

# Creating the First Page

---



**Gill Cleeren**

CTO Xpirit Belgium

@gillcleeren – [xpirit.com/gill](http://xpirit.com/gill)



# Module overview



**Introducing the MVC pattern**

**Creating the model and the repository**

**Creating the controller**

**Adding the view**

**Styling the view**



# Demo



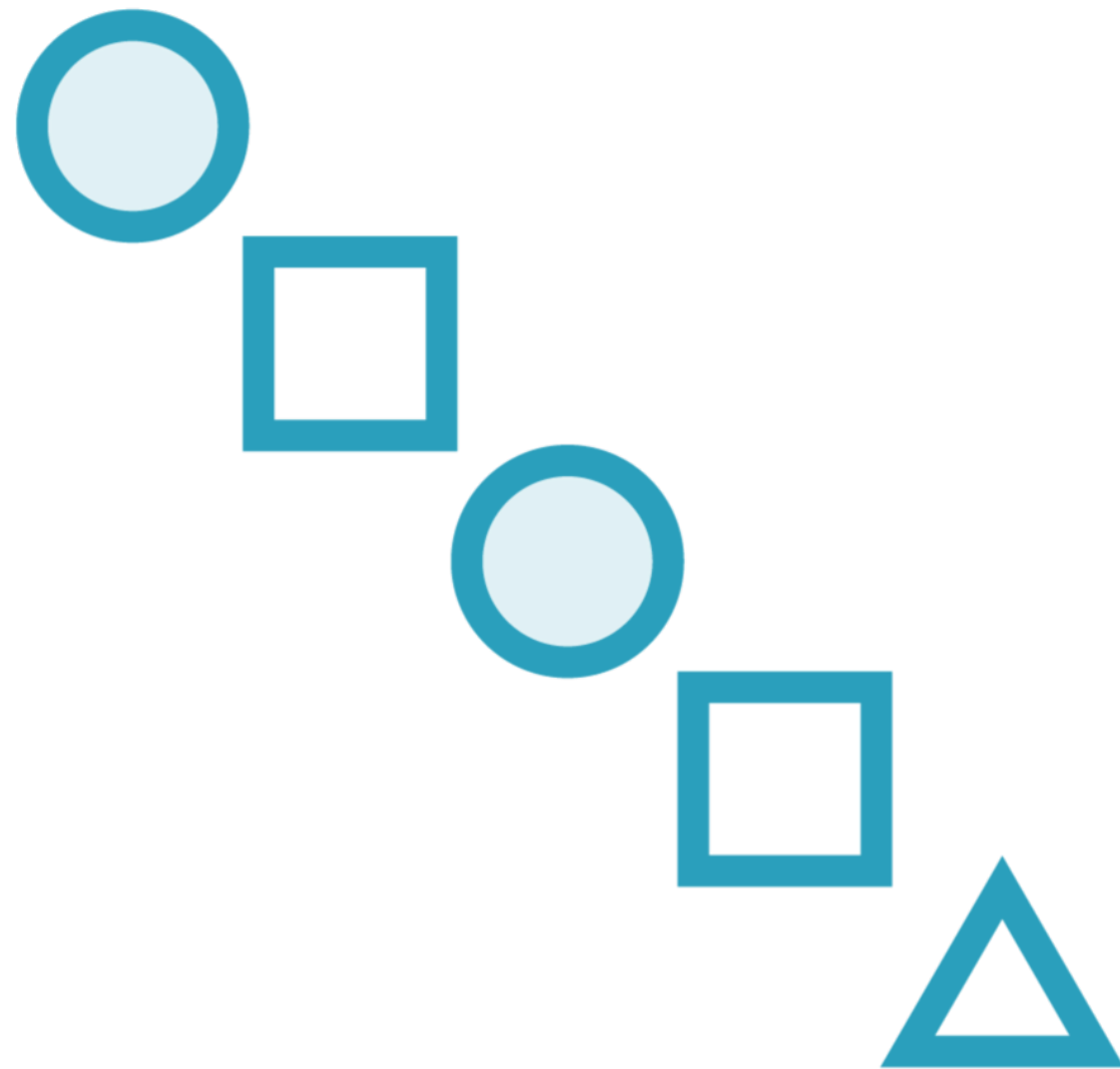
