# Creating the Overview Page

**Gill Cleeren**
ARCHITECT

@gillcleeren www.snowball.be

## Overview

The MVC pattern

Creating the model, repository and controller

Creating the view

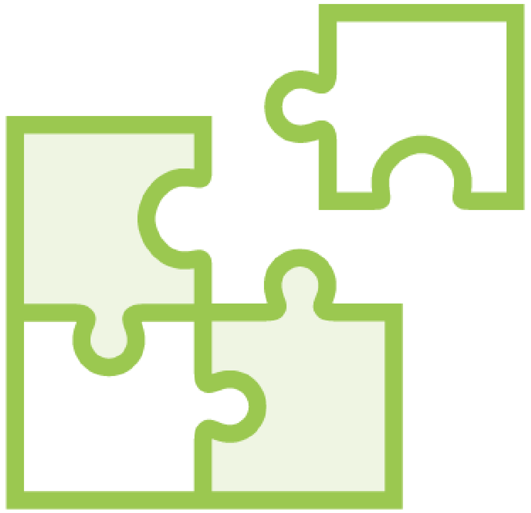
Adding styles

## Demo

## The overview page

## The MVC Pattern

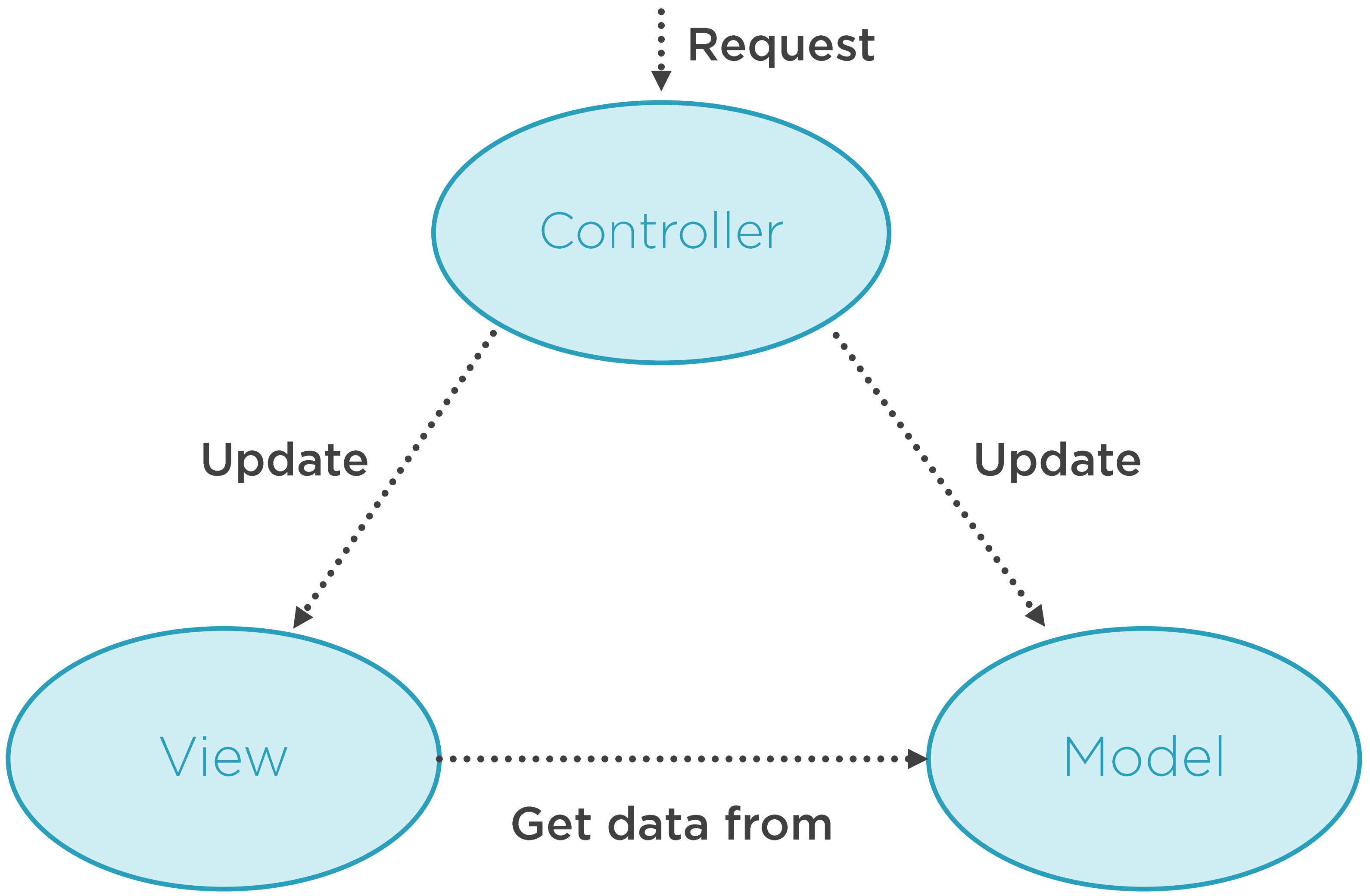
# The MVC in ASP.NET Core MVC



**Model-View-Controller**

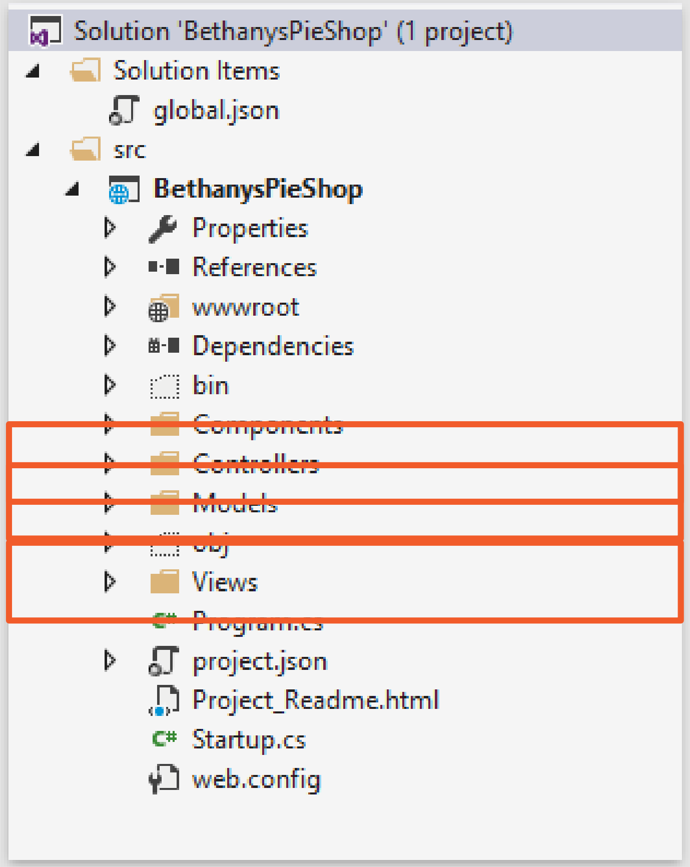
- Architectural pattern
- Separation of concerns
- Promotes testability and maintainability

# The MVC in ASP.NET Core MVC



## Creating the model, repository and controller

## Creating the Correct Folders



Solution 'BethanysPieShop' (1 project)
Solution Items
global.json
src
BethanysPieShop
Properties
References
wwwroot
Dependencies
bin
Components
Controllers
Models
obj
Views
Program.cs
project.json
Project_Readme.html
Startup.cs
web.config

## The Model in MVC

Domain data

Logic for managing the data

Simple API

Hide details of data management

```
public class Pie
{
    public int PieId { get; set; }
    public string Name { get; set; }
    public string ShortDescription { get; set; }
    public decimal Price { get; set; }
    public int CategoryId { get; set; }
    public virtual Category Category { get; set; }
}
```

