

TRY

NETMUG

Level 1

# Model View Controller

Level 1 – Section 1

# Model View Controller

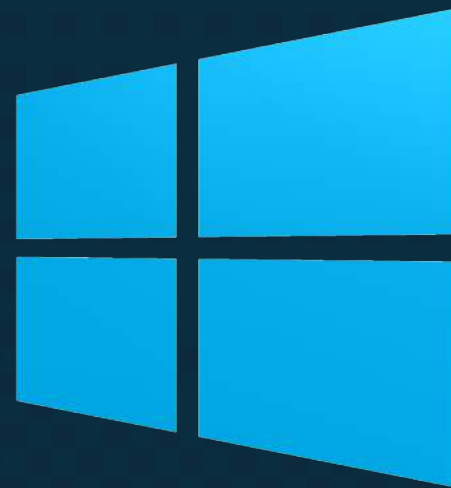
---

Introduction to MVC

# What Is ASP.NET MVC?

ASP.NET MVC is a .NET framework for the rapid development of web applications.

- Helps keep code clean and is easy to maintain
- Handles everything from the UI to the server environment and everything in between
- Runs on Linux, Windows, and OS X



*Other names for ASP.NET MVC*



*.NET MVC*

*ASP.NET Core*

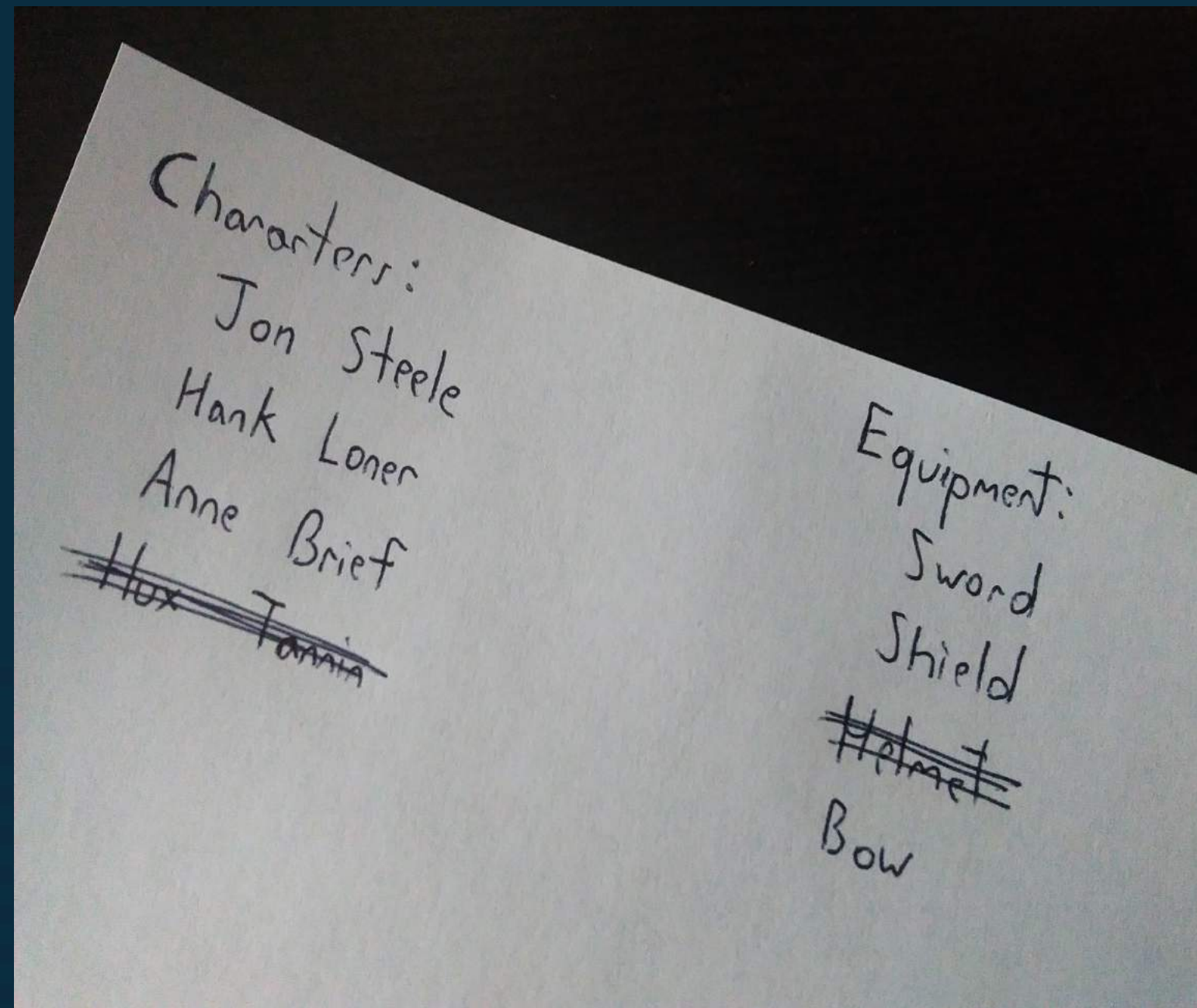




# The Problem...

---

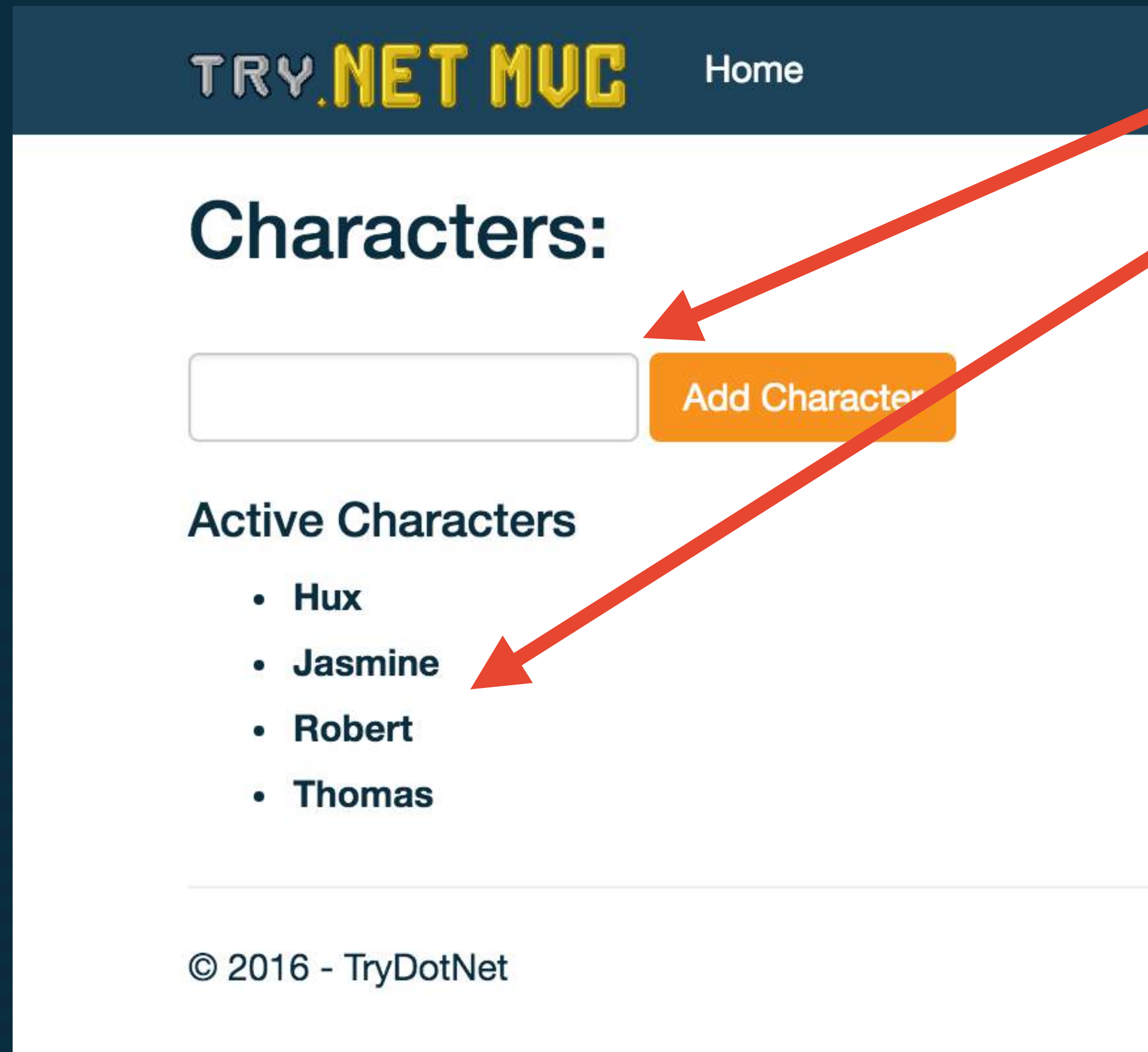
We're playing a tabletop game and keeping game information written on notebook paper, which is getting troublesome.



- Lots of paper to organize
- Hard to search
- Updating/erasing is messy

# The Solution: a .NET MVC Application

We're going to create a .NET MVC application to keep track of the different parts of our game so we can ditch the paper.



