- Custom Helpers
- Inline Razor Helpers
- Lab: HTML Helpers

Helpers

Introduction

- ASP.NET MVC provides a collection of helper methods by defining extension methods for classes `HtmlHelper`, `UrlHelper`, and `AjaxHelper`
- Make it easier to generate common pieces of view content and help with maintenance by dynamically generating content based on things like the routing configuration or model fields
- Properties of `WebViewPage` make it easy to access helpers

```
public HtmlHelper<object> Html { get; set; }  
public AjaxHelper<object> Ajax { get; set; }  
public UrlHelper Url { get; set; }
```

Helpers

HTML Helpers

Helper	Description
ActionLink	Renders a hyperlink element
BeginForm	Marks the start of a form
CheckBox	Renders a check box
DropDownList	Renders a drop-down list
Hidden	Renders a hidden form field
ListBox	Renders a list box
Password	Renders a text box for entering a password
RadioButton	Renders a radio button
TextArea	Renders a text area (multi-line text box)
TextBox	Renders a text box

Helpers

Html.ActionLink

- One example of an HTML helper is the ActionLink method
- Allow you to supply an action name and controller name
- Generates the correct URL based on the application's RouteTable
- Makes maintenance easier since links do not have to be updated when the routing configuration changes

```
@Html.ActionLink("Create New", "Create", "Course")
```



```
<a href="/Course/Create">Create New</a>
```


Helpers

Url.Action

- Sometimes, it is preferable to use a helper to generate only the URL and not the entire link element

```
<a class="btn" href="@Url.Action("Index", "Department")">Learn more</a>
```

Helpers

Strongly-Typed Helpers

- More advanced helpers will sometimes accept a lambda expression to specify a model property
- All strongly-typed helpers end with For
- Some helpers can dynamically choose the HTML element to use based on the type of the model property

```
<div>  
    @Html.LabelFor(model => model.FirstName)  
</div>  
<div>  
    @Html.EditorFor(model => model.FirstName)  
    @Html.ValidationMessageFor(model => model.FirstName)  
</div>
```

Helpers

Additional HTML Attributes

- Many helpers provide the ability to specify additional attributes to be rendered as part of the HTML

```
@Html.ActionLink("Click Me", "Details", null, new { attr = "val" })
```



```
<a href="/Course/Details" attr="val">Click Me</a>
```

Helpers

Additional HTML Attributes

- Attribute names that are also C# keywords need to be specified using @

```
@Html.ActionLink("Click Me", "Details", null, new { @class = "cool" })
```



```
<a href="/Course/Details" class="cool">Click Me</a>
```

Helpers

Custom Helpers

- To create a custom HTML helper, define a new extension method for `HtmlHelper`, `UrlHelper`, or `AjaxHelper`
- Extension methods are static methods in a static class that use the `this` keyword to tell the compiler the type being extended
- If generating HTML, return an `HtmlString` to prevent automatic encoding by the view engine

```
public static HtmlString Image(this HtmlHelper html,
                              string src, string alt)
{
    string str = String.Format("<img src=\"{0}\" alt=\"{1}\" />",
                              src, alt);
    return new HtmlString(str);
}
```

Helpers

Inline Razor Helpers

- Razor offers the ability to write custom helpers in a `cshtml` file using Razor syntax
- Allows for simple reuse without having to define an extension method
- Can be reused across multiple views by placing in `App_Code`

```
@helper DisplayPrice(Decimal price) {
    if (price == 0) {
        <span>FREE!</span>
    }
    else {
        @String.Format("{0:C2}", price)
    }
}
```

```
<li>
    @product.Name for
    @DisplayPrice(product.Price)
</li>
```

Lab 8

HTML Helpers

- Modify the homepage view so links are generated using `Url.Action`

ASP.NET MVC 5

Strongly-Typed Views

- Introduction
- Creating
- Passing Model Objects
- Lab: Strongly-Typed View
- Controller Inheritance
- Lab: Controller Base Class

Strongly-Typed Views

Introduction

- WebViewPage contains several properties used for accessing data provided by a controller

```
public ViewDataDictionary ViewData { get; set; }  
public dynamic ViewBag { get; }  
public object Model { get; }
```

Strongly-Typed Views

ViewData and ViewBag

- ViewData is a dictionary
- ViewBag is a dynamic variable that provides access to ViewData
- The two can be used interchangeably (see below)

```
public ActionResult Index()  
{  
    ViewData["Something"] = "Some Data";  
    ViewBag.SomethingElse = "Some Other Data";  
}
```

```
<h1>@ViewData["SomethingElse"]</h1>  
<h1>@ViewBag.Somthing</h1>
```

