Working with New Items

Gill Cleeren

CTO Xebia Microsoft Services Belgium

@gillcleeren

Overview

ASP.NET Core Features for Forms

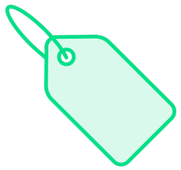
Adding a Create page

Adding validation and ModelState

Handling errors

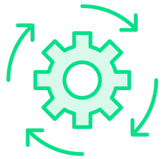
ASP.NET Core Features for Forms

Tag Helpers in ASP.NET Core



Server-side tags, look like regular HTML

Code execution



Built-in or custom

Navigation, forms...

A Simple Tag Helper

Razor code

```
<a
    asp-controller="Pie"
    asp-action="List">
        View Pie List
</a>
```

Resulting HTML

```
<a href="/Pie/List">View Pie List</a>
```

Form Tag Helpers

Form

Input

Label

Textarea

Select

Validation

```
<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**

```
@model Category

<h2>Details for @Model.Name</h2>
<hr />
```

Using @model and @Model

Accessing the Posted Data

Adding Model Binding

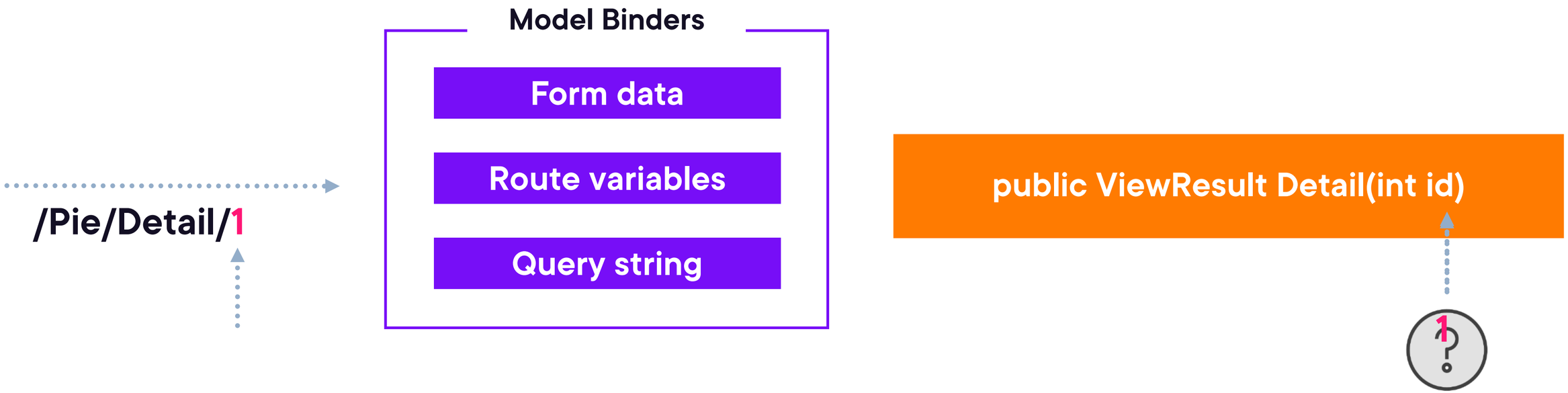
Data from request

Various sources

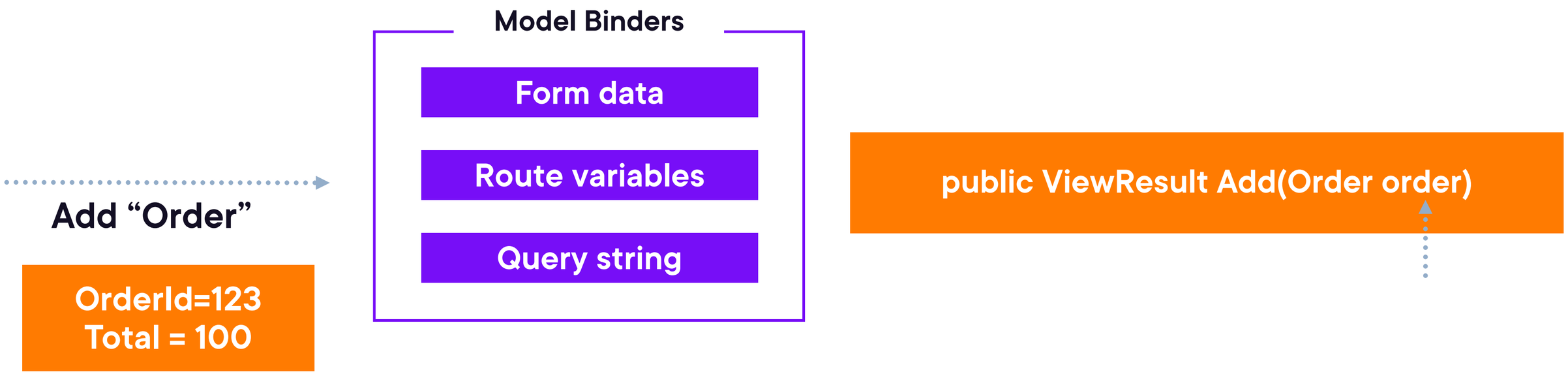
Passed to actions on controllers or pages

Simple and complex types

Understanding Model Binding



Binding to Complex Types



Adding a Create Page

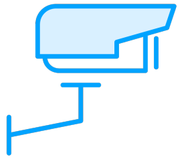
```
_bethanysPieShopDbContext.Categories.Add(category);

