Creating the List Page

Gill Cleeren
CTO XPIRIT BELGIUM

@gillcleeren www.snowball.be

Overview

Hello MVC

Creating the model and the repository

Creating the controller

Adding the view

Styling the view

Demo

Looking at the final page

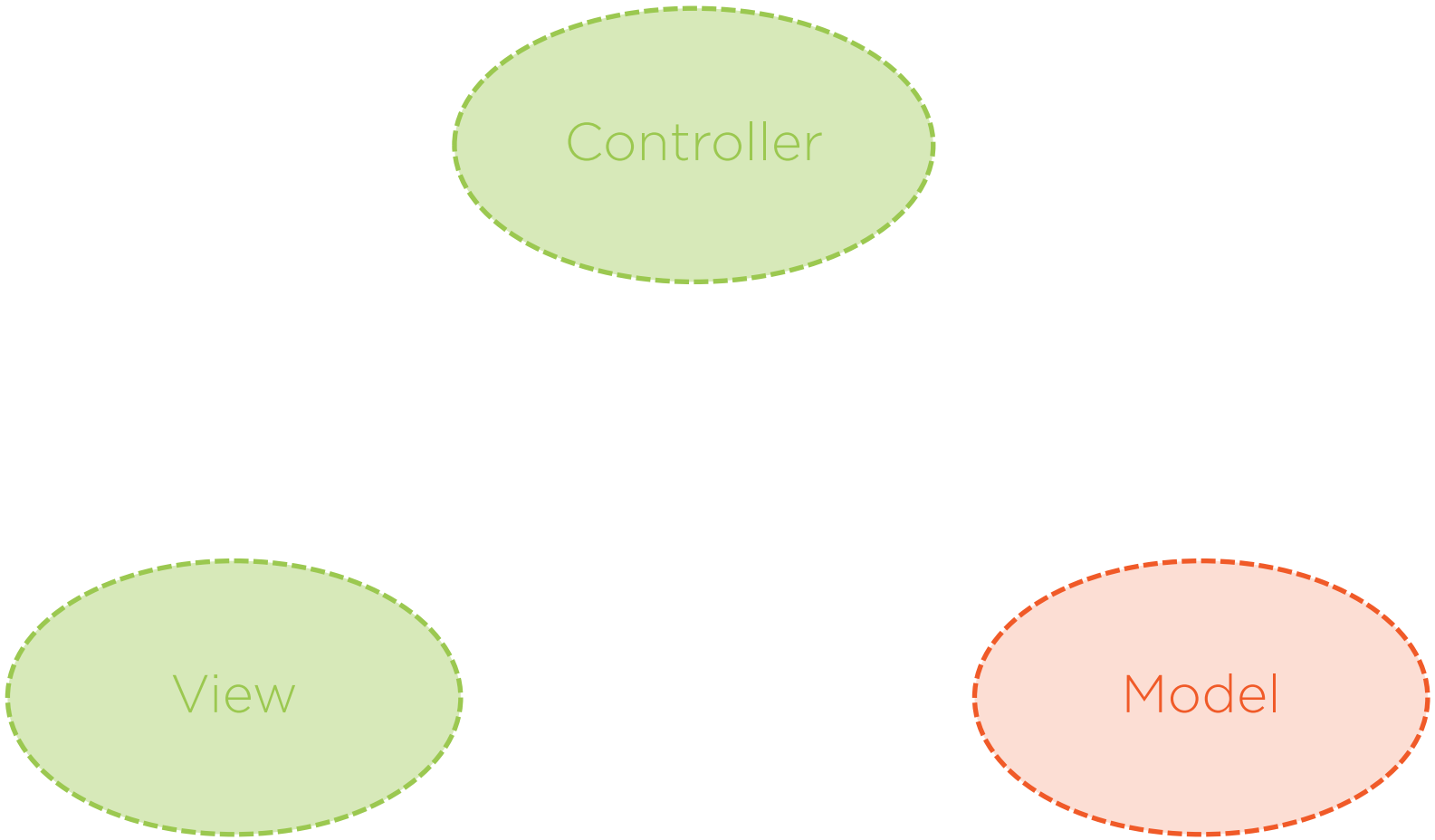
