TRY .NET MVC

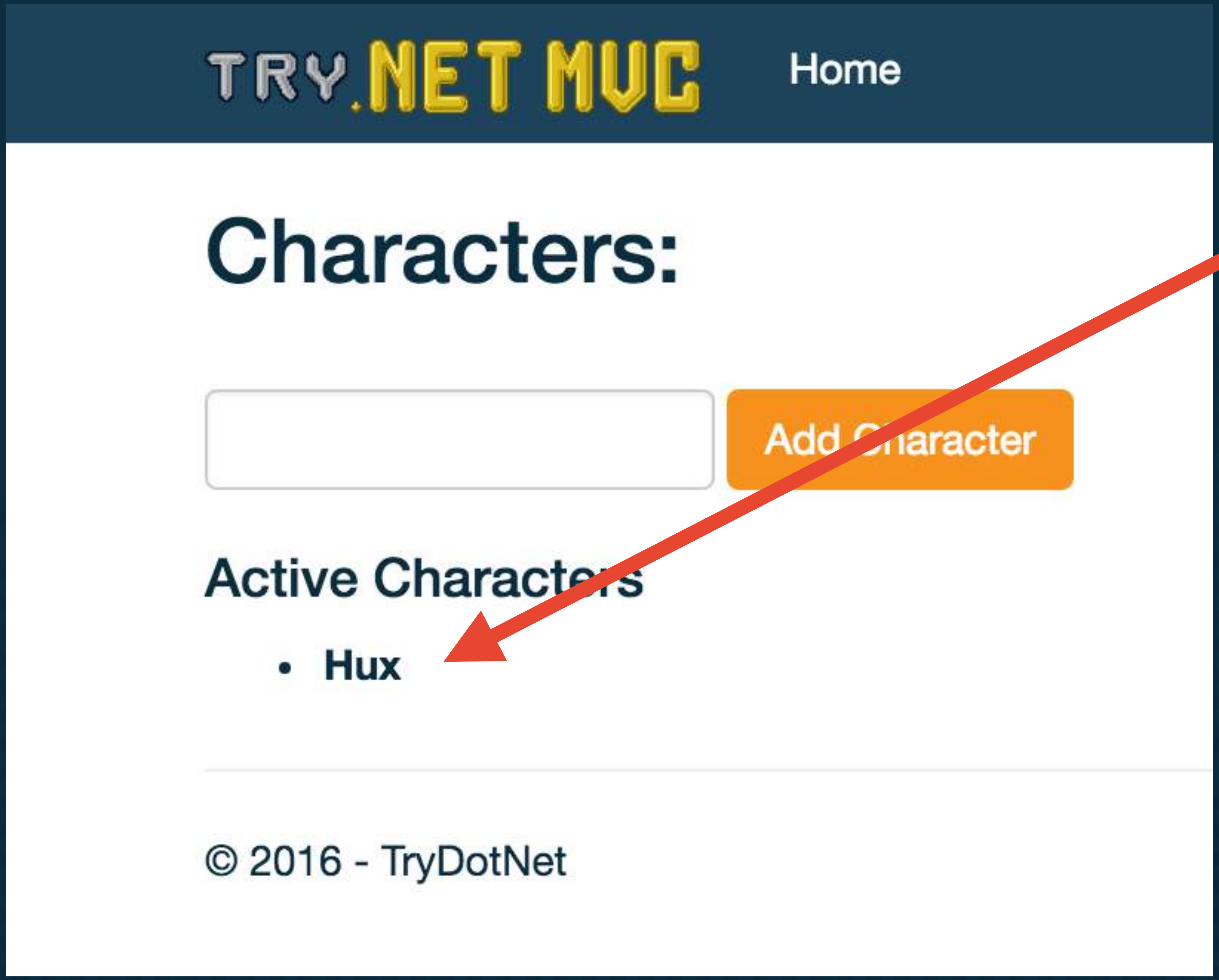
Level 3 – Section 1

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...



TRY.NET MVC Home

Characters:

Add Character

Active Characters

- Hux

© 2016 - TryDotNet

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.

