ASP.NET MVC5/(Core)

Landscape, Margin all 0.1 except bottom 2.5, font, 14

App\_Start

* Bundle/Filter/Identity/Route/Swagger/WebApiConfig.cs, Startup.Auth.cs

HttpContext

* Application-wide: static HttpContext.Current.xxx
* Controller only: HttpContext.xxx
* Common Usage: Access User, Request, Session(WebForm), Server, Application, Cache, Response, GetOwinContext().GetUserManager<*ApplicationUserManager*>()

(Core) Built-in Services: IApplicationBuilder/IApplicationEnvironment/IHostingEnvironment/ILoggerFactory/IserviceCollection

* IApplicationEnvironment: provide ApplicationName, ApplicationBasePath
* IhostingEnvironment: Environment name, webRootPath

(Core) Startup --- Configuration Settings (JSON,XML,INI,Environment)

public Startup(IHostingEnvironment env, IApplicationEnvironment ae, ILoggerFactory lf) {

var builder = new ConfigurationBuilder()

.AddJsonFile("appsettings.json") // Add Built-in Providers

.AddEnvironmentVariables(); // Add Built-in Providers

var config = builder.Build();

config.Set(“myKey”, “myValue”); // Write

var settingValue = config[“myKey”]; // Read

}

public IConfigurationRoot Configuration { get; set; }

(Core) Dependency Injection / Services – Startup.cs

* Register DI Mappings
* ConfigureServices(IServiceCollection services) { services.AddSingleton/Transient<IMyService, MyService>(); }
* Usage in Controller:
  + Constructor: MyCtor(IMyService srv) { }
  + Action Parameter: public IActionResult Foo([FromService] IMyService srv) { … }
* Usage in View: @inject IMyService myService // myService is the property name.

(Core) Middleware / ApplicationBuilder //ConfigureServices will be called BEFORE Config

* Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory, IApplicationEnvironment ae)
  + app is used to chain middleware together. env.IsDevelopment() is used to conditionally chain middleware
  + Built-in Middleware: Authentication/CORS/Diagnostics/Routing/Session/StaticFiles

Controller (C)

* **Action** method returns a corresponding View with optional Model data. **ActionResult** **MyAction**(…)
* **Action** method **Parameters** passing
  + **Route Map**: Special **‘ID’** param **w/o** using **url** **query** **param**: store/details/**5** => ActionResult Details(**int Id**) // Id can be ID or id
  + **Auto** mapped to url **query param**. E.g. store/browse**?genre** => ActionResult Browse(**string** **genre**)
* **Specify** **V** in **C**:
  + **Convention**: return **View()**; // View(…) is a overload **methods. Default to View with the same name as Action method**
  + **Configuration**: return **View(“myOtherView”, model)**; // View(“**~/Views/**someController/someAction.cshtml”)
* **Simple** **“M(s)”** in **C**:
  + **Prefer** - Passing **strongly typed** ViewModel to view
    - return **View**(object **Model**)/**View**(string ViewName, object **Model**);
  + Passing data from **Controller to Controller**
    - Use **RedirectToAction** with **routeValues** param to pass data. **new { Id =** someId **}**
    - **TempData[“xyz”]: Passing data across subsequent Html requests when redirect in Action Method**
      * **Once TempData is read, it will not be available to the next request unless TempData.Peek(“xyz”)**
  + Passing loosely typed data from Controller to View **- dynamic**
    - **ViewData**[“xyz”] OR in *~~MVC3 ViewBag~~.* Obsolete
* **ActionResult Types**
  + **View**
    - return View(); // default view
    - return View(**model / viewName / viewName, model**);
    - return View("~/xyz.aspx", model); // explicit path to a specific view template
  + **Redirect with controllerName, actionName and routeValues(segment params) as anonymous Object**
    - return **RedirectToAction**("actionName", "controllerName", **new { param1="...", param2=...}**);  
      return **RedirectToRoute**("routeName", **new { param1="value1", param2=value2 }**);  
      return **Redirect**("http://www.xyz.com or ~/some/relativeUrl");
    - [What’s the difference?](http://www.dotnet-tricks.com/Tutorial/mvc/4XDc110313-return-View()-vs-return-RedirectToAction()-vs-return-Redirect()-vs-return-RedirectToRoute().html)
  + **Http Status Code**
    - **return new HttpStatusCodeResult(…);**
    - **return new HttpNotFound();**
    - **return new HttpUnauthorized();**
  + **PartialView – for composition.**
    - return **PartialView**(…);
  + **Formatted Response**
    - **Textual** **Data** - e.g. **Plain text, XML, RSS**
      * return **Content**("this is plain text", "**text/plain**" **OR** "**application/rss+xml**");
    - **JSON**
      * return **Json**(new [] { new {city="London", temperature=68}, new  {city="Cairo", temperature=110} });
    - **JavaScript** Commands
      * return **JavaScript**("alert('Hello World!');");
    - **Files**
      * return **File**(**filename**, "application/pdf", "xyz.pdf");
    - **Binary Byte Array, Stream**
      * return **File**(**data**, "application/pdf", "xyz.pdf"); // where **data: is byte[] type**
      * return **File**(**stream**, "text/html"); // where **Stream stream = webClient.OpenRead("someUrl");**
* **Securing Application**
* **CSRF** (CrossSiteRequestForgery) Attack.
  + **Controller Action: [ValidateAntiForgeryToken]** on All [HttPost] Action method
  + **View**: Inside a <form> **@Html.AntiForgeryToken()** … </form>
* **Exception Handling – Controller Level**
  + protected override void **OnException**(ExceptionContext filterContext) {

filterContext.ExceptionHandled = true;

filterContext.Result = new **ViewResult**() {

ViewName = "Error", // **Error.cshtml can use @model HandleErrorInfo to access exception message!**

ViewData = new ViewDataDictionary(new HandleErrorInfo(

filterContext.Exception, controllerName, actionName))};

}

* + **Global Level -** [**http://www.codeproject.com/Articles/731913/Exception-Handling-in-MVC**](http://www.codeproject.com/Articles/731913/Exception-Handling-in-MVC)
* **Advanced: ActionFilters – provide pre/post action hook**
  + Authorization / Action / Result / Exception

