Chapter 17: Creating a Form

Gill Cleeren

@gillcleeren

Agenda

- Tag Helpers
- Model binding
- Model validation

Tag Helpers

Tag Helpers

Tag Helpers enable server-side code to participate in creating and rendering HTML elements in Razor files

- Server-side
- Trigger code
- Built-in or custom
  - We've already seen a custom one… <email>
  - Built-in commonly used in combination with Forms
- Replace HTML Helpers

Custom tag helpers

```
<email address="info@@bethanyspieshop.com"
  content="Contact us"></email>

<a href="mailto:info@bethanyspieshop.com">Contact us</a>
```

Built-in tag helpers (form-related)

- Form tag helper
- Input tag helper
- Label tag helper
- Textarea tag helper
- Select tag helper
- Validation tag helpers

Using the Label tag helper

```
<label asp-for="FirstName">
</label>
```

Label Tag Helper

```
<label for="FirstName">
  FirstName
</label>
```

Resulting HTML

```
<label for="FirstName">
  First name
</label>
```

Attributes on Model

```
<label for="FirstName"
  class="SomeClass">
  First name
</label>
```

Other HTML attributes

The Form tag helper

- Generates the form action attribute
- Generates request verification token

```
<form asp-action="Checkout" method="post"
  asp-antiforgery="true" class="form-horizontal"
  role="form">
  ...
</form>
```

The Form tag helper example output

```
<form asp-controller="Demo" asp-action="Register" method="post">
<!-- Input and Submit elements --> </form>
```

```
<form method="post" action="/Demo/Register">
     <!-- Input and Submit elements -->
     <input name="__RequestVerificationToken" type="hidden"
value="<removed for brevity>" />
    </form>
```

The Form tag helper

- Defines
  - asp-controller
  - asp-action
  - asp-route-*
  - asp-route
  - asp-antiforgery

The input tag helper

```
<input asp-for="PieName" />
```

- Generates id and name HTML attributes for asp-for value
- type is set based on model data and optional annotations
- Can optionally generate HTML5 validation attributes
- Feature-wise, very close to @Html.TextBoxFor/@Html.EditorFor
- Note that you will get IntelliSense + compilation errors if we change the asp-for value!

The input tag helper

```
using System.ComponentModel.DataAnnotations;

namespace FormsTagHelper.ViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

```
@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput"
method="post">
    Email:  <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>
```

```
<form method="post" action="/Demo/RegisterInput">
      Email:
      <input type="email" data-val="true"
            data-val-email="The Email Address field is not a valid e-
mail address."
            data-val-required="The Email Address field is required."
            id="Email" name="Email" value="" /> <br>
      Password:
      <input type="password" data-val="true"
            data-val-required="The Password field is required."
            id="Password" name="Password" /><br>
      <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden"
value="<removed for brevity>" />
  </form>
```

Tag Helpers

```
public class RegisterAddressViewModel
{
    public string Email { get; set; }

    [DataType(DataType.Password)]
    public string Password { get; set; }

    public AddressViewModel Address { get; set; }
}
```

```
@model RegisterAddressViewModel

<form asp-controller="Demo" asp-action="RegisterAddress"
method="post">
    Email:  <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    Address: <input asp-for="Address.AddressLine1" /><br />
    <button type="submit">Register</button>
</form>
```

Tag Helpers

- Also exist for
  - Anchor
  - Image
  - CSS
  - JavaScript
  - …

We'll learn about these later in the course!

CSS Tag Helper

