# Working with Forms and Model Binding

**Gill Cleeren**
CTO Xpirit Belgium

@gillcleeren – xpirit.com/gill

# Module overview

- Creating a form using tag helpers
- Understanding model binding
- Adding validation
- Recreating the form using a Razor Page

# Creating a Form Using Tag Helpers

# Built-in Form Tag Helpers

**Form tag helper**

**Input tag helper**

**Label tag helper**

**Textarea tag helper**

**Select tag helper**

**Validation tag helpers**

```
<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

# Form Tag Helpers

**asp-controller**

**asp-action**

**asp-route-***

**asp-route**

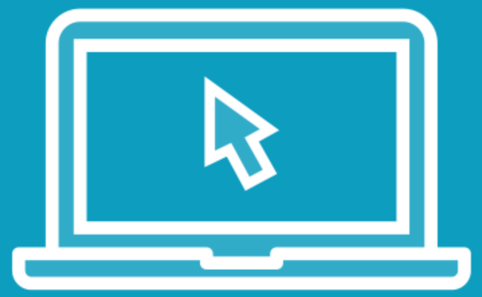
**asp-antiforgery**

```
<form asp-action="Checkout" method="post"
  role="form">
  ...
</form>
```

# Form Tag Helper

**Generates <form> action attribute**
**Generates hidden token against Cross-site request forgery**

# Demo



**Adding support for the Order creation**

**Creating the Order form**

**Navigating to the Order form**

# Understanding Model Binding

Our pages will need data to work with, coming from the request.

This can be a route value, posted data...

Extracting that data manually is a lot of work. Over and over again.

Model binding is a process in ASP.NET Core that will extract data from the request and provide that to the controller actions.

# Model Binding

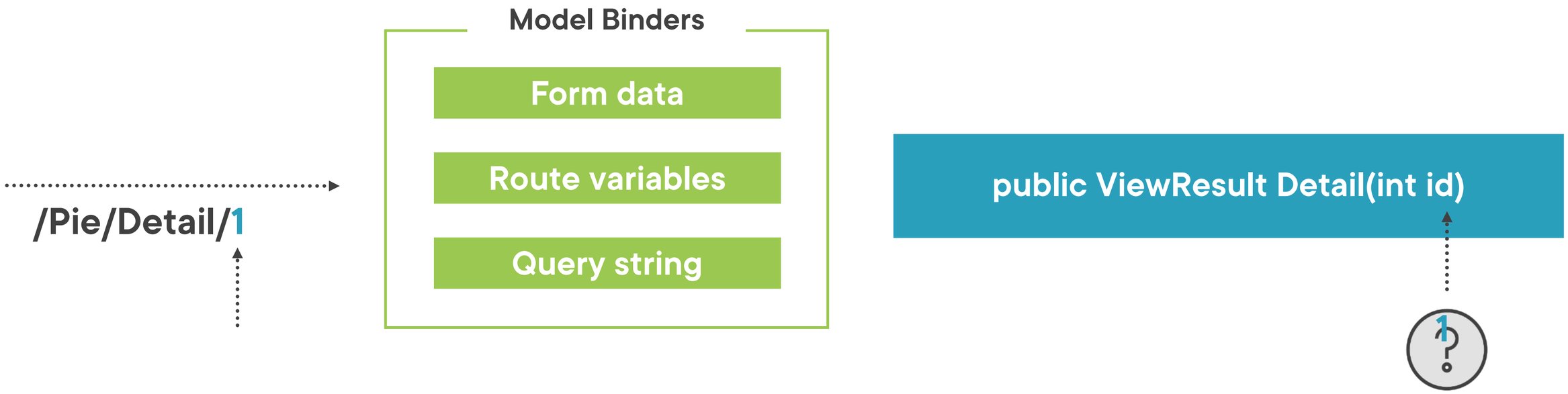
**Data from request**

**Various sources**

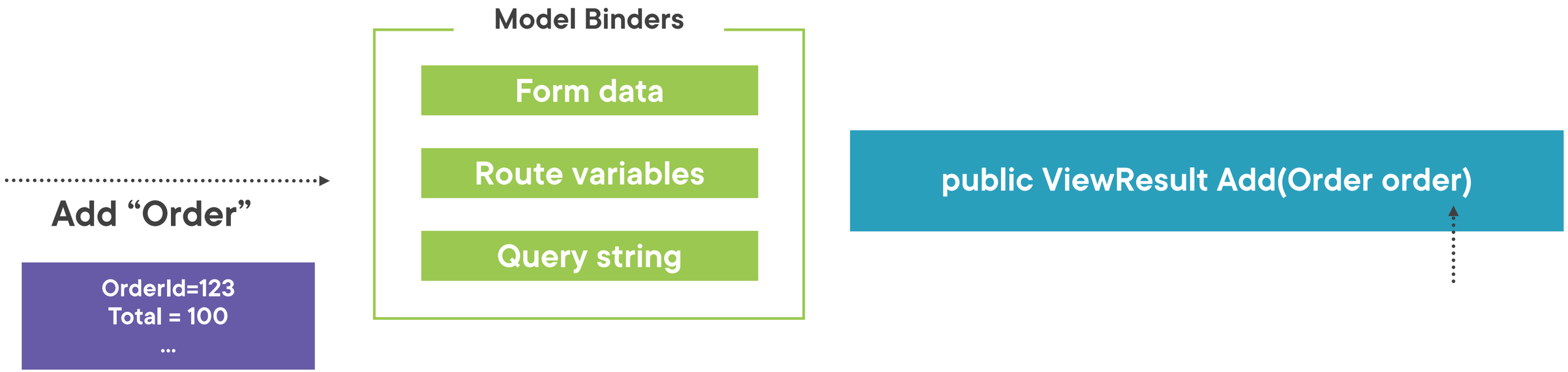
**Passed to controller actions**

**Complex types**

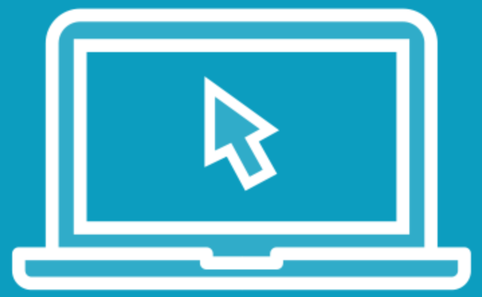