The screenshot shows a web application interface. At the top, there is a dark blue header with the text 'TRY.NET MVC' in yellow and 'Home' in white. Below the header, the main content area is white. It starts with the heading 'Characters:'. Underneath this heading is a text input field and an orange button labeled 'Add Character'. Below the input field and button is the heading 'Active Characters'. Under this heading is a bulleted list of names: 'Hux', 'Jasmine', 'Robert', and 'Thomas'. At the bottom of the page, there is a footer with the text '© 2016 - TryDotNet'.

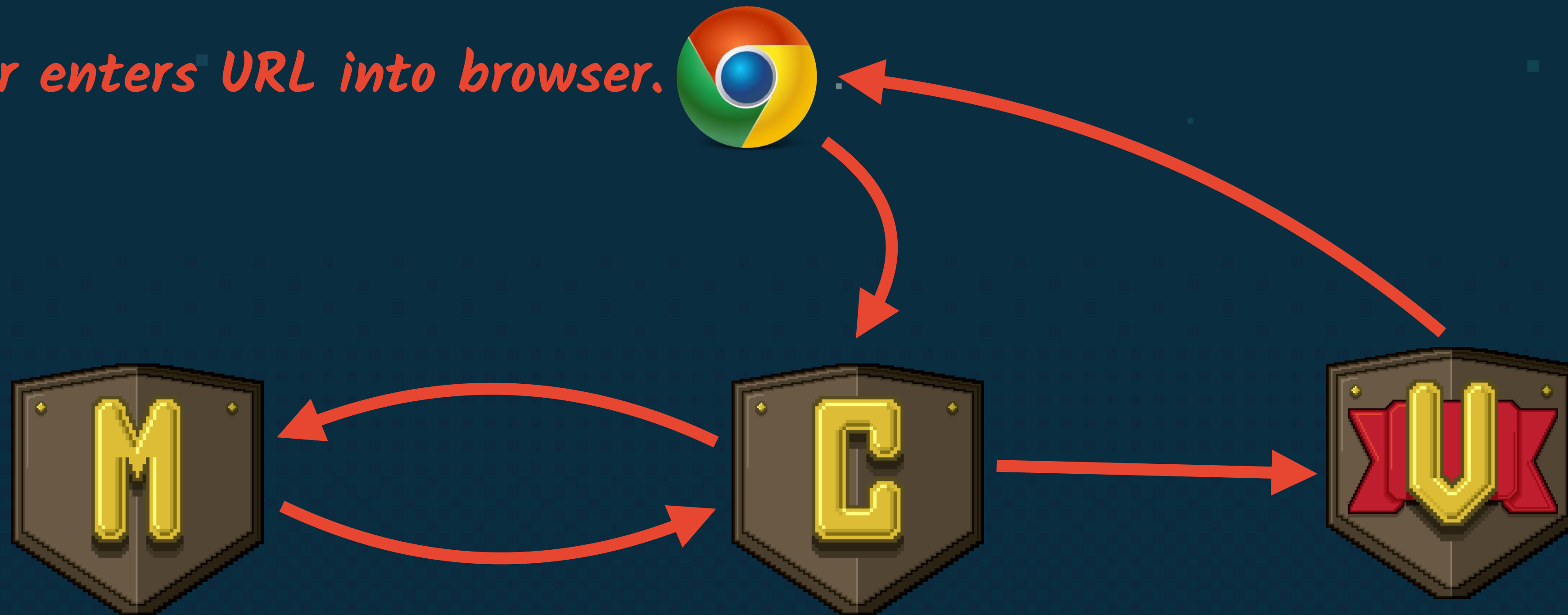
Allow users to add their own characters

See all added characters

# How Data Flows Through an MVC Application

MVC is a structural pattern for organizing code in a logical way.

*User enters URL into browser.*



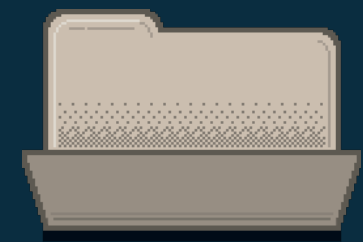
*Controller gets data from model based on the URL that was entered.*

*Controller gives data to the view so it can display it in the browser.*



# Structure of an ASP.NET MVC Project

Our ASP.NET MVC project primarily deals in the Models, Views, and Controllers folders.