```
<link rel="stylesheet"
   asp-href-include="lib/bootstrap/dist/css/*.min.css" />
```

DEMO

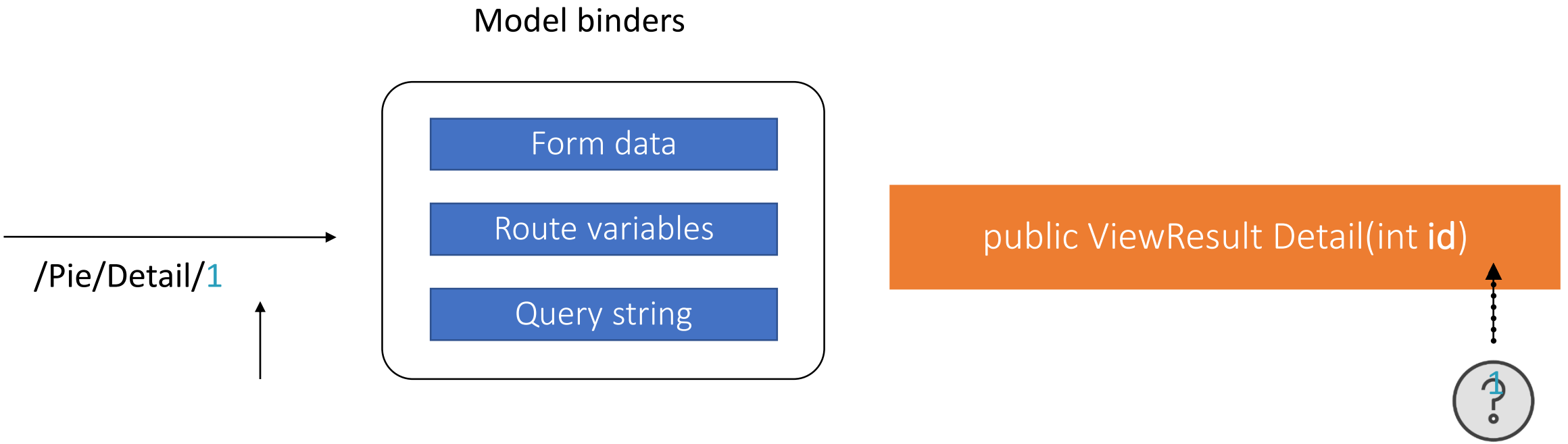
Creating a form using tag helpers

Model binding

Model binding

- Model binding maps the incoming HTTP requests to action method parameters
  - Can be simple (string, int)
  - Can be complex (type)
  - Can be list
- Model binding takes away the repetitive process that we would otherwise have to do this all the time again

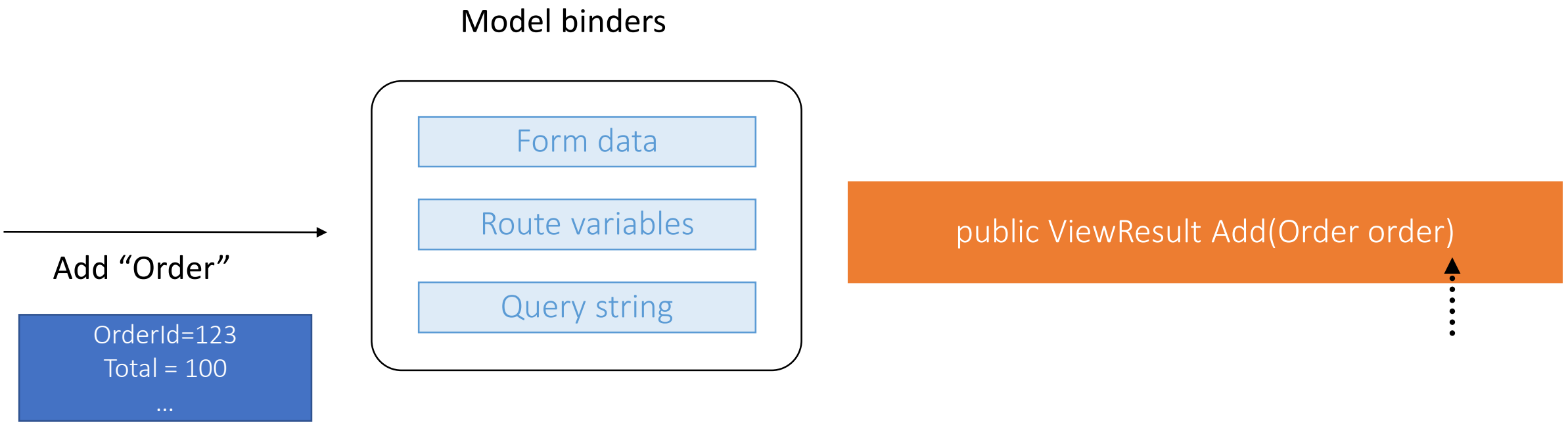
Model binding



How model binding works

- ASP.NET will first check which action method needs to get called
  - Based on route data
- Will then bind values from the request to the method parameters
- ASP.NET will search in different locations for these values
  - Works through model binders
    - Form values
    - Route values
    - Query strings
    - Are evaluated in this order
  - Also “.” is searched
    - ParameterName.PropertyName

Model binding for complex types

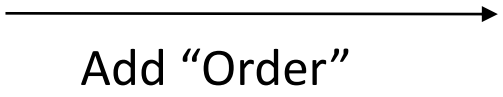


DEMO

Model binding

Validation

Validation is required on the controller



Add “Order”

OrderId=“ABC”
Total = “Hello world”
…

Validation

public ViewResult Add(Order order)

Model validation

- Before data is stored, it needs to be validated
  - Even security risks!
  - Format must be checked
  - Custom rules must be checked
- Can be tedious to implement
- MVC automates this quite a lot
  - Client and server

Validation attributes

- Configuration of model so it's consistent to the data model in DB
  - Constraints, required, format…