**Looking at the page we will create**



# Introducing the MVC pattern

---



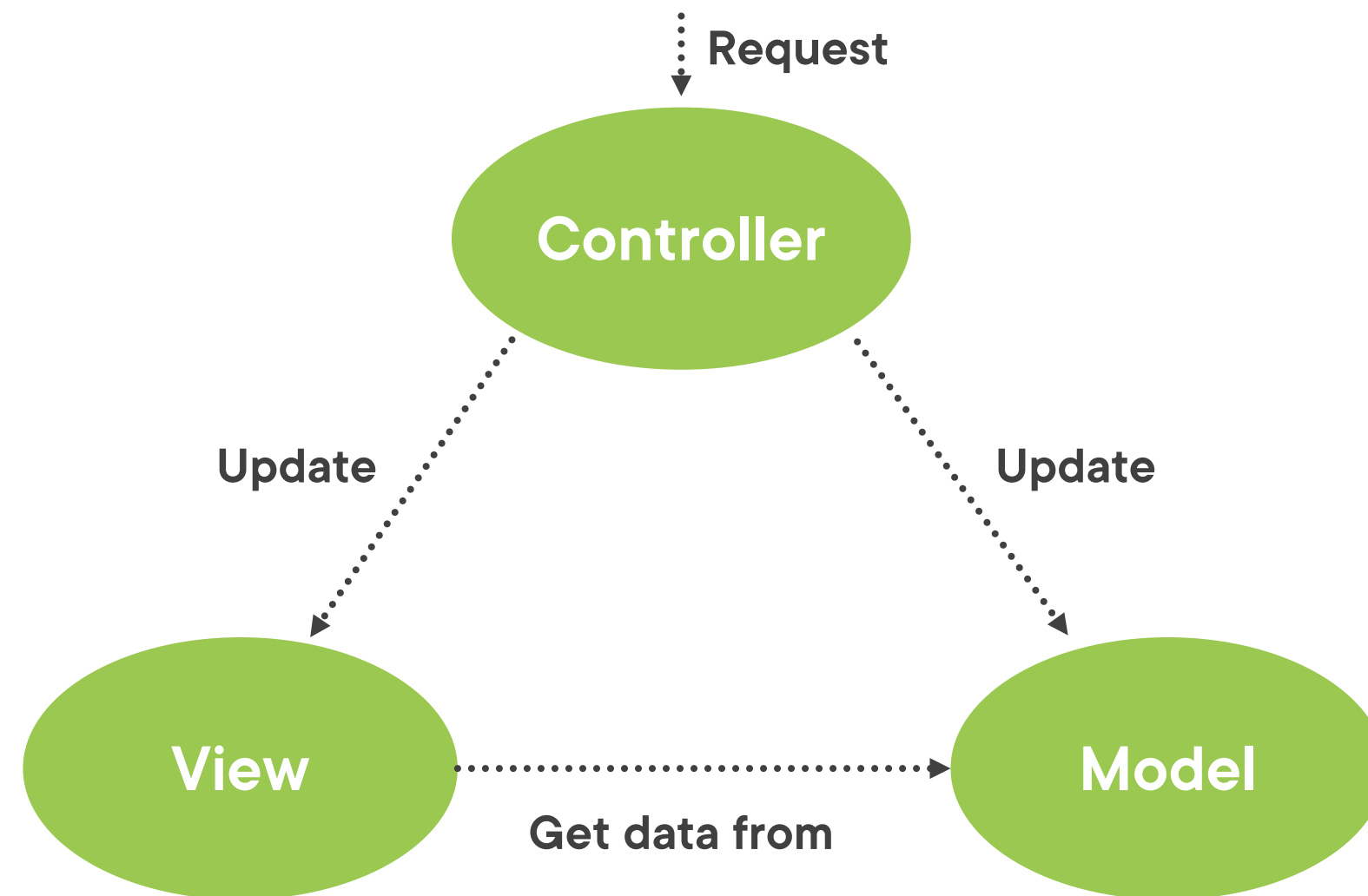


## Model – View – Controller

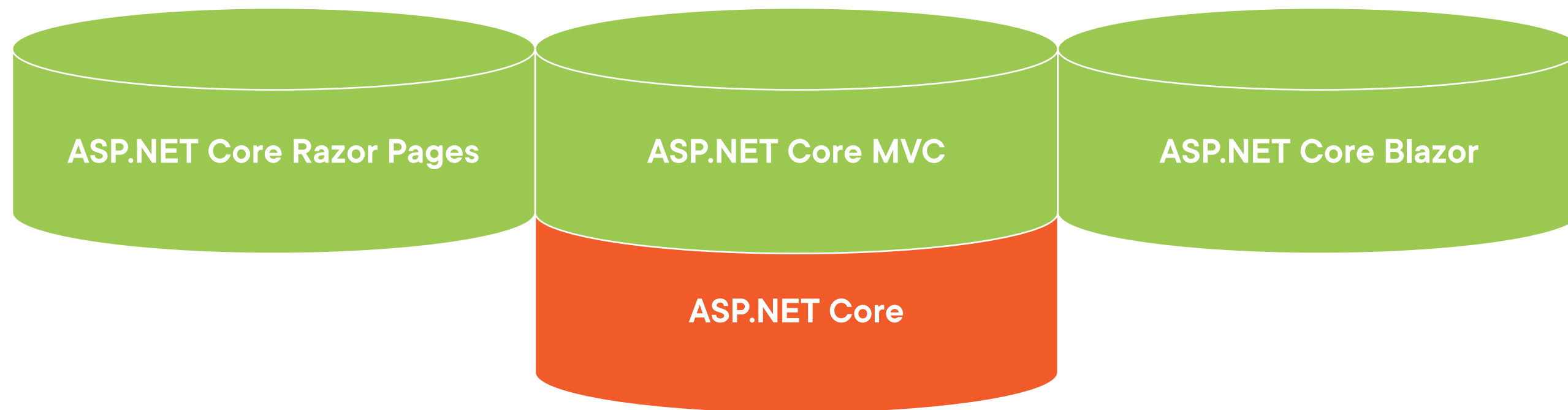
- Common design pattern
- Separation of concerns
- Less dependencies
- Promotes testability and maintainability



