
FORGING AHEAD
with
.NET
MVC

Level 4

# Creating Logical URLs

## Custom Routes

# Making Our URLs More User Friendly

Some of our URLs are hard to remember and understand. We should simplify them.

*Current URL to Our Details Page*

localhost:5000/Character/Details?Name=Hux

*Desired URL to Our Details Page*

localhost:5000/Character/Hux/Details

- It's clearer where a URL will take us
- Search engines use URLs in their algorithms
- We should avoid query strings

# Where Do We Want to Set Our Custom Route?

Application-wide routes are typically set in Startup.cs — otherwise, they can be set on the specific action affected.

*Our default route is application-wide, so it's set up in our Startup.cs file.*

localhost:5000/Character/Details?Name=Hux

*Our new routes will be "per action" and will live in our CharacterController.cs file.*

localhost:5000/Character/Hux/Details

# Adding a Custom Route

To create a custom route, we just add a route attribute with the route details to our action.

**Controllers/CharacterController.cs**

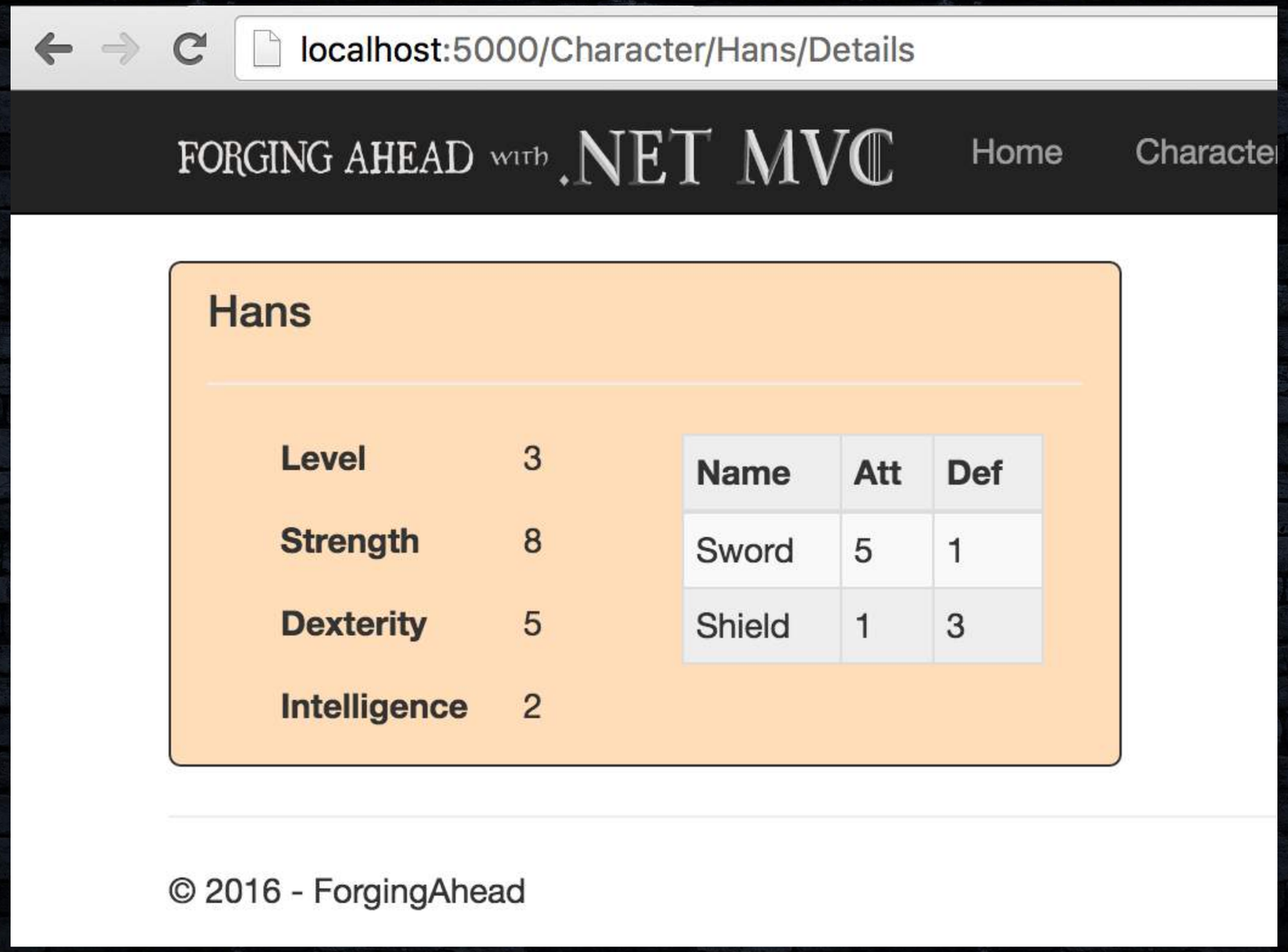
CS

```
…
public class CharacterController : Controller
{
…
  [Route("Character/{name}/Details")]
  public IActionResult Details(string name)
  {
    ViewData["Title"] = name;

    var model = _context.Characters.FirstOrDefault(e => e.Name == name);
    return View(model);
  }
…
}
```

