

FORGING AHEAD

with

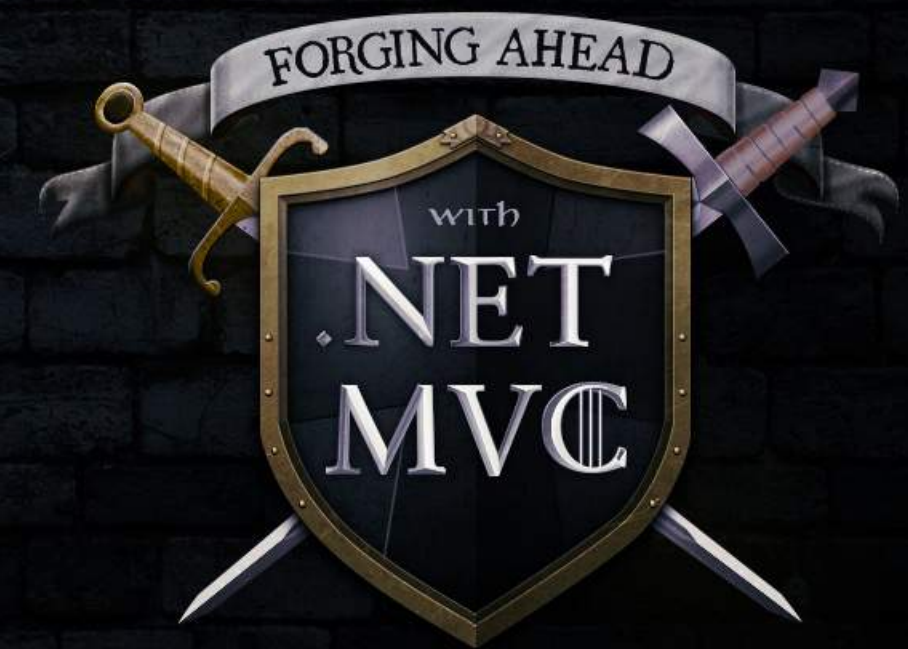
.NET
MVC



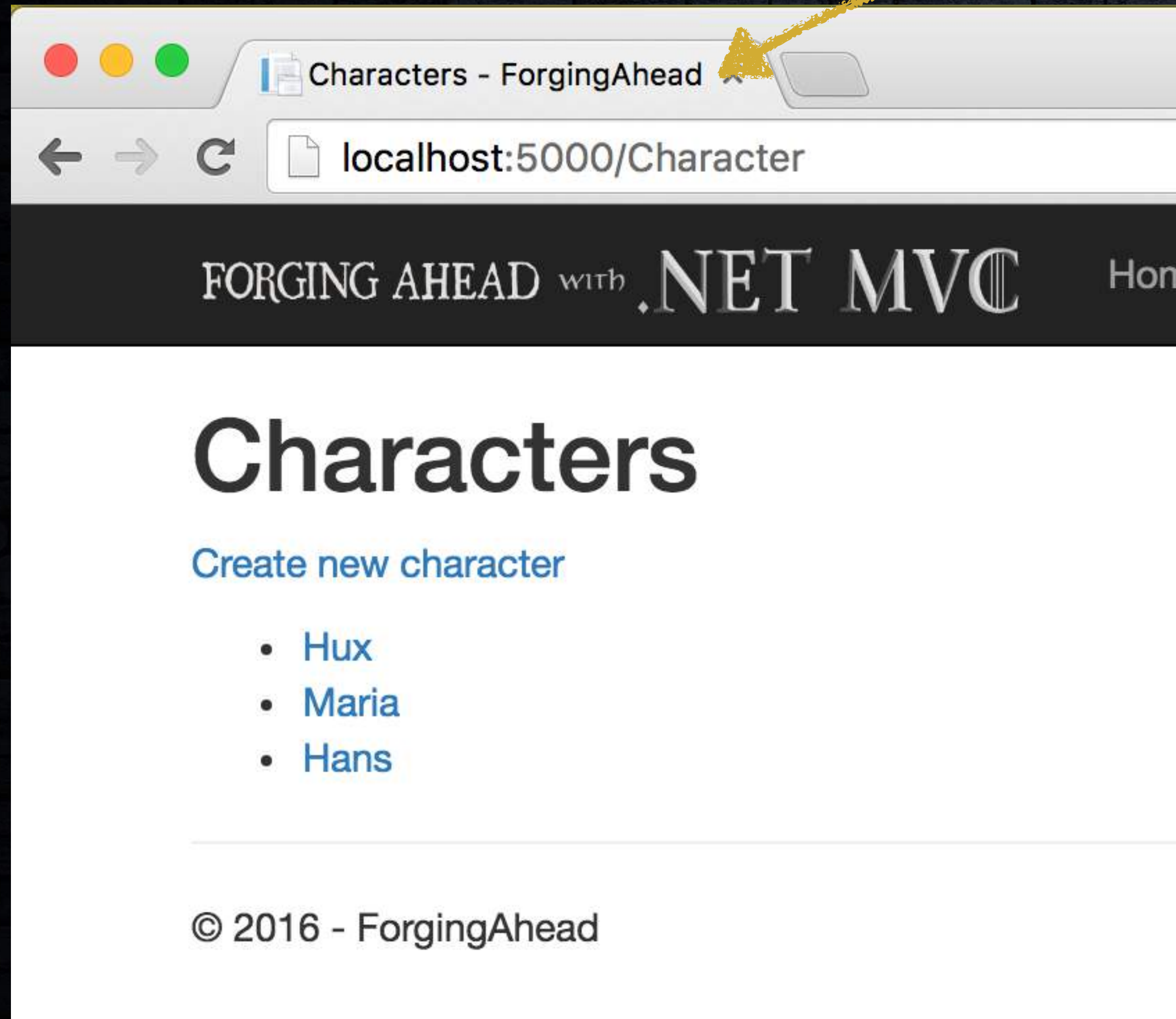
Level 3 – Section 1

Making Our Presentation Smarter

Layouts



We Want Our Title and Headers to Be Smarter



We want clear and concise titles to show up in both the title and search results.

Search Results Example

[Characters - ForgingAhead](#)
localhost:5000/Character
Lists all characters in ForgingAhead.