# The MVC Pattern



# The ASP.NET Core MVC Framework



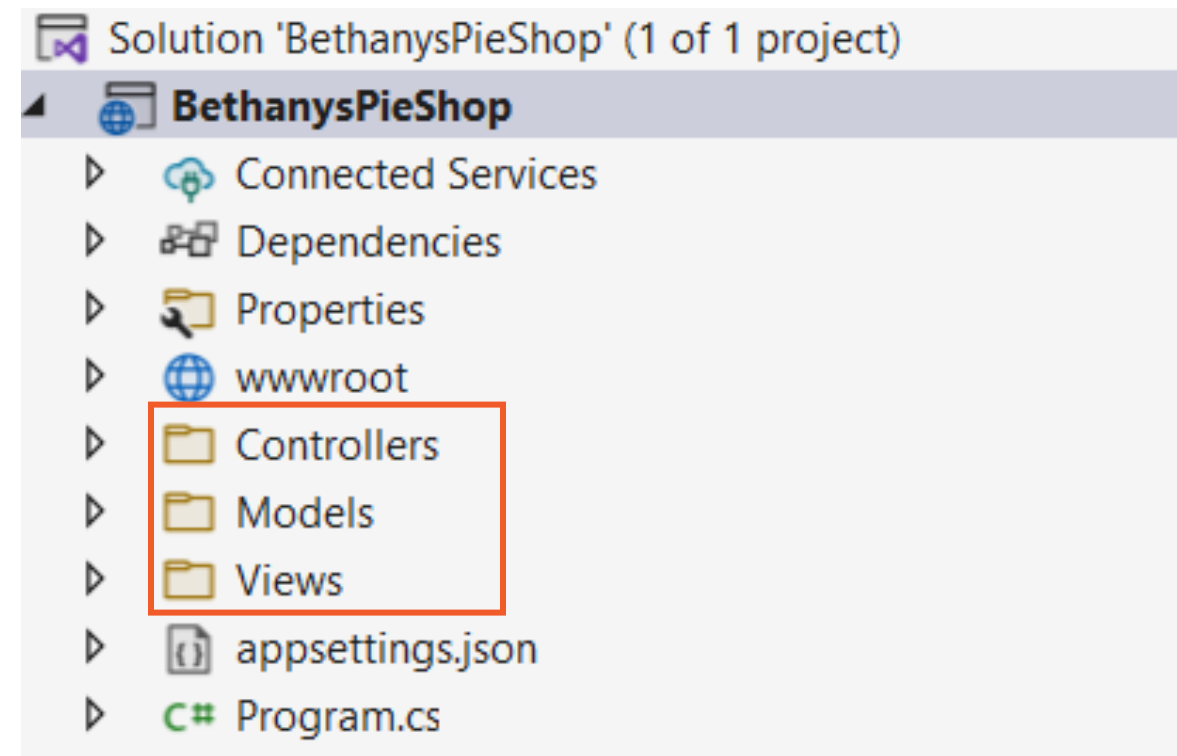
# Creating the Model and the Repository

---

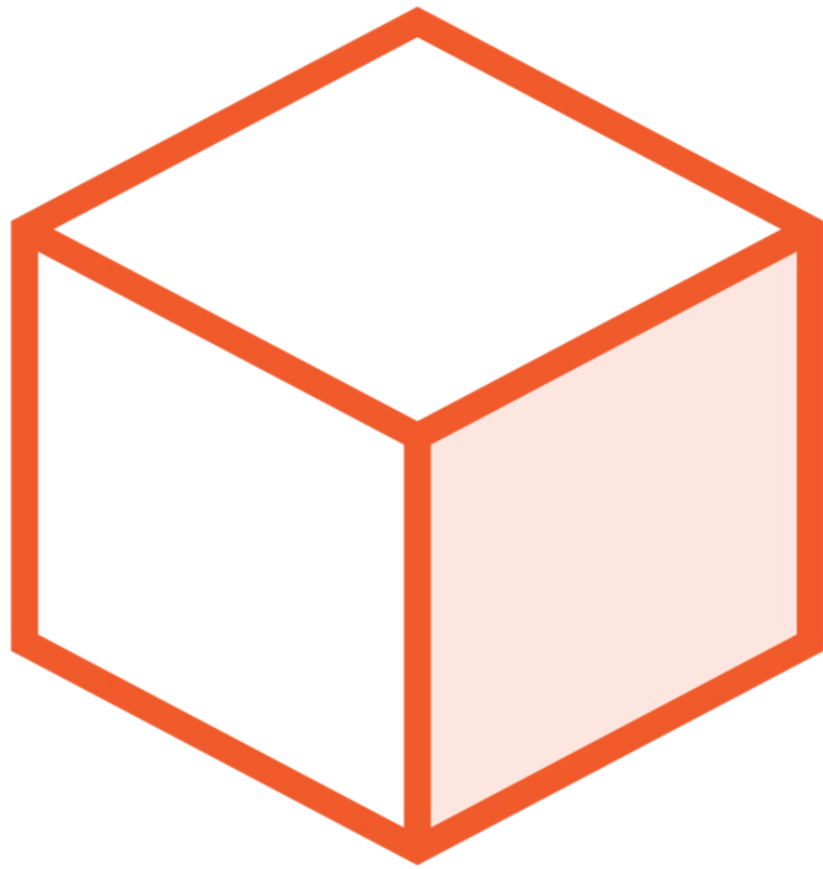




# Convention-based Folders



# The Model



**Domain data & logic to manage data**

**Simple API**

**Hides details of managing the data**



# Sample Model Class

Nullable is used here

```
public class Pie
{
    public int PieId { get; set; }
    public string Name { get; set; }
    public string? ShortDescription { get; set; }
    public string? LongDescription { get; set; }
    public string? AllergyInformation { get; set; }
    public decimal Price { get; set; }
    public string? ImageUrl { get; set; }
    public string? ImageThumbnailUrl { get; set; }
    public bool IsPieOfTheWeek { get; set; }
    public bool InStock { get; set; }
    public int CategoryId { get; set; }
    public Category Category { get; set; }
}
```



The repository allows our code  
to use objects without knowing  
how they are persisted.  
We will create an interface and  
its implementation.