Strongly-Typed Views

ViewData and ViewBag

- A major disadvantage of ViewData is the the lack of design-time support when creating views (Intellisense)
- A better approach would be to specify a type for the view's model object
 - Visual Studio will ensure the correct type is provided (compile-time error) and you will have Intellisense support when accessing model properties in a view

Strongly-Typed Views

Creating

- To create a strongly-typed view, use a @model directive as the first line of code in the view
- To be able to omit the namespace, add a using element to the Web.config file in the Views folder

```
@model Course
```

```
<namespaces>  
  <add namespace="Socrates.Models" />  
</namespaces>
```

Strongly-Typed Views

Passing Model Objects

- Use the View convenience method to pass a model object to a strongly-typed view
- Use the Model property of WebViewPage to access the object in the view

```
public ActionResult Details(int id)
{
    Course course = db.GetCourse(id);
    return View(course);
}
```

```
<h1>@Model.Title</h1>
```

Lab 9

Strongly-Types Views

- Create DepartmentController
- Create an Index action
- Define the Index view as a strongly-typed view
- Pass the collection of departments to the view
- Display the collection of departments as a bulleted list

ASP.NET MVC 5

Controller Inheritance

- Very often, you will have common code in multiple controllers
- It can be beneficial to move common code into a controller base class from which your other controllers can inherit
- The Initialize method can be overridden to perform required tasks before the controller handles a request

Lab 10

Controller Inheritance

- Move common controller code into a base class
- Override the controller's Initialize method

ASP.NET MVC 5

HTML Forms

- Introduction
- Helpers
- Lab: Edit Form
- Action Selectors
- Model Binding
- Lab: Handling a POST

HTML Forms

Introduction

- An HTML form allows a user to submit information to the server
- Action attribute specifies where to send the information
- Method attribute specifies whether to submit the information via an HTTP GET or POST
 - GET is appropriate for idempotent, read-only type operations
 - POST should be used for operations that modify state on the server

```
<form action="http://www.bing.com/search" method="GET">  
  <input name="q" type="text" />  
  <input type="submit" value="Search!" />  
</form>
```

HTML Forms

Helpers

- In ASP.NET MVC, many helpers exist to assist with creating forms

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Department</h4>
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.Id)
        <div class="form-group">
            @Html.LabelFor(model => model.Name)
            <div class="col-md-10">
                @Html.EditorFor(model => model.Name)
                @Html.ValidationMessageFor(model => model.Name)
            </div>
        </div>
    </div>
}
```

Helpers

Html.BeginForm

- The BeginForm helper will generate the opening <form> tag
- If no parameters are specified, the form will POST to the same action that returned the form
- Returns an object that implements IDisposable
 - Can be using with the C# using statement
 - Closing </form> tag rendered when the block completes

Helpers

Html.AntiForgeryToken

- The AntiForgeryToken helper generates a hidden form field and a cookie with a random token value
- Used to prevent cross-site request forgery (CSRF)
- Action that receives the form submission must include the ValidateAntiForgeryToken attribute

```
[ValidateAntiForgeryToken]  
public ActionResult Edit(FormCollection collection)  
{  
    ...  
}
```

Helpers

Html.LabelFor

- Html.LabelFor displays a name for a property
- Will use the value of a the Display attribute if present in the model

```
public class Course  
{  
    [Display(Name="Available")]  
    public bool IsActive { get; set; }  
    ...  
}
```

Helpers

Validation

- `Html.ValidationSummary` and `Html.ValidationMessageFor` are used to display messages when validation rules are violated
- More on this later

Lab 11

HTML Forms

- Create an action and view for editing a department

HTML Forms

Action Selectors

- An action selector is an attribute that helps the action invoker determine which action to invoke for a request
- One action selector allows you to specify an HTTP verb
- Allows for two controller actions with the same name
 - One to deliver the form and one to receive the submission

```
public ActionResult Edit(int id) { ... }  
  
[HttpPost]  
public ActionResult Edit(FormCollection collection) { ... }
```

HTML Forms

Model Binding

- When a form is submitted, the model binding system attempts to populate the parameters of the action with values in the request
 - URL parameters
 - Query string values
 - Form fields

HTML Forms

Model Binding

- If an action accepts an object parameter, the model binding system will create an instance of that class and attempt to populate its public properties with values from the request
- If validation errors occur during the model binding process, the IsValid property of the ModelState property with return false

```
public ActionResult Edit(Department department)
{
    if (ModelState.IsValid) { ... }
    ...
}
```

HTML Forms

Model Binding

- Sometimes, it is desirable to accept the incoming data differently
 - As separate field values
 - As a FormCollection
- Provides some flexibility in handling the data
- Direct use of the Request object should be avoided
 - Introduces a dependency that makes the controller more difficult to write unit tests for