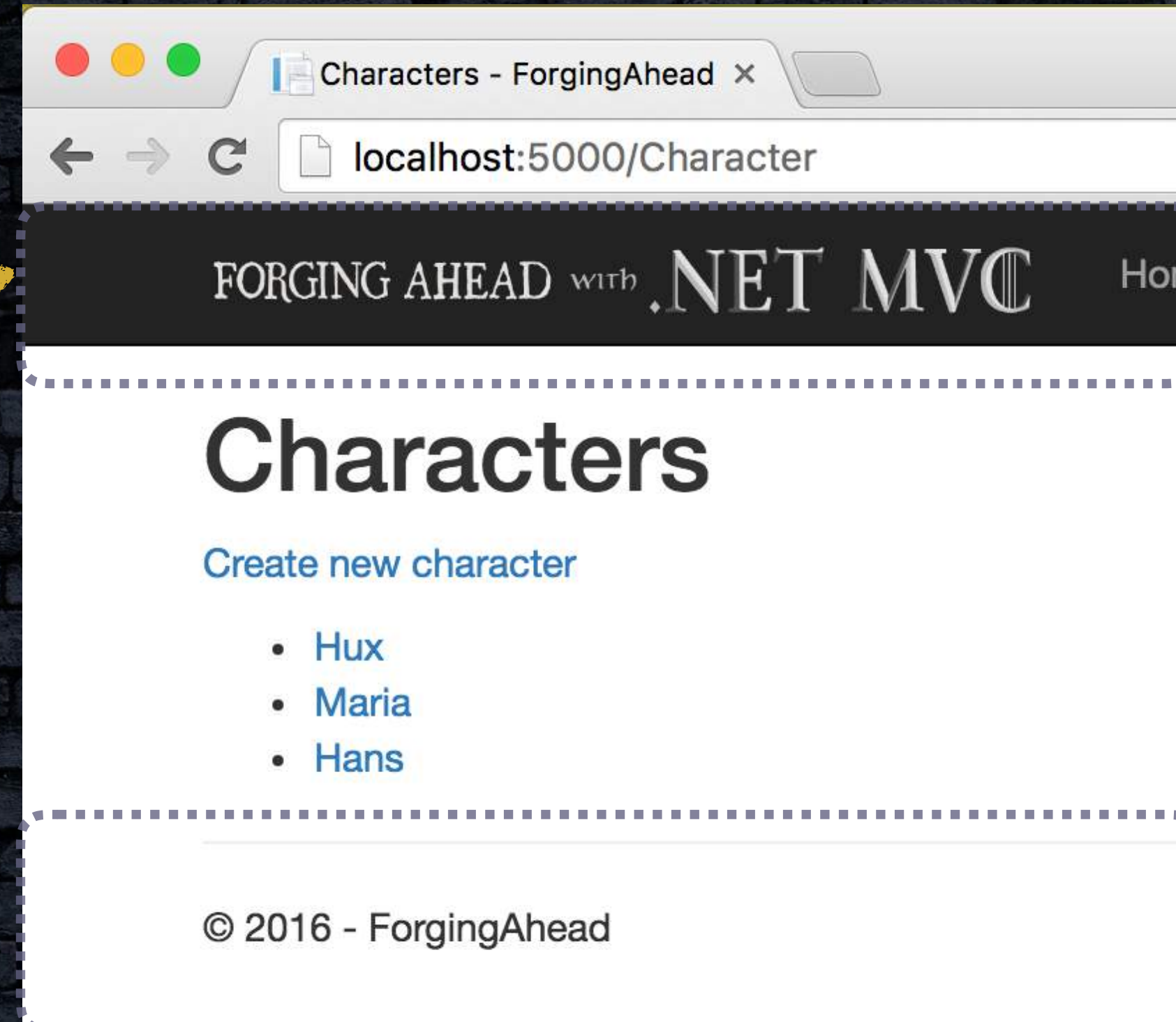
We want to specify our description data for our search results.

We want to handle this application-wide, but where could we set this?

Some Content Is the Same on All Pages

When viewing our application, you'll notice parts of the page don't change page-to-page.

Navigation is the same on all pages.



The only content that is changing is this part in the middle, known as the body.

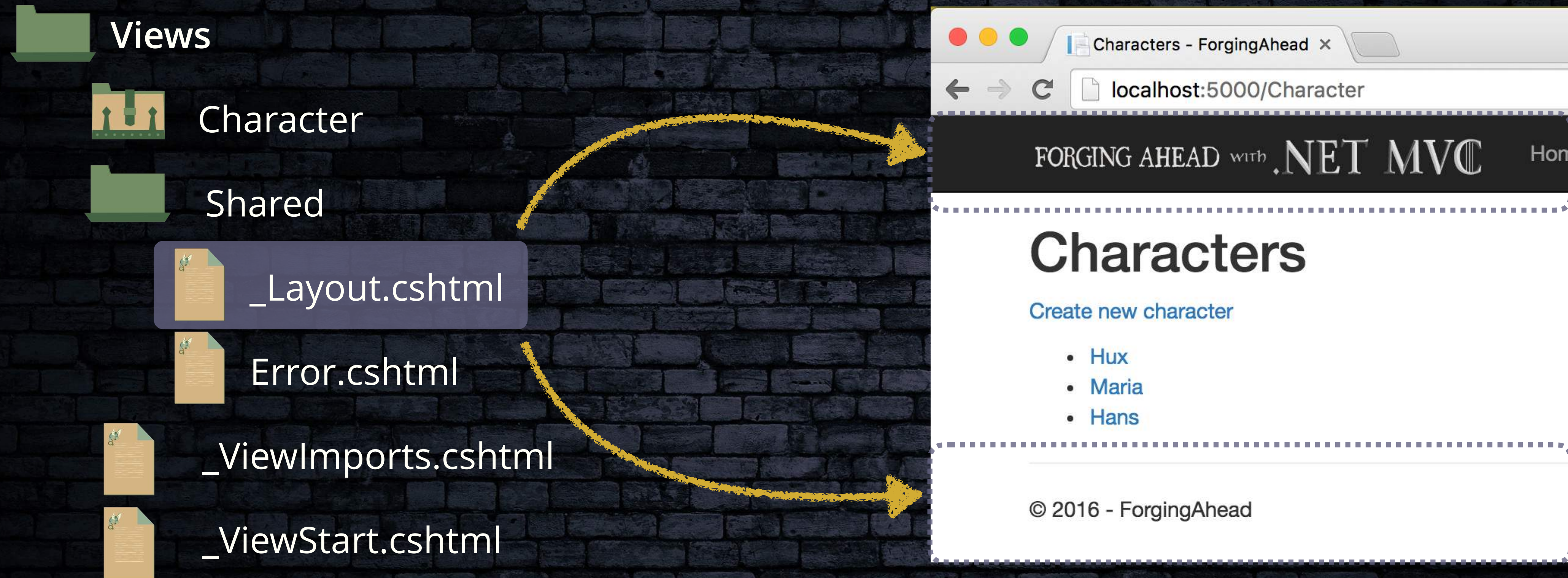


Footer is the same on all pages.



