Creating the List Page

Overview

- Hello MVC
- Creating the model and the repository
- Creating the controller
- Adding the view
- Styling the view

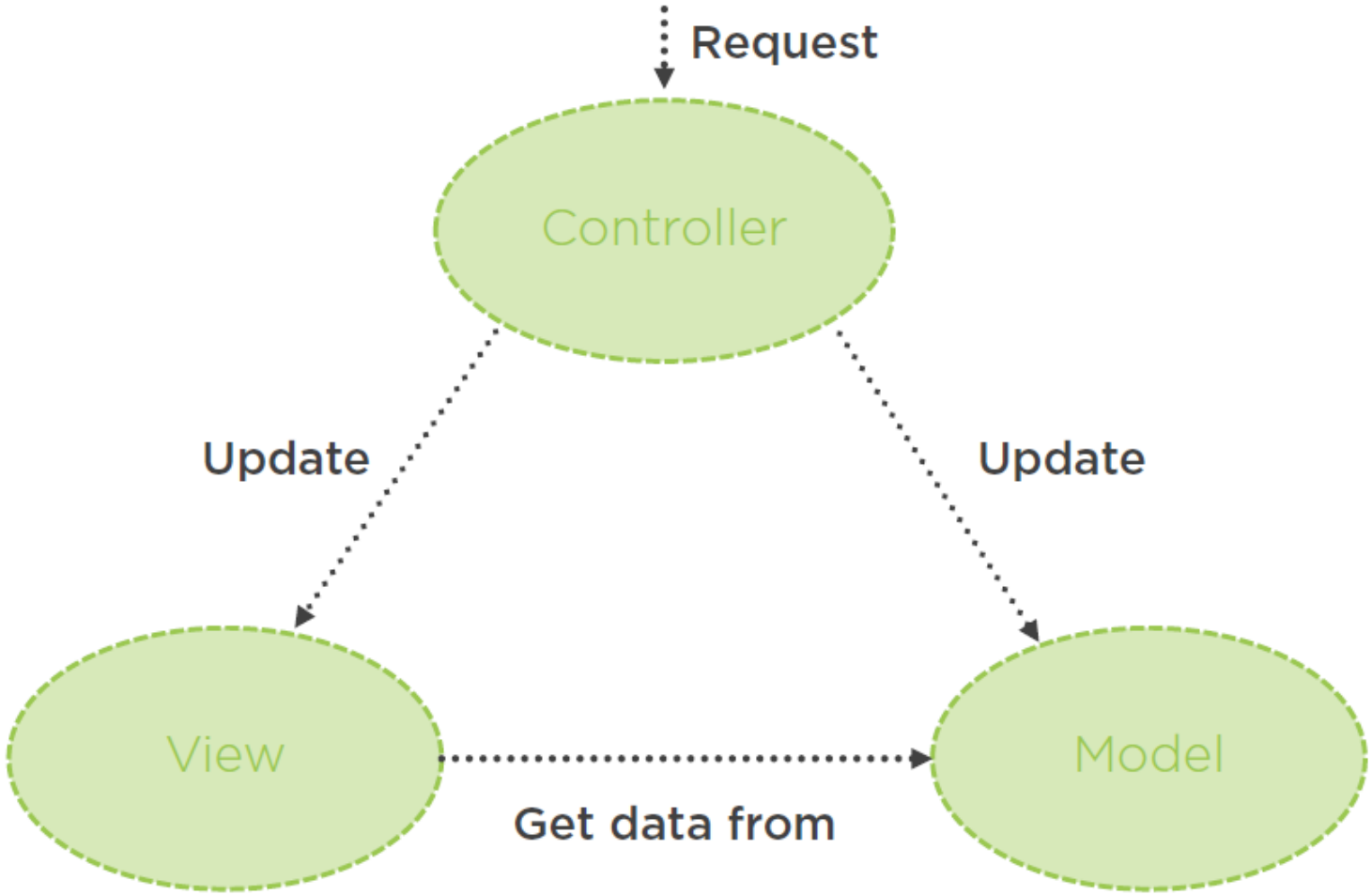
Hello MVC

The MVC in ASP.NET Core MVC

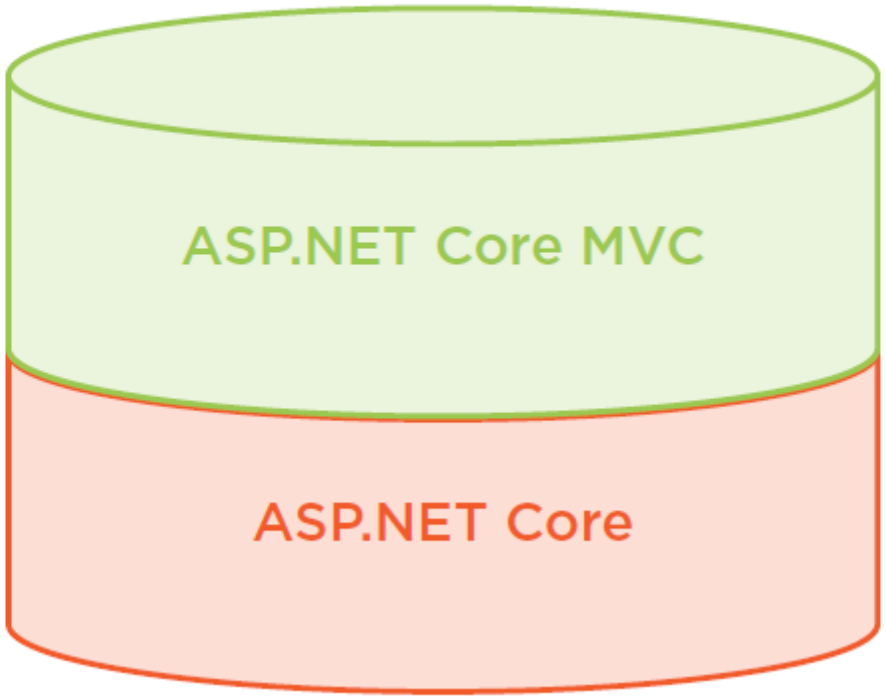
Model-View-Controller

- Architectural pattern
- Separation of concerns
- Promotes testability and maintainability

The MVC in ASP.NET Core MVC



The MVC in ASP.NET Core MVC



Creating the Model and the Repository

The Model

- Domain data + logic to manage data
- Simple API
- Hides details of managing the data

Sample Model Class

```
14 references
public class Pie
{
    5 references
    public int Id { get; set; }
    4 references
    public string Name { get; set; }
    4 references
    public string ShortDescription { get; set; }
    4 references
    public string LongDescription { get; set; }
    0 references
    public string AllergyInformation { get; set; }
    4 references
    public decimal Price { get; set; }
    4 references
    public string ImageUrl { get; set; }
    4 references
    public string ImageThumbnailUrl { get; set; }
    4 references
    public bool IsPieOfTheWeek { get; set; }
    4 references
    public bool InStock { get; set; }

    0 references
    public int CategoryId { get; set; }
    4 references
    public Category Category { get; set; }
}
```

The repository allows our code to use objects without knowing how they are persisted

Repository Interface

```
namespace PieShop.Models
{
    4 references
    public interface IPieRepository
    {
        2 references
        IEnumerable<Pie> AllPies { get; }
        0 references
        Pie GetPieByIdAsync(int pieId);
    }
}
```