```
public class Order
{
    [Required(ErrorMessage = "Please enter your first name")]
    [StringLength(50)]
    public string FirstName { get; set; }

    [Required(ErrorMessage = "Please enter your phone number")]
    [StringLength(25)]
    [DataType(DataType.PhoneNumber)]
    public string PhoneNumber { get; set; }
}
```

Available (built-in) attributes

- Required
- StringLength
- Range
- RegularExpression
- DataType
  - Phone
  - Email
  - Url
- CreditCard
- Compare

- All derive from ValidationAttribute
- Live in System.ComponentModel.DataAnnotations

Validation trigger

```
if (ModelState.IsValid)
{
    _orderRepository.CreateOrder(order);
    return RedirectToAction("CheckoutComplete");
}
else
{
    return View();
}
```

Model state

- Model state represents validation errors in submitted HTML form values
- Model validation happens when binding occurs
  - We need to check validity ourselves through ModelState.IsValid
- Custom rules can also be created

Client-side validation

- Saves us from roundtrip to the server
  - Higher responsiveness of the app
- ASP.NET Core MVC will add client-side validation automatically
  - Will add data-* attributes on client-side generated code
  - Works in combination with jQuery Validation
- Correct scripts need to be added

Client-side validation

```
<div class="form-group">
    <label asp-for="ReleaseDate" class="col-md-2 control-label"></label>
    <div class="col-md-10">
        <input asp-for="ReleaseDate" class="form-control" />
        <span asp-validation-for="ReleaseDate" class="text-danger"></span>
    </div>
</div>
```

Generated code

```
<form action="/movies/Create" method="post">
  <div class="form-horizontal">
    <h4>Movie</h4>
    <div class="text-danger"></div>
    <div class="form-group">
      <label class="col-md-2 control-label" for="ReleaseDate">ReleaseDate</label>
      <div class="col-md-10">
        <input class="form-control" type="datetime"
        data-val="true" data-val-required="The ReleaseDate field is required."
        id="ReleaseDate" name="ReleaseDate" value="" />
        <span class="text-danger field-validation-valid"
        data-valmsg-for="ReleaseDate" data-valmsg-replace="true"></span>
      </div>
    </div>
    </div>
</form>
```

Visualizing client-side validation errors

```
<form asp-action="Checkout" method="post">
    <div asp-validation-summary="All" class="text-danger">
    </div>
</form>
```

DEMO

Simple validation

Summary

- Model binding takes away need to search for correct request data
- Validation is required
- ASP.NET Core MVC handles this for us