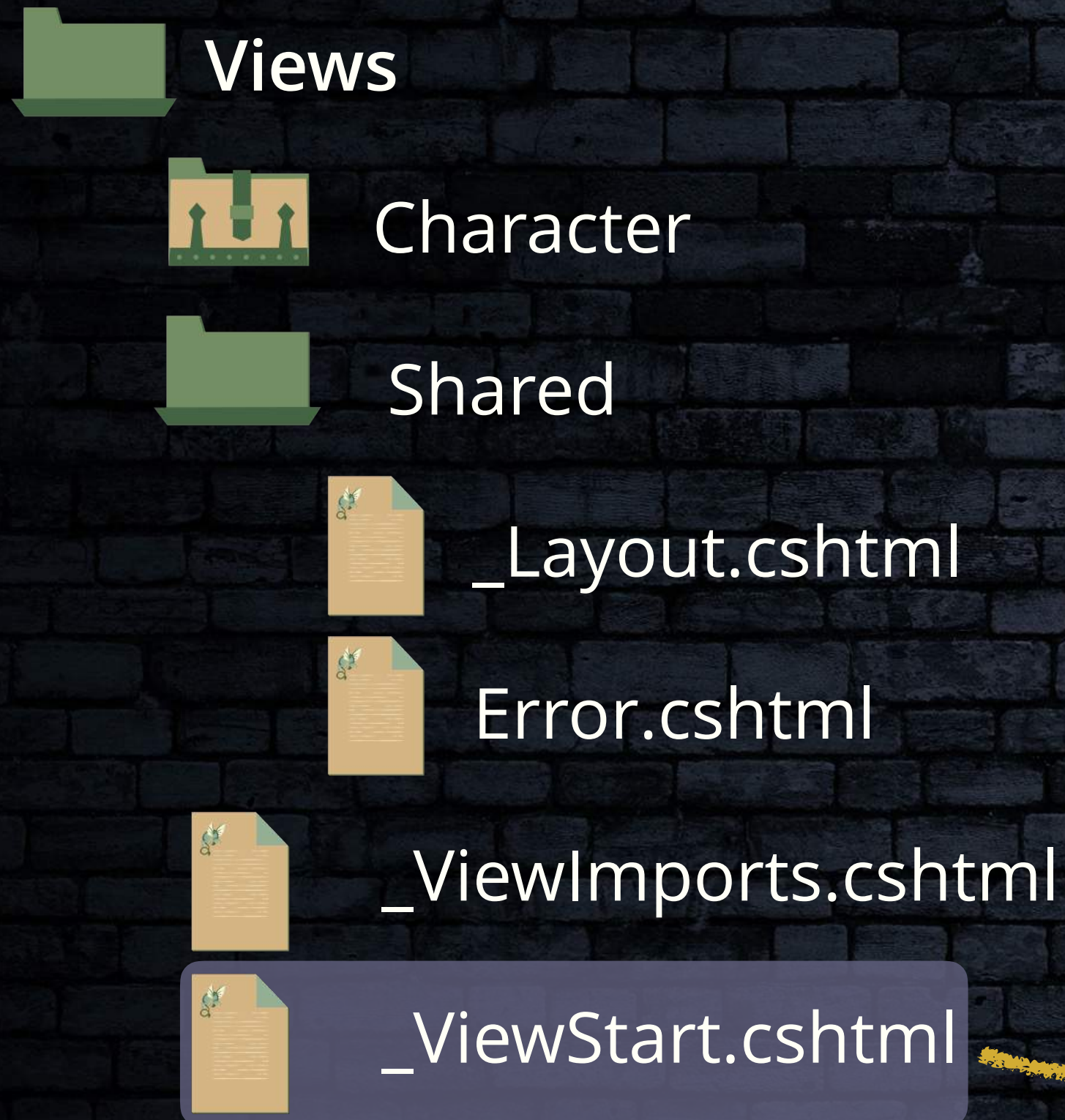
Default Layout Page

The `_Layout.cshtml` file contains elements we can use on multiple pages.



Where Is Our Layout Set?

_ViewStart allows us to set code that applies when you return a view — for example, "Return View();".



Views/_ViewStart.cshtml

CSHTML

```
@{  
    Layout = "_Layout";  
}
```



Layout is a keyword in Razor that sets what layout the current view will use. We don't need the .cshtml after the layout's file name.

Default Layout Page

Views/Shared/_Layout.cshtml

CSHTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - ForgingAhead</title>

  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
  @RenderSection("header", required: false)
</head>
<body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      ...
```

So here is all the HTML we've been seeing and weren't sure where it was coming from!

Layout Parts: ViewData

Views/Shared/_Layout.cshtml

CSHTML

```
...  
<head>  
  <meta charset="utf-8" />  
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
  <title>@ViewData["Title"] - ForgingAhead</title>  
  
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />  
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />  
  @RenderSection("header", required: false)  
</head>  
...
```

ViewData is a dictionary that allows us to pass data from our controller to our view.



A dictionary contains key-value pairs where ["string"] specifies the key, and we set and use it like a variable.

Layout Parts: RenderSection

Views/Shared/_Layout.cshtml

CSHTML

```
...
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - ForgingAhead</title>

  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
  @RenderSection("header", required: false)
</head>
...
```



RenderSection lets us render content to specific named sections of our page. They can be required or optional.

Layout Parts: RenderBody

Views/Shared/_Layout.cshtml

CSHTML

```
<!DOCTYPE html>
<html>
  <body>
    <div class="navbar navbar-inverse navbar-fixed-top">...</div>
    <div class="container body-content">
      @RenderBody()
      <hr />
      <footer>
        <p>&copy; 2016 - ForgingAhead</p>
      </footer>
    </div>
    ...
  </body>
</html>
```

RenderBody is where all content not set to a specific section will be rendered.

Putting Our Layout to Use

Now that we see our layout's parts, let's use them to solve our problem.

- Use ViewData for our page titles
- Use Sections to set some page metadata
- ✓ Use RenderBody to display our content



*RenderBody works automatically and
has been working this whole time.*

Adding ViewData to Our Controller

Controllers\CharacterController.cs

CS

```
...
public class CharacterController : Controller
{
    ...
    public IActionResult Create(Character character) {...}

    public IActionResult Index()
    {
        ViewData["Title"] = "Characters";
        var model = _context.Characters.ToList();
        return View(model);
    }
}
...
```

ViewData is a dictionary, so we need to set it as such. _Layout is expecting the "Title" key, so we need to set that for it to map up.



ViewData is a property of the controller that is directly passed into the ViewData property of the view automatically when a view is returned.

ViewData Is Carried From Controller to View

CS

./Controllers/CharacterController.cs