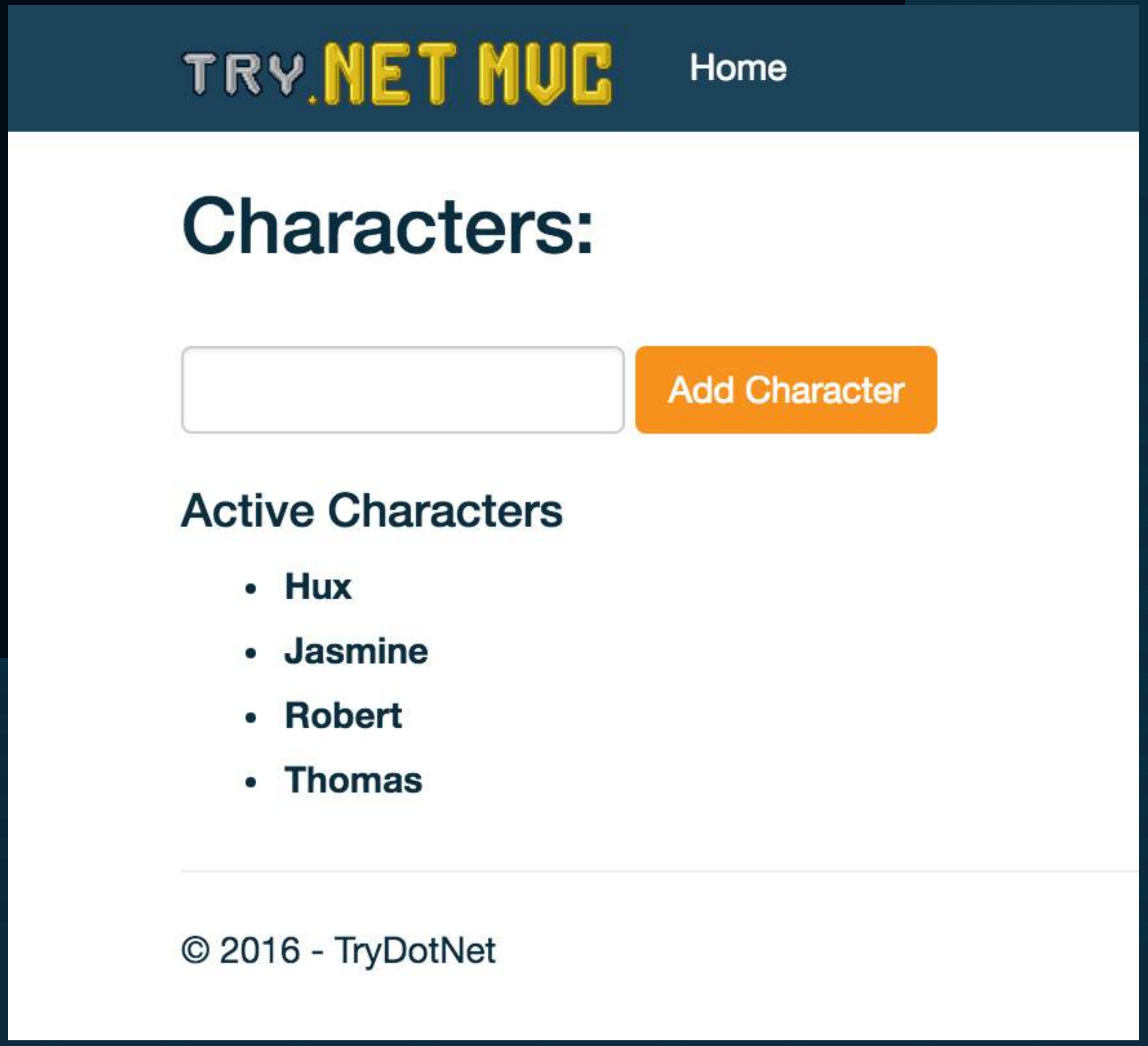
Set Index to Use Our Character.GetAll

./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.