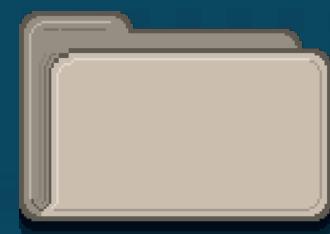
CharacterSheetApp

← *The root folder matches the name of our project.*

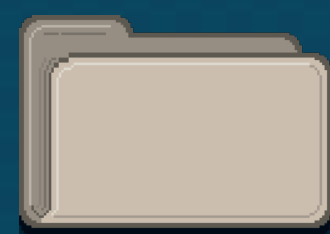
...



Models



Views



Controllers



*We will be doing most of our work in these folders.*



Properties

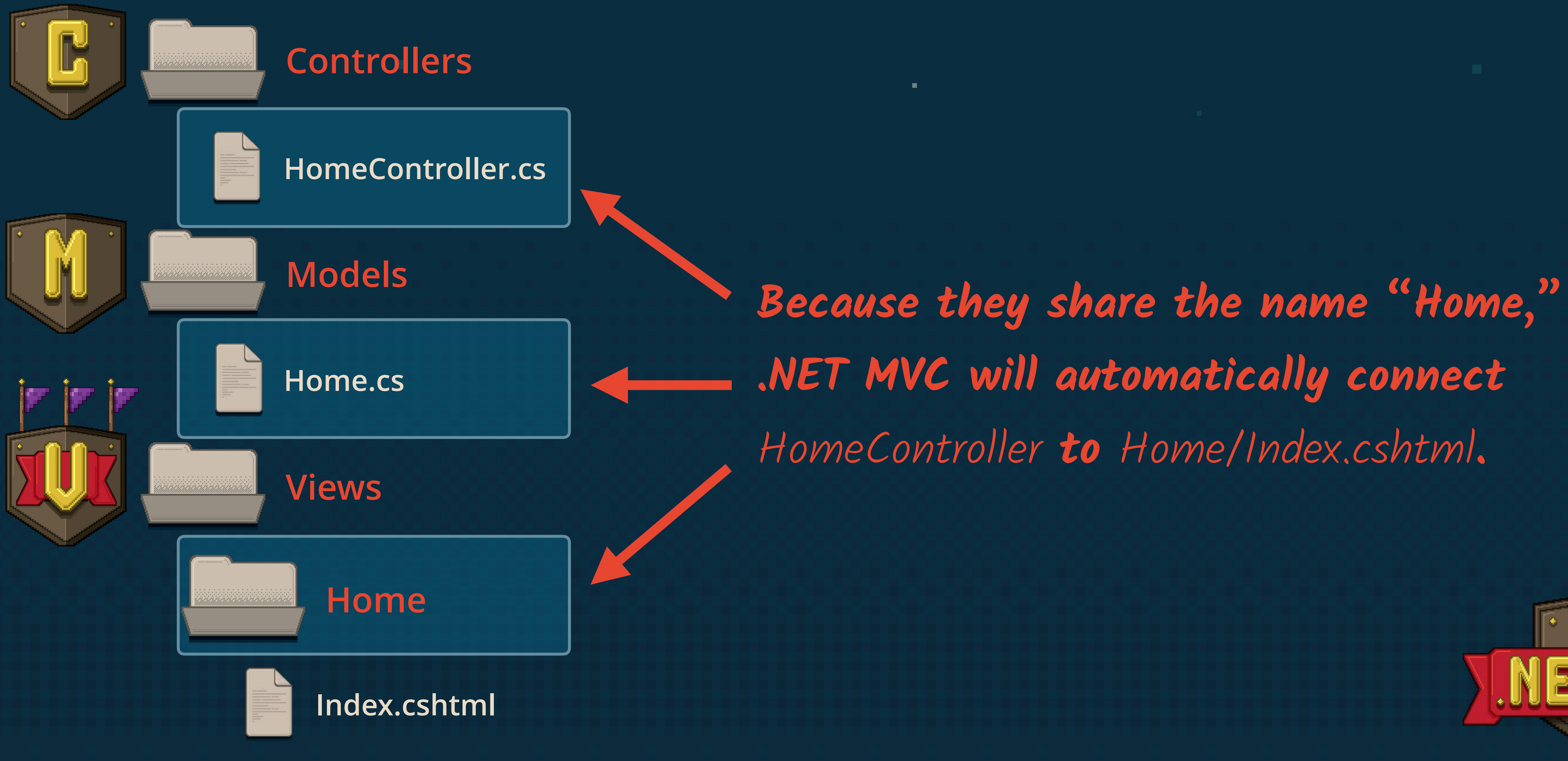
...





# File and Folder Names Are Important

The files typically follow a structure that makes it easy to see what's related.



# Creating a View in Views/Home

We need to create our new `Index.cshtml` view under Views/Home.



*How you create a new file depends on the tools you're using.*

Visual Studio  
Visual Studio Code  
Xamarin Studio  
MonoDevelop  
SharpDevelop



*cshtml is a template file that lets us use HTML- and C#-like code to generate our pages.*



# Raw HTML in a View

./Views/Home/Index.cshtml

CSHTML

```
<h2>Characters:</h2>
<div>
  <ul>
    <li>Hux</li>
    <li>Jasmine</li>
    <li>Robert</li>
    <li>Thomas</li>
  </ul>
</div>
```

*This HTML displays our list of characters, but we have no way to update the list without directly editing this file.*



TRY.NET MVC

Home

## Characters:

- Hux
- Jasmine
- Robert
- Thomas

© 2016 - TryDotNet

# Our Names Need to Be Dynamic, But How?

./Views/Home/Index.cshtml

CSHTML

```
<h2>Characters:</h2>
```

```
<div>
```

```
<ul>
```

```
<li>Jasmine</li>
```

```
</ul>
```

```
</div>
```

*We want to be able to set this dynamically.*



To update the name dynamically, we need to:

- Update the view to accept data
- Update the controller to send data



# Setting Our View's Model

In Razor, we have the keyword `@model` that tells our view what kind of data is coming in.

`./Views/Home/Index.cshtml`

CSHTML

```
@model String
<h2>Characters:</h2>

<div>
  <ul>
    <li>Jasmine</li>
  </ul>
</div>
```

*Our view is expecting a single string of data.*



*When you're mixing HTML and C#, you're using a built-in engine called Razor.*

# Accessing Our Model Data in Our View

@Model gives us access to data passed into our view from a controller.

./Views/Home/Index.cshtml

CSHTML

```
@model String
<h2>Characters:</h2>

<div>
  <ul>
    <li>@Model</li>
  </ul>
</div>
```

*Use the uppercase @Model to access whatever data gets passed into our view.*



# Pay Attention to the Capitalization of Model

./Views/Home/Index.cshtml

CSHTML

*Lowercase “defines the type of data coming into the view”*

```
@model String  
<h2>Characters:</h2>
```

*Uppercase “accesses the model data passed into the view”*

```
<div>  
  <ul>  
    <li>@Model</li>  
  </ul>  
</div>
```



# Getting Ready to Send Data to the View

We need our HomeController to send a string to the view.