WebAPI 2

* **Reference** - <http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/action-results>
* **public IHttpActionResult Get(…)**: <https://github.com/vincenthome/WebApi.2.HttpResults>
  + return **Ok()/Created()/CreatedAtRoute()/CreatedAtAction(); // Content Negotiated**
* **Routes**: <https://github.com/vincenthome/WebApi.Routes>
* **Exception**: <https://github.com/vincenthome/WebApi.ExceptionHandling>
* **Parameter Binding**: <https://github.com/vincenthome/WebApi.ParameterBinding>
* **CORS**: <https://github.com/vincenthome/WebApi.CORS>
* **OAuth**: <https://github.com/vincenthome/WebApi.OAuth>
* **File Upload**: <https://github.com/vincenthome/WebApi.FileUpload>

(Core)

* public **IActionResult** Index() { … return View(); }
* **WebApi** // **needs update** from Doc
  + **DI for property** – add attribute **[FromServices]**
  + **[Route(“api/[controller]”)]**
  + [**HttpGet/Post**(“**id:int**”, **Name** = “GetByIdRoute”)] attribute with optional segment to param mapping

public **IActionResult** GetById(**int id**) {

return **new ObjectResult(poco)**; // returns serialized JSON

// OR return **HttpNotFound()**;

}

* Accessing **Configuration Settings** in Controller using **IOptions<MySettings>**: link [here](https://docs.asp.net/en/latest/mvc/controllers/dependency-injection.html#accessing-settings-from-a-controller) and [here](https://docs.asp.net/en/latest/fundamentals/configuration.html#using-options-and-configuration-objects)

View (V)

* Derived from **WebViewPage<TModel>**
* **Strongly Typed M/VM** in **V** w/ **intellisense:**

@**model** **myNamespace.myModelType** // specify what type the keyword “Model” is mapped to! Support intellisense.

**@Model.someProperty** reference the **M** passed in **View**(object **Model**)/**View**(string ViewName, object **Model**); in **C**

* **T4 Add View Scaffolding**: [Visual Studio Install Directory]\Common7\IDE\ItemTemplates\CSharp\Web\MVC 3\CodeTemplates\AddView\CSHTML\

Razor

* **Syntax**: <http://www.asp.net/web-pages/tutorials/basics/2-introduction-to-asp-net-web-programming-using-the-razor-syntax>
* **Expression**:
  + **Implicit: @someExpression** // **Empty** **space** **after** the **expression** marks the end.
  + **Explicit: @(**someExpression**)** // anything after the () will treat as literal text. Useful for generic helper method
  + **Escape:** the **@** sign is **@@**.
* **Block**
  + **Code Only**

**@ {**

Some block **code** …

**}**

* + **Code & Markup**

**@ foreach(**var **item** initems**) {** // **‘@’ sign** will **switch** to **Razor Mode**

**<span>** Item: **@item.Name** **</span>** // **‘<…>’** will **switch** to **Literal Mode**

**<text> some plain text </text>**

**If (…) { // ‘if’ is NOT within any open/close tag. NO @**

**…**

**}**

**<div> // when the ‘if’ is within open/close tag, MUST use the @if**

**@if(…) {**

**…**

**}**

**</div>**

**}**

* **Html Helpers/model expression: always prefix with @.** E.g. @Html.xxx @model.xxx
* Modify Html attribute w/ **UNIQUE** **element** **ID**
  + **@Html.HiddenFor(e => e.SomeProperty, new { @id = "some\_property", @Name = "some\_property" });**
  + **@{**  // start **‘Razon Block’** mode

var **resultID** = **"result\_" + item.MovieId**;

<div **id=@resultID**> // ‘<div>’ switch to **Literal** mode. @resultID switch to **‘Razor Expression’** mode

@Ajax.ActionLink(..., new AjaxOptions { UpdateTargetId = **"result\_" + item.MovieId**, ...}) // @Ajax switch to **‘Razor Expression’** mode

</div> // ‘</div>’ switch to **Literal** mode again.

**}**

* **Comment**: **@\*** some comment here! **\*@**
* **Generic Html Helper Method**: **generic** **use** angle brakets **<>**, must use **explicit code expression**. E.g. **@(Html.SomeMethod<AType>())**
* **HtmlEncode implicit (Auto).** 
  + **Reason: prevent XSS** – Cross Site Script Injection Attack
  + **Implicit**: i.e. @**someVariable** will **AUTO** **HtmlEncode**(**someVariable**)
  + **Explicitly Disable** **Encoding** use: @**Html**.**Raw**(**someVariable**)
* **JavaScript String Encoding**: var text = **‘$Ajax.JavaScriptStringEncode(someString)’**;

Layout Page

* **Layout** Page (~/Views/Shared/**\_Layout.cshtml**) – **reference** Body & Sections:
  + @**RenderBody**
  + @**RenderSection**(“Header/Footer/Script”, true/false/\* **required** \*/) // **CAN’T** be **Nested** **inside** the **RenderBody**
  + **Default Content** for **Section**:

**@if** (**IsSectionDefined**(“Header/Footer/Script”)) {

**@RenderSection**(“Header/Footer/Script”); // **DON’T FORGET THE ‘@’** sign here

}

**else** {

<span>This is the default header/footer.</span>

}

* + It supports Url/Html Helpers as well. E.g. include 1+ Html.ActionLink(…) to create a **Menu**

View – define the Body & Section:

* **Default Layout**: At **\_ViewStart.cshtml** use the **Layout** **property** to specify the **default** **layout** page for **ALL Views**.
* **Override** **Layout property**: @ { **Layout** = “~/Views/Shared/**someLayoutPage.cshtml**”; } // At the **beginning of your View**
* **Turnoff Layout:** @ {Layout = **null**; }
* **Add Section inside the Content Page:** 
  + **@section** **Header/Footer/Script** { … code and/or markup … }. You can put it anywhere. But commonly the bottom of the page.