Hello MVC

The MVC in ASP.NET Core MVC

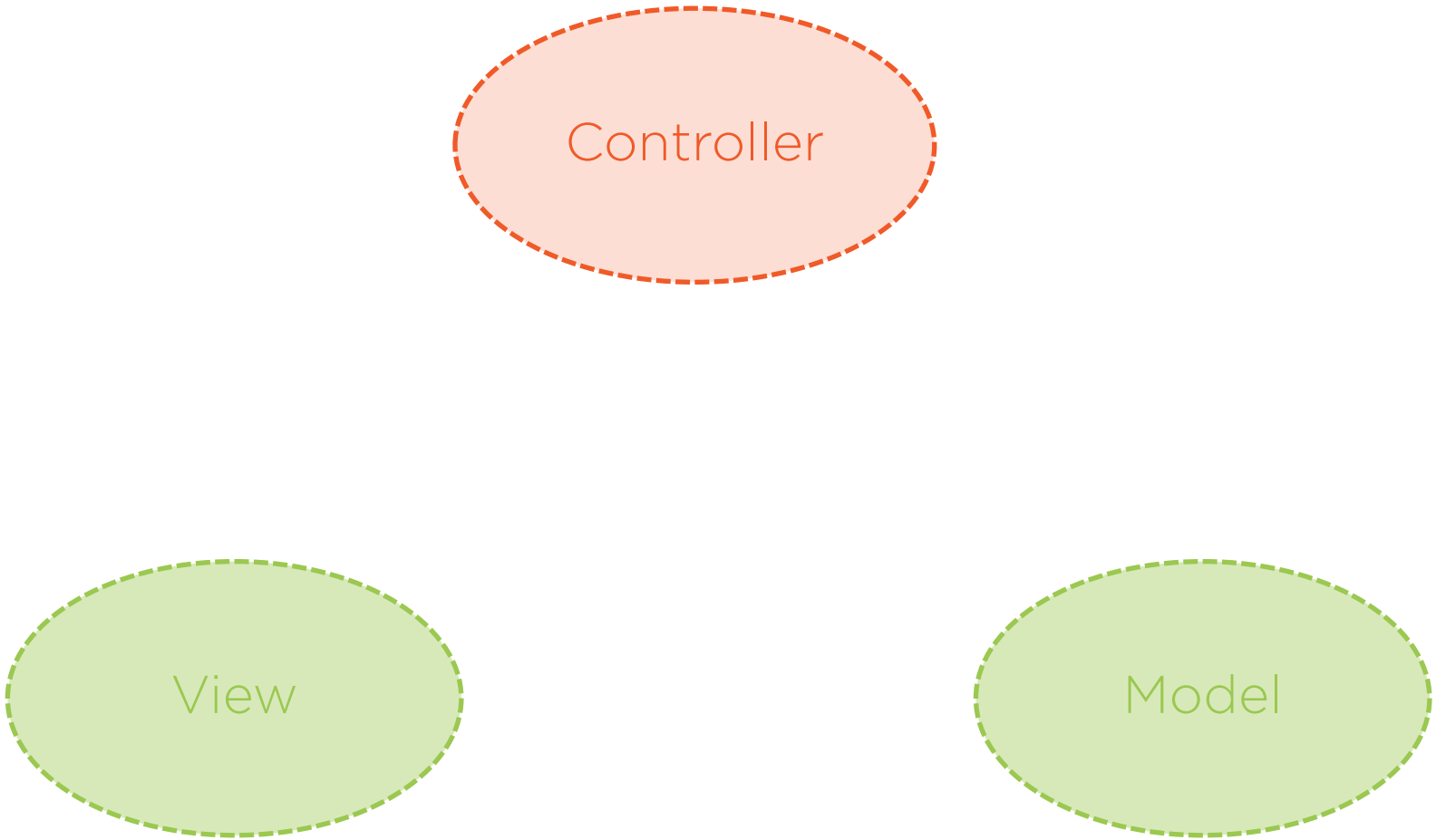
Model-View-Controller

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

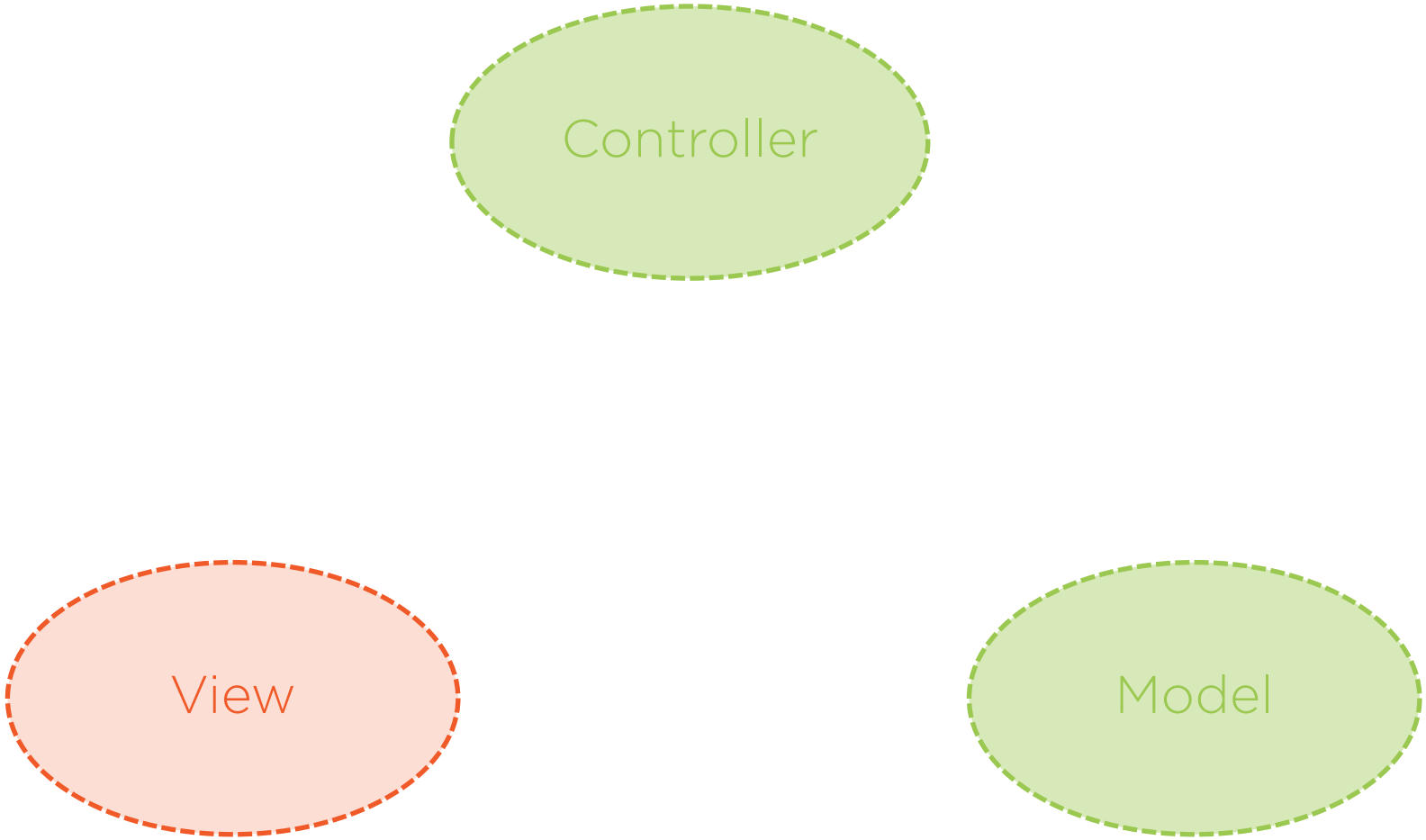
The MVC in ASP.NET Core MVC



