
TRY
.NET MUC

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

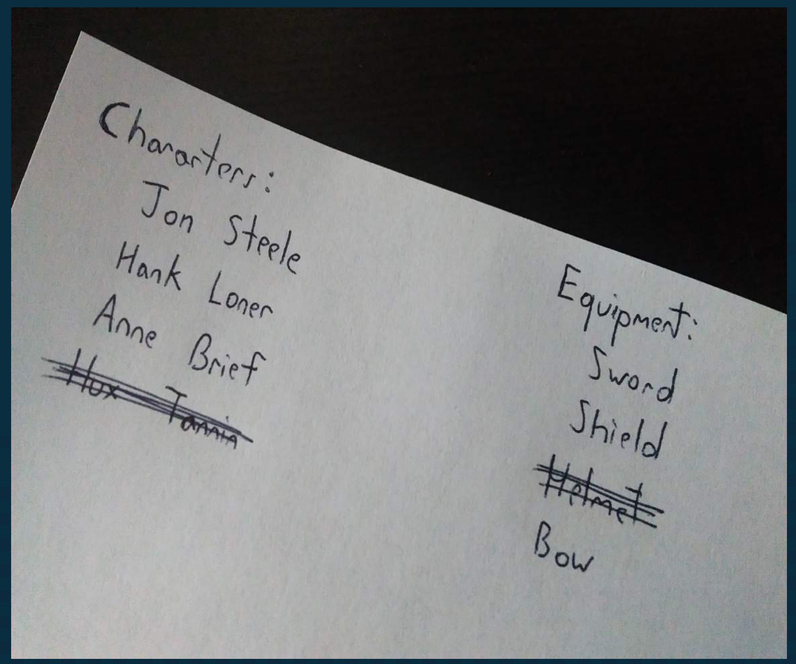
*Other names for ASP.NET MVC*

*.NET MVC*

*ASP.NET Core*

# The Problem...

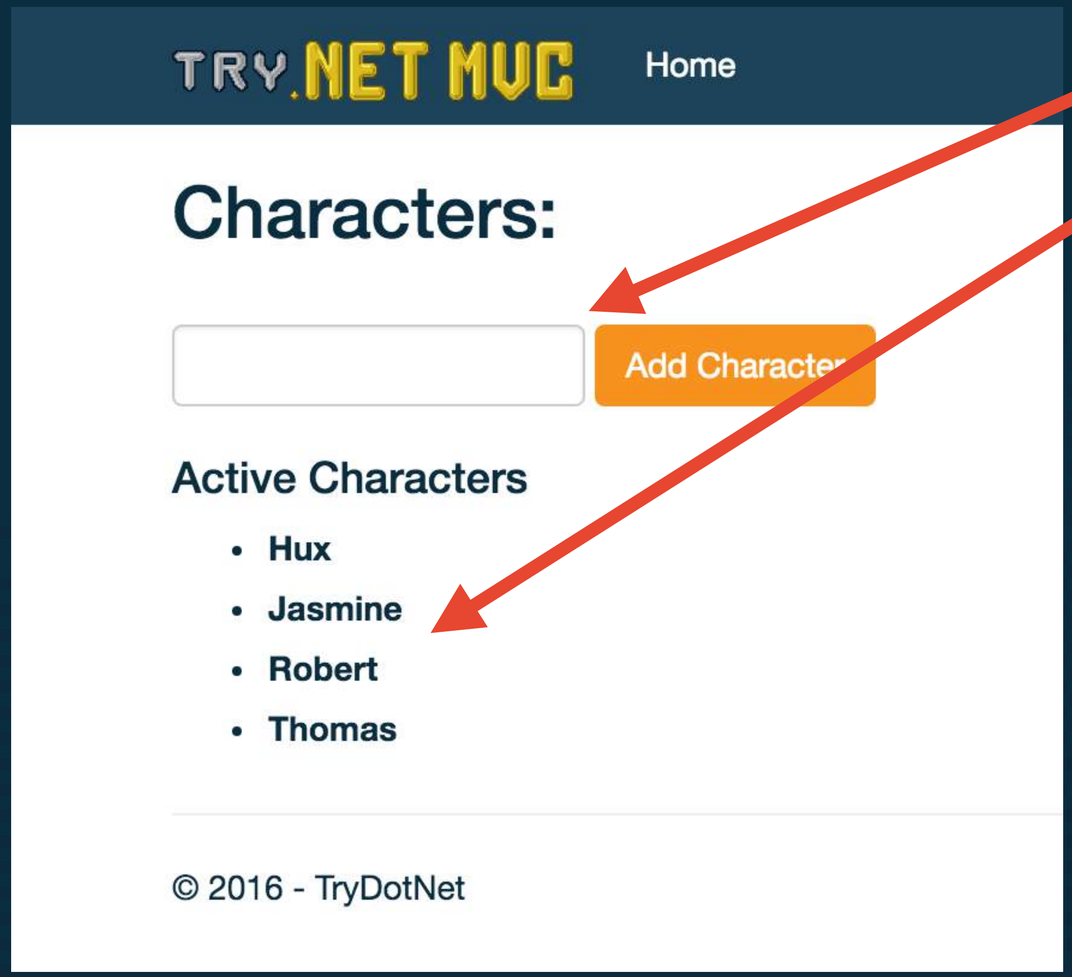
We're playing a tabletop game and keeping game information written on notebook paper, which is getting troublesome.




- 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.