```
public class CharacterController : Controller
{
    public IActionResult Index()
    {
        ViewData["Title"] = "Characters";
        var model = _context.Characters.ToList();
        return View(model);
    }
}
```

This will set our layout's ViewData, resulting in "Characters - ForgingAhead".

Views/Shared/_Layout.cshtml

```
<head>
    ...
    <title>@ViewData["Title"] - ForgingAhead</title>
    ...
</head>
```


Setting Our View to Populate a Section

CSHTML

./Views/Character/Index.cshtml

```
@model List<ForgingAhead.Models.Character>
```

```
@section header{  
    <meta name="description" content="Lists all characters in ForgingAhead.">  
}
```

```
<h1>  
    Characters  
</h1>  
...
```

We'll set metadata that search engines will display when listing our page in our header section.

Views/Shared/_Layout.cshtml

```
...  
<head>  
    ...  
    @RenderSection("header", required: false)  
</head>  
...
```


@section Replaces the Matching @RenderSection

CSHTML

./Views/Character/Index.cshtml

```
@model List<ForgingAhead.Models.Character>
```

```
@section header{
```

```
    <meta name="description" content="Lists all characters in ForgingAhead.">
```

```
}
```

```
<h1>
```

```
    Characters
```

```
</h1>
```

```
...
```

@section will match up with any @RenderSection with a matching name and replace it with its content.

Views/Shared/_Layout.cshtml