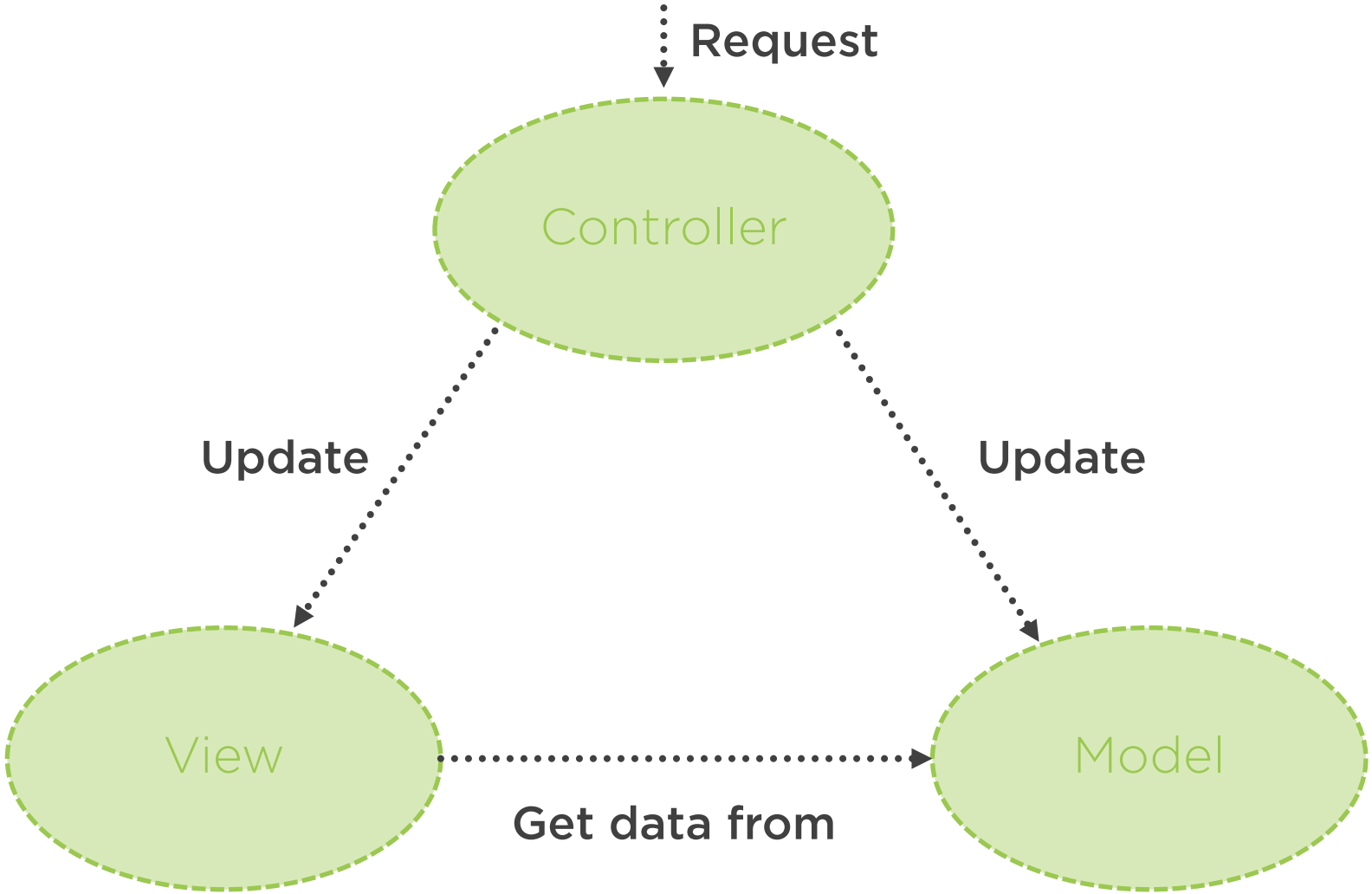
The MVC in ASP.NET Core MVC



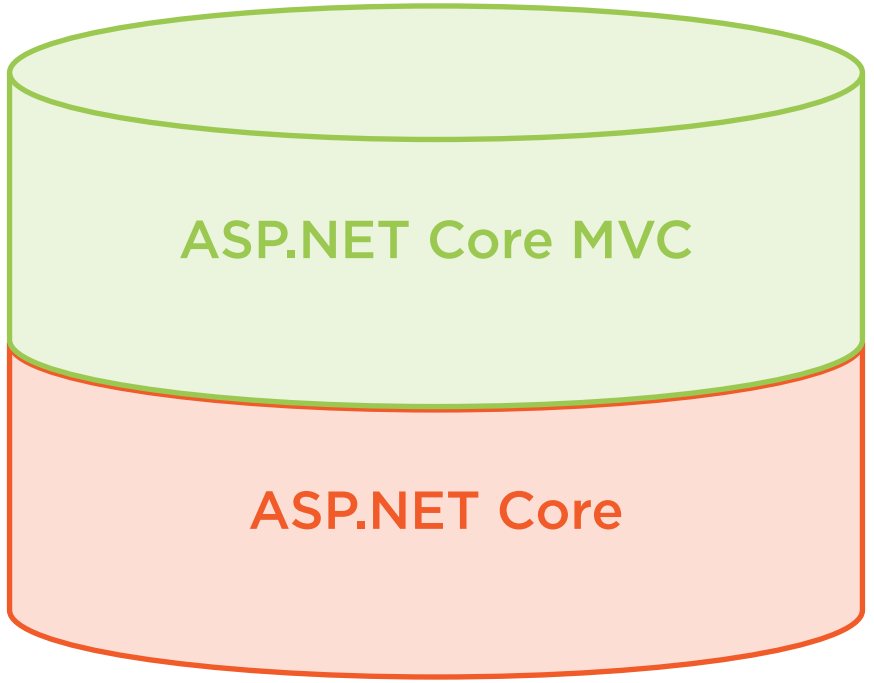
The MVC in ASP.NET Core MVC



The MVC in ASP.NET Core MVC

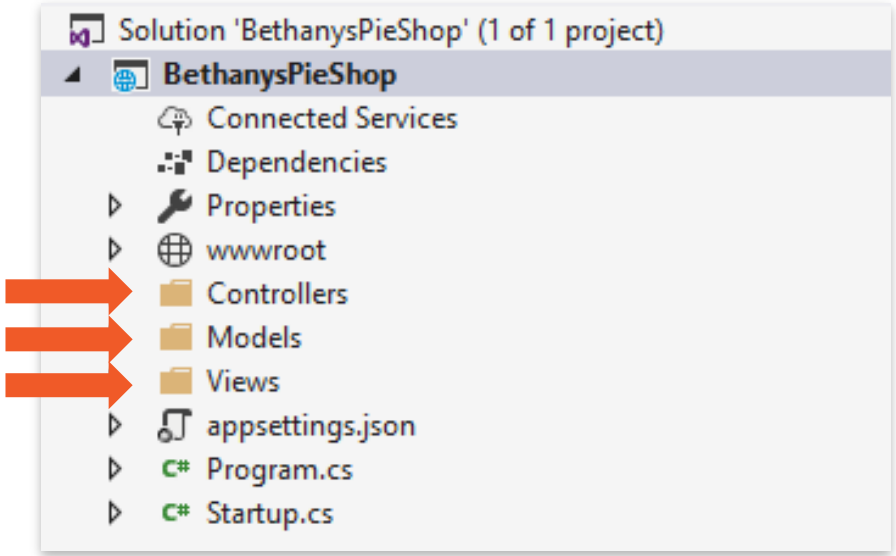


The MVC in ASP.NET Core MVC



Creating the Model and the Repository

Using the Correct Folders



The Model

Domain data + logic to manage data

Simple API

Hides details of managing the data

Sample Model class

```
public class Pie
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string ShortDescription { get; set; }
    public string LongDescription { get; set; }
    public decimal Price { get; set; }
    public string ImageUrl { get; set; }
    public string ImageThumbnailUrl { get; set; }
    public bool IsPieOfTheWeek { get; set; }
    public Category Category { get; set; }
}
```