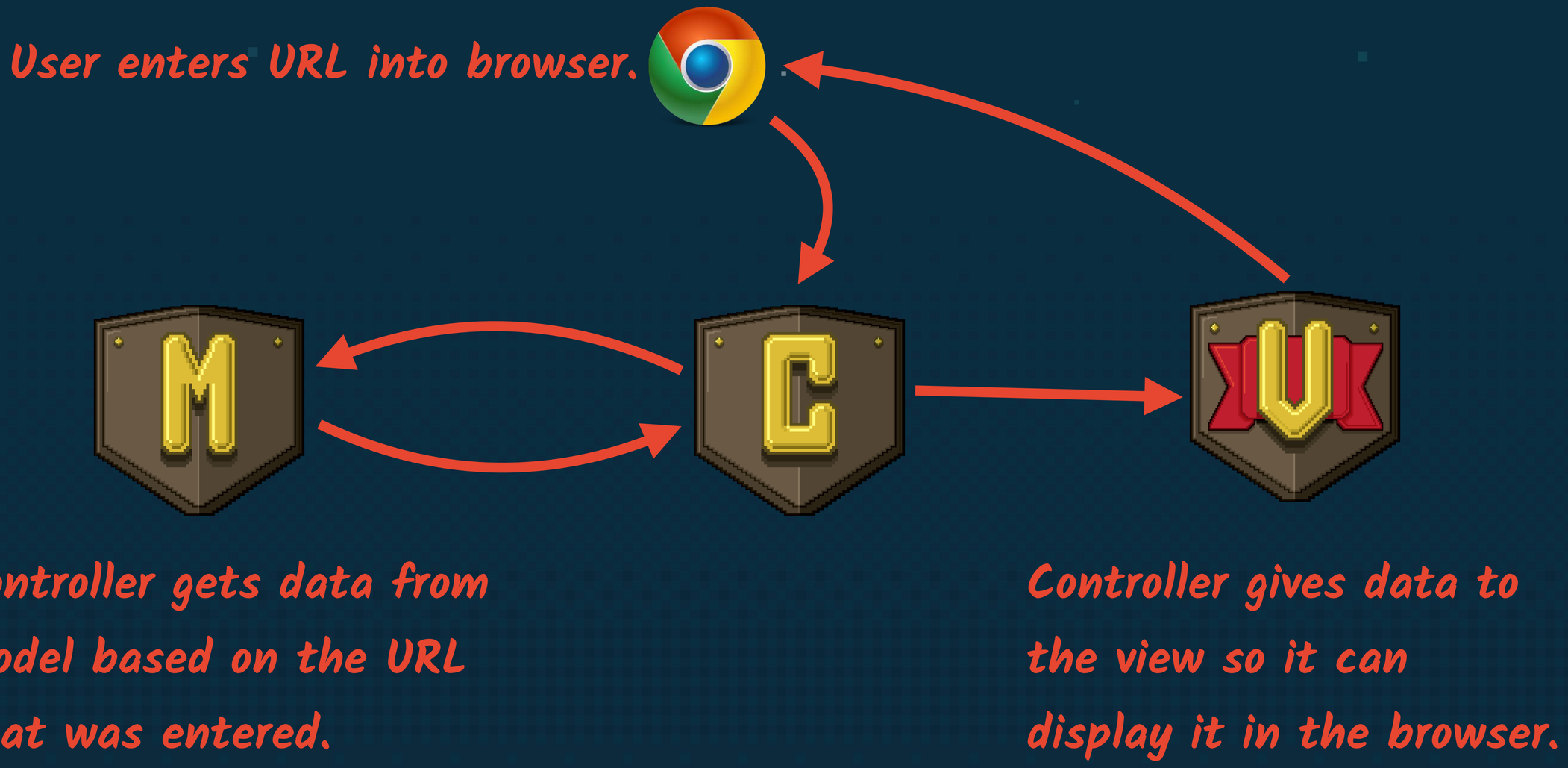



Allow users to add their own characters

See all added characters

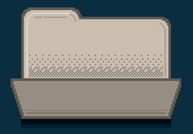
# How Data Flows Through an MVC Application

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



# 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.


***Because they share the name "Home," .NET MVC will automatically connect*** *HomeController* ***to*** *Home/Index.cshtml.*

# 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.*

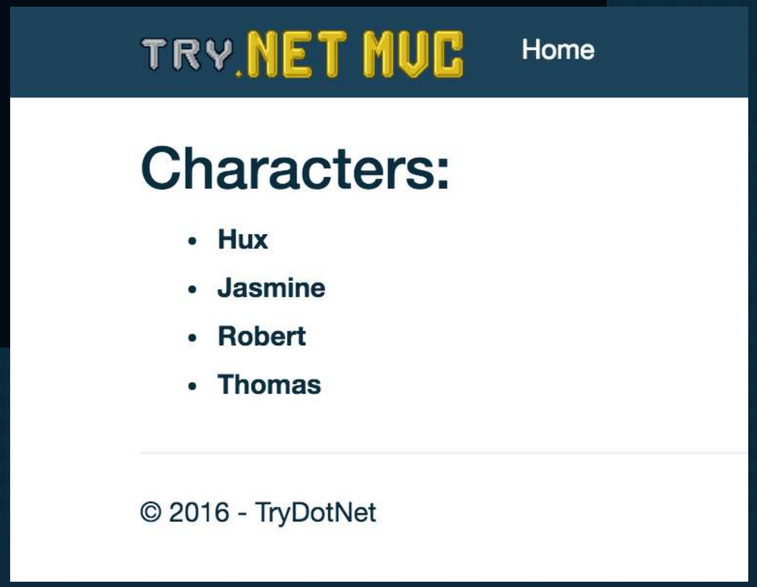
# Raw HTML in a View

./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.*



# 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

Our view is expecting a single string of data.

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

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

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

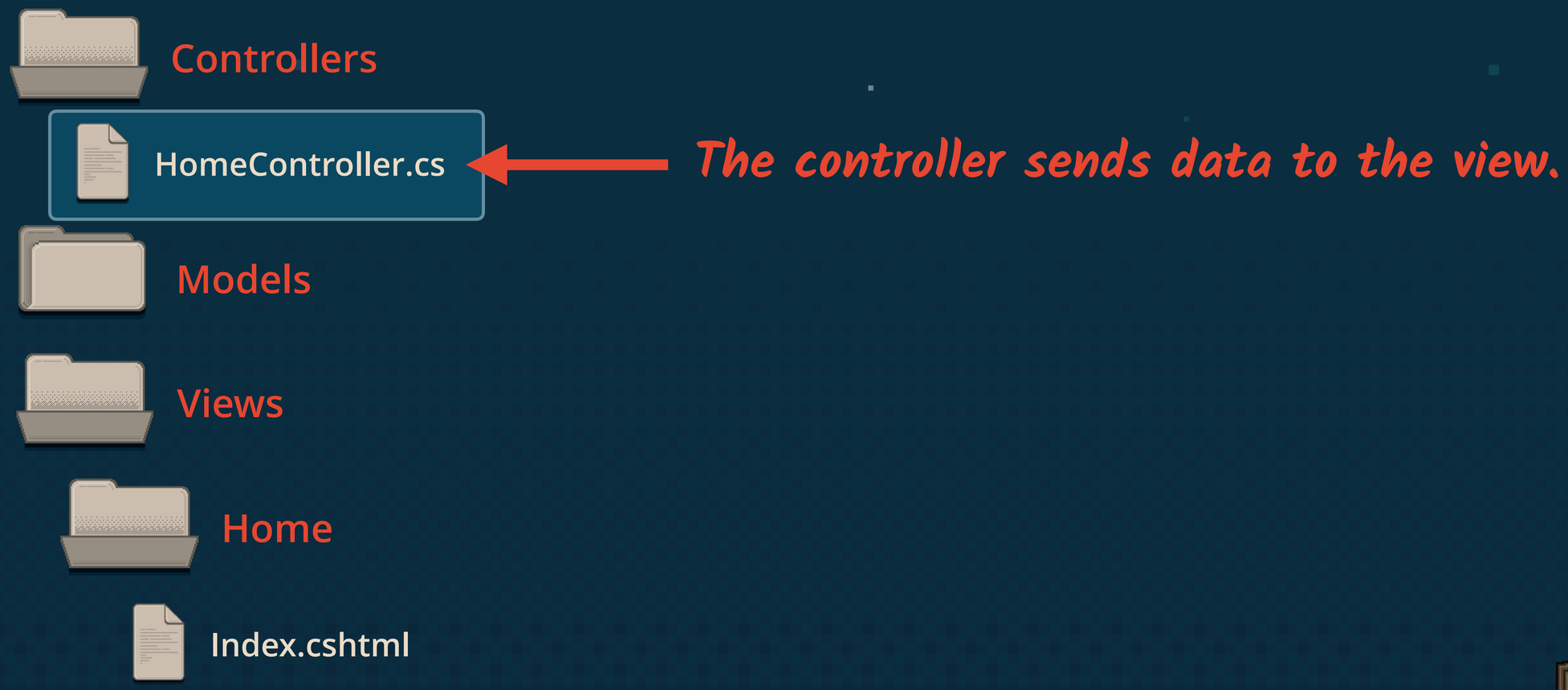
Lowercase "defines the type of data coming into the view"

