

# Creating the List Page

---



**Gill Cleeren**

@gillcleeren [www.snowball.be](http://www.snowball.be)



# 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

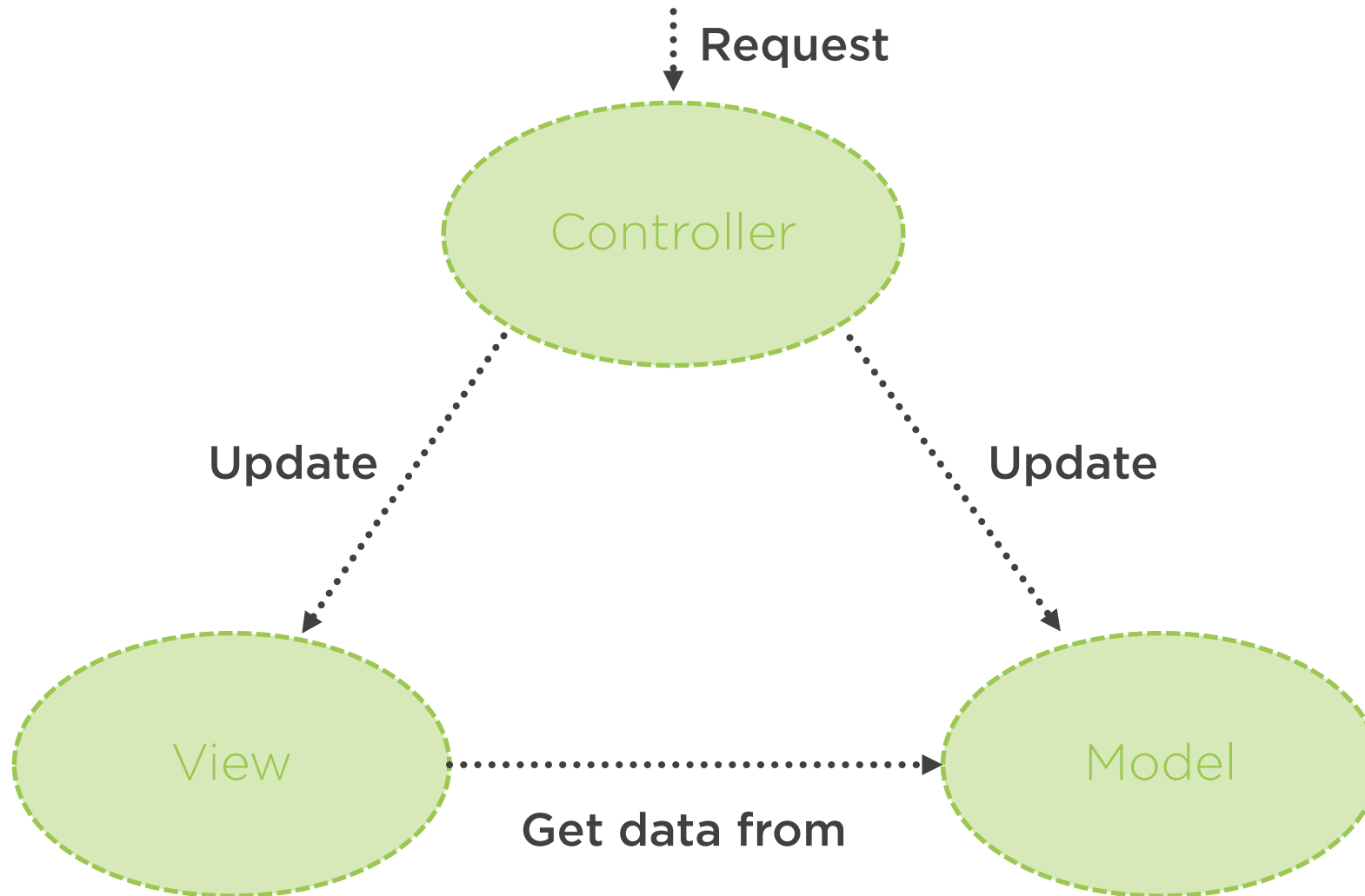


## Model-View-Controller

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



# The MVC in ASP.NET Core MVC



# 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

```
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; }
}
```