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

```
public interface IPieRepository
{
    IEnumerable<Pie> GetAllPies();
    Pie GetPieById(int pieId);

}
```

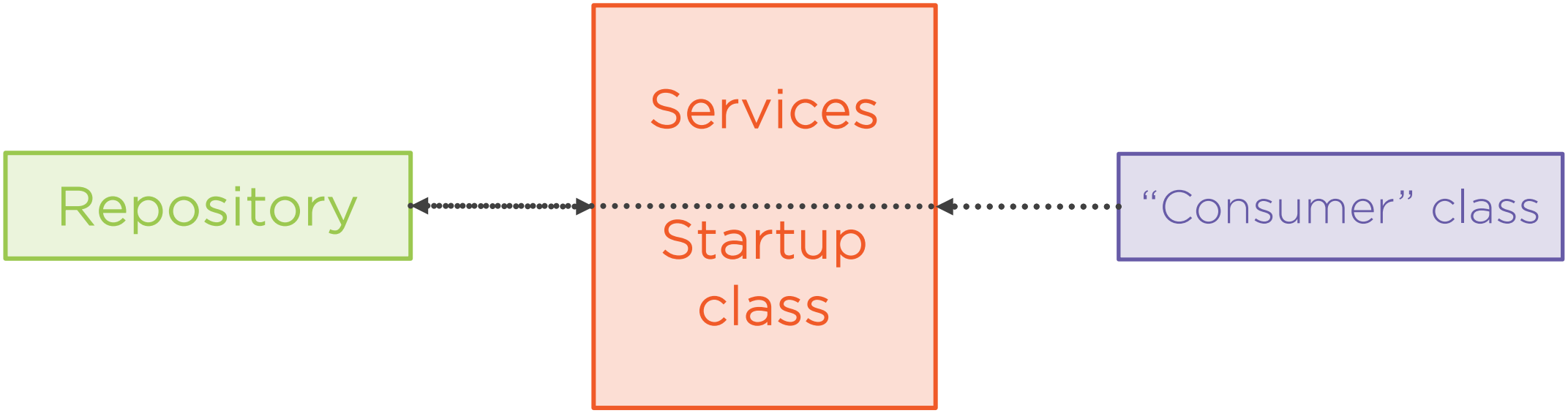
Pie Repository Interface

Mock Implementation

```
public class MockPieRepository : IPieRepository
{
    public IEnumerable<Pie> Pies
    {
        get { ... }
    }

    public Pie GetPieById(int pieId)
    { ... }
}
```

Registering the Repository



```
public void ConfigureServices(IServiceCollection services)
{
    //register framework services
    services.AddMvc();

    //register our own services
    services.AddScoped<IPieRepository, MockPieRepository>();
}
```