```
public ActionResult Edit(string number, string title)
```

```
public ActionResult Edit(FormCollection collection)
```

HTML Forms

Model Binding

- Model binding can be done explicitly using the UpdateModel and TryUpdateModel methods
- UpdateModel will throw an exception if a validation error is encountered
- TryUpdateModel will return a boolean value

```
public ActionResult Create(FormCollection collection)
{
    var dept = new Department();
    if (TryUpdateModel(dept))
    {
        // save to database
    }
    ...
}
```

HTML Forms

Model Binding

- UpdateModel and TryUpdateModel will use the Request object directly unless the source of the data is specified
- Source must implement IValueProvider (FormCollection does)

```
public ActionResult Create(FormCollection collection)
{
    var dept = new Department();
    if (TryUpdateModel(dept, collection))
    {
        // save to database
    }
    ...
}
```

Lab 12

HTML Forms

- Create an action for the submission of the department edit form

Lab 13

HTML Forms

- Add the ability to create a new department

ASP.NET MVC 5

Partial Views and Child Actions

- Partial Views
- Lab: Partial Views
- Child Actions
- Lab: Course List and Details

Partial Views and Child Actions

Partial Views

- Some views can be designed as a partial view
- Useful for separating and reusing parts of a view
- A partial view can be strongly-typed
- Used by other views via `Html.Partial()`
- Some actions may return a partial view for consumption by client-side AJAX code
- Popular convention in MVC 5 is to name partial views starting with an underscore and end with the word `Partial`
- Partial views in the `Shared` folder can be used by all controllers

Lab 14

Partial Views

- Use a partial view to eliminate duplicate view code

Partial Views and Child Actions

Child Actions

- In addition to using the results of a partial view, a view can also inject the results of another controller action
- Performed using `Html.Action()`
- Action may return any content that can be rendered in the view including a `PartialViewResult`
- `ChildActionOnly` attribute can be used to restrict access to an action so it cannot be invoked directly via a URL

```
<div>
  The current time is:
  @Html.Action("GetTime")
</div>
```

```
[ChildActionOnly]
public ActionResult GetTime()
{
    return Content(DateTime.Now);
}
```

ASP.NET MVC 5

Display and Edit Annotations

- Other attributes are available to make additional information available to HTML helpers and other components

Attribute	Description
Display	Sets a "friendly" name used by DisplayNameFor and LabelFor
ScaffoldColumn	Prevents a property from being included in scaffold templates
DisplayFormat	Specify a format string for a property value (ex. dates)
ReadOnly	Prevent the model binder from trying to update a property
DataType	Additional information for editor helpers (ex. MultilineText)
UIHint	Name of the template to use with helpers like EditorFor. Allows for the use of custom templates.
HiddenInput	Tells helpers to render property using a hidden form field

Lab 15

Courses

- Create CourseController
- Create an action and a view to display the list of courses
- Implement the ability to view the details for a course

ASP.NET MVC 5

ViewModels

- Introduction
- Model Binding
- Inheritance
- Lab: ViewModel

ViewModels

Introduction

- Sometimes, a model and a view do not match up exactly
 - View may require more data than is present in the model
 - View may only be displaying a portion of the model object
- Additional data could be sent using ViewData/ViewBag
- Another option is to create an additional object that is specifically designed to be the model for the view

ViewModels

Introduction

- Common uses of a ViewModel include...
 - Multiple model objects of different types
 - Model object plus property value choices (select lists)
 - Addition of application-specific artifacts (Remote attribute)

ViewModels

Model Binding

- If your ViewModel has a nested object, HTML helpers displaying data for that object will prepend the name of the parent object when choosing a form field name
- In your POST action, you can specify the prefix
 - Using an attribute on the action parameter
 - As a parameter when calling UpdateModel/TryUpdateModel

```
public ActionResult Edit([Bind(Prefix="Course")]Course course)
```

```
if (TryUpdateModel(course, "Course", collection)) { ... }
```

ViewModels

Inheritance

- A very powerful use of ViewModels is to create a hierarchy of ViewModel classes that match your view hierarchy
- Define your layout view to be strongly-typed to your ViewModel base class
- Define the views based on your layout to be strongly-typed to a subclass of the layout's ViewModel

Lab 16

ViewModels

- Implement the ability to edit a course
- Use a ViewModel to provide the course and list of departments to the view

ASP.NET MVC 5

Data Validation

- Introduction
- Validation Attributes
- ModelState
- IValidatableObject
- Lab: Data Validation
- Remote Validation

Data Validation

Introduction

- When receiving input from a user, it is important to validate the input
- Client-side validation provides a good user experience and reduces server traffic
- Server-side validation is always necessary
 - Client-side validation is easily circumvented
 - Complex validation is sometimes difficult to accomplish on the client
- Client-side validation that uses server-side data via an Ajax call can be a powerful option

Data Validation

Validation Attributes

- Several attributes related to data validation are available in `System.ComponentModel.DataAnnotations`
- `Required`, `StringLength`, `RegularExpression`, `Range`