# Model Binding



# Binding to Complex Types



# Demo



## Accessing posted data using model binding

# Adding Validation

# The Need for Validation

Add “Order”

OrderId=“ABC”
Total = “Hello world”
...

Validation

public ViewResult Add(Order order)

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

# Validation

**ModelState contains binding and validation errors**

# ModelState Properties

**IsValid**

**GetValidationState**

**AddModelError**

# Validation

**Attributes on the model classes**

**Constraints, required, regex patterns...**

**Custom attributes**

# Validation Attributes

**Required**

**StringLength**

**Range**

**RegularExpression**

**EmailAddress**

**Phone**

# Adding Validation Attributes

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

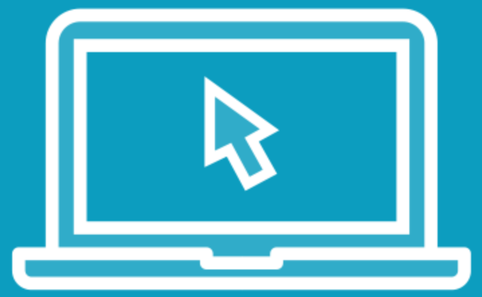
```
<form asp-action="Checkout" method="post">
    <div asp-validation-summary="All" class="text-danger">
    </div>
</form>
```

# Displaying a Validation Summary

# Demo

## Adding validation to the form

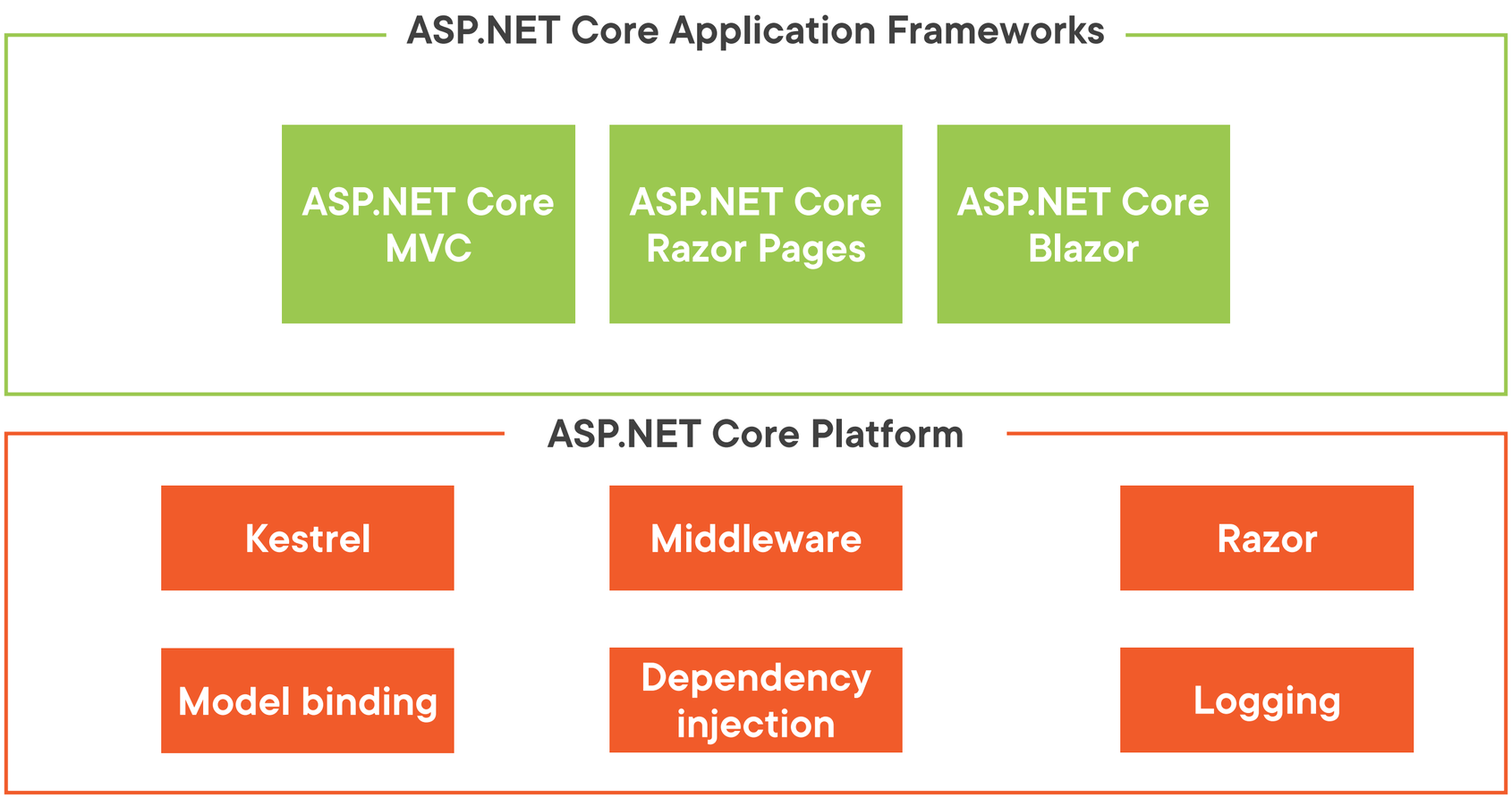
# Demo



## Adding client-side validation

# Recreating the Form Using a Razor Page

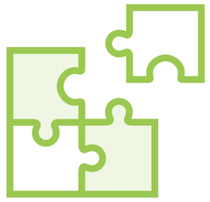
# ASP.NET Core Application Frameworks



# Introducing Razor Pages

**Simpler form of creating pages**

**Based on PageModel and code-behind**



**Can co-exist in single application with MVC**

**Most concepts apply**

# Configuring the Application for Razor Pages

## Changes to Program.cs