await _bethanysPieShopDbContext.SaveChangesAsync();
```

Creating a New Entity

EF Core will create the SQL to add the entity to the database

Understanding the EF Core Change Tracker

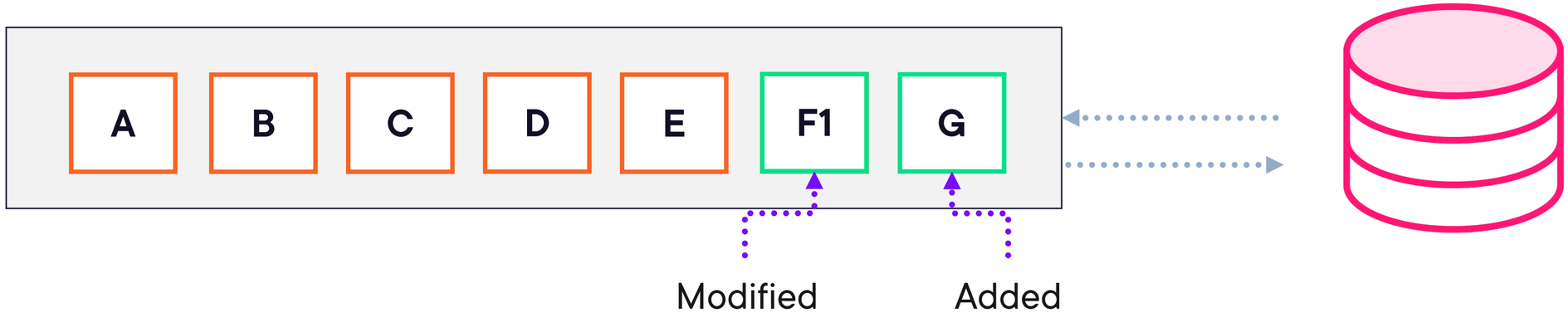


DbContext tracks items returned from query

Changes made to entities can be saved in one go to the database

Can introduce overhead for just reading data

How the Change Tracker Works



Available Entity States

Unchanged

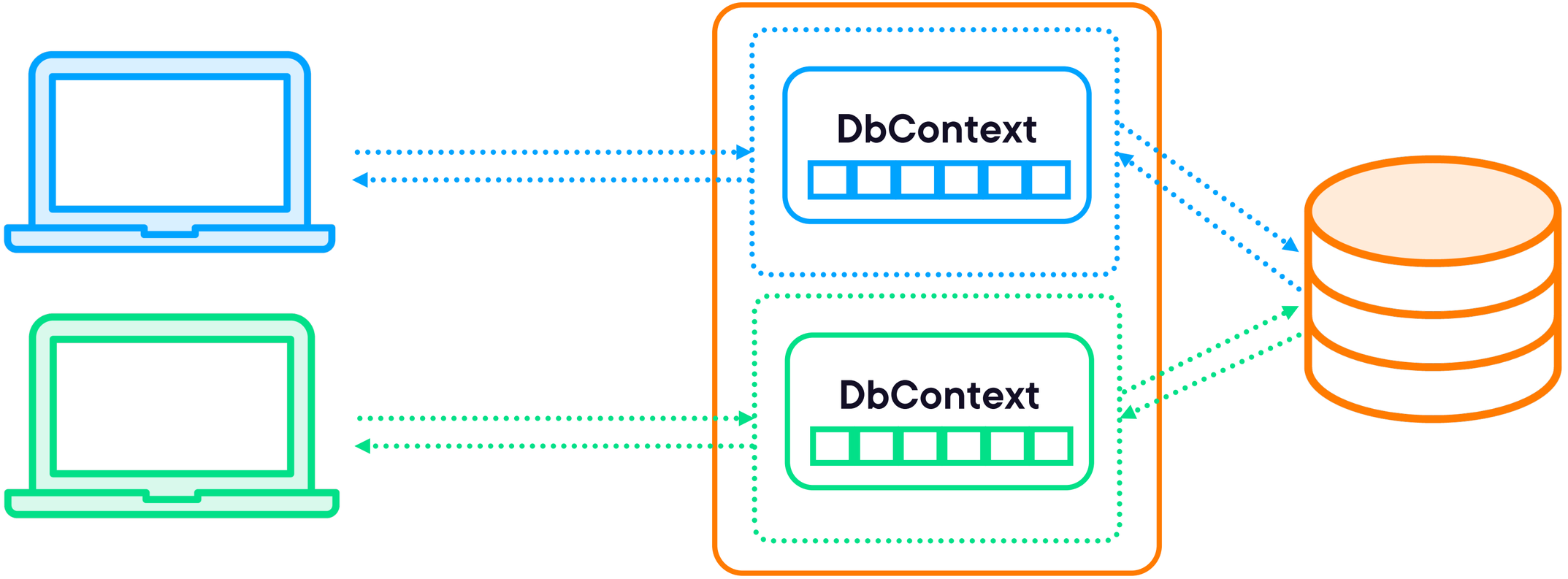
Added

Modified

Deleted

Detached

The Change Tracker and Dependency Injection



Using AsNoTracking()

