JOHN ABBOTT
CEGEP/COLLEGE

(420-PS4-AB)
Developing ASP .NET MVC (2)

Aref Mourtada

Summer 2018

Outline

- Action Results
- Custom Routes Issues
- Attribute Routing
- Constrains in Attribute Routing
- Passing Data to Views
- View Models
- Razor View Engine
- Partial Views
- Navigation
- Exercise

Action Results

ActionResult

- Base class for all action results in ASP .NET MVC.
- Depending on what an action does
  - It will an instance of one the classes that derive from action result.
- View method
  - A helper method inherited from the base Controller class.
  - Provide easy creation of a view result.

Action Result Sub Types

Type	Helper Method	Notes
ViewResult	**View()**	Most popular
PartialViewResult	**PartialView()**	Return partial view (print function)
ContentResult	**Content()**	Simple Text
RedirectResult	**Redirect()**	Go to a URL
RedirectToRouteResult	**RedirectToAction()**	Redirect to an action
JsonResult	**Json()**	Return serialized JSON object
FileResult	**File()**	Example file download
HttpNotFoundResult	**HttpNotFound()**	Page not found / 404 error
EmptyResult		When an action does need to return any value (void)

Demo: ActionResult

```
return Content("Hello World");

return HttpNotFound( );

return new EmptyResult( );

return RedirectToAction("Index","Home");

return RedirectToAction("Index","Home", new
{page=1,sortBy="name"});
```

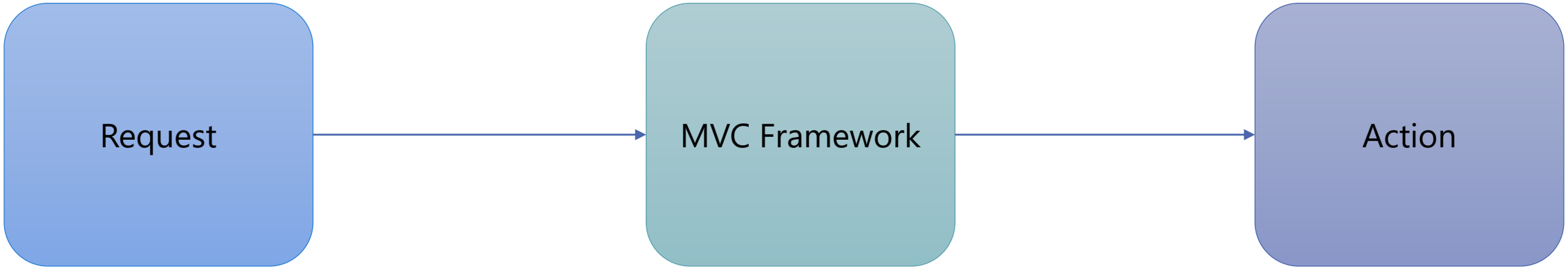
Final Word – Action Results

- No need to know all the details.
  - Know as you use.
  - MSDN is an excellent reference.
- Most used ActionResults
  - View( )
  - RedirectToAction( )
  - Redirect( )
  - HttpNotFound( )

Passing Data to Actions

Action Parameters

- Action parameters are the inputs to our actions.
- When a request comes ASP .NET MVC automatically maps request data to parameter values.
  - If an action requires a parameter, the framework looks for a parameter with the same name in the request data.
  - parameter with that name exists the framework will automatically pass the value of that parameter to the target action.



Form of Action Parameters

- Embedded in URLs:

  /media/edit/1
- In the query string

  /media/edit?id=1
- In the form data

  id=1

Demo: Action with parameter

- Create an action called “edit”
- Pass an integer value in the parameter list called “id”
- The action returns simple content
  - Value of the pass parameter.

- Test
  - Using URL passing
  - Query String

Demo: Action with parameter

- Change the passed integer name to “medaId”
- Test again
  - Using URL passing
  - Query String

  - Check route configuration.

Demo: Actions with multiple parameters

- Create an action method called index.
- Index takes two parameters:
  - int pageIndex
  - string sortBy

- Lets make the parameters optional
  - Provide a default value

  pageIndex =1 , sortBy = name

Custom Routes

- The default route works for most scenarios.

"{controller}/{action}/{id}“

- There are situations where we need a route with multiple parameters.

- Example:

/media/released/2017/7

Demo: Custom Routes

- Create action with two parameter to test it.
  - Action: released
  - Inputs: Year & Month

- Create custom route.

Issues with Custom Routes

- Once you work on large applications:
  - The RouteConfig file will grow really large with a lot custom routes.
  - Over time, it will be come a mess.
  - In addition, you always have to go back and forth between the actions and custom routes when update it required. (in action name or parameters).

Solution: Attribute Routing

- In ASP .NET MVC 5: MS introduced a cleaner and better cleaner way to create a custom route.
- Instead of creating a route in RouteConfig file, it can be applied using an attribute to the corresponding action.
- You need to know both ways of how create a custom route.
  - You might be working on an existing solution that uses convection based custom routes.

How to use Attribute Routing

- In order to use attribute routing, it needs to be enabled
- In RouteConfig file call this method:

  routes.MapMvcAttributeRout( );

- Above each action, add the attribute route, in the following format

[Route("controllerName/actionName/{par1}/{par2}"]

Demo: Attribute Routing

- Delete the custom route you added for the “released” action. (or comment it)
- Add attribute routing

[Route(“medias/released/{year}/{month}”]

- Add constraints
  - regex method
  - range method

Constrains in Attributes Routes

- min
- max
- minlength
- maxlength
- int
- float
- range
- regex

- Web search: ASP .NET MVC Attribute Route Constrains for any needed constrains.