```
...
```

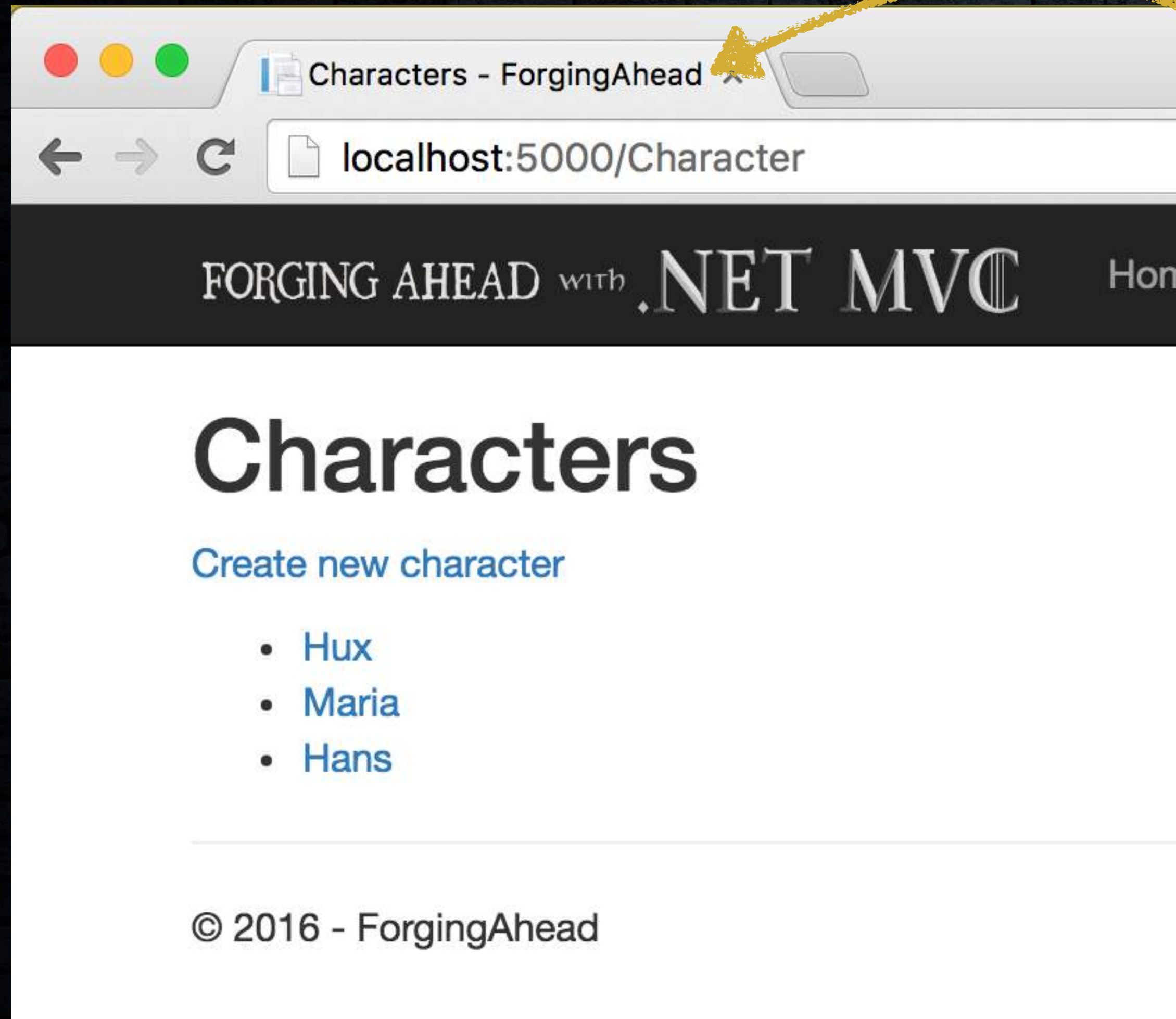
```
<head>
```

```
    @RenderSection("header", required: false)
```

```
</head>
```

```
...
```


The Result of Using Our Layout



Our ViewData has set the titles in both our browser and search results.

Search Results Example

Characters - ForgingAhead
localhost:5000/Character
Lists all characters in ForgingAhead.

Our header section set the text used to describe our page in search results.

Some Conventions to Note

There are several conventions you should be aware of when it comes to layouts.

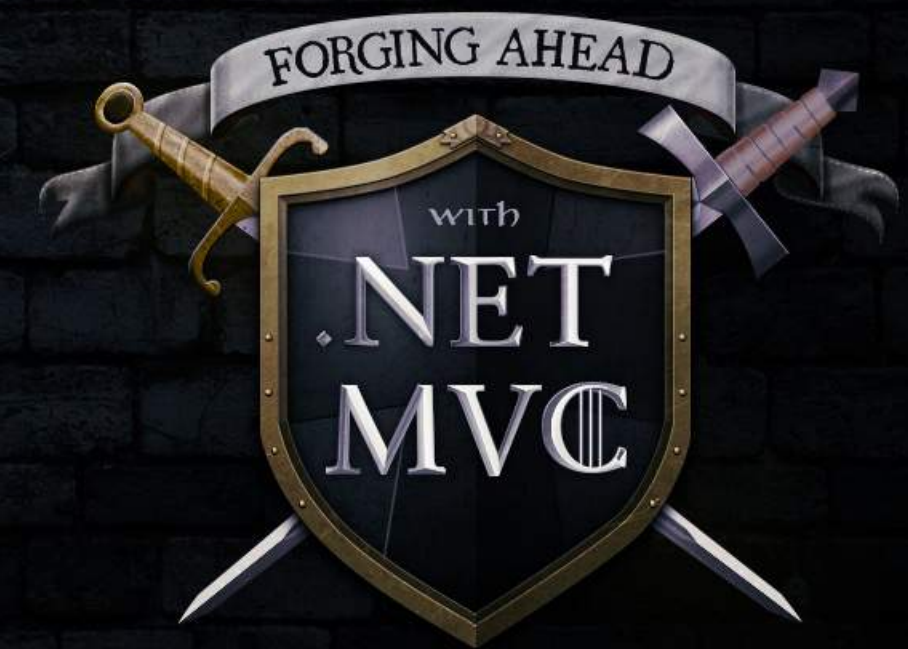
- Layout naming convention is `_PascalCase` (note the underscore).
- All layouts typically go in the `Shared` folder.
- Section names are typically lowercase.
- If you have content that will only be displayed on some but not all pages using your layout, then that content shouldn't be in your layout.



Level 3 – Section 2

Making Our Presentation Smarter

Partials



Reusing Our Views With Partial Views

Our character details appear multiple times here and will be on our Details page as well.

We should make this into a partial view to avoid duplicating code.



FORGING AHEAD with .NET MVC [Home](#) [Characters](#) [Equipment](#)

Active Characters

Hux

Level	3	<table><thead><tr><th>Name</th><th>Att</th><th>Def</th></tr></thead><tbody><tr><td>Sword</td><td>5</td><td>1</td></tr><tr><td>Shield</td><td>1</td><td>3</td></tr></tbody></table>	Name	Att	Def	Sword	5	1	Shield	1	3
Name	Att	Def									
Sword	5	1									
Shield	1	3									
Strength	6										
Dexterity	6										
Intelligence	3										

Maria

Level		
Strength		
Dexterity		
Intelligence		

Hans

Level	3	<table><thead><tr><th>Name</th><th>Att</th><th>Def</th></tr></thead><tbody><tr><td>Sword</td><td>5</td><td>1</td></tr><tr><td>Shield</td><td>1</td><td>3</td></tr></tbody></table>	Name	Att	Def	Sword	5	1	Shield	1	3
Name	Att	Def									
Sword	5	1									
Shield	1	3									
Strength	8										
Dexterity	5										

Creating Our _CharacterSheet Partial View



Pascal case preceded by an underscore is the standard naming convention for partial views, editor views, and layouts.

Our _CharacterSheet Partial View

Controllers\Shared_CharacterSheet.cshtml

CSHTML

```
@model ForgingAhead.Models.Character
```

```
<div class="col-md-5 bordered-div">
```

```
<h4>
```

```
    @Model.Name
```

```
</h4>
```

```
<hr/>
```

```
<div class="col-md-6">
```

```
    <div class="form-group">
```

```
        <label asp-for="Level" class="control-label col-md-8"></label>
```

```
        <div class="col-sm-3">
```

```
            <p class="form-control-static">@Model.Level</p>
```

```
...
```

_CharacterSheet will just be a formatted view for all of our characters' properties.

Using Our Partial View in a View

Our Details view will just have its model declaration and the @Html.Partial helper.

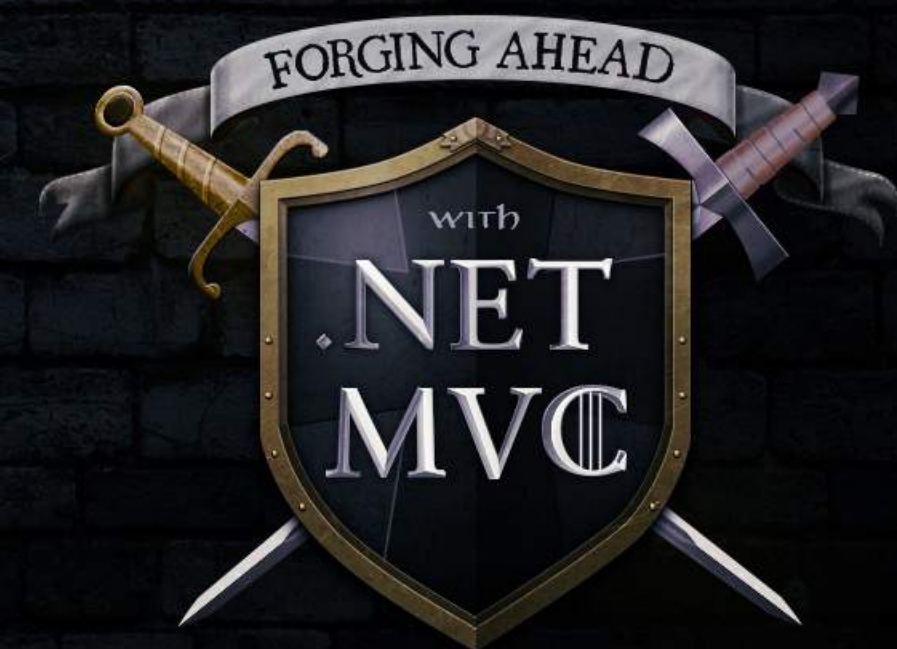
Views/Character/Details.cshtml

CSHTML