Include Javascripts and bundle them

* **BundleConfig**.js - bundles.Add(new **Script|StyleBundle**(“~/bundles/myjs|~/Content/css”).**Include**(“a.js|a.css”,”b.js|b.css”, …));

Referencing JavaScript

* \_**Layout**.cshtml: Put all your **Global** references there.
  + @**Scripts|Style**.**Render**(“~/bundles/jquery|~/Content/css”)
  + @**RenderSection**(“**Scripts**”, required:false) right after @Scripts.Render(…)
* **V**:

@**section Scripts** { // At **bottom** of the View

@**Scripts.Render**("~/plugins/somebundle") // **View level** **bundled** script

<script src=”/Scripts/someScript.js” ></script> // **View level** **external** reference script

**<script type="text/javascript">**…**View level** **raw** script … </script>

}

* + **Global Scripts**: reference them like this - @if (false) { <script src=”/Scripts/someGlobal.js” ></script> }
  + Gives you **Devlopment Time Intellisense**, at runtime it will be removed and scripts referenced at \_Layout will kick in.
    - **.js intellisense**: **/// <reference path="**/Scripts/jquery-1.5.1.js" **/>**

Common Javascript Operations

* **Conditionally Disable** a input control: ***p.s. CSS doesn’t support disabling control .* But Hide is BETTER**
  + **Loadtime:** 
    - @{

**object myhtmlAttributes;**

if (Model.ShowProperty) {

**myhtmlAttributes** = new { htmlAttributes = new { @class = "form-control" } };

}

else {

**myhtmlAttributes** = new { htmlAttributes = new { @class = "form-control**", disabled="disabled"** } };

}

}

@Html.EditorFor(m => m.MyTextbox, **myhtmlAttributes**)

* + - **@Html.HiddenFor(…) // Tips**: Without .HiddenFor, the disabled control’s value **will not be submitted by Form Post**
  + **Runtime:**
    - **Enable/Disable** a input control:
    - $("#MyCheckbox").**change**(function () {

$("#MyTextbox").**prop('disabled', !this.checked); // toggle the checked**

});

* **Hide/Show** an input control: using CSS class
  + **Bootstrap**: [.show / .hidden](http://getbootstrap.com/css/#helper-classes-show-hide)
  + **Hide:** [**visibility**](http://www.w3schools.com/cssref/pr_class_visibility.asp)**: hidden** – take up space. [**display**](http://www.w3schools.com/cssref/pr_class_display.asp)**: none;** no space taken.
  + **Show:** **visibility**: **visible**(default) ; **display**: inline(default)/block/inline-block.
  + **Loadtime**
    - @Html.EditorFor(m => m.MyTextbox, new { htmlAttributes = new { @class = Model.ShowProperty ? "form-control" : "form-control **hidden**" } })
  + **Runtime**
  + **Show/Hide** an input control:
    - $("#MyCheckbox").**change**(function () {

$("#MyTextbox").**toggleClass(“hidden”);**

//$("#MyTextbox").**toggle(**500**); // alternatively toggle the visibility**

});

* Use **data-\*=** to 1 event handler for **n+ event publishers**
  + $(“[myEventFiringObject]”).on(“click”, function() {

// **$(this).data**(“xyz”).

})

* Cascade Dropdown – the 2nd one
* Controller - for 2nd dropdown action return Json
  + public ActionResult **GetStates**(string id) {

List<**SelectListItem**> *states* = new List<SelectListItem>();

return **Json**(new **SelectList**(*states*, "**Value**", "**Text**")); } // **Value/Text** are properties of **SelectListItem**

* + - Alternative **directly using Complex Type**
      * **List<Person>** people = new List<Person>(); //Custom ComplexType
      * people.Add(new Person { **Id** = 1, **Name** = "Vince", Age = 49 });
      * **SelectList** items = **new SelectList**(people, **"Id", "Name"**); // with list, **valueFieldName, textFieldName**
      * // *People* is - **IEnumerable<SelectListItem>**
      * return View(new HomeVM { SelectedId = 2, *People* = items });
* View -
  + 2nd dropdown initially empty
    - @Html.**DropDownList**("State", new **SelectList**(string.Empty, "**Value**", "**Text**"), "Select …", new { @class = "xxx" })
  + 1st dropdown on change, call ajax action method, populate 2nd dropdown with <option value=”…”>…</option>
    - $("#**Country**").**change**(function () {

**$("#State").empty();**

$.**ajax**({

type: 'POST',

url: '@Url.Action("**GetStates**")', // we are calling json method

dataType: 'json',

**data: { id: $("#Country").val() }, // 1st dropdown selection**

success: function (**states**) {

**$.each**(**states**, function (**i, state**) {

**$("#State").append('<option value="' + state.Value + '">' + state.Text + '</option>');**

});

},

error: function (ex) {

alert('Failed to retrieve states.' + ex);

}

});

return false;

});

PartialView

* **With Action [Optional]: C** Action method will **return PartialView()**;
* Create View: Visual Studio **Add View** … – **check “Create as Partial View”**,

Paging – a Nuget package for paging an IEnumerable/IQueryable/StaticPagedList into pages and display via Bootstrap

* [PagedList](https://github.com/TroyGoode/PagedList) / [X.PagedList](https://github.com/kpi-ua/X.PagedList)

Error Page

* **Error.cshtml**
* With @model System.Web.Mvc.**HandleErrorInfo,** allow access to **.ControllerName, .ActionName, .Exception.Message**

(Core)

* TagHelper
  + ActionLink: <a asp-controller=”Home” asp-action=”Index” asp-route-id=”@item.Id” >…</a>
  + Label: <label asp-for=”FirstName” ></label>
  + <div asp-validation-summary=”ValidationSummary.All” class=”text-danger”></div>
  + <span asp-validation-for=”FirstName” class=”text-danger”></span>
  + Form: <form asp-controller=”Home” asp-action=”Edit”> …
* Custom TagHelper - <https://docs.asp.net/en/latest/mvc/views/tag-helpers/authoring.html>
* View Components - <https://docs.asp.net/en/latest/mvc/views/view-components.html>

Model (M)

Model View Attributes

* **Display**: customize the **Display Text** & **Order**
* **\*DisplayFormat\***: alternate **text for null** value, **turn-off Html encoding**, **DataFormatString** {c/d/m …} , **ApplyFormatInEditMode**.
* **ReadOnly**
* **DataType**(DataType.CreditCard/Currency/Date/DateTime/Duration/EmailAddress/Html/ImageUrl/Password /PostalCode/PhoneNumber/Text/Time/MultilineText/Upload/Url)] // **ONLY** used by **DisplayFor** & **EditorFor**
* **HiddenInput(DisplayValue=false**/true**)**: <input type=”hidden” … />
* **ScaffoldColumn(false)**: **Hide** the property **from** **Scaffolding Template Helpers** like **Display/EditorForModel**

CodeFirst EF ForeignKey / Navigation Property

* **Navigation** & **Foreign Key Property**: When a M has 2 properties **associated** to the same thing, e.g. **Album** has **2 properties ArtistId & Artist**, the 1st property **ArtistId** is a “**Foreign Key**” Property while the 2nd **Artist** property is a “**Navigation**” property, so you can navigate from the M Album to Artist using **myAlbum.Artist** **dot** **operator**.