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

The IPieRepository Interface

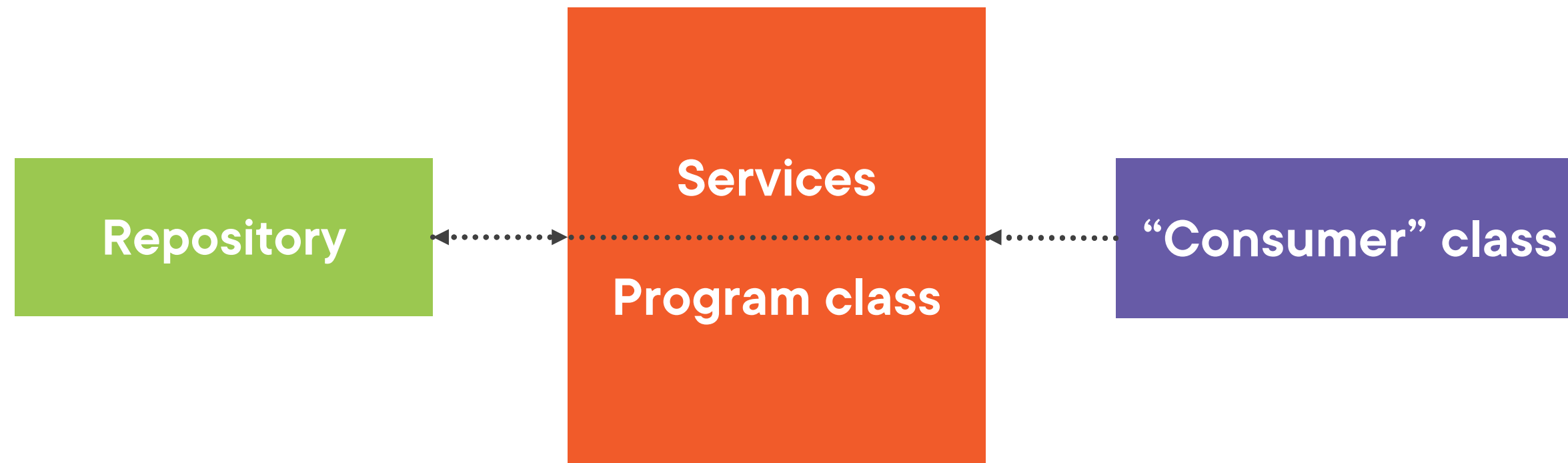
# Creating the Implementation

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

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



# Registering the Repository



# Registering the Repository

**Program.cs**

```
var builder = WebApplication.CreateBuilder(args);  
  
builder.Services.AddScoped<IPieRepository, MockPieRepository>();  
var app = builder.Build();
```