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

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

Uppercase "accesses the model data passed into the view"

# Getting Ready to Send Data to the View

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




# 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

Class

Method

```
namespace CharacterSheetApp.Controllers
{
    public class HomeController : Controller
    {
        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
.NET MUC

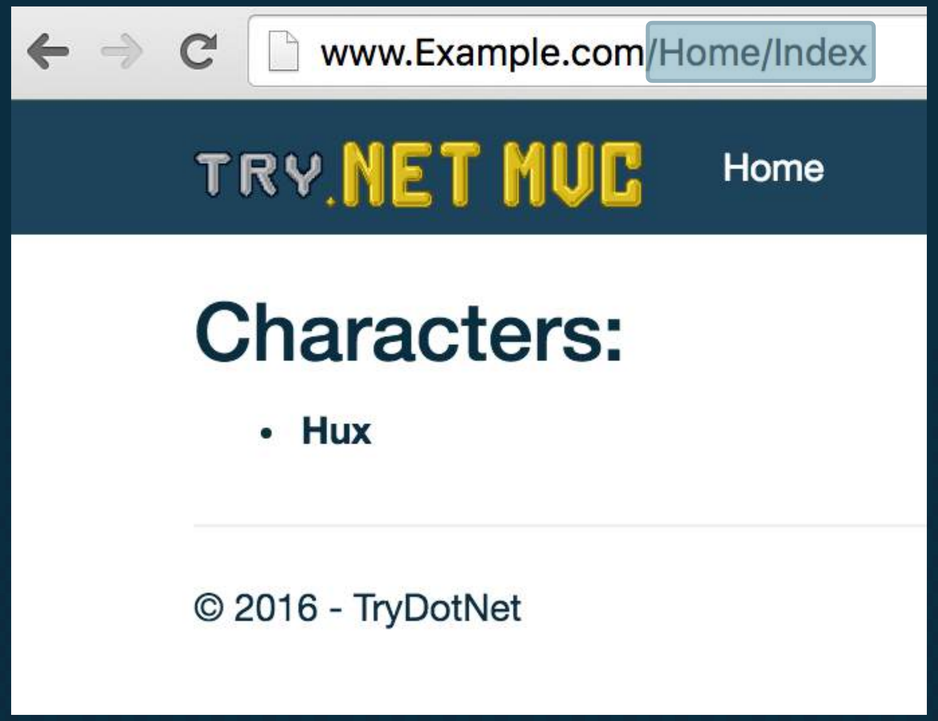
Level 1 – Section 2

# 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

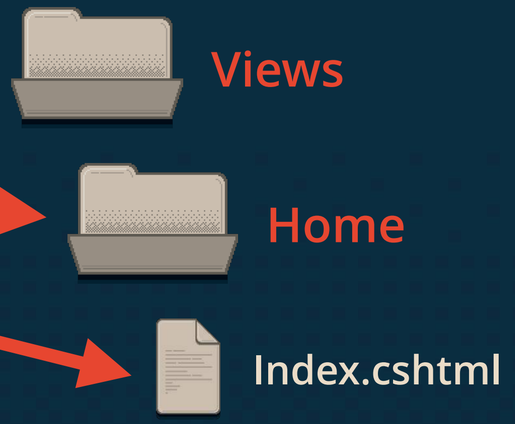
```
namespace CharacterSheetApp.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