EntityFramework

* **Code First Model**: Define your Model properties using virtual. So EF can hook the properties to provide change tracking.
  + **Files Created**: **DBContext** derived class( w/ **DBSet** tables), **Controller** **w/ private DBContext** derived & **Action** methods **w/ LINQ**, **View** for Create Edit Delete Index Details **scaffolding**.
  + **LINQ**: **Eager Loading** Query using **“.include”**. The **alternative**(**default**) is **Lazy** **Loading**.
  + **Create/Update/Delete:** myDBSet.**Find**/.**Add**/.**Remove, myDBSet.Entry(model).State = .Modified**, myDBContext.**SaveChanges**
  + **EdmMetadata**: **track** **M** **property changes** against the DB **table** **schema by computing & save the hash of M.**
* **Database Initializer**:
  + Put it in **Global.asax.cs Application\_Start.** You can find **sample** **code** in the generated **M** class
  + **Database.SetInitializer**(new **DropCreateDatabaseIfModelChanges**<**myDBContextDerived**>());
  + Database.**SetInitializer**(new **DropCreateDatabaseAlways**<**myDBContextDerived**>());
* **Seeding DB**
  + Create a **SeedingDB class derived from** **DropCreateDatabaseAlways<T>/DropCreateDatabaseIfModelChanges<T>.**
  + **Override** the **Seed** **method** and see data to various DBSet(tables).
  + Call this in your DBContext class ctor. Database.**SetInitializer**(new **mySeedingDB()**);
  + <http://www.entityframeworktutorial.net/code-first/seed-database-in-code-first.aspx>
* **Code-based Migration**
  + Repeat the following everytime schema changed:
    - **add-migration** “my-migrate”: It will prepare/scaffold file based on the changes in your Model class
      * File created in “Migration” folder : date\_my-migrate.cs
    - **update-database -verbose**: It will apply the changes in the scaffold files to the database
      * Support rollback db changes: update-database –TargetMigration “my-migrate”
  + <http://www.entityframeworktutorial.net/code-first/code-based-migration-in-code-first.aspx>

Identity

* [Book Pro ASP.NET MVC5 Platform Apress p.296](http://www.amazon.com/Pro-ASP-NET-MVC-5-Platform/dp/1430265418)
* Define 3 classes (**Auto created** by VS)
  + public class **ApplicationUser** : **IdentityUser** { … public string **customProperty**; } // **IdentityModels**.cs
  + public class **ApplicationUserManager** : **UserManager**<**ApplicationUser**> { … } // for manage ApplicationUser. **IdentityConfig**.cs
* Other Identity Managers:
  + **AuthenticationManager**: HttpContext.GetOwinContext().Authentication; // for login/logout in **AccountController**.cs
  + **RoleManager**: new RoleManager<IdentityRole>(new RoleStore<IdentityRole>(dbcontext)); // for manage roles
  + public class **ApplicationSignInManager** : **SignInManager**<**ApplicationUser**> { … } // **IdentityConfig**.cs
* **Get Currently Logged in ApplicationUser** By userId using ***ApplicationUserManager***
  + ***AppliicationUser user = UserManager*.FindById(User.Identity.GetUserId());** // i.e. **GetUserId() extension method**
  + Account/ManageController: ***UserManager* property auto created by Visual Studio**
  + Other Controllers. Needs to manually create UserManager property:
    - Implementation: **HttpContext.GetOwinContext().GetUserManager<*ApplicationUserManager*>();**
* **Update** ApplicationUser’s **custom properties** using ***ApplicationUserManager***
  + **usermanager.Update(user);**
* Cookie Authentication Timeouts - [ref](http://www.jamessturtevant.com/posts/ASPNET-Identity-Cookie-Authentication-Timeouts/)
  + **CookieAuthenticationOptions**.**SlidingExpiration** = true
  + CookieAuthenticationOptions.**ExpireTimeSpan** = TimeSpan.FromMinutes(30)

Model Binder at Action Method – implicit(param) & explicit(TryUpdateModel) with Data from View & Validation

* Works for both **GET** & **POST**
* **Routing Engine Param Mapping**: Map **Url Segments/Form** **Post => Url** **params**, etc…
* **Implicit Bind Model with Data from View & Validate** and pass to **Action** method **param:**
  + **Simple Type param:** e.g. myAction(**int** customerId)
  + **Complex Type param:** e.g. myAction(**Customer** c)
* **Explicit Build Your Own Model (BYOM). Explicit Bind Model with Data from View & Validate** using **Try/UpdateModel(**myModel**):**

var **myCustomer** = new Customer();

if (!**TryUpdateModel**(myCustomer)) // Bind Model with **Data from View** & **Validate**

return View(myCustomer); // when failed

else

return RedirectToRoute("Success");

// OR try { **UpdateModel**(myCustomer); … } **catch** { … }

* **ModelState**: shows the result of Model Binding, if **failed** it will show **propertyName**, attempted **value**, error **message**.

HTML Helpers

* **Html Encode** when **Output** **Model** **values** to avoid XSS attack.