```
[Required]  
[RegularExpression(@"[A-Z]{3}-\d{3}")]  
public string Number { get; set; }  
  
[Required]  
public string Title { get; set; }
```

Data Validation

Validation Attributes

- When you use a helper in a view, the helper will check for validation attributes
- If present, the helper will generate markup that uses the unobtrusive JavaScript and the jQuery Validate plug-in to provide client-side validation

```
<input data-val="true"  
       data-val-required="The Number field is required."  
       id="Course_Number" name="Course.Number" type="text"  
       value="MBL-120" />
```


Data Validation

Validation Attributes

- Every validation attribute allows you to provide a custom error message
- Also allows you to specify a resource type and name for localized error messages

```
[Required(ErrorMessage="Must Provide a Course Number")]  
public string Number { get; set; }
```

```
[Required(ErrorMessageResourceType=typeof(ErrorMessages),  
           ErrorMessageResourceName="CourseNumRequired")]  
public string Number { get; set; }
```

Data Validation

Model State

- On the server-side, the model binding system also looks for validation attributes
- When a validation rule is violated, an error object created and added to an object called ModelState
- When errors are present, ModelState.IsValid returns false
- Errors can also be added to ModelState manually in a controller action

```
ModelState.AddModelError("Number", "Invalid Course Number");
```

Data Validation

Model State

- ValidationMessageFor and ValidationSummary helpers examine the ModelState property when rendering output
- Visual appearance of output defined using CSS

```
@Html.ValidationSummary(true)
```

```
@Html.ValidationMessageFor(model => model.Number)
```

Data Validation

Custom Validation

- The model binder will also check if the target type implements IValidatableObject
- If so, model binder will call Validate as part of the process
- Errors returned from call to Validate are added to ModelState
- Helpful for implementing server-side equivalent of custom client-side validation

```
public interface IValidatableObject
{
    IEnumerable<ValidationResult> Validate(ValidationContext
                                         validationContext)
}
```

Lab 17

Data Validation

- Add data validation for editing a course
- Use a validation attribute
- Implement custom server-side validation

Data Validation

Remote Validation

- ASP.NET MVC defines a Remote attribute that enables client-side validation with a server callback
- The callback should return a boolean value in JSON format

```
[Remote("CheckNumber", "Course")]  
public string Number { get; set; }
```

```
public ActionResult CheckNumber(string number)  
{  
    bool b = context.Courses.Where(c => c.Number == number).Count() > 0;  
    return Json(b, JsonRequestBehavior.AllowGet);  
}
```

ASP.NET MVC 5

jQuery and Ajax

- jQuery
- Unobtrusive JavaScript
- Ajax Helpers
- Lab: Asynchronous Form

jQuery and Ajax

jQuery

- jQuery makes it easy to find, traverse, and manipulate HTML elements
- Easy to wire up event handlers, perform animations, and build AJAX interactions
- MVC project templates include jQuery in the Scripts folder by default
- The jQuery function is used to gain access to jQuery features
 - Named jQuery but also aliased to the \$ sign

jQuery and Ajax

jQuery Selectors

- Selectors are the strings you pass to the jQuery function to select elements in the DOM

Example	Meaning
<code>\$("#header")</code>	Find the element with an id of "header"
<code>\$(".editor-label")</code>	Find all elements with a class name of "editor-label"
<code>\$("div")</code>	Find all <div> elements
<code>\$("#header div")</code>	Find all <div> elements that are decedents of the element with an id of "header"
<code>\$("#header > div")</code>	Find all <div> elements that are children of the element with an id of "header"

jQuery and Ajax

Unobtrusive JavaScript

- Practice of keeping JavaScript code separate from markup
- Follows the same separation of concerns approach the MVC pattern promotes
- Keeping all of your script in separately downloaded files can increase performance by allowing for browser caching
- Ajax helpers in ASP.NET MVC use metadata attributes and external script files to attach behavior to elements
- Supports progressive enhancement