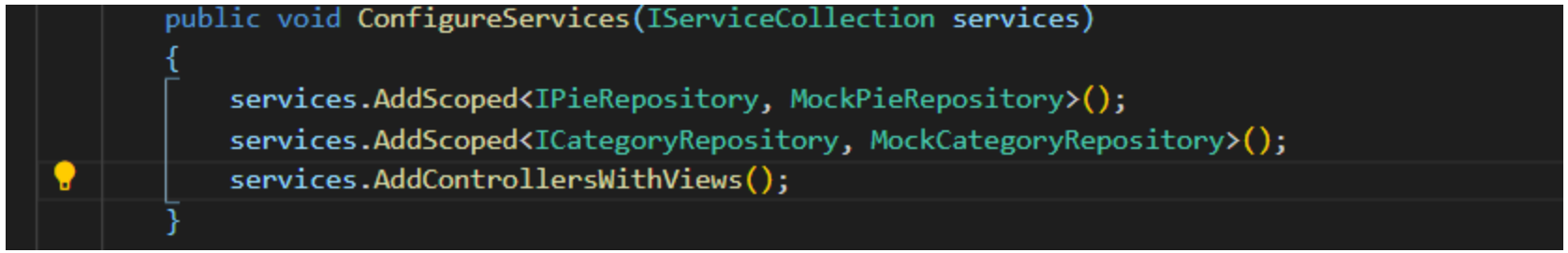
Mock Implementation

```
namespace PieShop.Models
{
    1 reference
    public class MockPieRepository : IPieRepository
    {
        4 references
        private readonly ICategoryRepository _categoryRepository = new MockCategoryRepository();
        2 references
        public IEnumerable<Pie> AllPies =>
            new List<Pie>
            {
                new Pie {Id = 1, Name="Strawberry Pie", Price=15.95M, ShortDescription="Lorem Ipsum"
                new Pie {Id = 2, Name="Cheese cake", Price=18.95M, ShortDescription="Lorem Ipsum",
                new Pie {Id = 3, Name="Rhubarb Pie", Price=15.95M, ShortDescription="Lorem Ipsum",
                new Pie {Id = 4, Name="Pumpkin Pie", Price=12.95M, ShortDescription="Lorem Ipsum",
            };

        0 references
        public IEnumerable<Pie> PiesOfTheWeek { get; }

        0 references
        public Pie GetPieByIdAsync(int pieId)
        {
            return AllPies.FirstOrDefault(p => p.Id == pieId);
        }
    }
```

Registering Services in ConfigureServices



Registration Options

AddTransient

AddSingleton

AddScoped

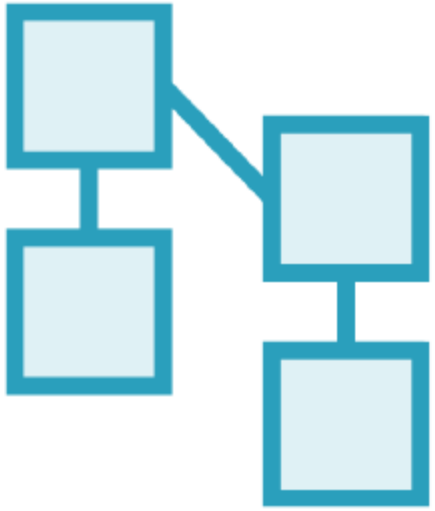
Demo

Creating the domain
Adding the repository
Registering with the services collection

Creating the Controller

Tasks of the Controller

Respond to user interaction



Update model

No knowledge about data persistence

A Simple Controller

```
public class PieController : Controller
{
        public ViewResult Index()
        {
                return View();
        }
}
```

Action

View to show

A Real Controller

```
public class PieController : Controller
{
        private readonly IPieRepository _pieRepository;
        public PieController(IPieRepository pieRepository)
        {
                _pieRepository = pieRepository;
        }
        public ViewResult List()
        {
                return View(_pieRepository.Pies);
        }
}
```

Demo

Adding the controller

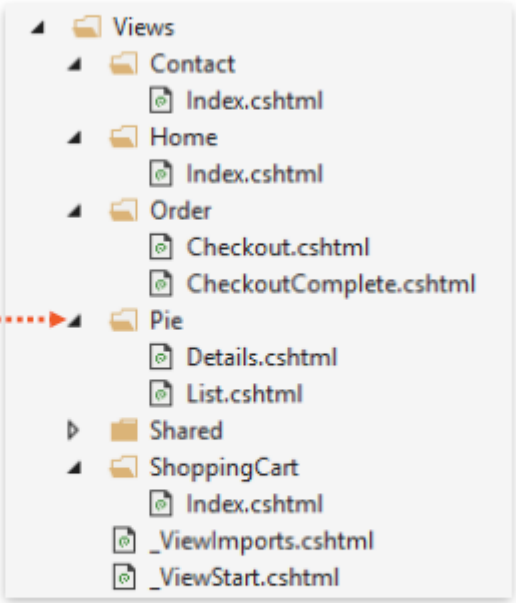
Adding the View

The Model

- HTML template
  - *.cshtml
- “Plain” or strongly-typed
- Uses Razor

Matching the Controller and its Views

PieController



Matching the Action With the View

```
public class PieController : Controller
{
    public ViewResult Index()
    {
        return View();
    }
}
```