Registering Services in ConfigureServices

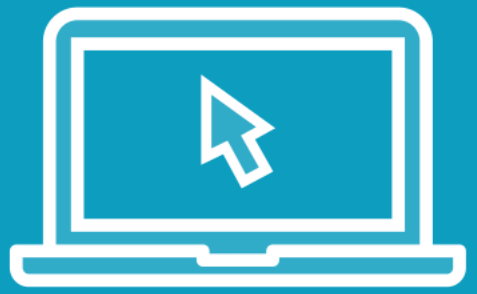
Registration options

AddTransient

AddSingleton

AddScoped

Demo



Creating the domain

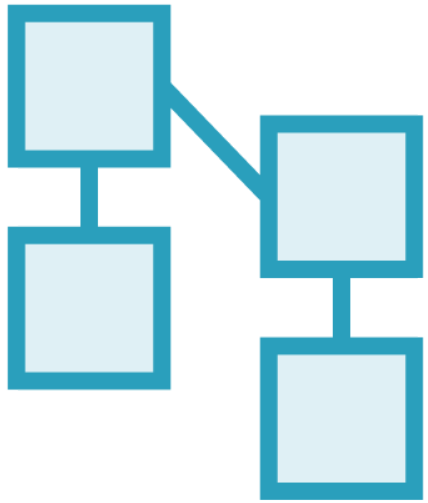
Adding the repository

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 View

HTML template

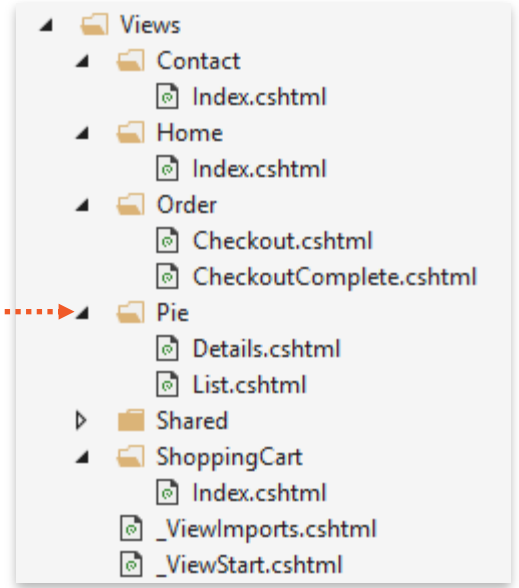
- *.cshtml

“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

Regular View

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <div>
      Welcome to Bethany’s Pie Shop
    </div>
  </body>
</html>
```

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>
```