```
<a data-ajax="true" data-ajax-update="#Test"
href="/Home/NextEngagement">Show Next Engagement</a>
```

jQuery and Ajax

Ajax Helpers

- Ajax helpers, like HTML helpers, help to easily generate page markup
- Build on or extend the functionality provided by jQuery
- Behave asynchronously
- Depend on the Microsoft jQuery Unobtrusive Ajax plugin
 - jquery.unobtrusive-ajax.js
 - Available via NuGet

Ajax Helpers

Ajax.ActionLink

- The Ajax.ActionLink helper method is similar to Html.ActionLink
- Instead of performing a standard navigation, it makes an Ajax call to the action method
- A LoadingElementId can be specified to provide some feedback to the user

```
@Ajax.ActionLink("Click here to get the current time",  
    "ServerTime",  
    new AjaxOptions  
    {  
        UpdateTargetId = "timeDisplay",  
        HttpMethod = "GET",  
        InsertionMode = InsertionMode.Replace,  
        LoadingElementId="progress"  
    })
```

Ajax Helpers

Ajax.ActionLink

- AjaxOptions also provides properties you can use to specify a JavaScript callback function to run in response to events that occur during an Ajax request
 - OnBegin
 - OnComplete
 - OnSuccess
 - OnFailure

Ajax Helpers

Ajax.BeginForm

- The Ajax.BeginForm helper method provides the ability to easily perform asynchronous form submissions

```
@using (Ajax.BeginForm("Search", new AjaxOptions {  
    HttpMethod = "GET",  
    InsertionMode = InsertionMode.Replace,  
    UpdateTargetId = "searchResults" }))
```

Lab 18

Ajax

- Add the ability to search for courses on the homepage
- Use Ajax to perform the search and display the results without reloading the page

ASP.NET MVC 5

jQuery UI

- Introduction
- DatePicker Widget
- Lab: DatePicker Widget
- Autocomplete Widget
- Lab: Autocomplete Widget
- Other Front-End Frameworks

jQuery UI

Introduction

- jQuery UI is a jQuery plugin that includes effects and widgets
 - jqueryui.com
 - Also available as a package via NuGet
- Relies on themes to provide presentation details
 - Includes a style sheet and images
 - NuGet package adds a "base" theme in the Content directory
 - Additional themes and a "ThemeRoller" utility are available on the jQuery UI web site

jQuery UI

Datepicker Widget

- The datepicker widget can be attached to an input field to provide a more pleasant user experience

```
$(function() {  
    $("#datefield").datepicker();  
});
```

```
<p>Date: <input type="text" id="datefield"></p>
```

Lab 19

Datepicker Widget

- Add a jQuery UI pop-up calendar for course availability date

jQuery UI

Autocomplete Widget

- Obtains suggestions from a service via AJAX and displays them to the user
- Can send the value of the input field as a query string value named term to a remote service set as source
- Expects to receive a JSON array of strings or an array of objects with a label and value property

```
$(document).ready(function () {  
    $("#searchField").autocomplete(  
        source:@Url.Action("QuickSearch", "Course");  
    });  
});
```

Lab 20

Autocomplete Widget

- Use a jQuery UI autocomplete widget to add autocomplete functionality to the course search form

jQuery UI

Other Front-End Frameworks

- Since views in ASP.NET MVC can be cleanly separated from other aspects of the application, other front-end frameworks such as Angular and React can also be used

ASP.NET MVC 5

Action Filters

- Introduction
- OutputCache
- HandleError
- RequireHttps
- Authorization
- Global Filters
- Custom Filters
- Lab: Custom Filter

Action Filters

Introduction

- Action filters can be used to inject functionality before and/or after an action method executes
- Several filters are built-in to ASP.NET MVC
- It is also possible to create custom filters

Action Filters

OutputCache

- The OutputCache filter is used to cache the output of an action method
- Can be very flexible if combined with actions that return partial views

```
[OutputCache(Duration=60, VaryByParam="none")]  
public ActionResult Index()  
{  
    return View();  
}
```

Action Filters

HandleError

- The HandleError filter is the default Exception filter included with ASP.NET MVC
- Use it to specify an exception type to handle and a View to display if an action method throws an unhandled exception
- If no exception type is specified, it handles all exceptions
- If no View is specified, it defaults to a View named Error

```
[HandleError(ExceptionType=typeof(ArgumentException), View="ArgError")]  
public ActionResult GetProduct(string name)  
{ ... }
```

Action Filters

RequireHttps

- The RequireHttps filter can be applied to action methods and controllers to prevent non-SSL access
- If a non-HTTPS GET request is received, it will be redirected to the HTTPS equivalent URL
- If a non-HTTPS request other than GET is received, an InvalidOperationException will be thrown