# The Controller Also Maps to the View

./Controllers/HomeController.cs

CS

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



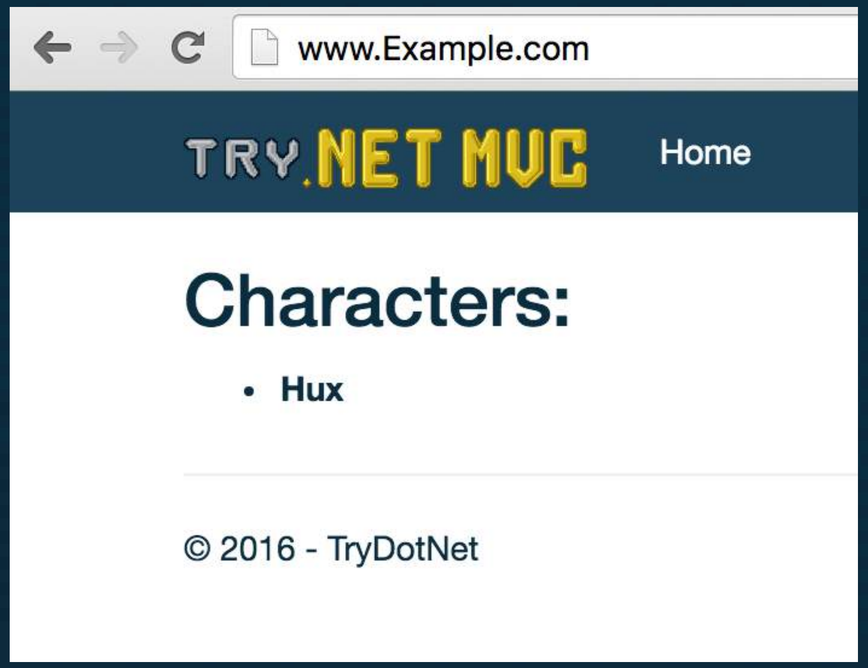

***The returned view returned is the one located in** Home/Index.cshtml*

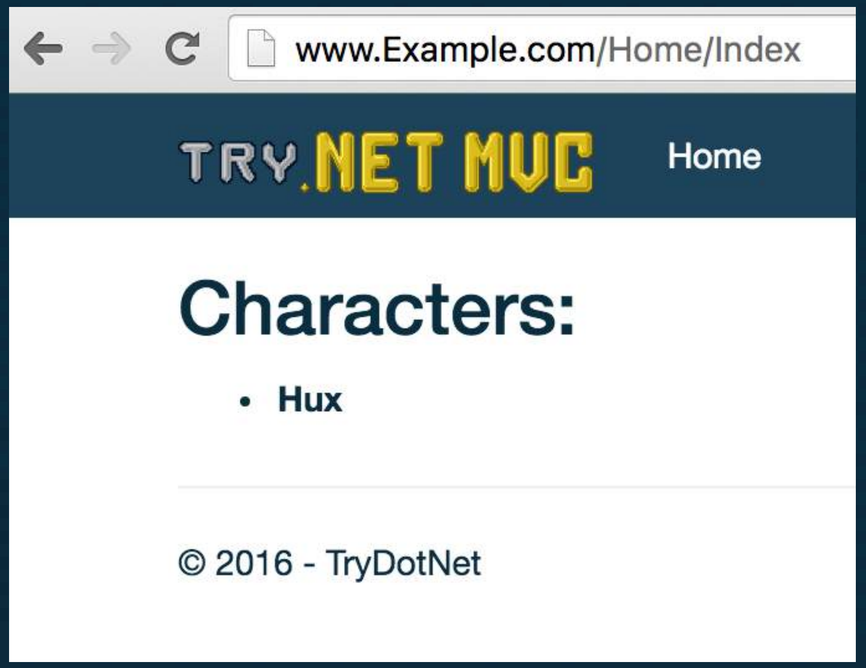
# What Happens if We Omit the Route?

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
.NET MUC

Level 1 – Section 3

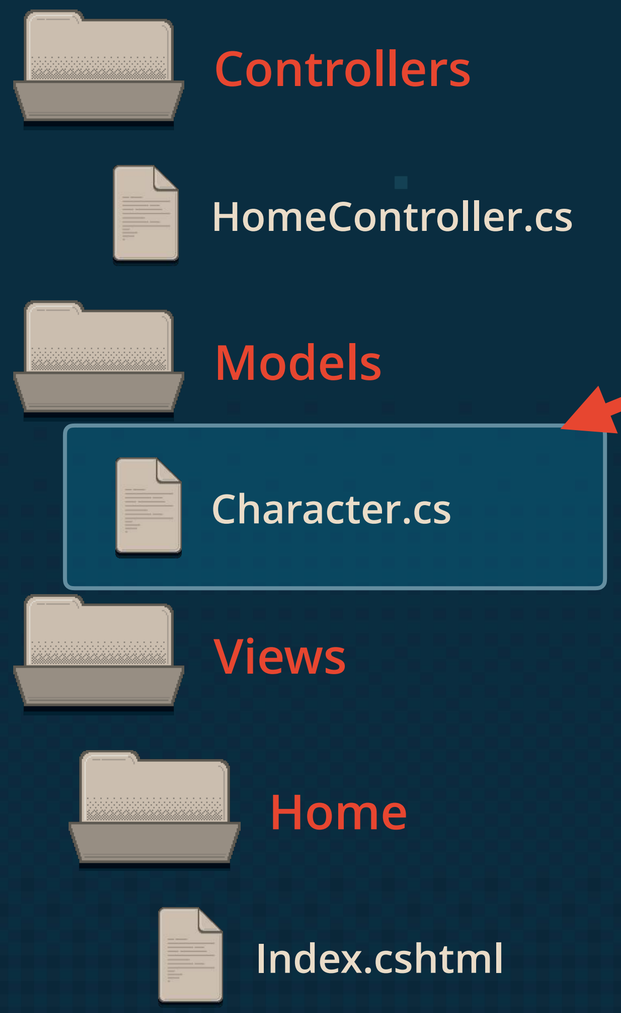
# Model View Controller

## 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




***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

./Controllers/HomeController.cs

CS

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

*We change our model to our Character model.*

*Set the field of an object using dot notation*

**HomeController.cs before**

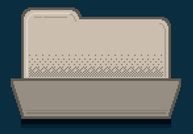
```
public IActionResult Index()
{
  var name = new string();
  name = "Hux";
  return View("Index",name);
}
```