Action

View to show: Index.cshtml

Using ViewBag from the Controller

```
public class PieController : Controller
{
        public ViewResult Index()
        {
            ViewBag.Message = "Welcome to Bethany's Pie Shop";
            return View();
        }
}
```

Dynamic Content Using ViewBag

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <div>
      @ViewBag.Message
    </div>
  </body>
</html>
```

Razor is a markup syntax which allows us to include C# functionality in our web pages

Calling a Strongly-typed View

```
public class PieController : Controller
{
    public ViewResult List()
    {
        return View(_pieRepository.Pies);
    }

}
```

A Strongly-typed View

```
@model IEnumerable<Pie>
<html>
 <body>
  <div>
   @foreach (var pie in Model.Pies)
   {
    <div>
     <h2>@pie.Name</h2>
     <h3>@pie.Price.ToString("c")</h3>
     <h4>@pie.Category.CategoryName</h4>
    </div>
   }
  </div>
 </body>
</html>
```

View Model

```
public class PiesListViewModel
{
    public IEnumerable<Pie> Pies { get; set; }
    public string CurrentCategory { get; set; }
}
```

_Layout.cshtml

Template

Shared folder

More than one can be created

_Layout.cshtml

```
<!DOCTYPE html>
<html>
    <head>
        <title>Bethany's Pie Shop</title>
    </head>
    <body>
        <div>
            @RenderBody()
        </div>
    </body>
</html>
```

Replaced with view

_ViewStart.cshtml

```
@{
    Layout = "_Layout";
}
```

_ViewStart.cshtml

```
@{
    Layout = "_Layout";
}
```

_ViewImports.cshtml

```
@using PieShop.Models
@using PieShop.ViewModels
```

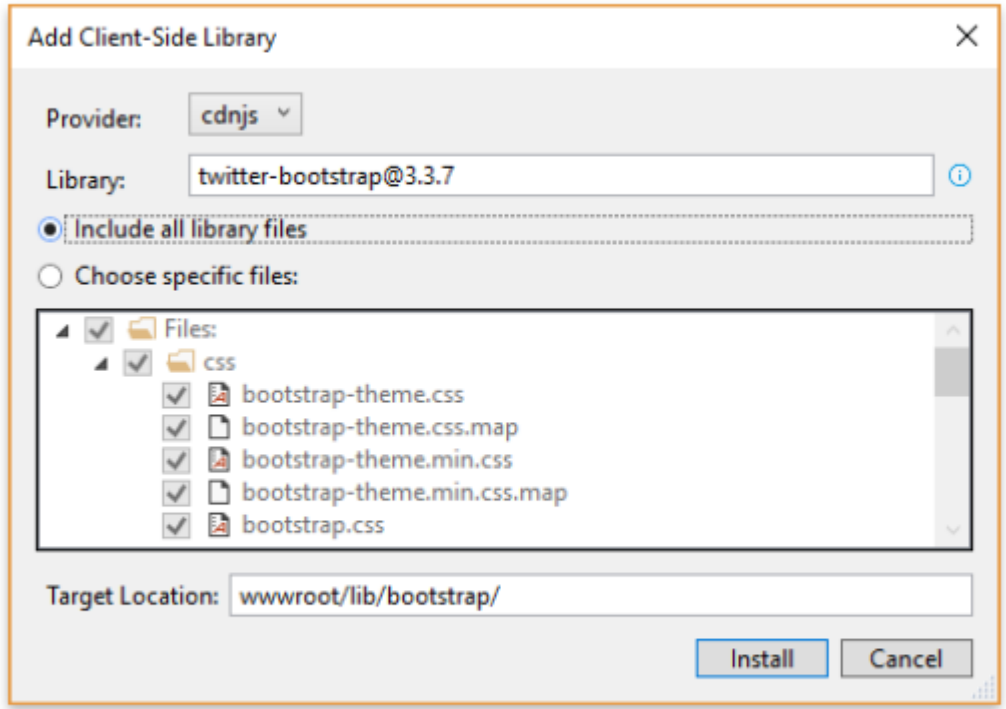
Demo

Creating the first view
Using a View Model

Styling the View

Using Library Manager (LibMan)

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "twitter-bootstrap@3.3.7",
      "destination": "wwwroot/lib/bootstrap/"
    }
  ]
}
```



Demo

Adding client-side packages using Library Manager

Add styles