```
[RequireHttps]
public ActionResult Index()
{
    return View();
}
```

Action Filters

Authorization

- You can use the Authorize filter to restrict access to actions
- Can be applied at the action level, controller level, or application-wide
- By default, it forbids anonymous access but allows all others
- Specific user and role names can be provided

```
[Authorize]
public class EngagementController
{
    public ActionResult Index() { ... }

    [Authorize(Roles="Admin")]
    public ActionResult Delete(int id) { ... }
}
```

Action Filters

Global Filters

- Filters can be applied for all actions within the application by using the `RegisterGlobalFilters()` method in `FilterConfig.cs`

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());
}
```

Action Filters

Custom Filters

- If you have a need to hook in some custom logic before and after an action method is executed, you can implement a custom filter
- Create a class that inherits from `ActionFilterAttribute` and override the appropriate methods
 - `OnActionExecuting`: Called after the action method has been located by the invoker but before it is called
 - `OnActionExecuted`: Called after the action is executed

Action Filters

Custom Filters

- After the model binding and validation process is complete, `OnActionExecuting` is invoked for each action filter applied to the action
- If none of the filters cause an `ActionResult` to be returned then the method is invoked with the appropriate parameter values
- If the method returns something other than a `ViewResult` then `OnActionExecuted` is invoked for each filter and the response is returned

Lab 21

Custom Filter

- Create a custom filter
- Apply custom filter to a controller
- Register a custom filter as a global filter

ASP.NET MVC 5

Asynchronous Controller Actions

- Introduction
- Example
- Timeouts

Asynchronous Controller Actions

Introduction

- ASP.NET MVC makes available an asynchronous request pipeline via AsyncController
- Allows the web server to handle long-running requests while remaining responsive to other requests
 - Worker process threads are able to be returned to the thread pool for the processing of other requests
- Useful when operations are IO-bound rather than CPU-bound
 - External service calls, etc.
- Can be used to perform multiple operations in parallel

Asynchronous Controller Actions

Example (Synchronous)

```
public ActionResult Search()
{
    string url = "http://search.twitter.com/search.atom?" +
        "q=guycode&rpp=100&result_type=mixed";
    var webClient = new WebClient();
    string xmlResult = webClient.DownloadString(url);

    return Json(ReadTwitterResults(xmlResult),
        JsonRequestBehavior.AllowGet);
}
```

Asynchronous Controller Actions

Example (Asynchronous)

```
public async Task<ActionResult> Search()
{
    string url = "http://search.twitter.com/search.atom?" +
        "q=guycode&rpp=100&result_type=mixed";
    var webClient = new WebClient();
    string xmlResult = await webClient.DownloadStringTaskAsync(url);

    return Json(ReadTwitterResults(xmlResult),
        JsonRequestBehavior.AllowGet);
}
```

Asynchronous Controller Actions

Timeouts

- AsyncTimeout attribute can be used to specify a timeout value for an asynchronous action (in milliseconds)
- Default is 45 seconds
- NoAsyncTimeout attribute can also be used if needed

```
[AsyncTimeout(60000)]  
[HandleError(ExceptionType = typeof(TaskCanceledException),  
View = "TimedOut")]  
public async Task<ActionResult> Search() { }
```

ASP.NET MVC 5

Mobile Clients

- Introduction
- Bootstrap
- Overriding Views
- jQuery Mobile

Mobile Clients

Introduction

- In the past, a major issue for web developers was ensuring their site would work correctly across multiple different web browsers
 - Although still a concern, the situation is better than it was
- A larger concern nowadays is ensuring your site will look and behave correctly on a wide array of client form factors
 - Desktop, tablet, phone, phablet?!
- ASP.NET MVC combined with some 3rd-party client-side frameworks can provide a wide range of options

Mobile Clients

Bootstrap

- Starting with ASP.NET MVC 5, the built-in project template includes the Bootstrap framework
- Offers the ability to build your views using Responsive Web Design (RWD) principles
- Starting with version 3, the Bootstrap framework takes a mobile first approach
 - The idea is that it is easier to scale up a page designed for a small screen than it is to scale down a page designed for a large screen

Mobile Clients

Bootstrap

- An advantage to a responsive design is that all layout decisions are made on the client-side based on the width of the browser
 - Page can adapt when a window is resized by the user or when a tablet is rotated from portrait to landscape
- However, there are some disadvantages...
 - CSS to support all widths has to be sent with the request
 - Only so much can be done with CSS and some designs require a significantly different layout for mobile clients

Mobile Clients

Overriding Views

- ASP.NET MVC includes a simple mechanism that allows you to specify individual alternate views for different user agents
 - Also works for layout and partial views
- To provide a mobile-specific version of a view, simply add .Mobile to the file name
- The view engine will use the user-agent string included with the request to select the appropriate view