Demo: Attribute Routing Constrains

- Add constrains to the created attribute route of the actin released

[Route("medias/released/{year}/{month}")]

Use range, regex

- Add an attribute route and constraints on the "index" action created in an earlier demo.
  - Test different options.

Passing Data to Views

Passing Data to Views

- Argument though the *View* method
  - Demonstrated in previous section, we saw that we can pass data to a view by passing as an argument though the View method.
  - Check “random” action in the Medias controller.

- ViewData Dictionary: a second type to pass data to views.
  - Every controller has property called ViewData

How to use ViewData with Views

- In the action method, add the object you need to access to the ViewData

  Syntax: ViewData[“SelectedName”] = objectName;

- In the view, instead on using model property, we use @ViewData and access the specified object
  - Requires casting to be able to access the object properties.

- ViewBag is an improved version of ViewData, but suffers from the same issues
  - Magic strings / properties
  - Casting

Final Word: Passing Data to View

- Old methods will be hard to maintain in big applications
- You need to know them if you come across them in a project.
- Stick to passing objects as an argument as it is safer and easier.

View Models

View Models

- So far we have seen that in a view, we managed to access a single object passed to it.
  - What if we need to access two objects from two different Models.
  - In our domain model there may not be a direct relationship between the models we want to display.

  Example: Display a media and customers who rented it

  Currently there is not relation between these two classes.
- A view model is a “Model” specifically built for a view.
- Includes any data and rules related to that view.

Demo: ViewModels

- Notes: Model folder is used only for the domain classes.
- Create a new folder: ViewModels.
- Create a new class inside that folder.
- ViewModel classes convection name: name+ViewModel suffix.
- Now add needed properties to that class
  - Media
  - List of Customers
- Do not forget the namespace

Demo: ViewModels (2)

- Create a new action “listCustomers”
  - Let us create a movie & a list of customers.
  - Create a ViewModel object and add them to that object.
  - Pass the new object to the view
  - Create a view and use @demo & @Demo to access our new object.

  - Access the data from that object in the view.

Razor Engine

Razor View Engine

- Allow us to swap between C# and HTML
- We add an ,“@” at sign, then start typing C# code.
- HTML code can be embedded within that code.
- You need to add a code block for any condition statement or loop even if it has one statement only.

Razor View Examples

```
@if (...)
{
    // C# code or HTML
}

@foreach (...)
{
}
```

Demo: Razor View

- Create a foreach loop to access the customer list from the previous demo.
- Add the items to a list
  - Use: <ul> & tags

- Add a custom message if there are not customers that rented that movie.
  - We need an “if” statement

Razor View Tip

- Adding comments in a razor view

```
@*
    Comment
    Using more than one line.
*@
```

Partial Views

Partial Views

- A partial view is like a mini version of a view that can be reused within different views.

- Partial view are necessarily for reusing markup.
  - They can be used to breakup a large view into smaller and more maintainable partial views.

Demo: Partial Views

- Open the "_Layout.cshtml" and lets examine it.
- Check navbar element
  - Maintainability of the layout can be improved by extracting the navbar into a partial view
- Under View → Shared → Add view
  - Name: _NavBar (convention to used _Name)
  - Tick partial view check box
  - Move the navbar element to the new partial view.
  - Use @Html.Partial("_NavBar") to add it is main layout.

Navigation

Creating Links

- The simplest way to create links in ASP.NET MVC application is to use raw HTML:

```
<a href="/Medias/Index">View Available Media</a>
```

- or Use *ActionLink* method of *HtmlHelper* class:

```
@Html.ActionLink("View Available Media", "Index",  "Medias")
```

- If the target action requires a parameter → use an anonymous object to pass parameter values:

```
@Html.ActionLink("View Available Media", "Index", "Medias",
new{id=1},null)
```

ActionLink VS raw HTML

- ActionLink queries the routing engine for the URL associated with the given action.
  - If you have a custom URL associated with an action and change that URL in the future, ActionLink will pick the latest URL, so you don't need to make any further changes.
  - But with raw HTML, you need to update your links when URLs are changed.

- Note: Avoid changing URLs as possible.
  - Because URLs are the public contract of your web application.
  - They can be referenced by other apps or bookmarked by users.
  - Changing them will create broken bookmarks and external references.

Exercise

(1) Updating Navigation Bar & Footer

- Change the navigation bar elements
  - Add your store name
  - Add a link for Media
  - Add a link for Customers.

  - Remove all other links
- Update foot to include store name.

(2) Customer Controller

- Update Index Action in Customer Controller to return a list of available customers
  - Write a method “getCustomers” to return 2 staged customers.
- Update the detail action
  - Check if the passed customer id exists using the method getCustomers.
  - Display customer details in the new view of 404 error if does not exsist.

(2) Customer Views

- Update Index View for Customers
  - Display passed customers in table.
  - Add to each element in the table add link to the details action to display the details of a customer.
  - And if there no customer to display a message that there are no customers.
- Update the detail view
  - Display name and ID of customer.

Exercise: Media Controller

- Update Index Action in Customer Controller to return a list of available customers
  - Write a method “getMedias” to return 2 staged Media objects.

Exercise: Media Views

- Update Index View for Medias
  - Display passed media objects in table.
  - Add to each element in the table.

Q & A