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

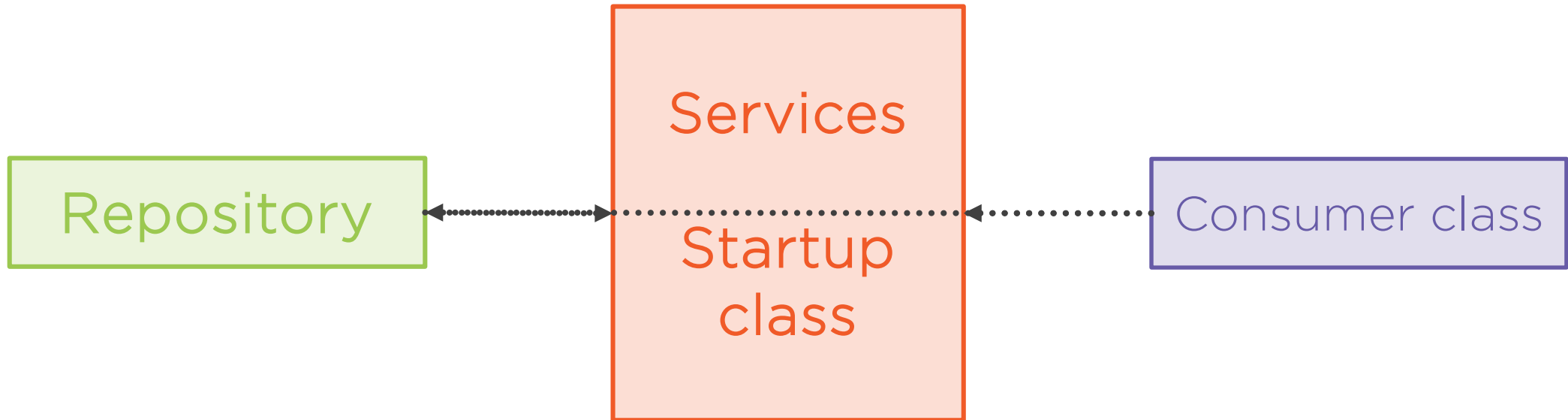


```
public interface IPieRepository
{
    IEnumerable<Pie> GetAllPies();
    Pie GetPieById(int pieId);
}
```

## Pie Repository Interface



# Registering the Repository



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

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

## Registering Services in ConfigureServices



# Demo



Creating the domain

Adding the repository



# Creating the Controller

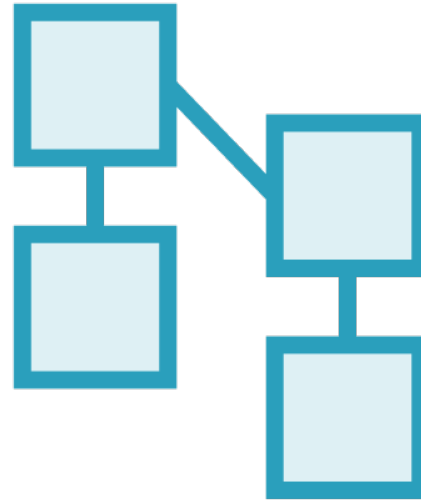
---



# 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()  Action
    {
        return View();  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**