```
builder.Services.AddRazorPages();

app.MapRazorPages();
```

```
@page

<h1>Welcome to Bethany's Pie Shop</h1>
<h2>The time is now @DateTime.Now</h2>
```

# A First Razor Page

**@page directive**
**Must be first directive**

```
@page
@model CheckoutPageModel

<p>
    @Model.Order.Total.ToString()
</p>
```

# Using the @model Directive

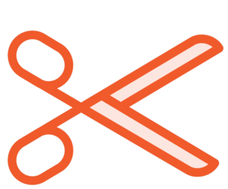
**Page model class**
**Links to class with same name as Page itself**
**@Model gives access to methods and properties**

```
public class CheckoutPageModel : PageModel
{
    public Order? Order { get; private set; }
}
```

# The PageModel Class

**Order defines a Total property**

# Page Handler Methods



**Allows for separating of page and its data**

**OnGet()**

**OnPost()**

**Async versions exist too**

```
public Order? Order { get; set; }


public IActionResult OnGet()
{
    Order = dbContext.Orders.First();
    return Page();
}
```

# Using OnGet()

# Most Concepts Are the Same

**Dependency injection**

**Layout page**

**Partial views**

**Tag Helpers**

**View Components**

# Routing to Pages


**Routing is simpler**

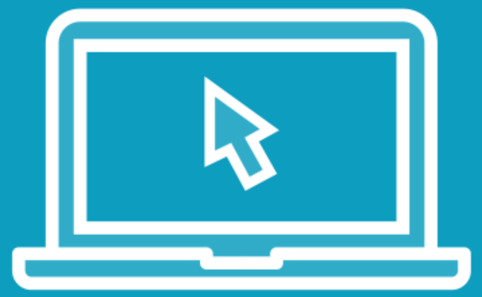
**Pages folder as root**

**Filename and location**

# Routing to Razor Pages

| Route | Page file |
| --- | --- |
| / | /Pages/index.cshtml |
| /Contact | /Pages/contact.cshtml |
| /Store/Pies | /Pages/Store/Pies.cshtml |
| /Store/Pies/3 | /Pages/Store/Pies.cshtml<br>@page "{id:int?}" |

# Demo



**Configuring the application for Razor Pages**

**Recreating the Checkout page using Razor Pages**

# Take a look at ASP.NET Razor Pages Fundamentals

# Summary

- Model binding helps access to data
- Tag helpers are used to create forms
- Razor Pages offer a simpler alternative to building pages

**Up next:**
Testing our code