
FORGING AHEAD
with
.NET
MVC

Level 2 – Section 1

Showing Off Our Data

Relating Data

Planning Out Our Data Structure

Before we can implement our data structure, we need to plan it out.

We know our players will be playing as characters.

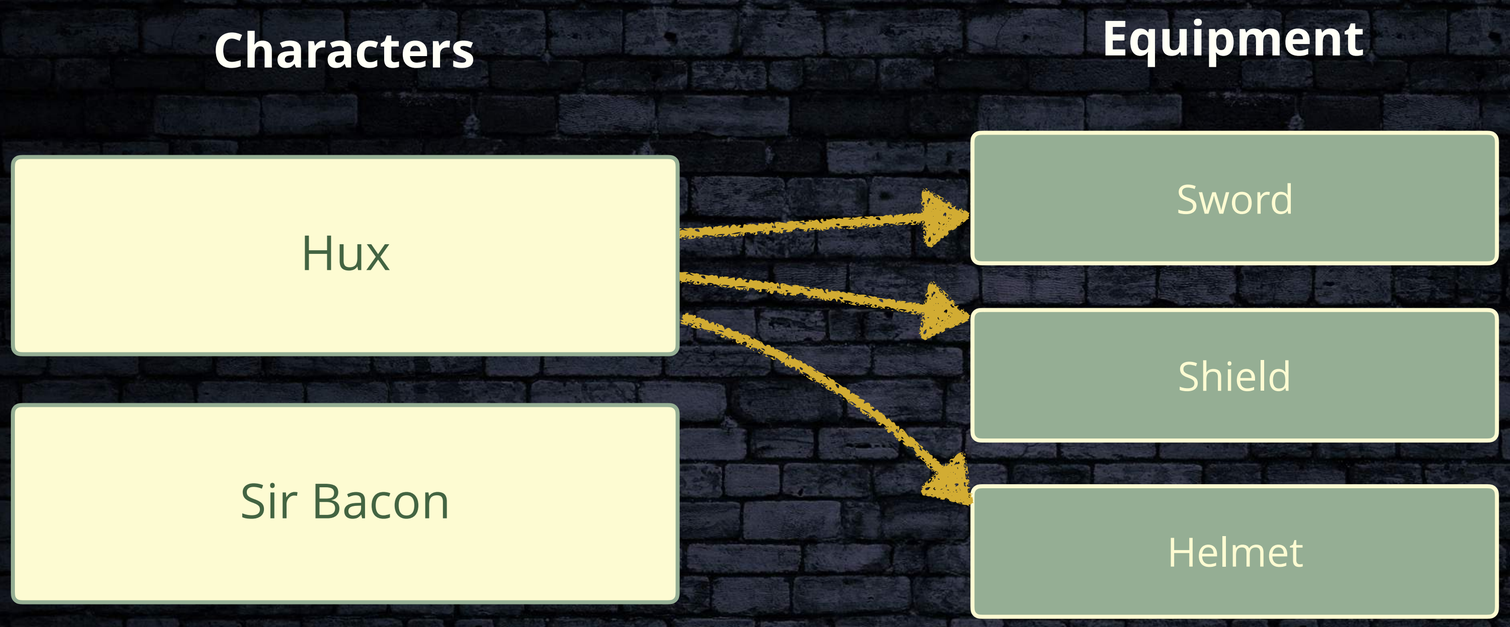
Characters

Hux

Sir Bacon

Characters Have Equipment

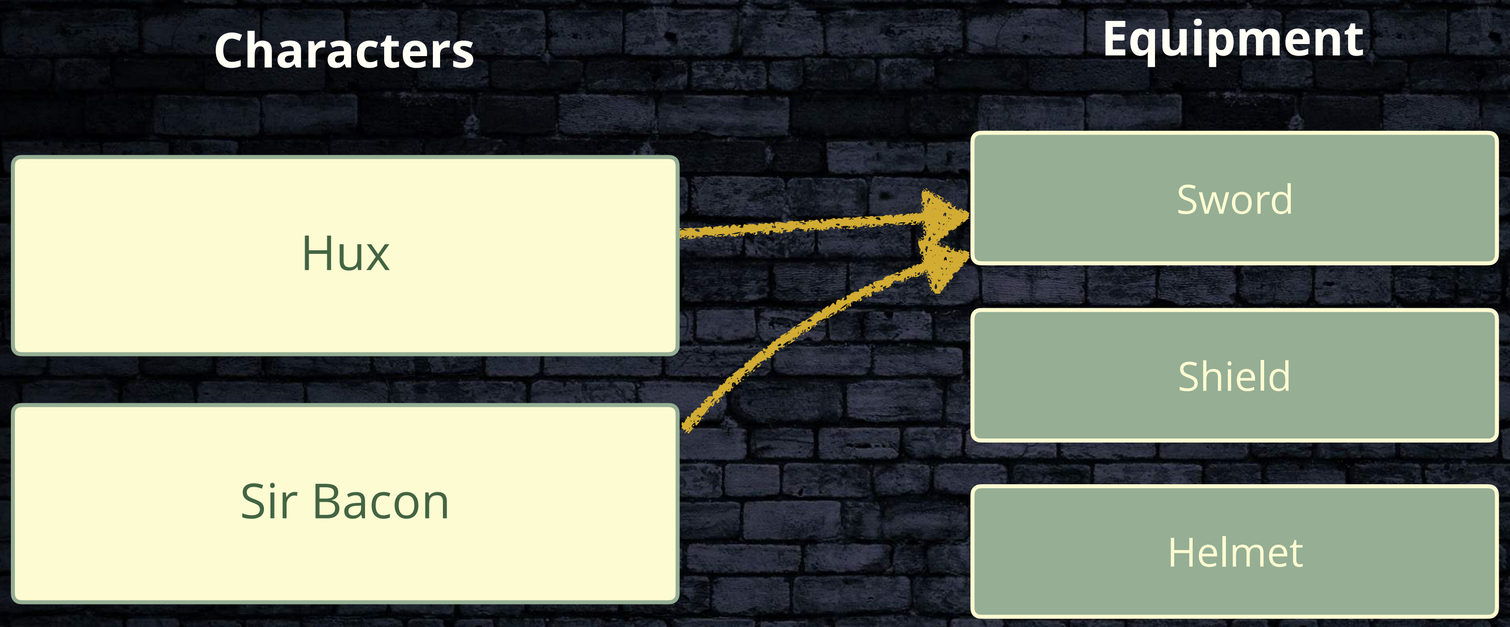
Each character can carry multiple pieces of equipment.



This implies a one-to-many relation between characters and equipment.

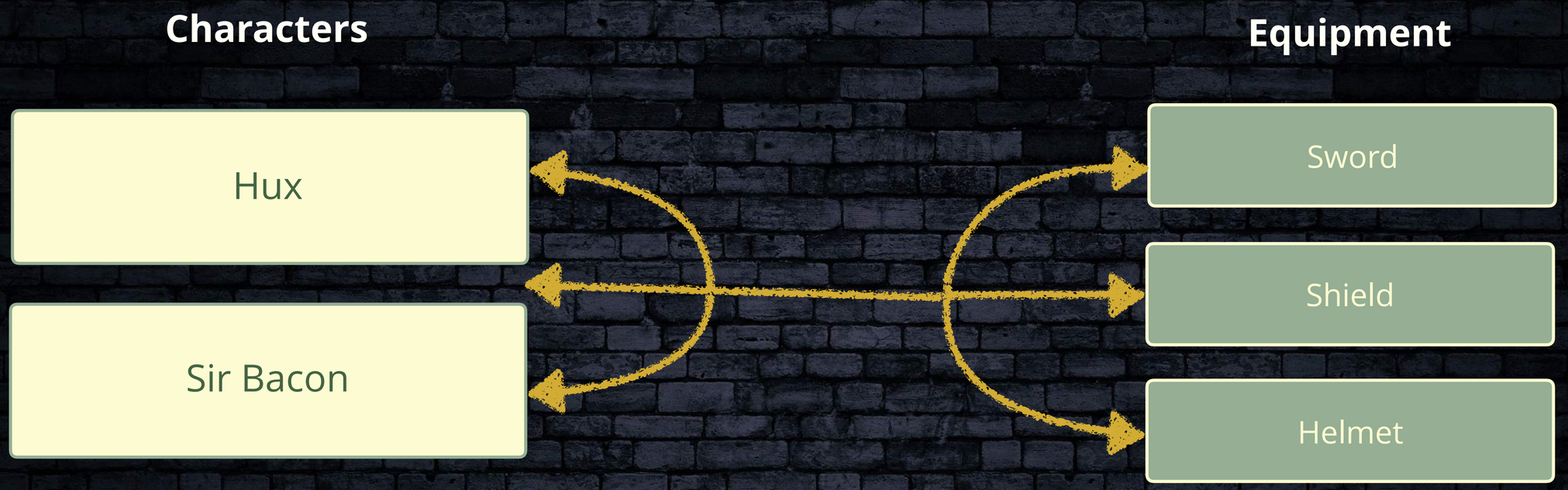
Equipment Isn't Unique

Multiple characters can also have the same type of equipment.



Many-to-many Relationship

This means characters and equipment have a many-to-many relationship.



How Databases Handle Many-to-many

Our database will need what's called a union table to accommodate this relationship.

Characters

Hux
Sir Bacon

Union Table

Hux : Sword
Hux : Shield
Sir Bacon : Shield

Equipment

Sword
Shield
Helmet

Object-relational mapping frameworks handle all this complicated database stuff so we don't have to, but you should be aware of what it's doing behind the scenes.

Creating Our Relationship in Code

Models/Character.cs

CS

```
using System.Collection.Generic;
using System.ComponentModel.DataAnnotations;

namespace ForgingAhead.Models
{
    public class Character
    {
        …
        public int Intelligence { get; set; }

        public List<Equipment> Equipment { get; set; }
    }
}
```

Making a List of Equipment in our Character class will create the relationship between Character and Equipment.