Form Helpers

* @**using** on **returned** **IDisposable** object: e.g. Html.BeginForm() will return an IDisposable object which writes the **closing tag </form>**
* **Form**
  + **Html.BeginForm() //** Default to **Post** to the **Current Url**
  + **Html.**BeginForm**(action, controller, routeValues , get\_or\_pos, htmlAttributes, …)**

e.g. **@using (Html.**BeginForm**())** { @Html.**ValidationSummary**(excludePropertyErrors: true) … }

* + **Ajax.BeginForm** – Action can **return PartialView** in **Html/Plain Text**:
  + Generate a **Form** when **submit** that sends **ajax** **request**
  + **@using (Ajax.BeginForm(action, controller,**

**new AjaxOptions** {

**InsertionMode**=**InsertionMode**.**Replace**,

HttpMethod=”**GET**”,

**OnFailure**=”searchFailed”, // callback

**LoadingElementId**=”**ajax-loader**”, // shows this element when async request is in progress

**UpdateTargetId**=”searchresults”,

})) {

<input type=”text” name=”q” />

<input type=”submit” value=”search” />

<**img** id=”**ajax-loader**” src=”@Url.Content(“~/Content/Images/ajax-loader-**spinner**.gif”)” style=”display:none”/>

}

* **Input** HtmlHelpers – ALL about **Model Property Binding**.
* Common params: object htmlAttributes **key**/value pairs
  + **keys** that aren’t allowed: replace **class** with **@class**, replace **‘-‘** with **‘\_’**
* Let **‘m’** be the **Model**
* **Html.Validation…**
  + **.ValidationSummary(bool excludePropertyErrors, errorMessage, htmlAttribs)** – list errors from ModelState dictionary
  + **.ValidationMessageFor**(**lambda, errorMessage, htmlAttribs**)
* **Html.LabelFor**(lambda, **labelText**)
* **Html.DropDownListFor & Html.ListBoxFor**: (**m => m.SelecteId, SelectListItems, optionEmptyItemLabel /\* dropdown only\*/, htmlAttribs**)
  + Action-side Preparation: Create an IEnumerable<**SelectListItem> SelectListItems**.

public class **Item** {

public int **Id** { get; set; }

public string **Name** { get; set; }

}

private **List<Item>** **items** = … populate it from db!

public **IEnumerable<SelectListItem>** **SelectListItems**

{

get

{

return **items**.Select(i => new **SelectListItem** {

**Value** = i.**Id**.ToString(),

**Text** = i.**Name**

});

}

}

* **EditorFor** **Enum** Model -> Custom **EditorTemplate** with **Unique Id RadioButtons** – [link](http://stackoverflow.com/questions/18542060/mvc4-enum-and-radio-button-list)

@model Enum

@foreach (var value in Enum.GetValues(Model.GetType()))

{

var **id** = TagBuilder.CreateSanitizedId(string.Format( "{0}\_{1}\_{2}", ViewData.TemplateInfo.HtmlFieldPrefix, Model.GetType(), value));

<div>

@Html.RadioButton(string.Empty, value, value.Equals(Model), new { **id** })

@Html.Label(value.ToString(), new { **@for = id** })

</div>

}

* **EditorFor IEnumerable<SelectListItem>** Model -> Custom **EditorTemplate** with **DropdownListFor** – [link](http://stackoverflow.com/questions/9517627/converting-html-editorfor-into-a-drop-down-html-dropdownfor)
* **Html.RadioButtonFor**(lambda**, valueToMatch, htmlAttribs**)
  + Create a group of **1+ RadioButtonFor** helpers**:** Use a **COMMON lambda propName** but **different valueToMatch** for each of the elements. If the **lambda** **property’s** **value** **=** **valueToMatch**, radio button will be checked.
* **Html.CheckBoxFor**(lambda**, htmlAttribs**)
  + Will generate **2 html elements (checkbox & hidden**).
  + hidden field is for when box is not checked and therefore value could not be posted.
* **Html.HiddenFor**(lambda, **htmlAttribs**) // e.g. .HiddenFor(model => model.Id)
* **Html.PasswordFor**(lambda, **htmlAttribs**)
* **Html.TextBoxFor & Html.TextAreaFor**: TextBoxFor(**lambda, htmlAttribs**), TextArea(**lambda, rows**, **columns, htmlAttribs**)

Templated Helpers

* The **choice** of **template** is based on the **Property’s** **C# Type & DataAnnotation instead of the HtmlHelper you chose**
  + Its **C# DataType**: bool for checkbox, string for TextBox or TextArea or Plain Text
  + Its **[DataType(DataType.XYZ)]** Attrib.: DataType.**MultilineText** for TextArea, Data.EmailAddress for <a href=’mainto: [x@y.com](mailto:x@y.com)’ … >[x@y.com</a](mailto:x@y.com%3c/a)> // **ONLY** used by DisplayFor & EditorFor
* **Html.DisplayFor**(lambda, additionViewData) **// using default built-in template if nothing found in */ Views/Shared/DisplayTemplates***
* **Html.EditorFor**(lambda, additionViewData) **// using default built-in template if nothing found in */ Views/Shared/EditorTemplates***
  + vs **TextBoxFor** which **always** output<input type=**text**>
* **Custom Html.DisplayFor/EditorFor Template**
  + **Create Folder:** Views->Shared->DisplayTemplates / EditTemplates
  + **Add [DataType].cshtml PartialView:** e.g. Int32.cshtml, String.cshtml, DateTime.cshtml, Enum.cshtml
  + **Creating the Partial View Templates**
    - @Html.TextBox(**“”**, **ViewData.**[**TemplateInfo**](https://msdn.microsoft.com/en-us/library/system.web.mvc.templateinfo%28v=vs.118%29.aspx?f=255&MSPPError=-2147217396)**.FormattedModelValue**, new { … })
      * 1st param “” by leaving it blank, it will auto discover the **Model’s property name**
      * 2nd param **TemplateInfo.FormattedModelValue** returns the **Model’s value**
      * [Example](http://cpratt.co/displaytemplates-and-editortemplates-for-fun-and-profit/)
* **Full Model Html Generation – DON’T LIKE IT. NOT PRACTICAL**
  + ~~DisplayForModel(templateName, …)~~ // for ALL properties of the entire Model
  + ~~EditorForModel(templateName, …)~~ // for ALL properties of the entire Model