Mobile Clients

Overriding Views

- It is also possible to create browser-specific views
- Add a new display mode in Application_Start

```
DisplayModeProvider.Instance.Modes.Insert(0, new DefaultDisplayMode("iPhone")  
{  
    ContextCondition = (context => context.GetOverriddenUserAgent().IndexOf  
        ("iPhone", StringComparison.OrdinalIgnoreCase) >= 0)  
});
```

Mobile Clients

jQuery Mobile

- For an even more optimized user experience for mobile clients, consider using the jQuery Mobile framework
- Provides a user interface framework that works across all of the major mobile browsers
- jquerymobile.com
- Also available as a NuGet package

ASP.NET MVC 5

Web API

- Introduction
- ApiController
- Action Parameters
- Exceptions
- Configuration
- Routing
- Model Binding
- Formatters
- Status Codes
- Hands-On Lab Exercise

Web API

Introduction

- ASP.NET Web API is a framework for building HTTP services on top of the .NET Framework
- Result of collaboration between ASP.NET MVC and WCF teams
- Web API is its own stand-alone framework and technically not part of ASP.NET MVC
- Very often used in conjunction with ASP.NET MVC

Web API

Introduction

- Standard ASP.NET MVC controllers can be use to create an HTTP service but that approach has some shortcomings
- WCF can also be used to create HTTP services but can be complex and difficult to configure
- Web API represents a middle ground
 - More natural dispatching to actions based on HTTP verb
 - Accepting and generating different content types
 - Content type negotiation
 - Hosting outside of the ASP.NET runtime stack

Web API

ApiController

- Like ASP.NET MVC, Web API maps HTTP requests to controller actions
- Controllers inherit from ApiController
- Actions directly render the resulting model object as the response (typically no views)
- Supports automatic content negotiation via the request Accept header (XML and JSON)

ApiController

Example

```
public class CourseController : ApiController
{
    // GET api/<controller>
    public IEnumerable<Course> Get() { ... }

    // GET api/<controller>/5
    public Course Get(int id) { ... }

    // POST api/<controller>
    public void Post([FromBody]Course course) { ... }

    // PUT api/<controller>/5
    public void Put(int id, [FromBody]Course course) { ... }

    // DELETE api/<controller>/5
    public void Delete(int id) { ... }
}
```

Web API

ApiController

- All Web API controllers are asynchronous by design
- No direct access to response object
- Actions are expected to return an object of type `HttpResponseMessage`
 - You can create and return an object of type `HttpResponseMessage` or Web API will create one implicitly

```
public HttpResponseMessage Post([FromBody]Course course)
{
    var response = Request.CreateResponse(HttpStatusCode.Created, course);
    response.Headers.Location = new Uri(Url.Link("DefaultApi",
                                                new { id = course.Id }));
    return response;
}
```

Web API

Action Parameters

- By default, Web API will assume...
 - Simple types are taken from non-body values
 - Complex types are taken from the body
- Only a single value can come from the body and that value must represent the entirety of the body
- Incoming non-body parameters are handled by a model binding system similar to ASP.NET MVC
- Incoming and outgoing bodies are handled by formatters

Web API

HttpResponseException

- When something other than model data must be returned, Web API allows for an `HttpResponseException` to be thrown

```
public Product Get(int id)
{
    Product p = db.GetProduct(id);

    if (p == null)
        throw new HttpResponseException(HttpStatusCode.NotFound);

    return p;
}
```

Web API

Configuration

- Web API configuration values are stored in an instance of `HttpConfiguration`
- `GlobalConfiguration.Configuration` gives access to a global instance for use in an ASP.NET application
- Contains properties for routing, filters, formatters, etc.

```
protected void Application_Start()  
{  
    WebApiConfig.Register(GlobalConfiguration.Configuration);  
}
```

Web API

Routing

- Routing system in Web API uses the same routing logic as ASP.NET MVC
- When self-hosting, Web API uses its own private copy of the routing code
- When web-hosted, wrapper Route objects are created and registered with the ASP.NET routing engine

```
public static void Register(HttpConfiguration config)  
{  
    config.Routes.MapHttpRoute(  
        name: "DefaultApi",  
        routeTemplate: "api/{controller}/{id}",  
        defaults: new { id = RouteParameter.Optional }  
    );  
}
```

Web API

Routing

- The {action} token in Web API is optional
- Action names that start with a verb name are selected automatically
- Default verb is POST if an action name does not start with a verb name and is not decorated with AcceptVerb attribute

Web API

Model Binding

- Web API uses parameter binders to provide values for individual parameters
- Attributes can be used to override default behavior

```
public Product Get([FromUri] int id) { ... }  
public void Post(int id, [FromBody] Product product) { ... }
```