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

View Model

Demo

Creating the first view

Using a View Model

_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

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

_ViewStart.cshtml

```
@using BethanysPieShop.Models
```

View Imports

Demo

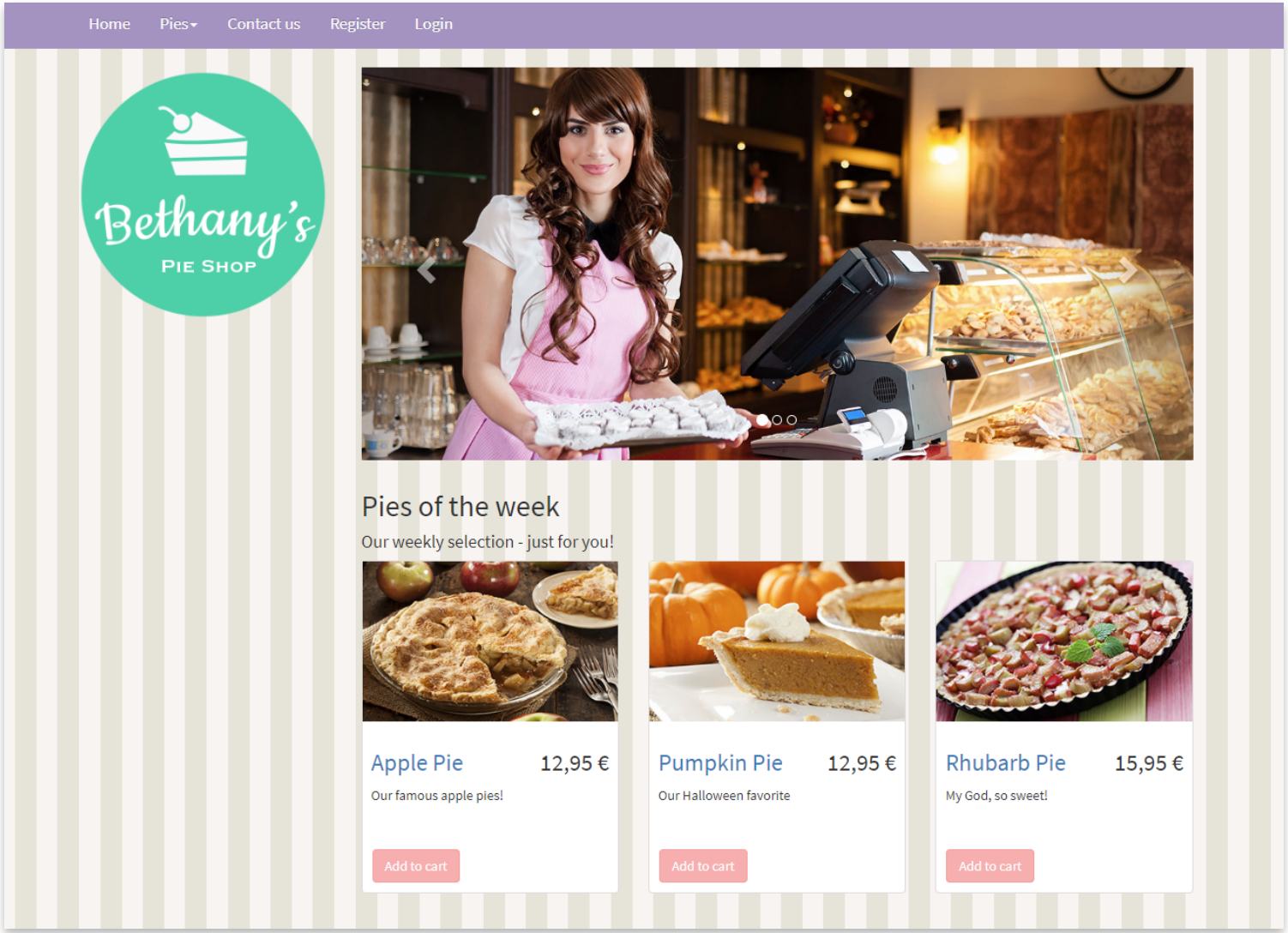
Adding a layout template

Creating the ViewStart file

Adding the ViewImports file

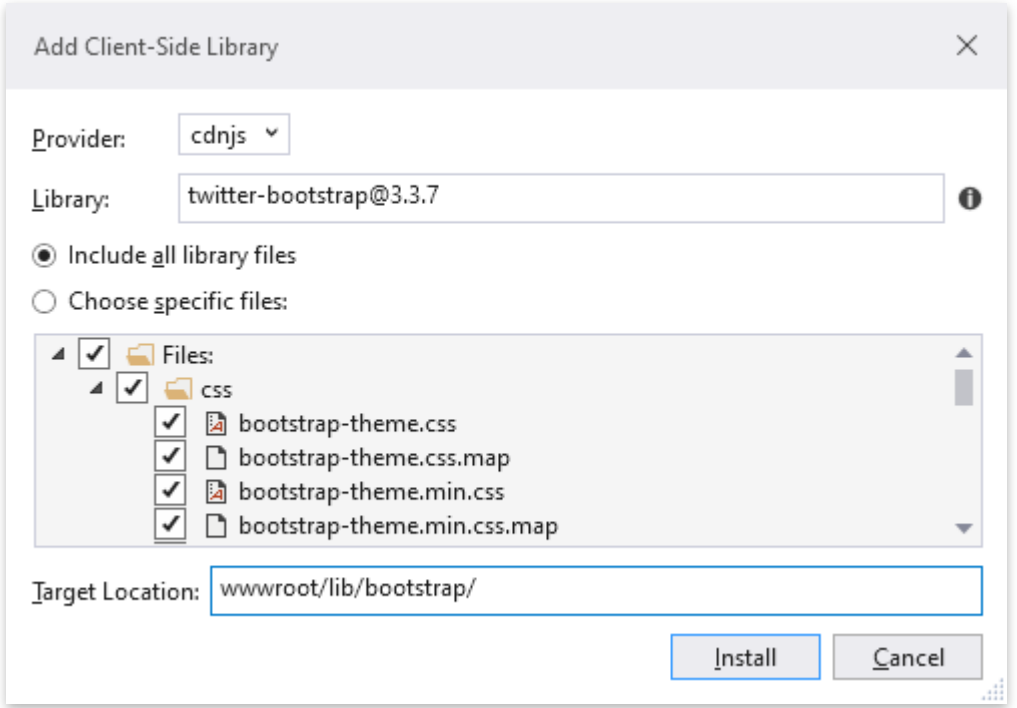
Styling the View

Where We Need to Get



Using Library Manager (LibMan)

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



Older versions of Visual Studio: beware!

- Bower.json not supported anymore
- Adding packages manually

Demo

Adding client-side packages using Library Manager

Add styles

Summary

Building a complete page

- M
- V
- C

Client-side packages added using Library Manager

Up next:
Adding Entity Framework Core