Rendering HtmlHelpers

* **routeValues**:
  + **Think** routeValues **AS** **Http Request Segment param values**.
  + **new { Id**=**value1**, someOtherProp=value2 **}** // pass in an **anonymous** object
  + **A FULL Model object:** e.g. new { boo = new SomeType {p1=v1, p2=v2 } }

Url & Link Generation

* **Html Link <a href=”someUrl”>** 
  + **Html.ActionLink**(**linkText**, **action**, **controller**, **routeValues, htmlAttributes**)
  + **Html.RouteLink**(**linkText**, **routeName**, **routeValues**)
    - **w/o** using **routeName**, returned Url will solely based on the **parsing** of the **routeValues**.
    - **w/ routeName**, it will **lookup** a route in RouteConfig.cs **registered** with that **name**
* **Url Literal**
  + **Url.Action / Url.RouteUrl**: same as above except it **doesn’t return <a>** but returns **ONLY the Url**.
  + **Url.Content(relativeUrl):** convert a **Relative** **Url** **to** an **Absolute** **Url**. E.g. Url.Content(“**~/**Scripts/jquery-1.4.1.js”)
* **Ajax Link - return PartialView** in **Html/Plain Text**:
  + **Ajax.ActionLink**
  + <**div** id="**result**" >

**@Ajax.ActionLink(linkText, action, controller,**

**new AjaxOptions** {

**UpdateTargetId**="**result**", // insert the response into an element w/ id=result

**InsertionMode**=InsertionMode.**Replace**, // replace <div id=result>

**HttpMethod**="Get",

**Url=””, Confirm=””, LoadingElementId=””, LoadingElementDuration=100**

**OnBegin=””, OnComplete=””, OnFailure=””, OnSuccess=””**

})

</div>

* **Or jQuery: $(‘#result’).load(**‘/home/message’); // call the Controller Action directly which **return** an **PartialView**
  + **Generates** the following **Html**:

<a **data-ajax**=”true” **data-ajax-method**=”GET” **data-ajax-mode**=”replace”

**data-ajax-update**=”#controller/action” href=”/Home/DailyDeal”>

linkText </a>

* + **Required** jquery-xyz-**unobtrusive**-xyz.js for processing.

PartialView Generation

* **Directly – View w/ Model**: **Html**.**Partial**(partialViewName, model): directly render PartialView **w/o using Action**.
  + p.s. RenderPartial output directly to stream. Partial is better
* **Indirectly** – **more flexible**: **Html**.**Action**(action, controller, routeValues): invokes Action method to returns **PartialView**
  + **routeValues**: You can **pass** in an **anonymous** object **OR**, pass in the **Model** if it is **available**

e.g. @Html.Action(someAction, someController, new { **boo** = **new** **SomeType** { p1=v1, p2=v2 } })

* + p.s. RenderAction output directly to stream, Action is better

Custom HtmlHelper

* namespace System.Web.Mvc.Html { public static class CustomExtensions {
* public static **MvcHtmlString** CustomHelperFor**<TModel, TValue>**(this **HtmlHelper<TModel>** helper,

**Expression<Func<TModel, TValue>>** expression, **object htmlAttributes = null**) {

var data = **ModelMetadata.FromLambdaExpression(expression, helper.ViewData);**

string propertyName = **data.PropertyName**; //extract model property **name** from lambda expression

var value = **data.Model**; // extract model property **value** from lambda expression

TagBuilder span = new **TagBuilder**("span");

span.Attributes.Add(**"name/id", propertyName**);

if (htmlAttributes != null) {

var attributes = **HtmlHelper.AnonymousObjectToHtmlAttributes(htmlAttributes)**;

span.**MergeAttributes**(attributes);

}

return **new MvcHtmlString**(span.ToString());

}

Server-Side Validation

DataAnnotation Attributes

* **Validation Attributes**
  + **Required**, **StringLength**, **RegularExpression**, **Range**(support *integers, doubles, dates, decimals*),
* **Special** **Attributes**
  + **Remote**: client-side **validation** with server-side callback. e.g.

// Model

[**Remote**(“**CheckUserName**”, “**Account**”)] // controller Account, action CheckUserName

public string UserName { get; set; }

// Controller’s Action

public **JsonResult** CheckUserName(string username) {

var result = Membership.FindUsersByName(username).Count == 0;

return **Json(result, JsonRequestBehavior.AllowGet)**; // **JSON wrapped true/false value**.

}

* + **Compare**: 2 properties on a model has the same value. E.g. email address enter twice has to be the same

[RegularExpression(@”[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}”)]

public string **Email** { get; set; }

**[Compare(“Email”)]**

public string **EmailConfirm** { get; set; }

* **Set Custom Error Messages**: use to override the default error message.
  + Use **{0}** to show the **property name** associated with the current Attribute.
  + e.g. [Required(**ErrorMessage**=”Your **{0}** is required.”]

ModelState (A Controller Property):

* + Model level errors,
  + Property level errors & attempted values
  + **Check Validity – in [HttpPost] Action method**:
    - ModelState.**IsValid == true/false**
    - ModelState.**IsValidField(“propName”)** == true/false / ModelState**[“propName”].Errors.Count** > 0

e.g. if(ModelState.IsValid) { … xyz.SaveChanges(); … return ReirectToAction(“Success”, …); }

else { return View(newOrder); // redisplay w/ errors

* + - **Valid**: **Save** **data** to backend (e.g. myDBCtx.SaveChanges()),ReirectToAction(e.g. SuccessView)
    - **InValid**: **Re-renders** the **same** **view** with error message and attempted values
  + Get Error Message (**Rarely** need this): ModelState[“propName”]**.Errors[0].ErrorMessage**
  + **Manually add error**:
    - **Property Level**: ModelState.**AddModelError**(string propertyKey, string errorMessage);
    - **Model Level**: ModelState.**AddModelError(“”, “error message”); // p.s. set key to empty**