*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>

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

*Change string to be our Character model*

# Using Our Model's Name Field

./Views/Home/Index.cshtml

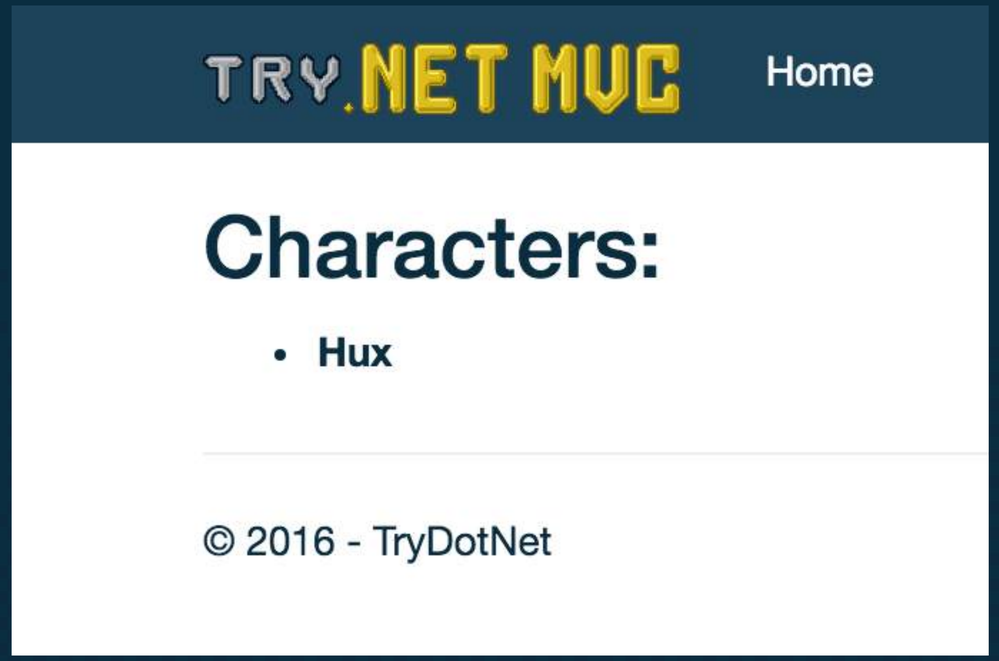
CSHTML

```
@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
.NET MUC

Level 2

# Getting User Input

Level 2 – Section 1

# Getting User Input

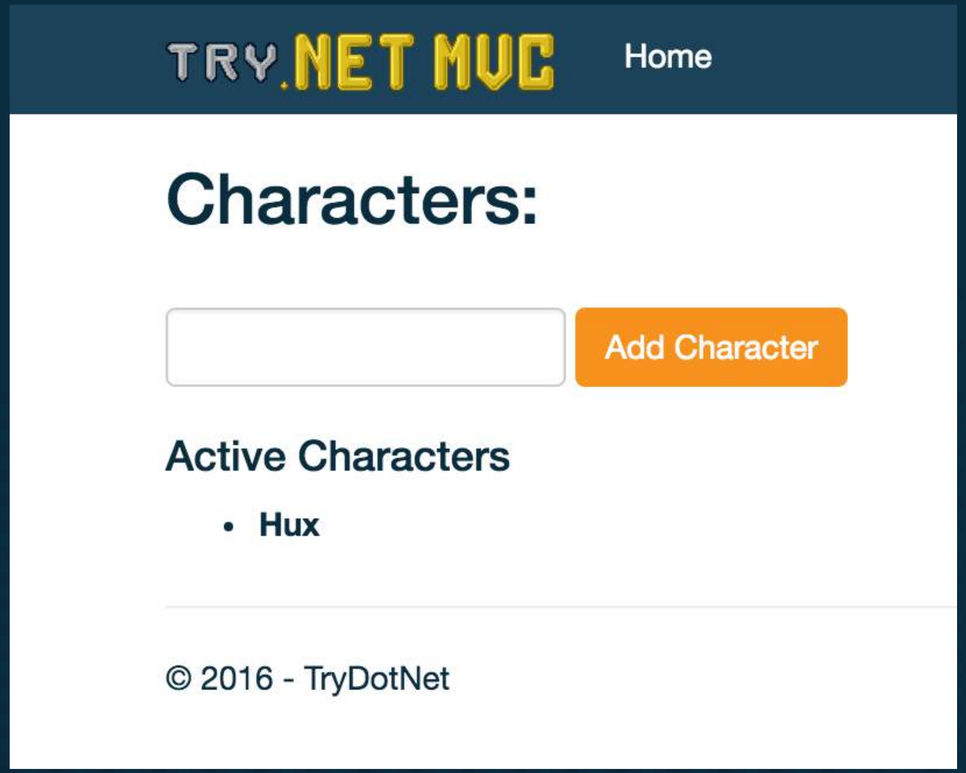
## Application Needs Input

# The Problem...

In the last level, we set up our view to show our names dynamically, but then we hard-coded that name in the controller. Instead, we need to use user input.

### Steps to the Solution

1. Create form in view
2. Add Create() method to controller
3. Move existing logic out of Index() into Create()



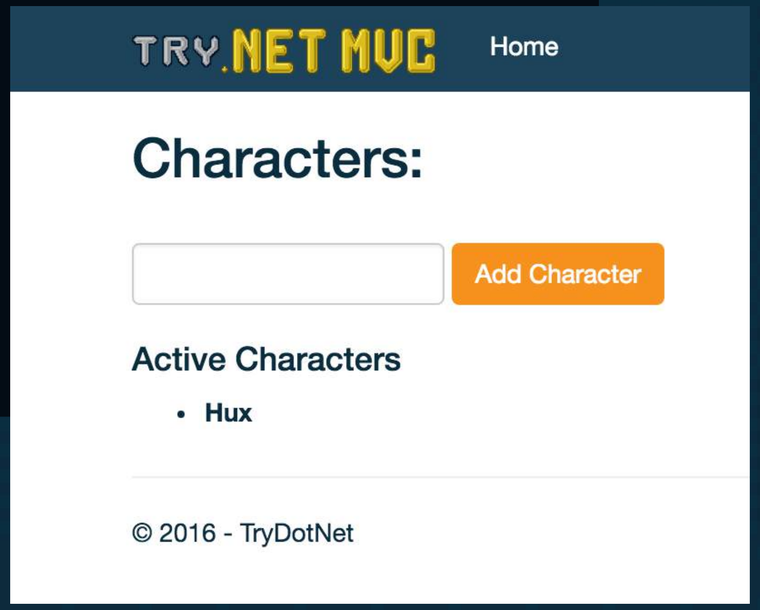
# Create Our New Form

./Views/Home/Index.cshtml

CSHTML

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

<form>
   <div>
      <input name="CharacterName" />
      <input type="submit" value="Add Character" />
   </div>
</form>
```

We've added some HTML to our view for a simple form.

# Inputs Are Matched to Parameters by Name

./Views/Home/Index.cshtml

CSHTML

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

<form>
    <div>
        <input name="CharacterName" />
        <input type="submit" value="Add Character" />
    </div>
</form>
```

*This input's name, CharacterName, will match up with our action's parameter, so these will automatically map up.*

./Controllers/HomeController.cs

CS

```
public IActionResult Create(string characterName)
{
    return View();
}
```

# Creating an Action With an Input Parameter

./Controllers/HomeController.cs

CS

```
public IActionResult Index()
{
    var model = new CharSheetApp.Models.Character();
    model.Name = "Hux";

    return View(model);
}

public IActionResult Create(string characterName)
{
    return View();
}
```

# Moving Our Model to Create

./Controllers/HomeController.cs

CS

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

public IActionResult Create(string characterName)
{
    var model = new CharSheetApp.Models.Character();
    model.Name = "Hux";

    return View(model);
}
```

*We don't want our index creating our character anymore, so let's move that code to our Create method.*

# Using Our Input Parameter

./Controllers/HomeController.cs

CS

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

public IActionResult Create(string characterName)
{
    var model = new CharSheetApp.Models.Character();
    model.Name = characterName;

    return View(model);
}
```

*Change our hard-coded value to our parameter*

# Set Create Action to Use the Index View

./Controllers/HomeController.cs

CS

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

public IActionResult Create(string characterName)
{
    var model = new CharSheetApp.Models.Character();
    model.Name = characterName;

    return View("Index",model);
```