## Sample Model class

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

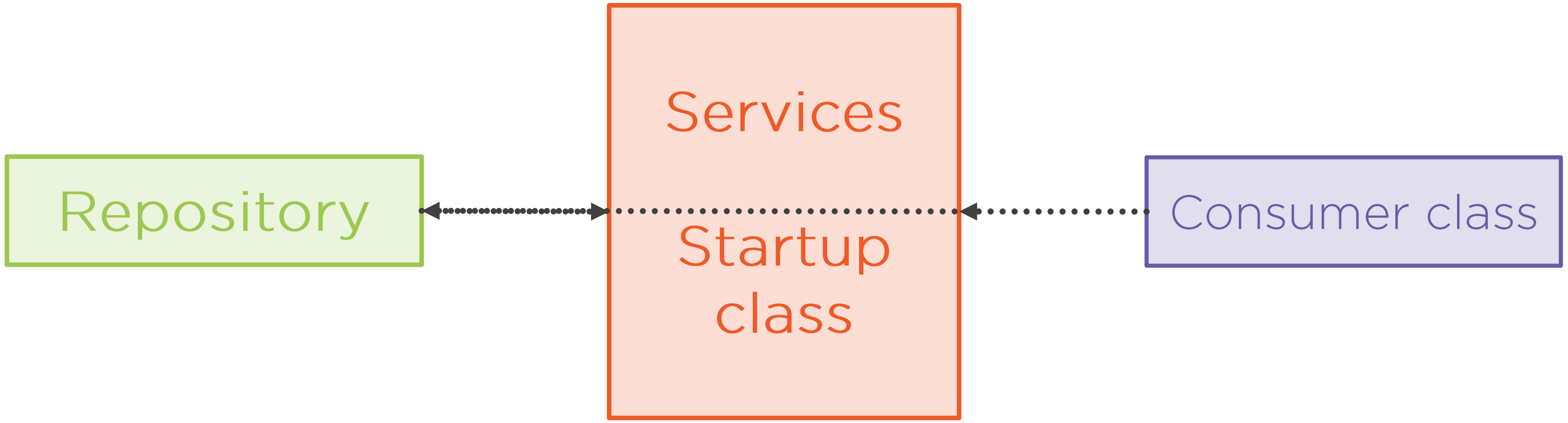
```
public interface IPieRepository
{
    IEnumerable<Pie> Pies { get; }
    IEnumerable<Pie> PiesOfTheWeek { get; }

    Pie GetPieById(int pieId);
}
```

## Adding the Repository Interface

## Mock Implementation

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

    public IEnumerable<Pie> PiesOfTheWeek
    {
        get
        { ... }
    }

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

# Registering the Repository



```
public void ConfigureServices(IServiceCollection services)
{
    services.AddTransient<IPieRepository, MockPieRepository>();
    services.AddMvc();
}
```

## Registering the Repository

## Registration options

AddTransient

AddSingleton

AddScoped

# The Controller

Respond to user interaction

Update model

No knowledge about data persistence

## A Basic Controller

```
public class PieController : Controller
{
    public ViewResult Index()          ◄·················· Action
    {
        return View();                 ◄·················· View to show
    }
}
```

# A Second Controller

```
public class PieController : Controller
{
    private readonly IPieRepository _pieRepository;

    public PieController(IPieRepository pieRepository)
    {
        _pieRepository = pieRepository;
    }

    public ViewResult List()
    {
        return View(_pieRepository.Pies);
    }
}
```

## Demo

Creating the domain classes

Creating the repository

DI registration

Adding the PieController

## Creating the View

## Types of Views

“Regular” View

Strongly-typed view

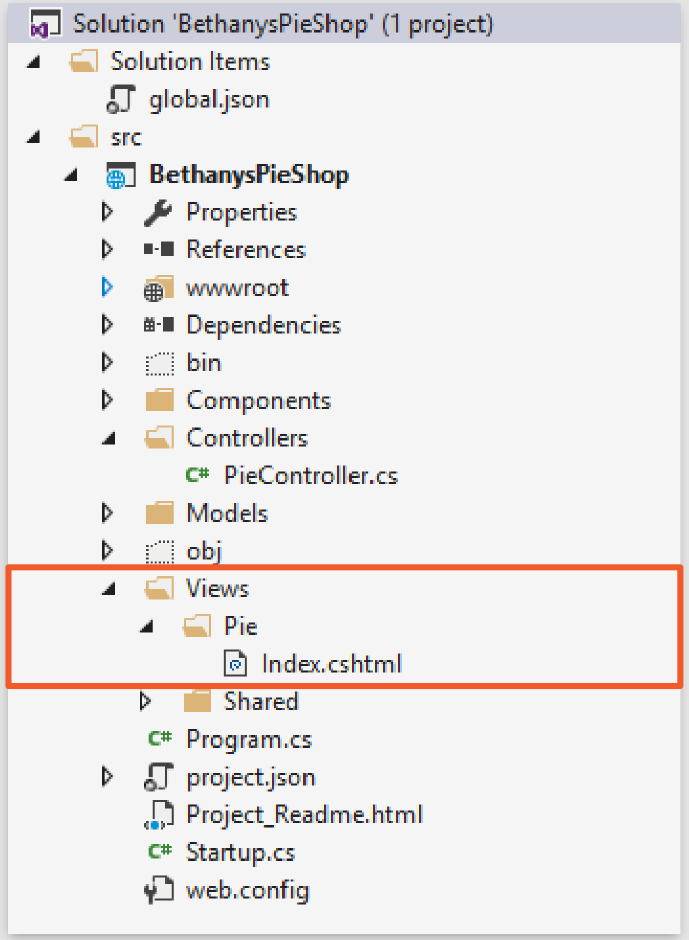
## Regular View

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

## Matching the Action With the View

```
public class PieController : Controller
{
    public ViewResult Index()        ◄··········  Action
    {
        return View();               ◄··········  View to show
    }
}
```