```
@model ForgingAhead.Models.Character  
  
@Html.Partial("_CharacterSheet", Model)
```

When this view loads, this line will be replaced with the contents of the _CharacterSheet partial.

We also need to pass our Character into the partial.



Results of Our Partial on Our Details Page

The end user viewing our Details page will have no way of telling we used a partial.

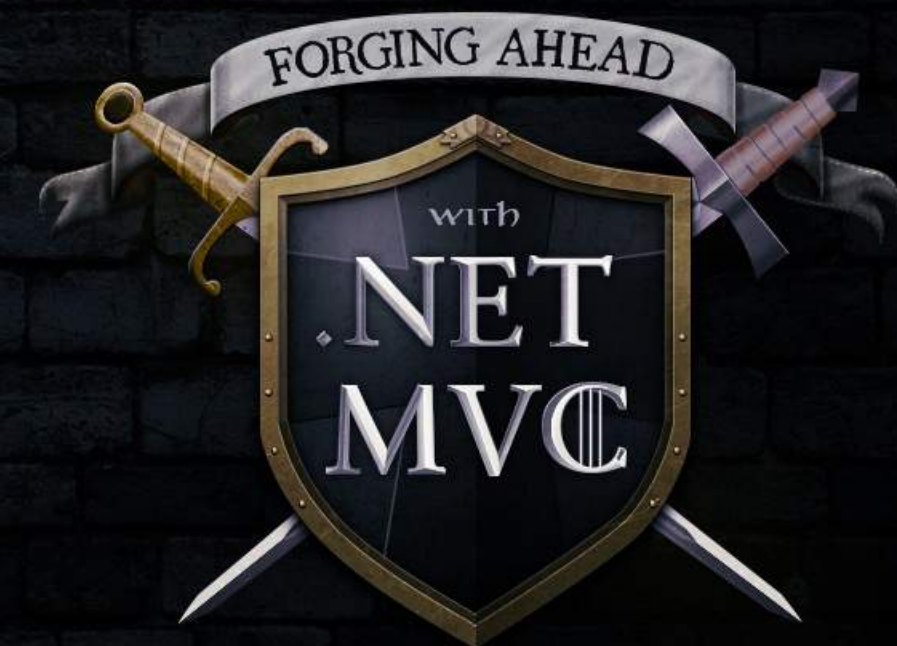
We can reuse this partial on the Index page and Character Details page.

FORGING AHEAD with .NET MVC Home Cr

Hux

Level	3	Name	Att	Def
Strength	6	Sword	5	1
Dexterity	6	Shield	1	3
Intelligence	3			

© 2016 - ForgingAhead



Set Our Home/Index to Use _CharacterSheet

Now we have what we need to set up our Active Characters List on Home/Index.



Views/Home/Index.cshtml

CSHTML

```
@model List<ForgingAhead.Models.Character>
<h1>
    Active Characters
</h1>
<div class="row">
    @foreach (var item in Model)
    {
        @Html.Partial("_CharacterSheet", item)
    }
</div>
```

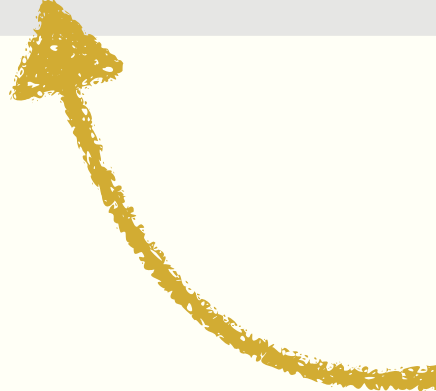
Instead of doing <Label>@item.Name</Label>, we'll pass item through our @Html.Partial.

Setting Up Home/Index to Get Active Characters

Controllers\HomeController.cs

CS

```
...  
public class HomeController : Controller  
{  
    ...  
  
    public IActionResult Index()  
    {  
        var model = _context.Characters.Where(e => e.IsActive).ToList();  
        return View(model);  
    }  
}  
...
```



We can just set our model to all characters where IsActive is true, then pass that back to our view.

We're All Set Up on Partials!

Now we can update _CharacterSheet once instead of updating both Details and Index.

Details Page

FORGING AHEAD with .NET MVC

HomeCh

Hux

Level	3	Name	Att	Def
Strength	6	Sword	5	1
Dexterity	6	Shield	1	3
Intelligence	3			

© 2016 - ForgingAhead

Index Page

FORGING AHEAD with .NET MVC

HomeCharac

Active Characters

Hux

Level	3	Name	Att	Def
Strength	6	Sword	5	1
Dexterity	6	Shield	1	3
Intelligence	3			

Hans

Level	3	Name	Att	Def
Strength	8	Sword	5	1
Dexterity	5	Shield	1	3
Intelligence	2			

© 2016 - ForgingAhead

Some Conventions to Note

There are several conventions you should be aware of when it comes to partials.

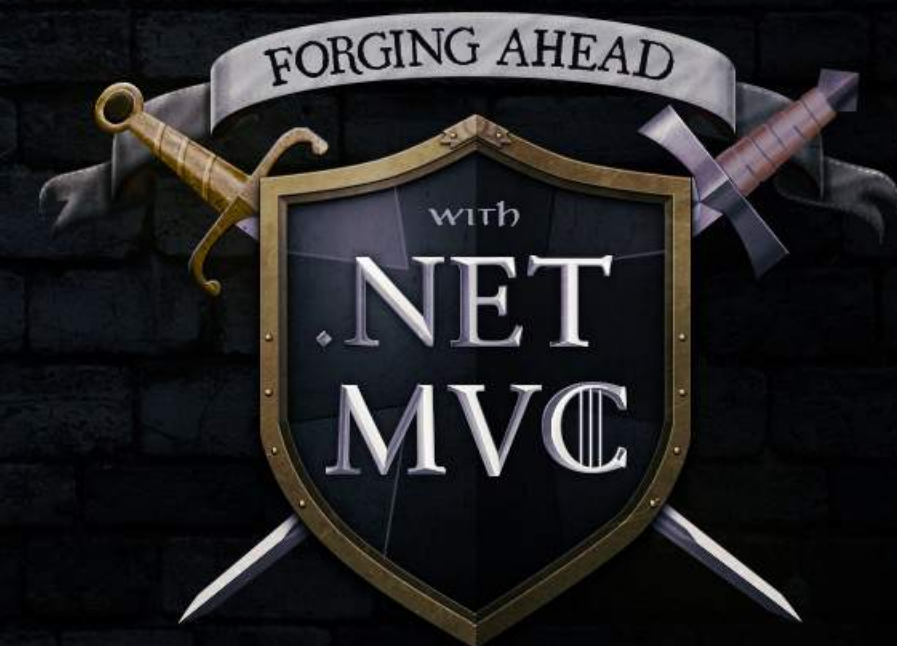
- Partials naming convention is `_PascalCase` (note the underscore).
- All partials can be placed in the **Shared** folder *or* partials can be placed in folders based on what model they use.

Partials in Shared



OR

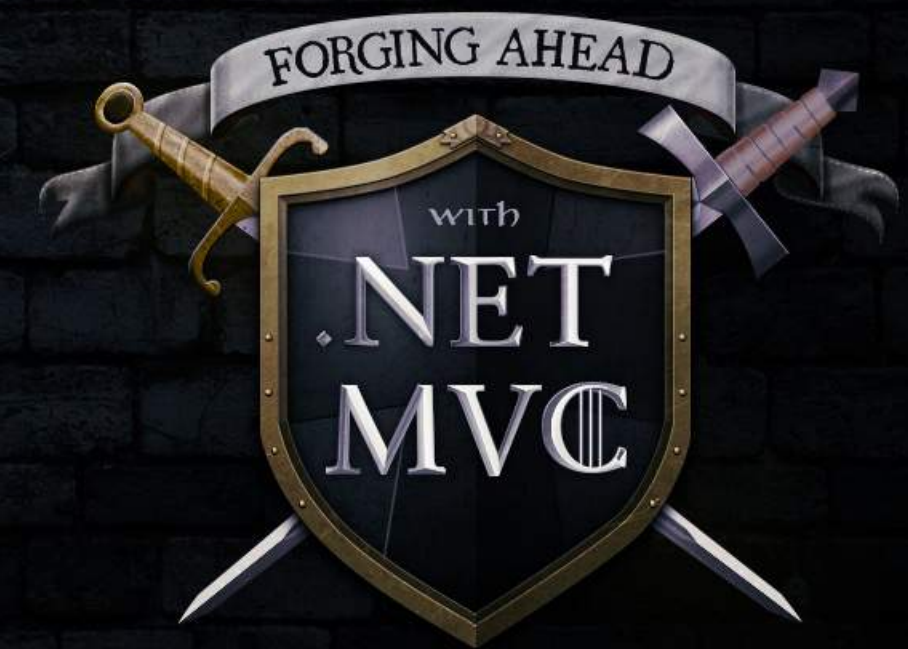
Partials Organized by Model



Level 3 – Section 3

Making Our Presentation Smarter

Editor Templates



Updating Our Characters

Our players will need to occasionally update their characters.

We need to make a page like our Create page so we can easily update our characters.

FORGING AHEAD with .NET MVC [Home](#)

Edit Character

Name

Is Active

☒

Level

Strength

Dexterity

Intelligence

© 2016 - ForgingAhead

Adding Edit Action to Controller


The only difference between our Edit action and Details action is the title and returned view.

Controllers/CharacterController.cs

CS