Custom Validation

* **IValidatableObject Self-validating Model**:
  + Good for **cross-properties** validation

public class Order : **IValidatableObject** {

public **IEnumerable<ValidationResult>** **Validate**(ValidationContext validationContext) {

if (LastName != null && LastName.Split(‘ ‘).Length > 10) {

**yield return new ValidationResult(“**The last name has too many words!”, new []{“LastName”});

}}}

* **Custom DataAnnotation** **Attribute**: **Derived** from **ValidationAttribute** and **override** the **IsValid** method to **return Validation.Success** or if failed **return new ValidationResult(“Error Message”)**. Add **custom** **properties** as Attribute’s **named** **parameters**

public class MaxWordsAttribute : **ValidationAttribute** {

private readonly int \_maxWords;

public MaxWordsAttribute(int maxWords) : **base**(“{0} has too many words.”) {

\_maxWords = maxWords;

}

protected **override** ValidationResult **IsValid**(object value, ValidationContext validationContext) {

if (value != null) {

var valueAsString = value.ToString();

if (valueAsString.Split(‘ ‘).Length > \_maxWords) {

var errorMessage = FormatErrorMessage(validationContext.DisplayName);

**return new ValidationResult(errorMessage);**

}}

**return ValidationResult.Success;**

}}

Debugging

* Usage: Add System.Diagnostics.**Trace/Debug**(…)
* Display on **Glimpse Trace Tab**
  + Install Glimpse: <http://getglimpse.com/Docs>
* **Azure**
  + In Server Explorer, right-click web app and select “**View Streaming Logs**” to see Error level
  + To change Log level, select “View Settings” and set Application Logging to “**Verbose**”
  + You can send logs to
    - file system,
    - Storage Table – good for online viewing
    - Storage blobs – good for download log file
  + <https://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-troubleshoot-visual-studio>

Client-side Validation

* **PURE jQuery Validation**: Read jQueryValidation.docx
* <script src=”@Url.Content(“~/Scripts/jquery.**validate**.min.js”)” type=”text/javascript” />
* <script src=”@Url.Content(“~/Scripts/jquery.**validate.unobtrusive**.min.js”)” type=”text/javascript” />

Client-side + Server-side Validation Integration

* **Client-side Validation** **enabled** by **default**. **Disable** **with ClientValidationEnable in web.config <appSettings**>
* For **View level** Client-side Validation, use **EnableClientValidation**.
* [http://forums.asp.net/t/1926186.aspx?What+does+EnableClientValidation+and+EnableUnobtrusiveJavaScript+actually+do+](%20http:/forums.asp.net/t/1926186.aspx?What+does+EnableClientValidation+and+EnableUnobtrusiveJavaScript+actually+do+)
* Generated element:
  + Html helper will turn **Data Annotation Attrib**. **=>** **data-\*** html attrib.
  + <input **data-val**=”true” **data-val-length**=”err msg” **data-val-length-max**=”160” **data-val-required**=”err msg” id=”Title” name=”Title” type=”text” value=”...” />
* **When**: jQuery run validation on every **keypress**, **focus** **events** and **block** **form** **submission** if have errors.
* **Custom IClientValidatable(Client-side):** <http://blogs.msdn.com/b/stuartleeks/archive/2010/07/28/asp-net-mvc-adding-client-side-validation-to-validatepasswordlengthattribute-in-asp-net-mvc-3-preview-1.aspx>

public class MaxWordsAttribute : **ValidationAttribute**, **IClientValidatable** {

public **IEnumerable<ModelClientValidationRule>** **GetClientValidationRules**(ModelMetadata metadata, ControllerContext context) {

var rule = **new** **ModelClientValidationRule**();

rule.ErrorMessage = FormatErrorMessage(metadata.GetDisplayName());

rule.ValidationParameters.Add(“wordcount”, WordCount); // ALL Lowercase

rule.ValidationType = “maxwords”; // ALL Lowercase

yield return rule;

}

}

* Above will generate html attrib.: data-val-maxwords=rule.ErrorMessage data-val-maxwords-wordcount=WordCount
* Custom JS validation code:
  + Add new **Adapter**: $.validator.unobtrusive.adapters.addSingleVal(“maxwords”, “wordcount”);
    - **1st** param is the **name** of this adapter, matches the **ValidationType** above, 2nd param matches on **ValidationParameters**
    - Adapter creation Method(jquery.validate.unobtrusive.js)s: addBool, addSingleVal, addMinMax, add
  + Add new **Validator**:

$.validator.**addMethod**(“maxwords”, function (value, element, maxwords) {

if (value) {

if (value.split(‘ ‘).length > maxwords) {

**return false;**

}

}

**return true;**

});

* + - **1st** param is the **name** of this validatior, **matches** the **name** of the **adapter**.
    - **2nd** param, **callback** when **validation** invoked.

Azure

WebJob

* Continuous: QueueTrigger[]
* [Custom Domain Name](https://azure.microsoft.com/en-us/documentation/articles/web-sites-custom-domain-name/)
* [Enable Https](https://azure.microsoft.com/en-us/documentation/articles/web-sites-configure-ssl-certificate/)
* [Create an ASP.NET MVC app with auth and SQL DB and deploy to Azure App Service](https://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-deploy-aspnet-mvc-app-membership-oauth-sql-database/)
* [Create a secure ASP.NET MVC 5 web app with log in, email confirmation and password reset](http://www.asp.net/mvc/overview/security/create-an-aspnet-mvc-5-web-app-with-email-confirmation-and-password-reset)
* [Advanced Account Confirmation and Password Recovery](http://www.asp.net/identity/overview/features-api/account-confirmation-and-password-recovery-with-aspnet-identity)
* [How to Send Email Using SendGrid with Azure](https://azure.microsoft.com/en-us/documentation/articles/sendgrid-dotnet-how-to-send-email/)
* [ASP.NET MVC 5 app with SMS and email Two-Factor Authentication](http://www.asp.net/mvc/overview/security/aspnet-mvc-5-app-with-sms-and-email-two-factor-authentication)
* [Code! MVC 5 App with Facebook, Twitter, LinkedIn and Google OAuth2 Sign-on](http://www.asp.net/mvc/overview/security/create-an-aspnet-mvc-5-app-with-facebook-and-google-oauth2-and-openid-sign-on)
* [Best practices for deploying passwords and other sensitive data to ASP.NET and Azure App Service](http://www.asp.net/identity/overview/features-api/best-practices-for-deploying-passwords-and-other-sensitive-data-to-aspnet-and-azure)
* [Creat Custom Domain in Azure](https://azure.microsoft.com/en-us/documentation/articles/web-sites-custom-domain-name/)