# Registering Services

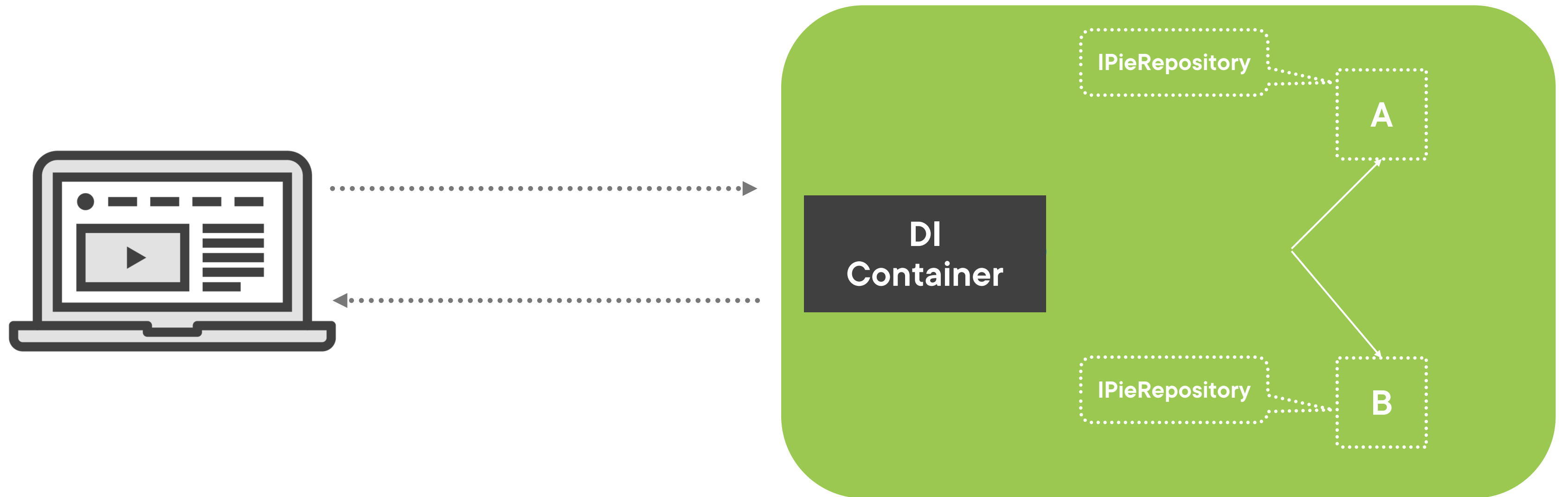
**AddTransient**

**AddSingleton**

**AddScoped**



# Understanding AddScoped



# Demo



**Creating the domain**

**Adding the repository**

**Registering with the services collection**



# Creating the Controller

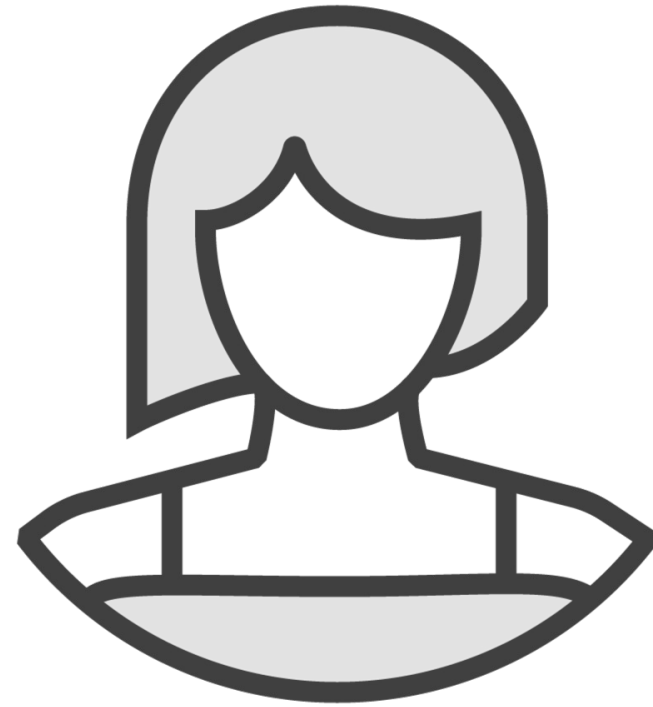