*This will redirect to /Home/Index after the model is created.*

# Using Tag Helpers to Set the Called Action

./Views/Home/Index.cshtml

CSHTML

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

<form asp-action="Create">
    <div>
        <input name="CharacterName" />
        <input type="submit" value="Add Character" />
    </div>
</form>
```

*asp-action* **is a tag helper that tells Razor what controller action we want to use on submit.**

# **Optional:** asp-controller **Tag Helper**

**./Views/Home/Index.cshtml**

CSHTML

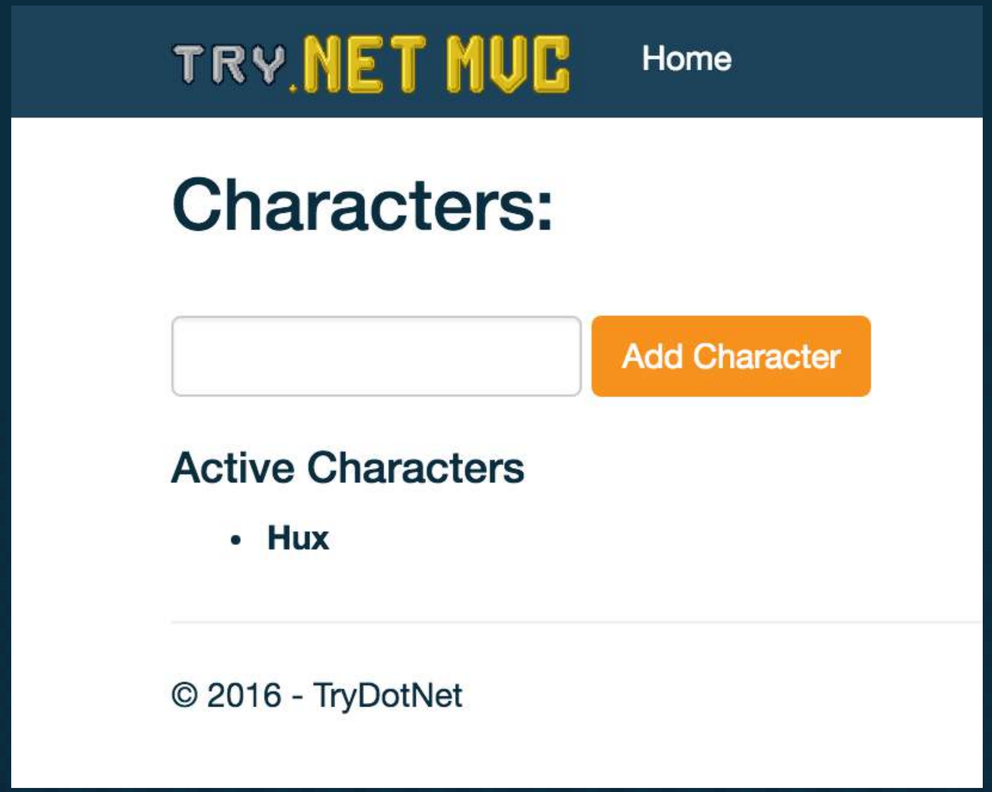
```
@model CharacterSheetApp.Models.Character
<h2>Characters:</h2>

<form asp-action="Create" asp-controller="Home">
    <div>
        <input name="CharacterName" />
        <input type="submit" value="Add Character" />
    </div>
</form>
```

***We can also use the** asp-controller **tag helper to let our form know which controller to find the action in.***


***Without** asp-controller **here, MVC will use the controller that matches your view's directory.***

# Now We Can Accept User Input!

We're finally able to accept user input to create our list of characters, which means our developer doesn't need to manually update our character list anymore.




TRY
.NET MUC

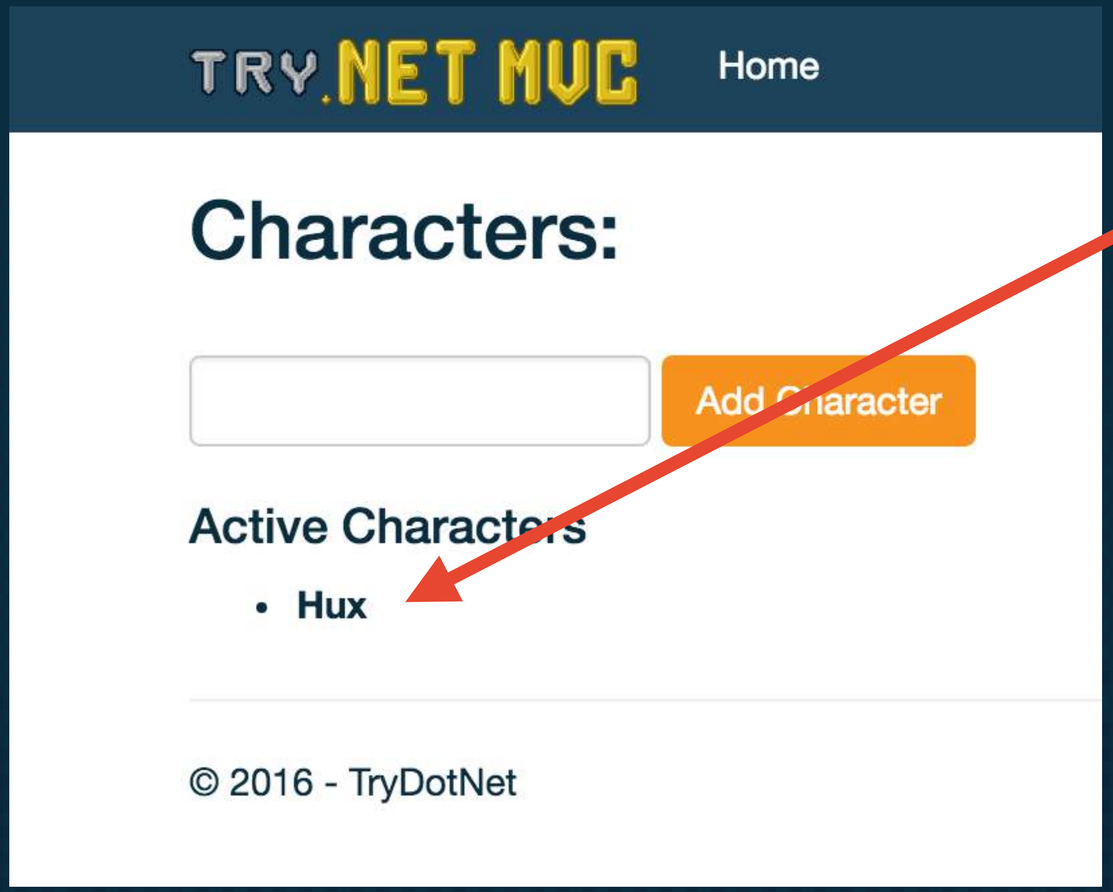
Level 3 – Section 1

# Retaining Data

## Remembering Our Input

# The Problem...

In our last level, we got everything set up so our users could enter character names, but it's not working quite as expected...




*Every new character name overrides the last!*

*We need to set up our application to remember names so we can create an actual list.*

# We Should Group Our Characters Using Lists

We could use either an array or a list to handle our group of characters.

**Adding a new object to an array**