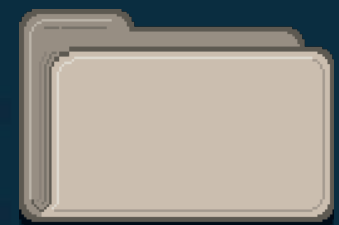
Controllers



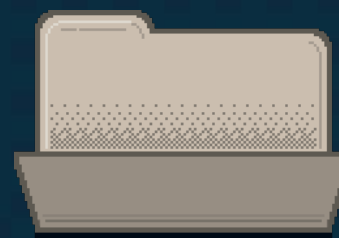
HomeController.cs



*The controller sends data to the view.*



Models



Views



Home



Index.cshtml





# A Look Inside the Controller

When you create a new ASP.NET MVC project, it automatically creates this controller, which already includes at least this code.

./Controllers/HomeController.cs

CS

*Namespace*

```
namespace CharacterSheetApp.Controllers
{
```

*Class*

```
public class HomeController : Controller
{
```

*Method*

```
public IActionResult Index()
{
    return View();
}
}
```



*Methods that return `IActionResult` are called “action methods.”  
Action methods provide responses usable by browsers.*

# Passing Our Name Back to the View

./Controllers/HomeController.cs

CS

```
namespace CharacterSheetApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            var name = "Hux";
            return View("Index", name);
        }
    }
}
```

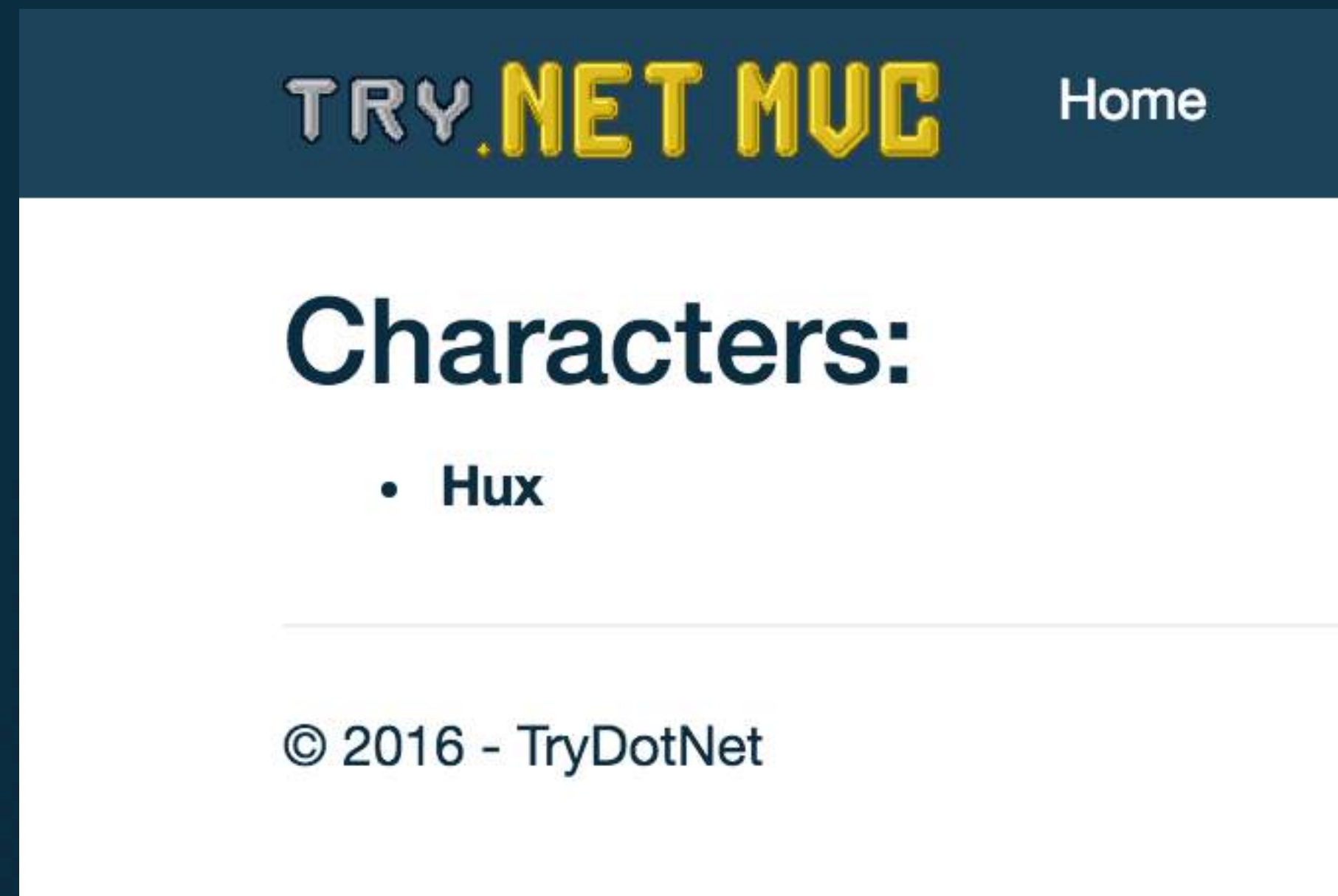
*We create the string we want to return to our view...*

*...and then pass the name of the Index view and the string as a parameter.*

# Our Index Action Working

---

Our view is now rendering the name dynamically using the value provided by the controller.