WebJob

* [Overview](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk/)
* [Create WebJob from Scratch & from an Existing WebApp using Azure Queues/Blobs with access to SQL Db](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk-get-started/)
* [SDK GitHub](https://github.com/Azure/azure-webjobs-sdk-extensions)

Routing Engine

Register Routes

* **Attribute Routing**
  + Enable in RouteConfig.cs - routes**.MapMvcAttributeRoutes();**
  + <http://blogs.msdn.com/b/webdev/archive/2013/10/17/attribute-routing-in-asp-net-mvc-5.aspx>
  + **Default**: [**Route**("books/lang/{lang**=**en}")]
  + **Optional**: [Route("books/{isbn**?**}")]
  + Controller level:
    - [**RoutePrefix**("reviews")] // one for **all actions**
    - [Route("{**action=**index}")] // **default action**
    - [RouteArea("Admin")] // ‘/Admin/Controller/Action/…’
  + Route **Constraints**
    - e.g. [Route("users/{id**:int**}"] // param id has to be an integrer
    - {x**:alpha/bool/datetime/decimal/double/float/guid/int/long**}
    - {x:**length(9)/length(1,9)/max(9)/maxlength(9)/min(9)/minlength(9)/range(1,9)/regex(…)**}
    - **1+constraints**: [Route("users/{id**:int:min(1)**}")]
    - **Optional constraint with ‘?’ at end**: [Route("greetings/{message:maxlength(3)**?**}")]
* Classic Routing
  + RouteConfig.cs – RegisterRoutes(RouteCollection routes) {

routes.MapRoute(name, urlParams, defaults, constrants); // defaults & constrants use new anonymous object

* + Http Request Segments & Url Parameters Mapping – case-insensitive
    - **RequestContext.RouteValueDictionary** stores the UrlParameters.
    - **General Mapping**
      * **Http Request Segments**: e.g. **/**albums**/**display**/**123
      * **Parameters**: e.g. “**{**first**}/{**second**}/{**third**}**” **IMPORTANT: START WITH ‘{‘ NOT ‘/’**
        + After mapping: **first=albums, second=display, third=123**
    - **Special Parameters – {controller}, {action}**
      * **Http Request Segments**: e.g. **/**albums**/**display**/**123
      * **Parameters**: e.g. “**{**controller**}/{**action**}/{**id**}**”
        + After mapping: **controller=albumsController, action=display, id=123**

p.s. the suffix controller is added by MVC

* + - * + **Any additional Url Parameters is available as Action Method Parameters**.

e.g. ActionResult **Display**(**int id**) { … }

**Que. If Display() use 0 argument, will it get called if Url Param has {id} and Url Segment has 123???**

* + - **More Mapping Pattern**
      * **Hard Code Prefix: hardcodedPrefix/{**controller**}/{**action**}/{**id**}**
      * **Mixing Params & Literals**: e.g.
        + {language}**-**{country}/{controller}/{action}/{id} // the **dash ‘-‘** literal.

e.g. /English**-**Uk/albums/display/123

* + - * + {controller}**.**{action}**.**{id} // the **period ‘.’** Literal

e.g. /albums.display.123

* + Route Defaults
  + **1+ Optional & Default Url Parameters**

routes.**MapRoute**(“simple”, “{controller}/{action}/**{id}**”, **new {id = UrlParameter.Optional, action=”index”}**);

* + **All** **consecutive** optional & default params has to be used as the **last params**
  + Route Constraints
  + **Apply RegEx on a Url Segment** test for matches, if **false** then move on to the **Next** **Route** **Mapping**.
  + Routing Engine **Auto** **wrap** the regular **expression** w/ **^ and $** to ensure an **EXACT** **MATCH**.
  + e.g. routes.**MapRoute**(“blog”, “{**year**}/{**month**}/{**day**}”, new {controller=”blog”, action=”index”}

, **new {year=@”\d{4}”, month=@”\d{2}”, day=@”\d{2}”}**);

routes.**MapRoute**(“simple”, “{controller}/{action}/{id}”); // **if** the **above** **Map** Test **failed**, **this** **Map** Test will be **Next**.

Register & Generate Routes - ???

* **Classic Routing**
* **Algorithm**
  + 1. **If routeName exists**, use routeName to **find** a **route**
  + 2. **If routeName NOT exists,** use **routeValues** to **find** a **matched** **route**
  + 3. **When** route **found**, **fill** in each of the **Url** **parameters** **w/** the supplied **routeValues**.
* **Named Routes** – Specifically **used** **by** **RouteLink** to **avoid** Url **parsing** **Ambiguity**
  + **Register** your route with **unique** **route** **name**. e.g. **.MapRoute(routeName**, …);
  + **Generate Url** using **route** **name** **instead** **of** letting Route Engine **parsing** **routeValues** to figure out where to go to.

@Html.**RouteLink**(

linkText: “some display text”,

**routeName**: “test123”, // **with** **routeName** specified, **NO Parsing** is needed!!!

**routeValues**: new {controller=”section”, action=”Index”, id=123}

)

* + Compare to ActionLink which **ALWAYS** relies on **action** & **controller** params
* **MVC Areas** – **partition** your Models, Views, Controllers into **Sections**(**Areas**) – provide better organize a **Complex** **Site**
  + Derive: public class MyArea : AreaRegistration
  + Override: AreaName, RegisterArea
* **CatchAll** Parameter
  + routes.MapRoute(“catchallroute”, “query/{query-name}/**{\*extrastuff}**”);
  + /query/select/**a/b/c** OR /query/select/**a/b/c/** => **extrastuff = “a/b/c”**
* **Multiple Url Parameters in a Segment**
  + **Url Parameters are matched greedily**, i.e. route tries to match text as much as possible with each Url parameter
  + e.g. {foo}**xyz**{bar} for /**xyzxyzxyzblah** => foo=”**xyzxyz**” & bar=”**blah**”
* **IgnoreRoute** & StopRoutingHandler
  + Global.asax -> RegisterRoutes(…) { routes.**IgnoreRoute**(“{resource}.axd/{\*pathInfo}”); … }
* **Debugging** Routes
  + NuGet – Route Debugger