```
Array.Resize(ref characters, characters.Length + 1)
characters[characters.Length - 1] = new Character();
```

*Resizing an array is a lot of code.*

**Adding a new object to a list**

```
characters.Add(new Character());
```

*Resizing a list is a single line of code.*

- *Use arrays for groups of fixed size.*
- *Use lists for groups of variable size.*

# Change Our View's Model to a List

./Views/Home/Index.cshtml

CSHTML

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

<form asp-action="Create">
    <div>
        <input name="CharacterName" />
        <input type="submit" value="Add Character" />
    </div>
</form>
```

*To let us use more than one character, we need to change it into a collection.*

# Our Current @Model Won't Work With Collections

./Views/Home/Index.cshtml

CSHTML

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

<form>…</form>

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

**We're a collection now, so @Model.Name won't give us an individual name anymore.**

# Iterating Through Collections in Razor

./Views/Home/Index.cshtml

CSHTML

```
<ul>
  @foreach (var item in Model)
  {
    <li>
      <label>@item.Name</label>
    </li>
  }
</ul>
```

We could use a *foreach* loop to create a list item tag with data for each item in our collection.

Add the @ symbol before any C# code in your view.

# Razor Parses C# and HTML Well

./Views/Home/Index.cshtml

CSHTML

```
<ul>
  @foreach (var item in Model)
  {
    <li>
      <label>@item.Name</label>
    </li>
  }
</ul>
```

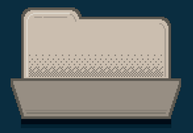
C#

HTML

*You can bounce between C# and HTML as much as needed in Razor!*

# Creating a Place to Store Our Characters

We'll create a class called GlobalVariables to store our characters and equipment.



CharacterSheetApp

GlobalVariables.cs

*We're using global variables here since we're not ready to jump into databases yet. That being said, use global variables sparingly as they can create a number of problems and concerns when misused.*

# Creating a GlobalVariables Class

./GlobalVariables.cs

CS

```
namespace CharacterSheetApp
{
    public class GlobalVariables
    {
        public List<CharacterSheetApp.Models.Character> Characters;
    }
}
```

New class

A variable that holds a list of characters

# Making the Collection Static

./GlobalVariables.cs

CS

```
namespace CharacterSheetApp
{
    public static class GlobalVariables
    {
        public static List<CharacterSheetApp.Models.Character> Characters;
    }
}
```

We only want one character list, so it and the *GlobalVariables* class should be static.


Marking a variable as static means you only need to instantiate it once — so each time you use it, it's the same exact data.

# We're Missing a Reference for List

./GlobalVariables.cs

CS

```
namespace CharacterSheetApp
{
    public static class GlobalVariables
    {
        public static List<CharacterSheetApp.Models.Character> Characters;
    }
}
```

The type or namespace name 'List<CharacterSheetApp.Models.Character>' could not be found (are you missing a using directive or an assembly reference?)


*We're getting an error because this class doesn't have access to the List class.*

# Accessing Other Namespaces in Code

./GlobalVariables.cs

CS

```
namespace CharacterSheetApp
{
    public static class GlobalVariables
    {
        public static System.Collections.Generic.List
            <CharacterSheetApp.Models.Character> Characters;
    }
}
```

*Using the entire namespace works, but it can get messy and repetitive.*

# Accessing a Namespace Via “Using Directives”

./GlobalVariables.cs

CS

```
using System.Collections.Generic;

namespace CharacterSheetApp
{
    public static class GlobalVariables
    {
        public static List<CharacterSheetApp.Models.Character> Characters;
    }
}
```

*Using directives lets us use classes from another namespace in our current namespace.*

# Put the Methods After Our Fields

./Models/Character.cs

CS

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

*Methods typically go after our fields.*

# Add a Character to Our Characters Collection

./Models/Character.cs

CS

```
        public string Name;

        public void Create(string characterName)
        {
            var character = new Character();
            character.Name = characterName;
            GlobalVariables.Characters.Add(character);
        }
    }
}
```

*List collections have an add() method you can use to add objects to the list.*

# Add a Character to Our Characters Collection

./Models/Character.cs

CS

```
        public string Name;

        public void Create(string characterName)
        {
            var character = new Character();
            character.Name = characterName;
            if(GlobalVariables.Characters == null)
                GlobalVariables.Characters = new List<Character>();
            GlobalVariables.Characters.Add(character);
        }
    }
}
```

We need to be careful that *Characters* isn't null or we'll get an error, so check if it's null and initialize it in the event that it is.

# Making a Method Static

./Models/Character.cs

CS

```
        public string Name;

        public static void Create(string characterName)
        {
            var character = new Character();
            character.Name = characterName;
            if(GlobalVariables.Characters == null)
                GlobalVariables.Characters = new List<Character>();
            GlobalVariables.Characters.Add(character);
        }
    }
}
```

*A static method means you don't need to instantiate the class first.*


**Calling Our** Create **Method**

C#

```
Models.Character.Create("Hux")
```

# Use Our New Character.Create Method

./Controllers/HomeController.cs

CS

```
public IActionResult Create(string characterName)
{
    Models.Character.Create(characterName);

    return View("Index",model);
}
```

*Since we made Create static, we can call it without instantiating it first.*

# **Set** Create **to Redirect to** Index

./Controllers/HomeController.cs

CS

```
public IActionResult Create(string characterName)
{
    Models.Character.Create(characterName);

    return RedirectToAction("Index");
}
```


*RedirectToAction* **keeps us from accidentally creating our character twice by refreshing, avoids duplicate code, etc.**