---



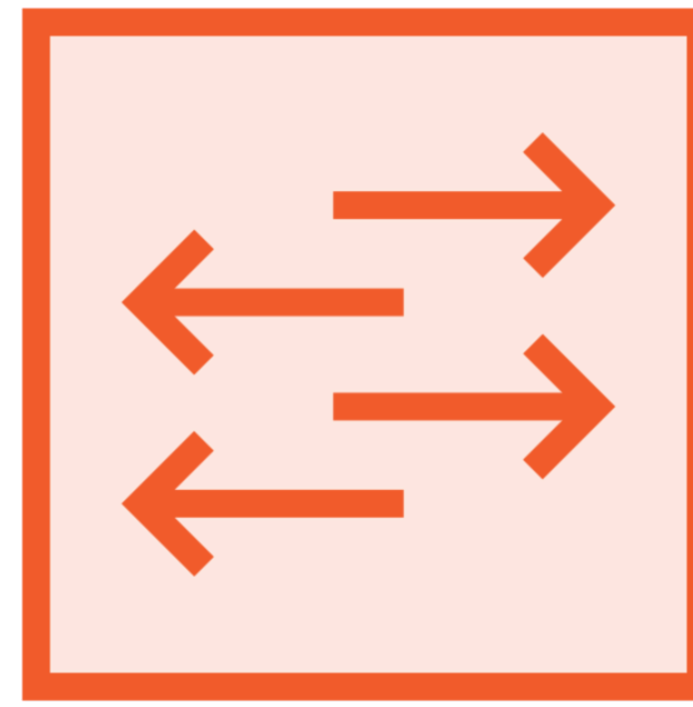
# The Controller



**Central role**



**Respond to user  
interaction**



**Interact with  
model and  
select view**



**No knowledge of  
data persistence**