Characters:

Active Characters

- Hux
- Jasmine
- Robert
- Thomas

© 2016 - TryDotNet

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>();
    }
}
```

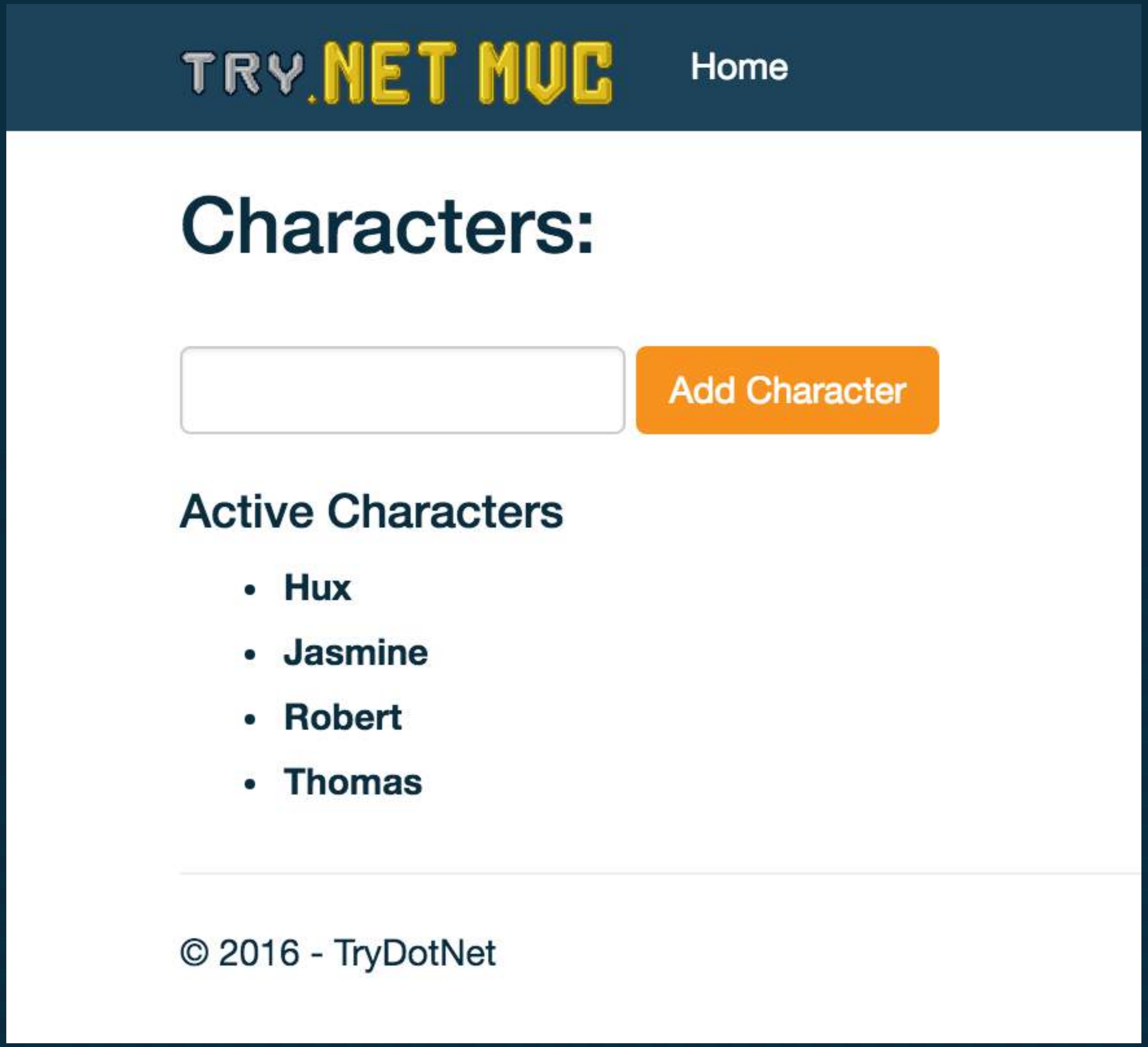
In the event our Characters property is null, it'll use whatever we set after the = as the default value.

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 MVC Home

Characters:

Add Character

Active Characters

- Hux
- Jasmine
- Robert
- Thomas

© 2016 - TryDotNet

TRY
.NET MVC