*Routes tied to specific actions should specify everything that isn't a parameter to reduce risk of conflict.*

# Now We Have Much Nicer URLs

This will help our users with our application, as well as help us show up on search results.



# We're Seeing Weird Stuff With Create

Looking at how users are accessing our website, we're seeing something we didn't expect.

*We expect our users to reach our Character Create page through:*

localhost:5000/Character/Create

*Then use the form we've made to create their character*

*But we're seeing users bypass our form to create characters using query strings...*

localhost:5000/Character/Create?Name=Test&IsActive=True&Level=1&Strength=5&Dexterity=5&Intelligence=5

*This can be a serious problem depending on what data is passed.*

# URLs Shouldn't Be Used to Submit Data

Users can directly create a character using query strings because we allow both GET and POST.

- URLs have a limited length.
- URLs can leak sensitive data.
- URLs posting data can accidentally be resubmitted.
- URLs use the verb GET, which is meant to request data, not submit it.

https://localhost:5000/Account/Login?UserName=Administrator&Password=InPlainText

*Even with encryption, passing sensitive data via query strings is EXTREMELY dangerous! GET requests can leak sensitive information easily!*

# HttpVerbs **Help Indicate Data Interaction**

**HTML forms and URLs only support** GET **and** POST**, so we're only going to worry about them.**

- GET **- Requests data from the application**
- POST **- Submits data to the application**

***Unless we specify a verb, all action methods will allow both*** *GET* ***and*** *POST* ***by default.***

# Examples of GET

GET is all about making a request for data.

- Index - Requests a list of all characters
- Details - Requests a character's data
- Edit - Requests a character's data

# Adding the HttpGet Attribute Allows GET

Specifying that Index is an HttpGet, it will only accept GET and no longer accept POST requests.

**Controllers/CharacterController.cs**

CS

```
…
public class CharacterController : Controller
{
…
  [HttpGet]
  public IActionResult Index()
  {
    ViewData["Title"] = "Characters";

    var model = _context.Characters.ToList();
    return View(model);
  }
…
}
```

*Any action that just returns data will typically use GET.*

# Examples of POST

POST is about making a request to submit or update data.

- Create - Requests to add new character data
- Update - Requests to update character data
- Delete - Requests to delete character data

# Adding the HttpPost Attribute Allows POST

Specifying that Create is an HttpPost means Create will no longer accept query strings.