# View Folder Structure



```
Solution 'BethanysPieShop' (1 project)
◢ Solution Items
    global.json
◢ src
  ◢ BethanysPieShop
    ▷ Properties
    ▷ References
    ▷ wwwroot
    ▷ Dependencies
    ▷ bin
    ▷ Components
    ◢ Controllers
        PieController.cs
    ▷ Models
    ▷ obj
    ◢ Views
      ◢ Pie
          Index.cshtml
      ▷ Shared
      Program.cs
    ▷ project.json
      Project_Readme.html
      Startup.cs
      web.config
```

## 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

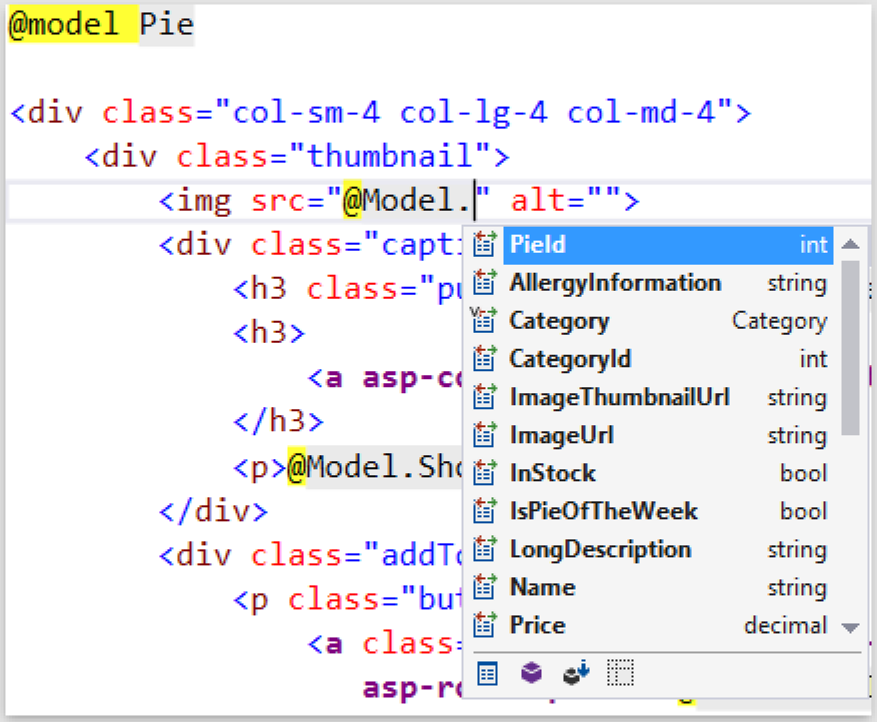
## Calling a Strongly-typed View

```
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>
```

# IntelliSense



```
@model Pie

<div class="col-sm-4 col-lg-4 col-md-4">
    <div class="thumbnail">
        <img src="@Model." alt="">
        <div class="capt
            <h3 class="pu
            <h3>
                <a asp-co
            </h3>
            <p>@Model.Sho
        </div>
        <div class="addT
            <p class="but
                <a class=
                    asp-ro
```

| | |
|---|---|
| PieId | int |
| AllergyInformation | string |
| Category | Category |
| CategoryId | int |
| ImageThumbnailUrl | string |
| ImageUrl | string |
| InStock | bool |
| IsPieOfTheWeek | bool |
| LongDescription | string |
| Name | string |
| Price | decimal |

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

## View Model

## Demo

Creating our first view

## _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()   ◀··················· Replaced with view
        </div>
    </body>
</html>
```

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

## _ViewStart.cshtml

```
@using BethanysPieShop.Models
```

## View Imports

Demo

Adding a Layout, ViewStart and ViewImports file

Updating the existing list view

## Adding Styles

## Client-side Package Management with Bower

Client-side package manager

bower.json

wwwroot

## Demo

Adding a bower.json file

Adding Bootstrap

## Summary

**MVC pattern**

- Model
- Controller
- View

**Specific view files**

**Bower for client-side package management**