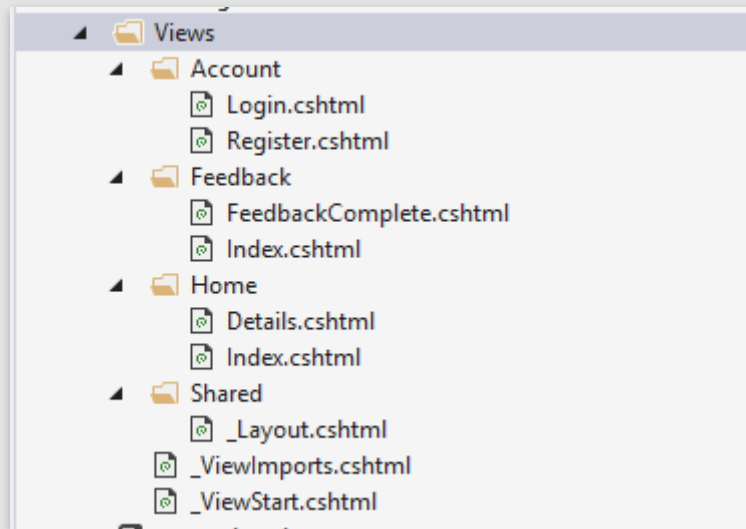
- \*.cshtml

**“Plain” or strongly-typed**

**Uses Razor**



# View Folder Structure



# Matching the Action With the View

```
public class PieController : Controller
{
    public ActionResult Index() ←..... Action
    {
        return View();           ←..... View to show: Index.cshtml
    }
}
```



# Using ViewBag from the Controller

```
public class PieController : Controller
{
    public ActionResult 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>
```