```
_bethanysPieShopDbContext
        .Categories
        .AsNoTracking()
        .OrderBy(c => c.CategoryId)
        .ToListAsync();
```

Demo

Creating the Add Pie page

Adding AsNoTracking() on the repositories

Demo

Looking at the Add Category page

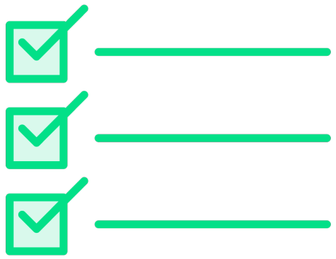
Understanding overposting

Adding Validation and ModelState

The Need for Validation

Add “Order”

OrderId=“ABC”
Total = “Hello world”



Validation

public ViewResult Add(Order order)

```
public class Pie
{
    public int PieId { get; set; }

    [Display(Name = "Name")]
    [Required]
    public string Name { get; set; } = string.Empty;

    [StringLength(100)]
    [Display(Name = "Short description")]
    public string? ShortDescription { get; set; }
}
```

Adding Validation Attributes

Built-in or custom

Validation Attributes

Required

StringLength

Range

RegularExpression

EmailAddress

Phone

```
try
{
     if (ModelState.IsValid)
     {
          await _categoryRepository.AddCategoryAsync(category);
          return RedirectToAction(nameof(Index));
     }
}
catch (Exception ex)
{
     ModelState.AddModelError("", $"Adding the category failed, please try again!");
}
```

Using ModelState

ModelState contains binding and validation errors

Demo

Adding validation

Using ModelState

Demo

Adding client-side validation

Handling Errors

Demo

Handling errors in the repository and UI

Summary

Tag helpers are used to create forms

ModelState contains information about the state of the posted model data

Validation attributes are an easy but limited way to bring in validation

Up Next:

Editing and Deleting Data