```
...
public class CharacterController : Controller
{
    ...
    public IActionResult Edit(string name)
    {
        ViewData["Title"] = "Edit " + name;
        var model = _context.Characters.FirstOrDefault(e => e.Name == name);
        return View(model);
    }
    ...
}
```

Change our Title to be "Edit Character.Name".



Using the EditorForModel Helper in the Edit View

We could make the edit view by writing all of the HTML by hand, but using editor helpers can simplify things for us.

Views/Character/Edit.cshtml

CSHTML

```
@model ForgingAhead.Models.Character
<h2>
    Edit Character
</h2>

<form asp-action="Update" asp-controller="Character">
    @Html.EditorForModel()
    <input type="submit" value="Update" class="btn btn-default" />
</form>
```

Set up our asp-action and asp-controller to work with Controller/Update.

@Html.EditorForModel automatically creates labels and input fields in this form based on properties in the model.

Automatically Generated Template

EditorForModel will automatically generate a template using its properties.

Models/Character.cs

CS

```
public class Character
{
    public string Name { get; set; }
    public bool IsActive { get; set; }
    public int Level { get; set; }
    public int Strength { get; set; }
    public int Dexterity { get; set; }
    public int Intelligence { get; set; }
}
```

FORGING AHEAD with .NET M

Edit Character

Name

Hux

Is Active



Level

3

Strength

6

Dexterity

6

Intelligence

3

Update

© 2016 - ForgingAhead

Our Template Works but Could Be Better

This looks kind of like our Create view, but the styling is a bit underwhelming.

FORGING AHEAD with .NET M

Edit Character

Name

Is Active
☒

Level

Strength

Dexterity

Intelligence

© 2016 - ForgingAhead

There are three different field types in this form.

String

Boolean

Int32 (Integer)

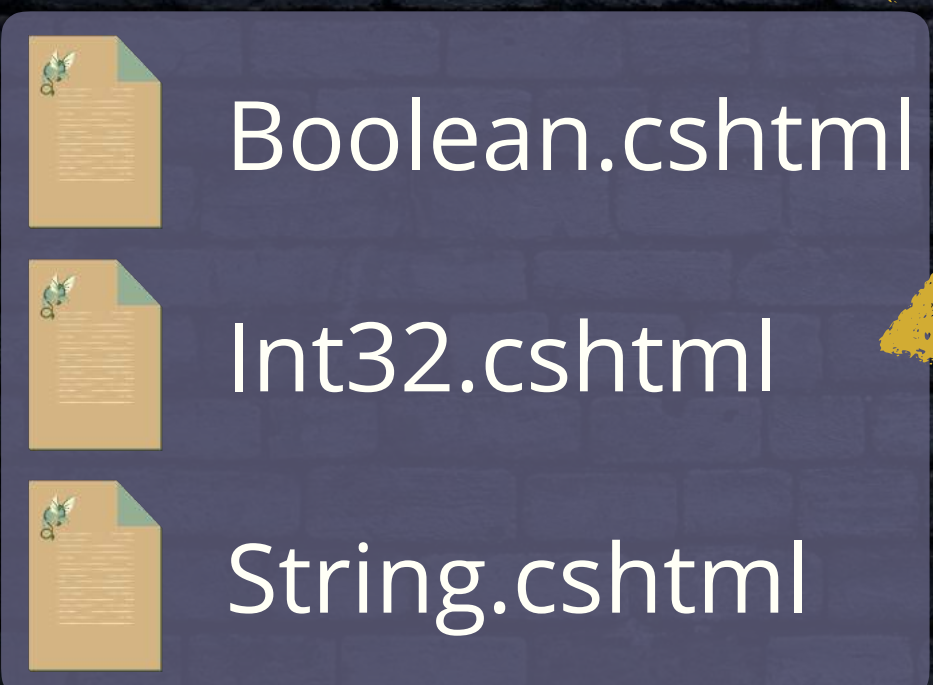
The spacing between our Update button and Intelligence field is especially bad.

Overriding Editor Templates

EditorFor will use the templates in our EditorTemplates folder instead when available.



First, create an EditorTemplates folder in our Views/Shared directory.



Then, add Boolean, Int32, and String.cshtml files to that new folder.



Editor templates can be applied to any datatype or object.

String.cshtml Editor Template

EditorFor will use this template any time it needs to handle a string.

Views/Shared/EditorTemplates/String.cshtml

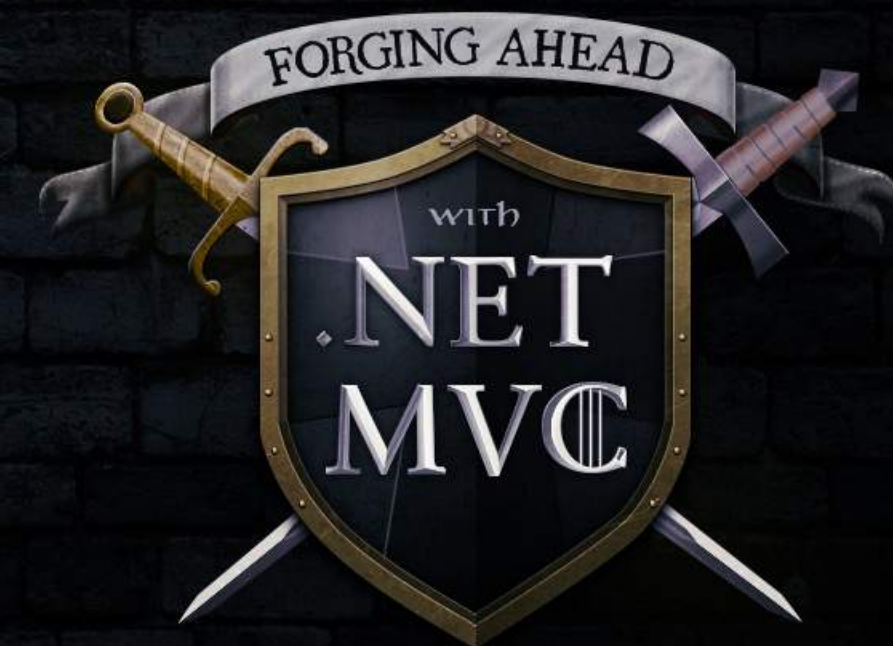
CSHTML