TRY

NETMUG



Level 1 – Section 2

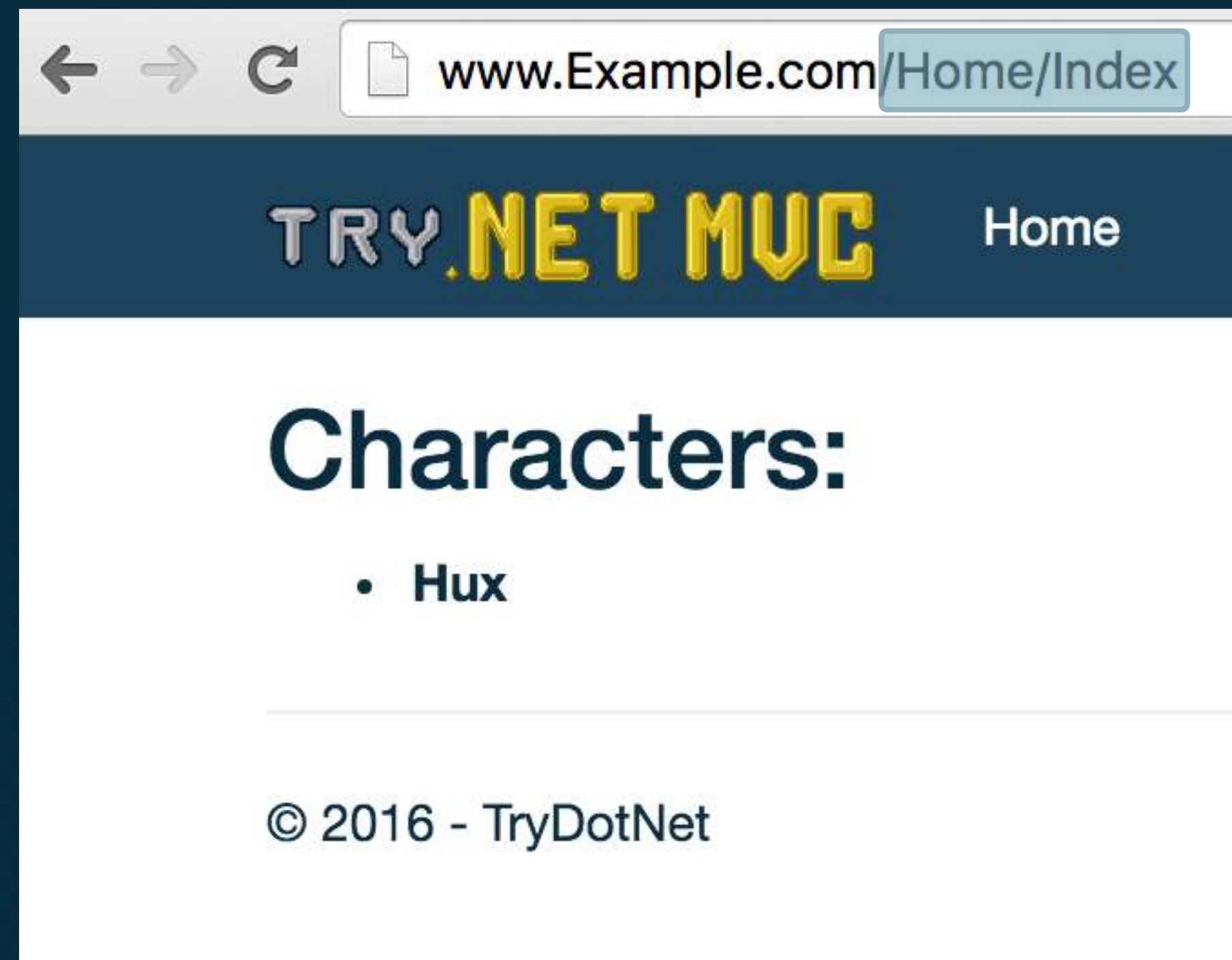
# Model View Controller

---

Basic Routes

# Accessing Our Website

The application decides which view to display using a system called routes that tells our web application which controller and controller action to access based on the URL.



*New projects' domain will be `http://localhost:PortNumber` and the port number will vary.*

# How Do Routes Map Up With Controllers?

By default, the first section of the route maps up to the controller of the same name, and the second section maps up to an action within that controller of the same name.

http://www.example.com/Home/Index

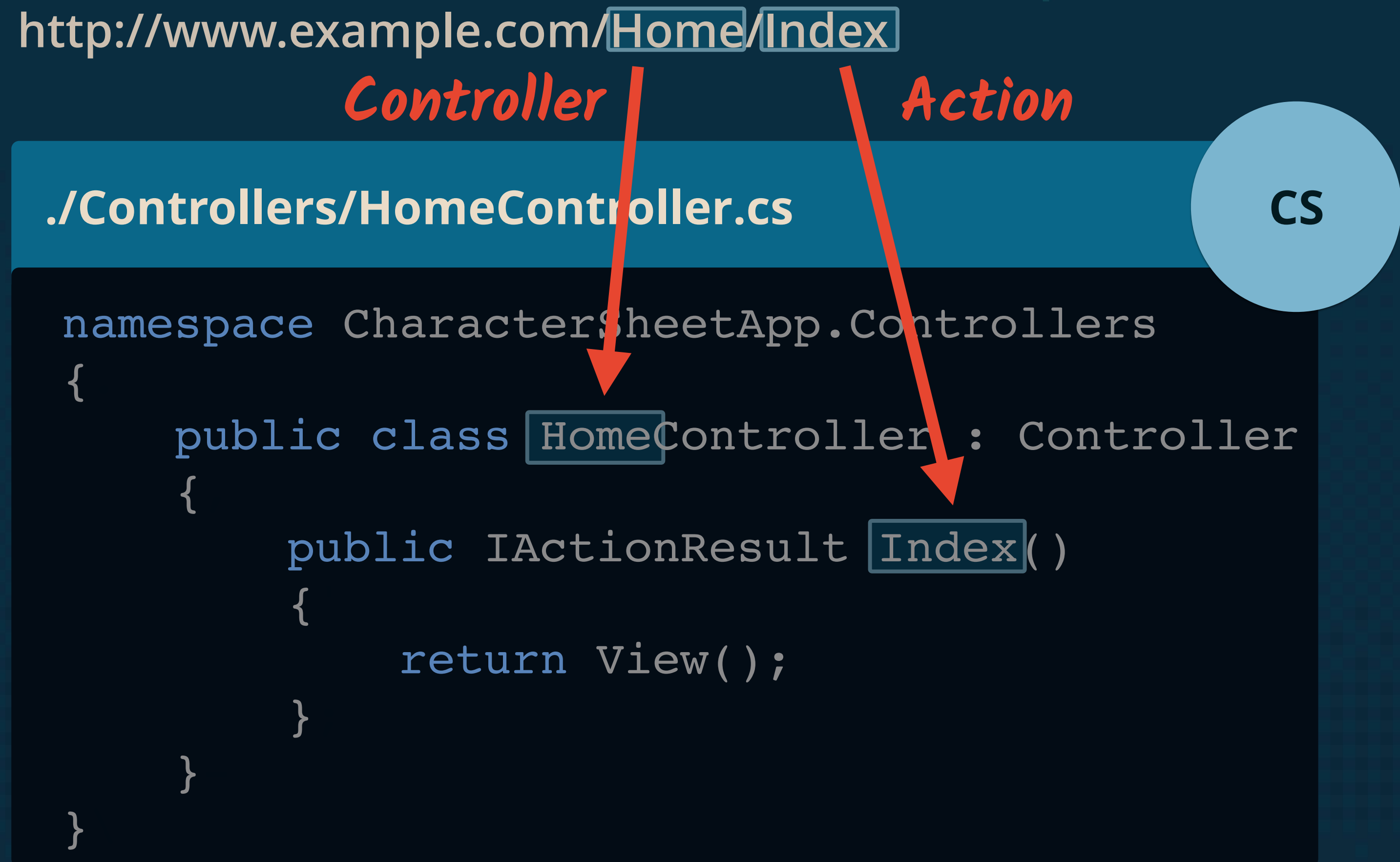
*Controller*

*Action*

**./Controllers/HomeController.cs**

CS

```
namespace CharacterSheetApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```



# The Controller Also Maps to the View

./Controllers/HomeController.cs

CS

```
namespace CharacterSheetApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```