**Controllers/CharacterController.cs**

CS

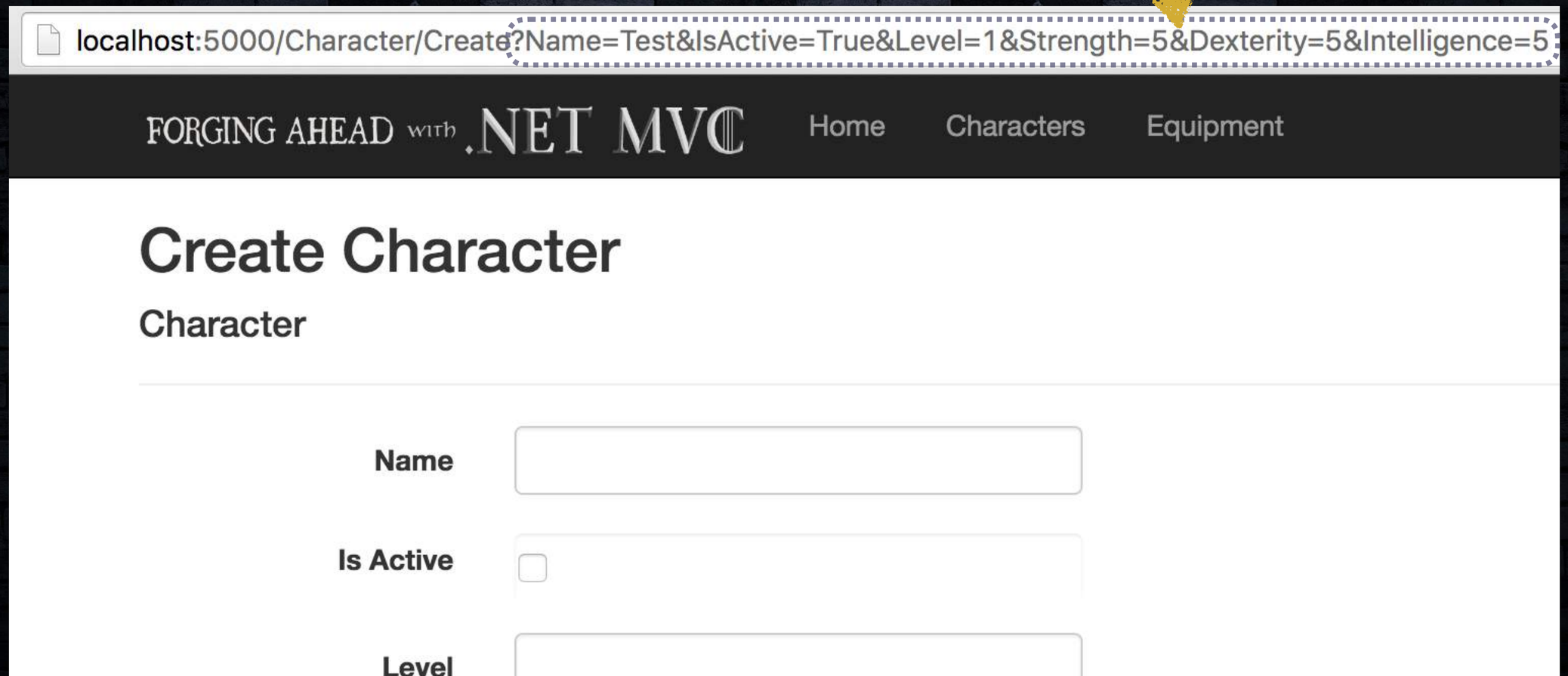
```
…
public class CharacterController : Controller
{
…
  [HttpPost]
  public IActionResult Create(Character character)
  {
    _context.Characters.Add(character);
    _context.SaveChanges();

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

*Any action we submit data to will typically use POST.*

# Query Strings No Longer Work

If we try to access Create using query strings like before, we'll be taken to the Create view.

*Now, no matter what query strings we put after Create in our URL, we'll be taken to the Create view.*



# What Verb Do We Use With Our Delete Action?

If we created an HttpRequest, we'd use Delete — but without it, we need to fallback to POST.

**Controllers/CharacterController.cs**

CS

```
…
  [HttpPost]
  [Route("Character/{name}/Delete")]
  public IActionResult Delete(string name)
  {
    var character = _context.Characters.FirstOrDefault(e => e.Name == name);
            _context.Characters.Remove(character);
            _context.SaveChanges();
    return RedirectToAction("Index");
  }
…
```

*We actually have HttpDelete as an attribute, but unless we use JavaScript, we'll need to use HttpPost instead.*

# What Verb Do We Use With Our Update Action?

Updates are tricky, as PATCH, POST, or PUT could be used. In our case, we'll want POST.

**Controllers/CharacterController.cs**

CS

*POST is the most appropriate for updating an entire record without creating one if it doesn't exist.*

```
…
  [HttpPost]
  public IActionResult Update(Character character)
  {
    _context.Entry(character).State = EntityState.Modified;
    _context.SaveChanges();
    return RedirectToAction("Index");
  }
…
```

# Now We're Using the Proper HttpVerbs

**Using the proper verbs is good for a variety of reasons that will benefit our application:**

- It helps keep sensitive information from leaking
- Prevents search engines from indexing actions for submitting data
- You'll need to know **HttpVerbs** if you host or consume a web service


FORGING AHEAD
with
.NET
MVC