# A Basic 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



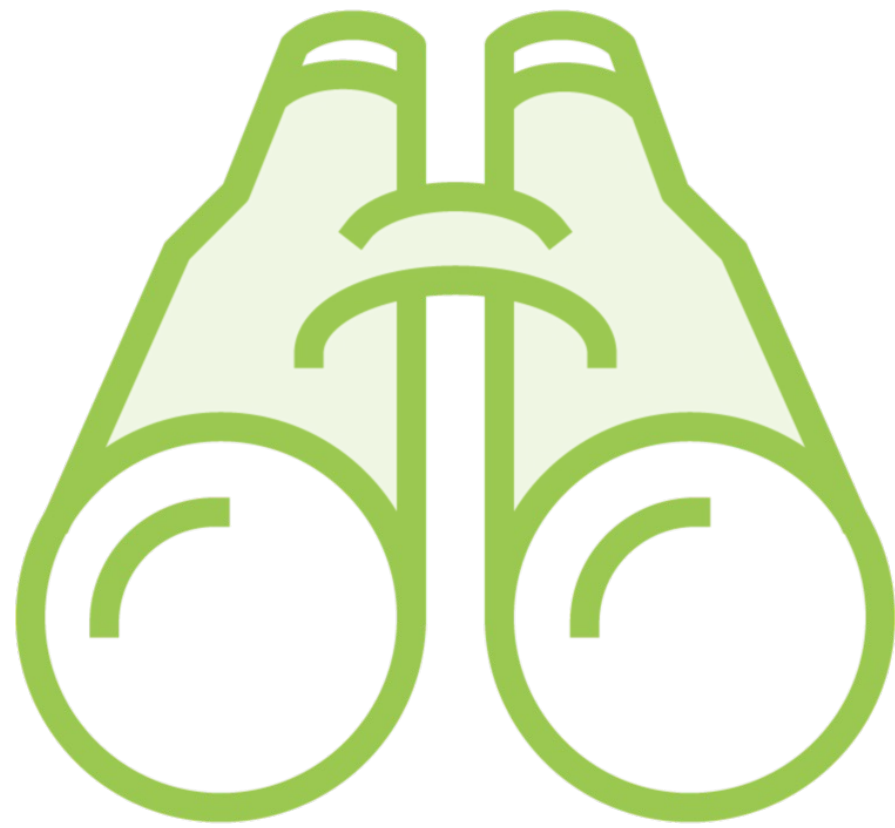


# Adding the View

---



# The View



**HTML template (\*.cshtml)**

**Use Razor code**

**(Almost) no logic**

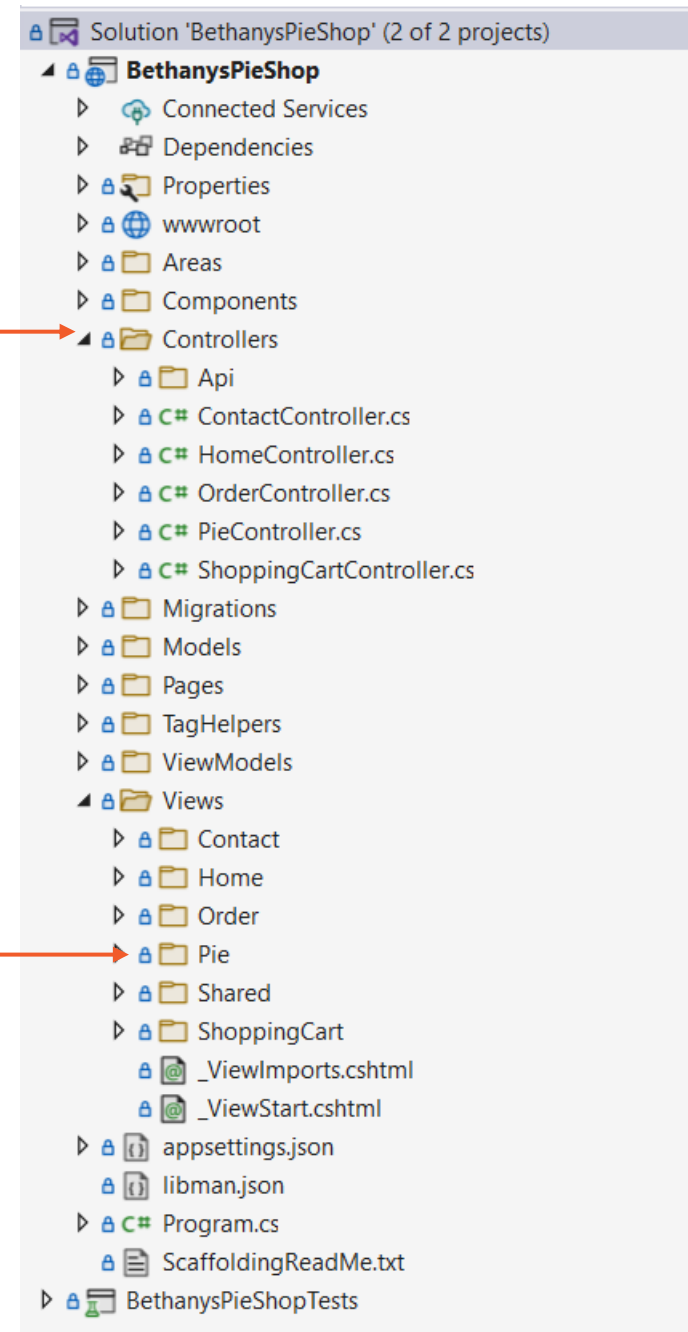
- Conditions, loops
- Add tag helpers and view components