# Creating Our Character.GetAll Method

./Models/Character.cs

CS

```
            if(GlobalVariables.Characters == null)
                GlobalVariables.Characters = new List<Character>();
            GlobalVariables.Characters.Add(character);
        }

        public static List<Character> GetAll()
        {
            return GlobalVariables.Characters;
        }
    }
}
```

*This new method will return our characters list.*

# Creating Our Character.GetAll Method

./Models/Character.cs

CS

```
            if(GlobalVariables.Characters == null)
                GlobalVariables.Characters = new List<Character>();
            GlobalVariables.Characters.Add(character);
        }

        public static List<Character> GetAll()
        {
            if(GlobalVariables.Characters == null)
                GlobalVariables.Characters = new List<Character>();
            return GlobalVariables.Characters;
        }
    }
}
```

*We need to make sure we handle the null here as well.*

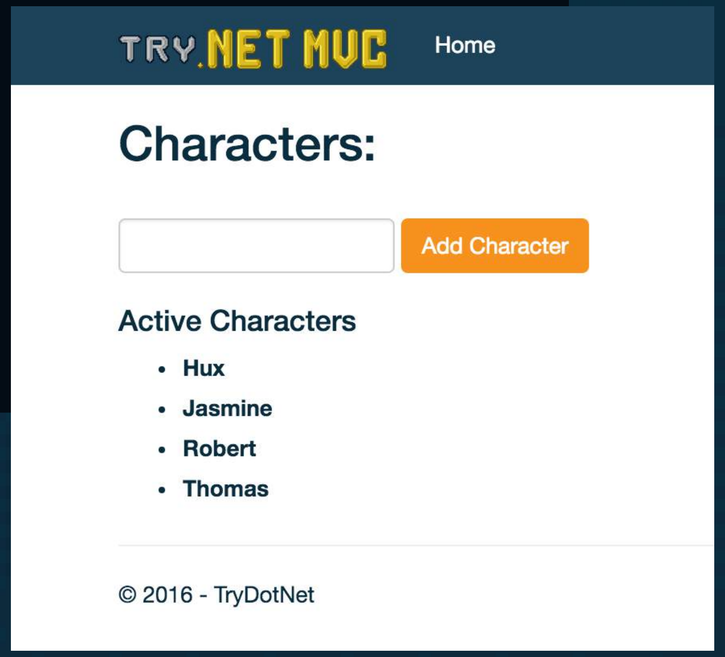
# **Set** Index **to Use Our** Character.GetAll

**./Controllers/HomeController.cs**

CS

```
public IActionResult Index()
{
    return View(Models.Character.GetAll());
}
```

***Our** GetAll **method gives us our list of characters, and using it in our** View **method will pass the entire list to our view so we can display them.***

# Manually Handling Nulls Is Less Than Ideal

Anytime we access Characters, we need to handle nulls — and that's a lot of duplicate code.

**Handling Nulls Manually**

C#

```
if(GlobalVariables.Characters == null)
        GlobalVariables.Characters = new List<Character>();
```

*Let's change Characters from a variable to a property.*

*Making a variable a property allows us to override what our application does when it retrieves or sets the value of the property.*

TRY .NET MVC

# Setting Up Our Get Setters

./GlobalVariables.cs

CS

```
using System.Collections.Generic;

namespace CharacterSheetApp
{
    public static class GlobalVariables
    {
        public static List<Character> Characters { get; set; }
    }
}
```

*Adding a get setter to a field or variable changes it into a property.*

# Give Characters a Default Value Instead

./GlobalVariables.cs

CS

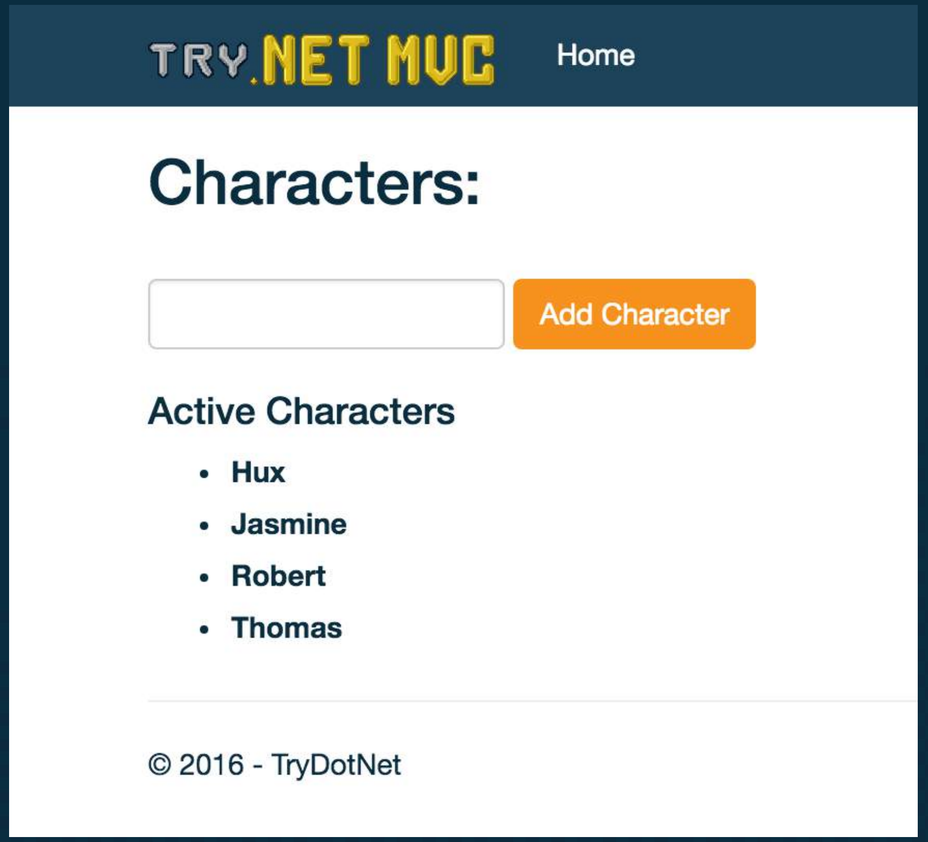
```
using System.Collections.Generic;

namespace CharacterSheetApp
{
    public static class GlobalVariables
    {
        public static List<Character> Characters { get; set; }
            = new List<Character>();
    }
}
```

*In the event our Characters property is null, it'll use whatever we set after the = as the default value.*

# Our First Application Is Complete

We had two goals at the start of this course, and our application now fulfills those goals.

- Users can add new characters.
- Users can see all added characters.




TRY
.NET MUC