Views



Home



Index.cshtml



*The returned view returned is the one located in Home/Index.cshtml*

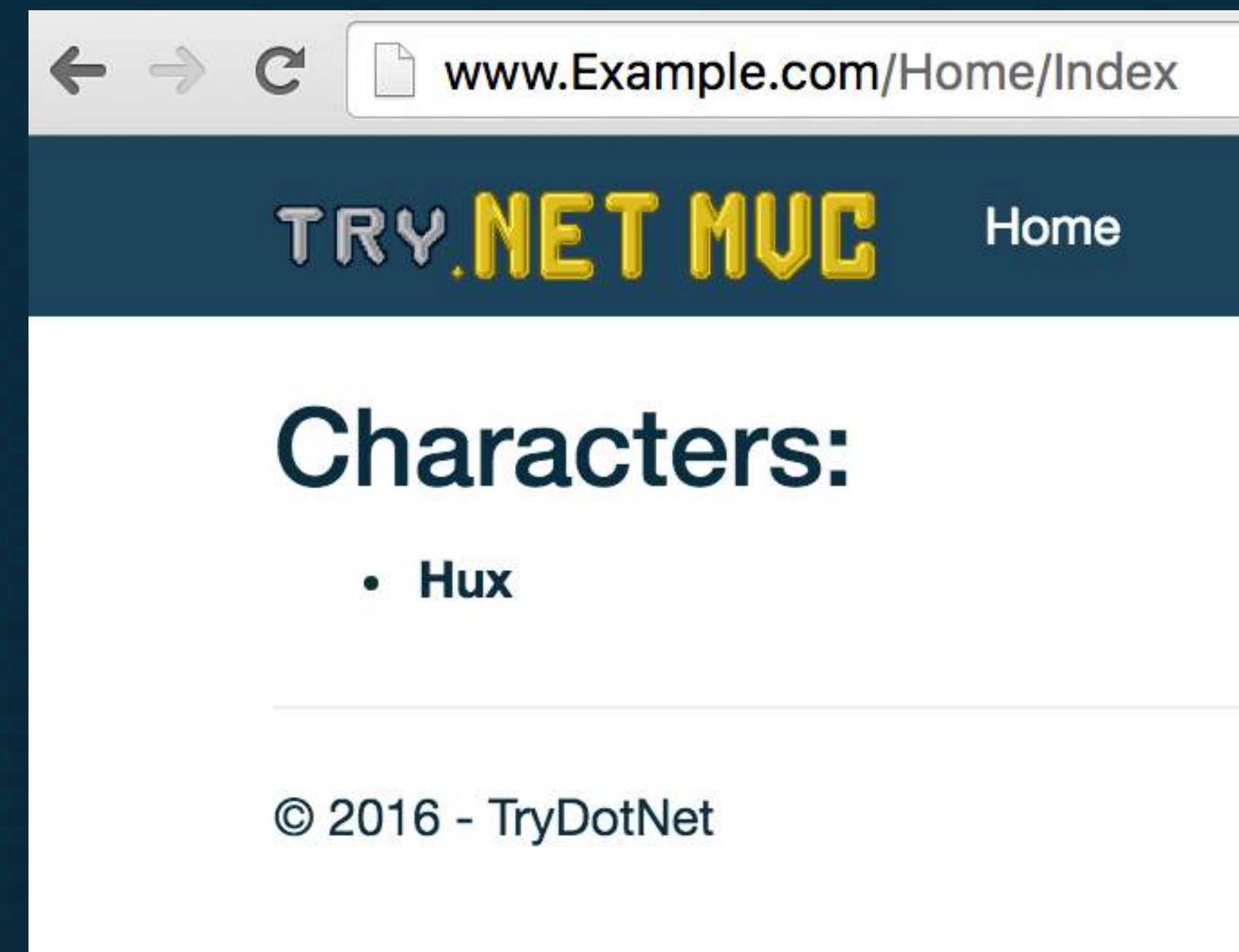
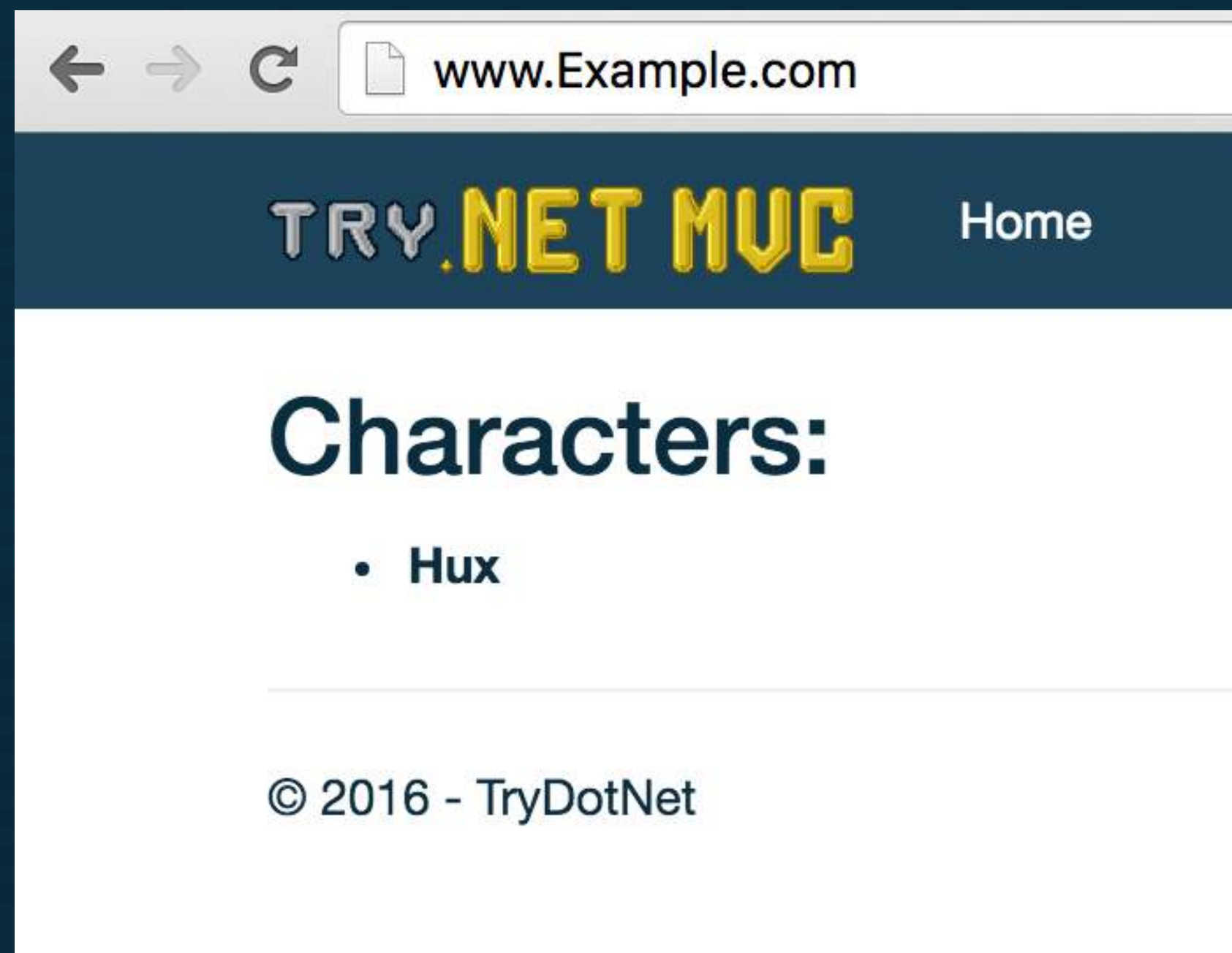


# What Happens if We Omit the Route?

When we don't have the route part of the URL, ASP.NET MVC will use a default route.