# Matching the Controller and Its Views

**PieController**

**Pie folder**



# Matching the Action With the View

Convention-based approach

```
public class PieController : Controller
{
    public ViewResult Index() ..... Action
    {
        return View(); ..... 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 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>
```



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





```
@ViewBag.Message
```

```
<p>@DateTime.Now</p>
```

```
@{
```

```
    var message = "Welcome to Bethany's  
    Pie Shop";
```

```
}
```

```
<h3>@message</h3>
```

◀ Using ViewBag in Razor

◀ Displaying a date in Razor code

◀ Using a code block

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

Calling a Strongly-typed View

# A Strongly-typed View

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



# Demo



## Creating the first view



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

## Introducing the View Model

# Demo



## Using a View Model



# Using the \_Layout.cshtml

**Template**

**Lives in Shared  
folder  
Searched by default**

**One or more  
View can specify**



# \_Layout.cshtml

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





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

```
@model PieListViewModel
```

View Can Specify the Layout

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

\_ViewStart.cshtml

**Contains logic shared by set of views**  
**Executed automatically**

```
@using BethanysPieShop.Models
```

View Imports

**Group commonly used using statements**

# Demo



**Adding a layout template**

**Creating the ViewStart file**

**Adding the ViewImports file**



# Styling the View

---



# Where We Need to Get


**BETHANY'S**  
PIE SHOP


SHOP ▾CONTACT

Q

REGISTER


LOGIN






### Pies of the week

Enjoy a weekly selection of our favorite pies




+ ADD TO CART

Apple Pie\$12.95



+ ADD TO CART

Pumpkin Pie\$12.95



+ ADD TO CART

Rhubarb Pie\$15.95



# Adding Client-side Libraries

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "libraries": [
    {
      "library": "bootstrap@5.1.3",
      "destination": "wwwroot/lib/bootstrap/"
    }
  ]
}
```

Add Client-Side Library

Provider:

cdnjs

Library:

bootstrap@5.1.3

☒ Include all library files

☐ Choose specific files:

☒

Files:

☒

css

☒

bootstrap-grid.css

☒

bootstrap-grid.css.map

☒

bootstrap-grid.min.css

☒

bootstrap-grid.min.css.map

Target Location:

wwwroot/lib/bootstrap/

Install

Cancel



# Demo

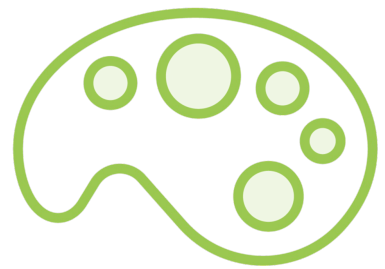


## Bringing in Bootstrap through Library Manager





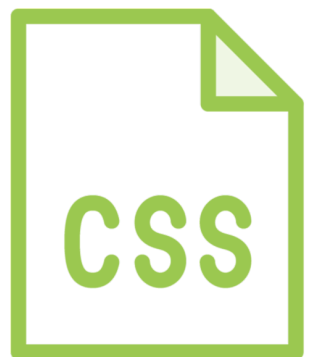
# Introducing CSS Isolation



**Styles specific for a page**



**Helps with avoiding style name conflicts**



**.cshtml.css**



# Demo



## Adding CSS isolation



# Summary



**MVC ensures good separation of concerns**

- M
- V
- C

**Models wrap data**

**Views are templates that show model data**

**Controllers direct the flow**





**Up next:**  
Accessing data in a database