```
@model string  
<input class="form-control" asp-for="@Model" /> <br />
```

This datatype would be int32 or boolean in the other templates.

This is a style from the Bootstrap framework that we can use to style this input.

Our Int32.cshtml and Boolean.cshtml will be the same with their respective datatypes as their models.



Now We Have a Proper Edit View

Our players can update their characters and, as an added bonus, the form looks decent.

before

FORGING AHEAD with .NET MVC

Edit Character

Name

Is Active

☒

Level

Strength

Dexterity

Intelligence

© 2016 - ForgingAhead

after

FORGING AHEAD with .NET MVC [Home](#)

Edit Character

Name

Is Active

☒

Level

Strength

Dexterity

Intelligence

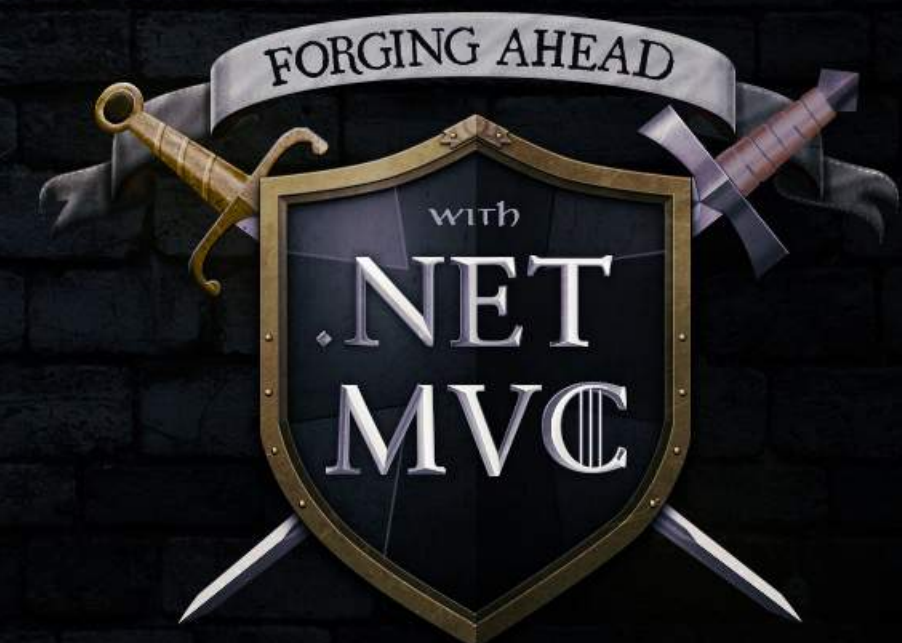
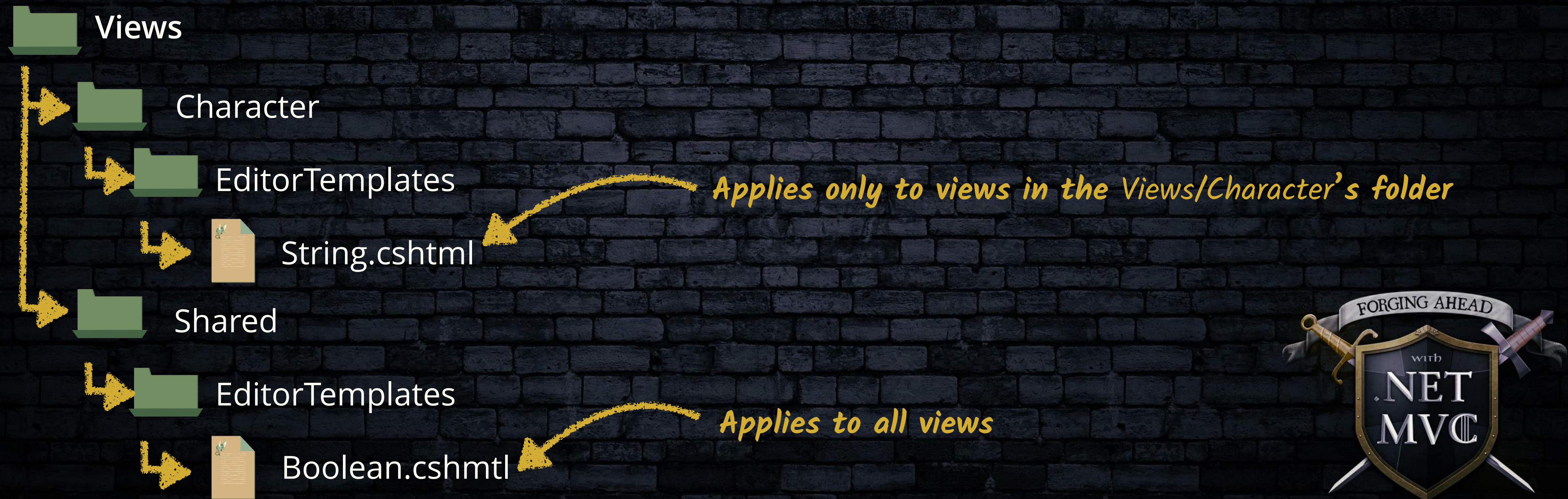
© 2016 - ForgingAhead

Editor templates are a way to set input styles globally throughout your application.

Some Conventions to Note

There are several conventions you should be aware of when it comes to editor templates.

- Editor templates naming convention is PascalCase.
- Editor templates go in an **EditorTemplates** subfolder, which can be located in different folders depending on the intended scope.



FORGING AHEAD

with

.NET
MVC