`http://www.example.com`  `http://www.example.com/Home/Index`

*The default route in a new project points to Home/Index.*



TRY

NETMUG

Level 1 – Section 3

# Model View Controller

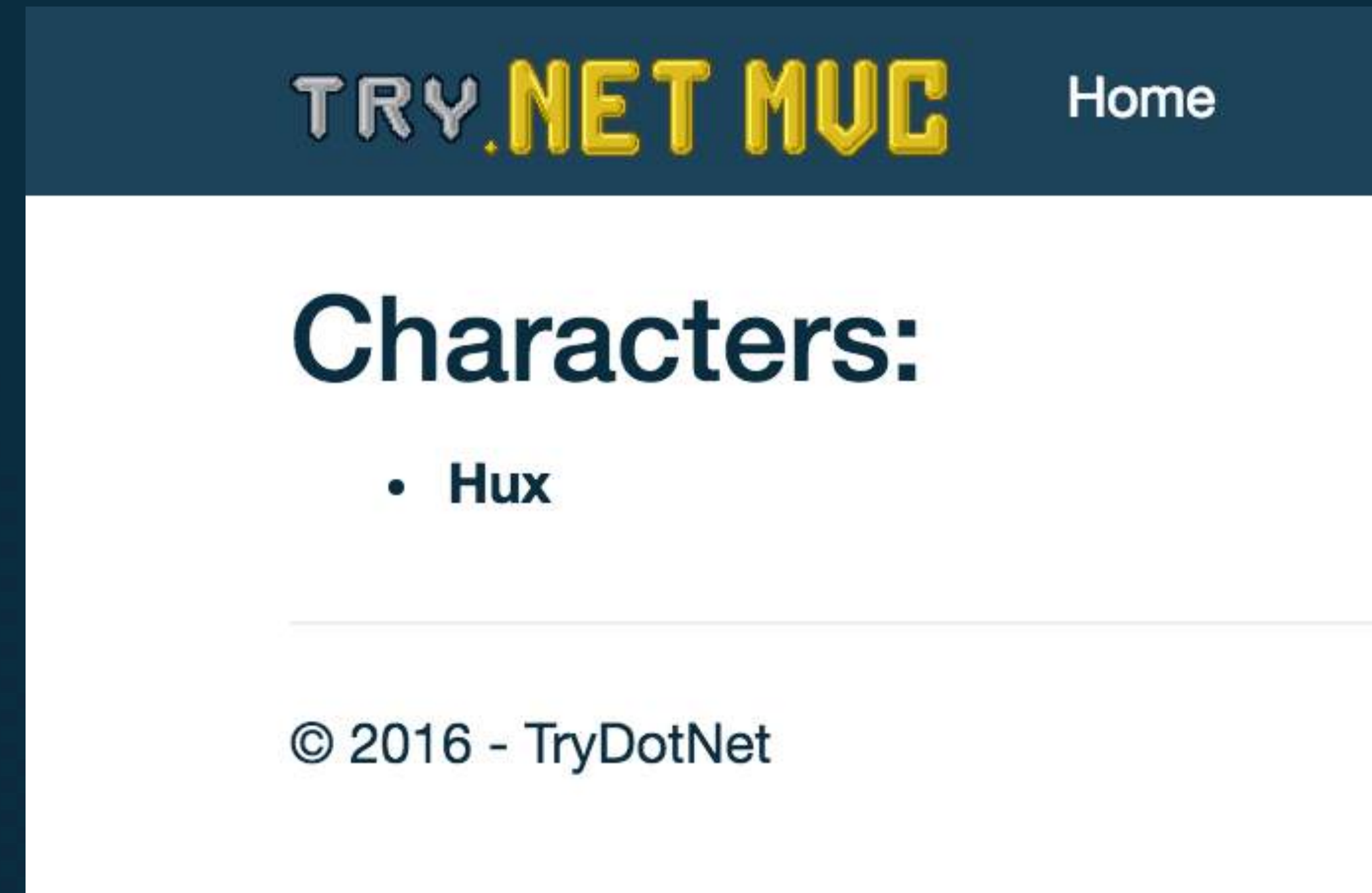
---

Models



# We Will Want Stats

We're going to want more than just a character's name — we'll want at least the most basic stats. A string won't be able to do that, so let's go ahead and create a character class to be ready.





# Creating a Model Class



Controllers



HomeController.cs



Models



Character.cs



Views



Home



Index.cshtml

*We create a new class, Character.cs, in the Models folder.*



# Add Fields to Our Model

./Models/Character.cs

CS

```
namespace CharacterSheetApp.Models
{
    public class Character
    {
        public string Name;
    }
}
```

*We'll add our Name field.*



# Setting Our Model to Our Character Object

CS

./Controllers/HomeController.cs

```
public IActionResult Index()
{
    var model = new Character();
    model.Name = "Hux";
    return View(model);
}
```

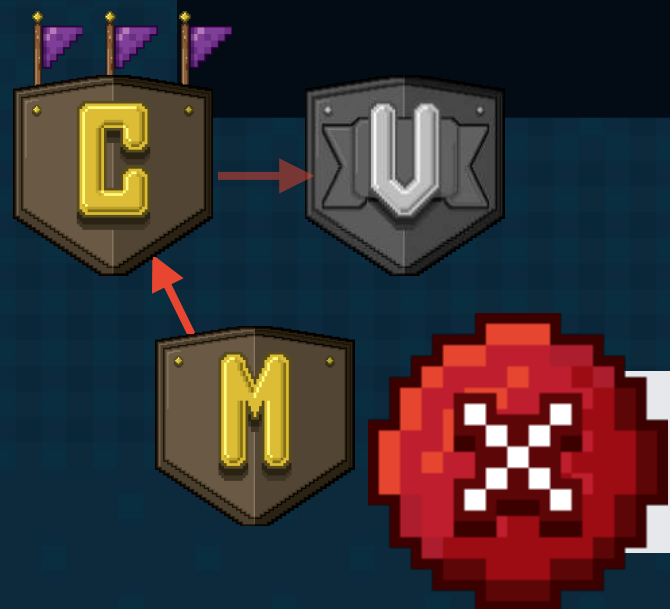
*We change our model to our Character model.*

## HomeController.cs before

```
public IActionResult Index()
{
    var name = new string();
    name = "Hux";
    return View("Index", name);
}
```

*Set the field of an object using dot notation*

*But doing this has caused us to get an error...*



The type or namespace name 'Character' could not be found (are you missing a using directive or an assembly reference?)