# 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>
```



\_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



# Demo



Creating the view

Adding a layout template

Creating the ViewStart file



# Styling the View


---




# Where We Need to Get

[Bethany's Pie Shop](#) [Feedback](#) [Register](#) [Log in](#)


## Welcome to Bethany's Pie Shop




**Apple Pie** £12.95  
Our famous apple pies!




**Blueberry Cheese Cake** £18.95  
You'll love it!




**Cheese Cake** £18.95  
Plain cheese cake. Plain pleasure.






**Cherry Pie** £15.95  
A summer classic!



**Christmas Apple Pie** £13.95  
Happy holidays with this pie!



**Cranberry Pie** £17.95  
A Christmas favorite





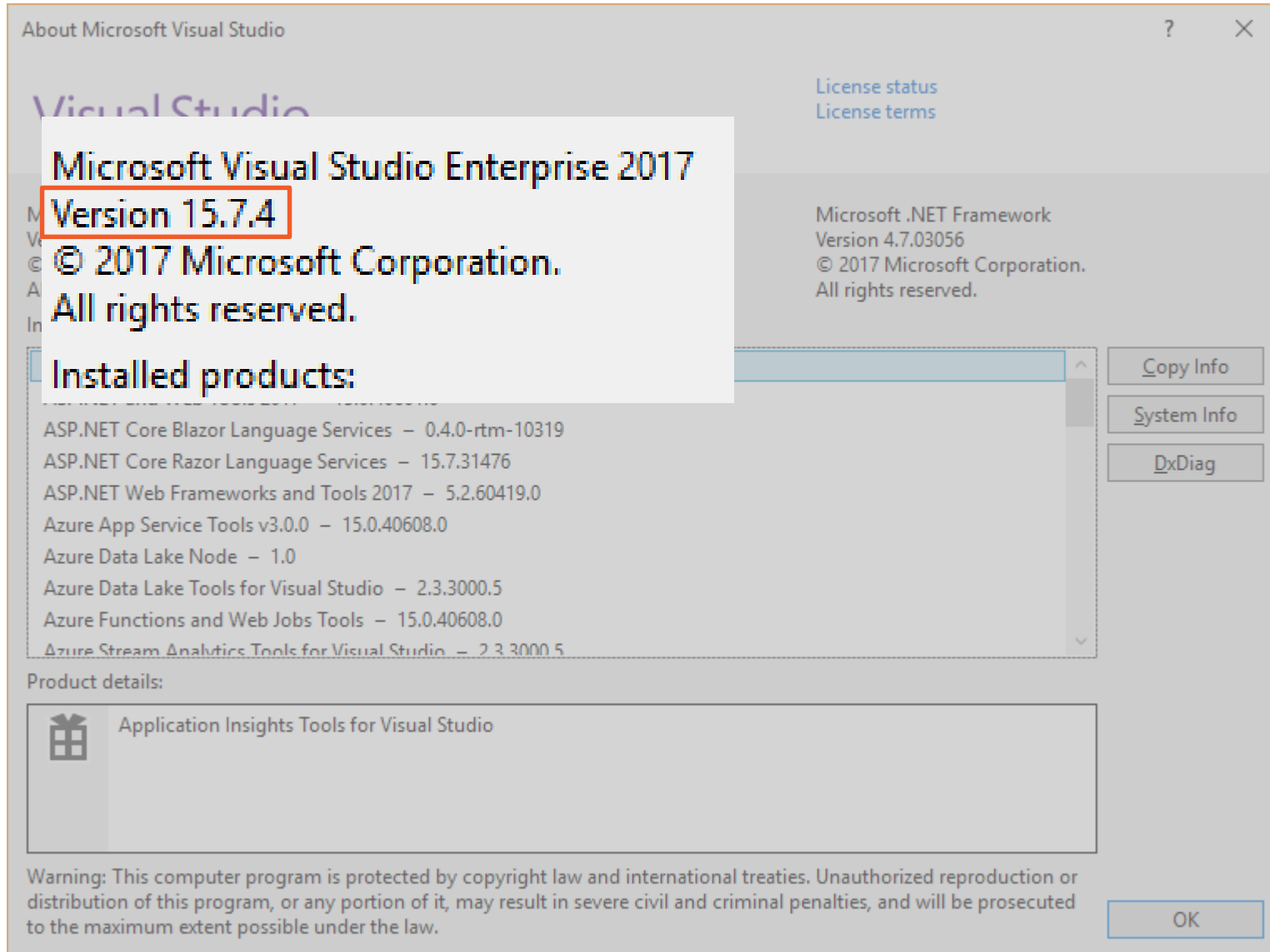




## Current state of client package managers in Visual Studio 2017

- Bower (up until 15.5)
- Manual process (15.6 – 15.7)
- Library Manager (15.8 or higher)

# Checking the Visual Studio Version



=< 15.5: first demo

15.5 → 15.7: second demo

>= 15.8: third demo



# Demo



Using Bower from Visual Studio  
Adding Bootstrap



# Demo

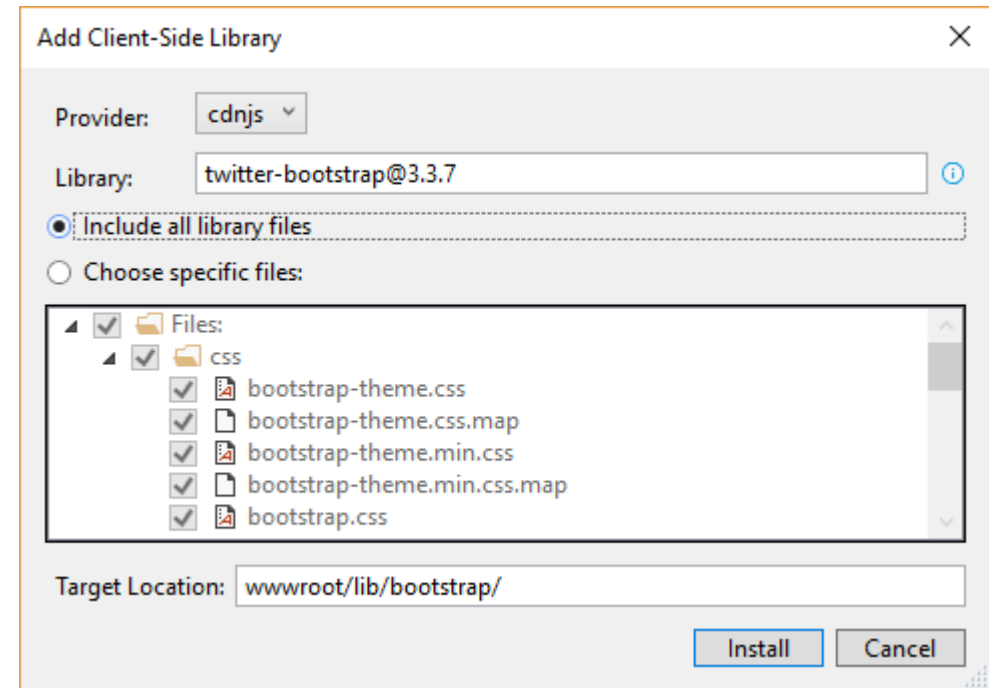


## Adding Bootstrap



# Using Library Manager (LibMan)

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



# Demo



## Using Library Manager



# Summary



## Building a complete page

- M
- V
- C

## Styling using Bower





**Up next:**  
Adding Entity Framework Core