Web API

Formatters

- Formatters are responsible for consuming and producing body content
- Three formatters are built into Web API
 - JSON (using Json.NET)
 - XML (XmlSerializer or DataContractSerializer)
 - URL encoded form data

Web API

HTTP Status Codes

- When using Web API, your API methods should return the proper HTTP status codes when appropriate
 - Default is 200 (OK)

```
throw new HttpResponseException(HttpStatusCode.NotFound);
```

```
public HttpResponseMessage Post(Product item)
{
    item = repository.Add(item);
    var response =
        Request.CreateResponse<Product>(HttpStatusCode.Created, item);

    string uri = Url.Link("DefaultApi", new { id = item.Id });
    response.Headers.Location = new Uri(uri);
    return response;
}
```

Lab 22

Web API

- Create a Web API controller for courses
- Retrieve a list of courses in both JSON and XML format

ASP.NET MVC 5

Security

- Authentication
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Over-Posting

Security

Authentication

- A new project entitled the Open Web Interface for .NET (OWIN) defines a standard interface between .NET web servers and web applications
- Provides a middleware implementation for .NET that allows developers to hook into a deeper pipeline to perform tasks like logging, exception handling, authentication, and more
- Allows for an easier implementation of 3rd-party authentication systems such as OAuth 2.0

Security

Authentication

- OWIN-based authentication is the default and preferred option for authentication in an MVC 5 application
- The ASP.NET MVC 5 templates provide the option to include OWIN-based authentication components including an AccountController class
- Other ASP.NET authentication options can still be used
 - Windows authentication
 - ASP.NET Membership System

Security

Cross-Site Scripting (XSS)

- Cross-site scripting (XSS) is an attack where someone attempts to cause your website to load a script into the user's browser
- Once accomplished, many bad things can be done
 - Cookie theft
 - Modify user settings
 - Download malware
 - Modify content
 - Account hijacking

Security

Cross-Site Scripting (XSS)

- ASP.NET will automatically reject any submitted data that appears to be HTML
 - Can be disabled using the ValidateInput attribute for an action
 - Can also use the AllowHtml attribute on a model property

```
[HttpPost]
[ValidateInput(false)]
public ActionResult Edit(int id, FormCollection collection)
{ ... }
```

```
[AllowHtml]
public string CustomerComments { get; set; }
```


Security

Cross-Site Scripting (XSS)

- Microsoft provides an AntiXSS library as a separate download that can be used to filter submitted HTML fragments

Security

Cross-Site Request Forgery (CSRF)

- A cross-site request forgery (CSRF) attack is one where another site presents the user with a form that submits data to your site
 - Submission includes existing authentication cookie
 - Data submitted is potentially not data the user supplied
- The AntiForgeryToken HTML helper combined with the ValidateAntiForgeryToken attribute can be used to prevent this

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

```
[HttpPost]  
[Authorize]  
[ValidateAntiForgeryToken]  
public ActionResult Edit(...)  
{ ... }
```

Security

Over-Posting

- Using model binding makes it possible for a user to submit values for properties that you didn't intend to be modifiable
- There are several ways to prevent this:
 - Use the Bind attribute on model classes or controller actions
 - Pass a bind list to UpdateModel
 - Use a view model class that excludes properties

```
[Bind(Include="Number, Title")]  
public class Course { ... }
```

```
UpdateModel(course,  
    new string[] { "Number", "Title" },  
    collection);
```

ASP.NET MVC 5

Deployment

- Introduction
- Web.config Transformations
- IIS Best Practices

Deployment

Introduction

- An ASP.NET MVC 5 application will run on IIS 7 and newer (integrated pipeline mode) without server modifications
- Deploying to IIS 6 requires some changes to ensure the routing system will work correctly

Deployment

Web.config Transformations

- A Web.config transformation file contains XML markup that specifies how to change the Web.config file when it is deployed
- You can specify different changes for specific build configurations (Debug, Release, etc.) and publish profiles
- Appear under the Web.config file in Visual Studio

Deployment

IIS Best Practices

- Understand all of the settings affecting when your ASP.NET MVC application process will be recycled
 - Scheduled recycling
 - Idle activity
 - File changes
- Use the application initialization module in IIS 8 (warm-up module in IIS 7) to mitigate the effect of application with a long initialization time (e.g. Entity Framework)

Deployment

IIS Best Practices

- Use the UrlRewrite module to enforce URL consistency
 - Lowercase URLs
 - Canonical URL
 - Add or remove trailing slashes
- Leverage caching for static resources
 - Use a CDN if possible