# Namespaces in .NET

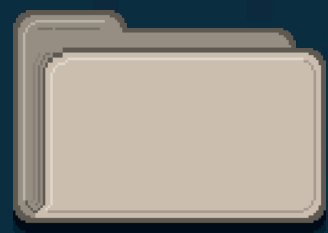
.NET really pushes division of concerns. One example is that namespaces often follow directories and keep us from accessing other parts of the code that we don't intend to.



CharacterSheetApp

...

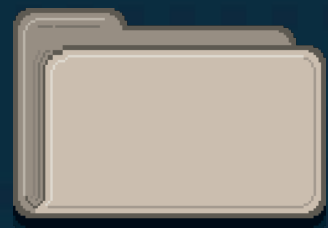
*Namespace*



Models



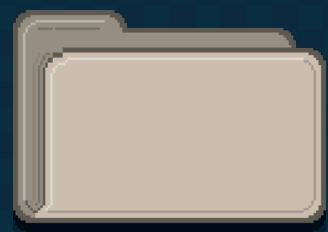
*CharacterSheetApp.Models*



Views



*CharacterSheetApp.Views*



Controllers



*CharacterSheetApp.Controllers*



Properties

...





# Using the Full Character Namespace

./Controllers/HomeController.cs

CS

*Since our Character model lives outside our controller,  
we need the full namespace and class to access it.*

```
public IActionResult Index()
{
    var model = new CharacterSheetApp.Models.Character();
    model.Name = "Hux";
    return View(model);
}
```

*Namespace*      *Class*

./Models/Character.cs

CS

```
namespace CharacterSheetApp.Models
{
    public class Character
    {
    }
```

*Namespace*      *Class*

*We can find the namespace by  
looking at the Model class.*



# Change Our View to Use Our Model

./Views/Home/Index.cshtml

CSHTML

```
@model CharacterSheetApp.Models.Character
```

```
<h2>Characters:</h2>
```

*Change string to be our Character model*

```
<div>
```

```
<ul>
```

```
<li>@Model</li>
```

```
</ul>
```

```
</div>
```



# Using Our Model's Name Field

./Views/Home/Index.cshtml

CSHTML

```
@model CharacterSheetApp.Models.Character
<h2>Characters:</h2>

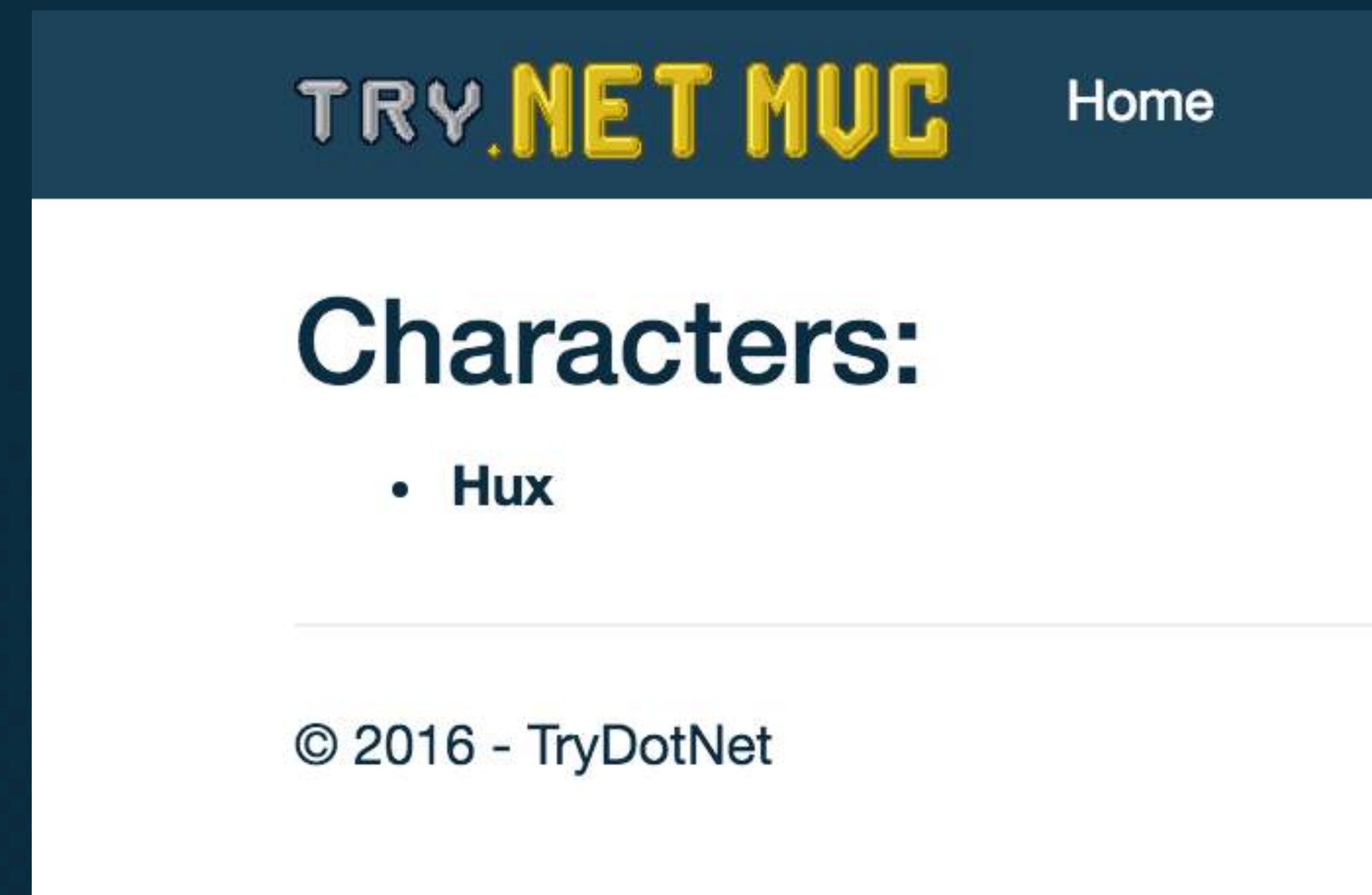
<div>
  <ul>
    <li>@Model.Name</li>
  </ul>
</div>
```

*Change to show the Name field instead of the model itself*



# We're Dynamic Now!

Our view looks close to what it was before, but now we can update it programmatically instead of having to update the HTML — we just need to give it the ability to let users input data.





TRY

NETMUG