Don't forget you'll need a using directive for System.Collection.Generic to use List.


FORGING AHEAD
with
.NET
MVC

Level 2 – Section 2

Showing Off Our Data

Presenting Data in Views

We Need to Create Our Views

We've created our action methods to pass data to our views, so we just need to make the views.

- Character Create Page
- Character Index Page

Views

Character

Create.cshtml

Index.cshtml

Create our Create and Index views inside our Views/Character folder.

Creating Our Index.cshtml Page

./Views/Character/Index.cshtml

CSHTML

```
@model List<ForgingAhead.Models.Character>
<h1>
    Characters
</h1>
<ul>
    @foreach(var item in Model)
    {
      <li>
         @item.Name
      </li>
    }
</ul>
```

We'll add a link to our Create action after our heading.

Adding a Link to Our Create Action

./Views/Character/Index.cshtml

CSHTML

```
@model List<ForgingAhead.Models.Character>
<h1>
    Characters
</h1>
<p>
    <a asp-action="Create" asp-controller="Character"/>Create new character</a>
<p>
<ul>
    @foreach(var item in Model)
    {
      <li>
        @item.Name
      </li>
    }
</ul>
```

We can use asp-action and asp-controller to specify which controller and action to use.

The template will render this URL based on the two attributes we provided.

```
<a href="/Character/Create"/>Create new character</a>
```

Adding Parameters Through asp-route

asp-route allows us to specify parameters by adding the parameter name after a hyphen.

./Views/Character/Index.cshtml

CSHTML

The list of names is a good place for us to link to our Details action.

```
…
@foreach(var item in Model)
{
  <li>
    <a asp-action="Details" asp-controller="Character" asp-route-name="@item.Name"/>
        @item.Name
    </a>
  </li>
}
…
```

Adds a query string Name with item.Name's value

```
<a href="/Character/Details?Name=Hans">
```

If you have a character named Hans, our Tag Helpers would render this to the browser.

Creating Our Create.cshtml Page

./Views/Character/Create.cshtml

CSHTML

```
@model ForgingAhead.Models.Character
<h2>Create</h2>
<hr/>
<form asp-action="Create" asp-controller="Character">
    <div class="form-horizontal">
        <h4>Character</h4>
        <hr/>
        <div class="form-group">
            <div class="row">
                <label class="control-label col-md-2" asp-for="Name"></label>
                <div class="col-md-10">
                    <input class="form-control" asp-for="Name"/>
                </div>
            </div>
        </div>
…
```

Our Create page is using CSS classes to keep things looking nice — otherwise, most of this is the same HTML and Razor we've seen before.

asp-for **Tag Helper**

./Views/Character/Create.cshtml

CSHTML

```
…
<h4>Character</h4>
<hr/>
<div class="form-group">
  <div class="row">
    <label class="control-label col-md-2" asp-for="Name"></label>
    <div class="col-md-10">
      <input class="form-control" asp-for="Name"/>
    </div>
  </div>
</div>
…
```

asp-for ***will populate the value of our*** *label* ***based on the field's*** *Name.*

it will also set our input tag's *type, id,* ***and*** *name* ***attributes.***

The tag helpers would render the resulting HTML.

```
<label … for="Name">Name</label>
<div class="col-md-10">
  <input … type="text" id="Name" name="Name"/>
</div>
```

What If We Want a Different Label?

./Views/Character/Create.cshtml

CSHTML

IsActive ***doesn't look very good as one word...***

We can fix this with *DataAnnotations.*

```
…
<div class="form-group">
  <div class="row">
    <label class="control-label col-md-2" for="IsActive">IsActive</label>
    <div class="col-md-10">
      <input class="form-control" type="checkbox" id="IsActive" name="IsActive" />
    </div>
  </div>
</div>
…
```

Setting Display Through DataAnnotations

Models/Character.cs

CS

Add a using directive for DataAnnotations

```
using System.ComponentModel.DataAnnotations;

public class Character
{
    public string Name { get; set; }
    [Display(Name = "Is Active")]
    public bool IsActive { get; set; }
    public int Level { get; set; }
    public int Strength { get; set; }
    public int Dexterity { get; set; }
…
```

The attribute Display sets the display name of our property, which tells asp-for what you want the label to show.

Rendered Output With IsActive Changes

Setting the IsActive property's Display attribute made it so the label is now Is Active instead of IsActive.

```
…
<div class="form-group">
  <div class="row">
    <label class="control-label col-md-2" for="IsActive">Is Active</label>
    <div class="col-md-10">
      <input class="form-control" type="checkbox" id="IsActive" name="IsActive"/>
    </div>
  </div>
</div>
…
```

The for, id, and name attributes are still IsActive, which is what we want, so everything wires up properly.

You Can Now Add and View Data

We've added the views necessary for our Create **and** Index **actions to work.**

- Character Create Page
- Character Index Page


FORGING AHEAD
with
.NET
MVC