ASP.NET Core

C# Programming

In this chapter we will learn ...

- What is ASP.NET Core
- How to use write a CRUD application using ASP.NET Core

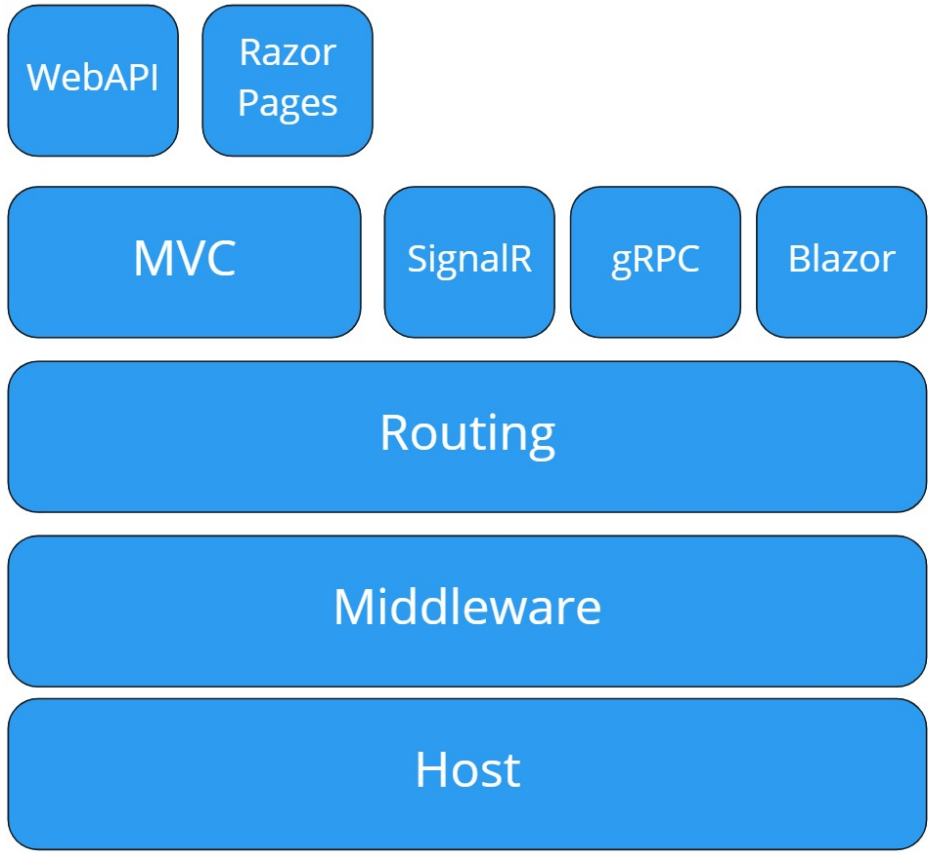
Developing a Web Application with ASP.NET

What is ASP.NET

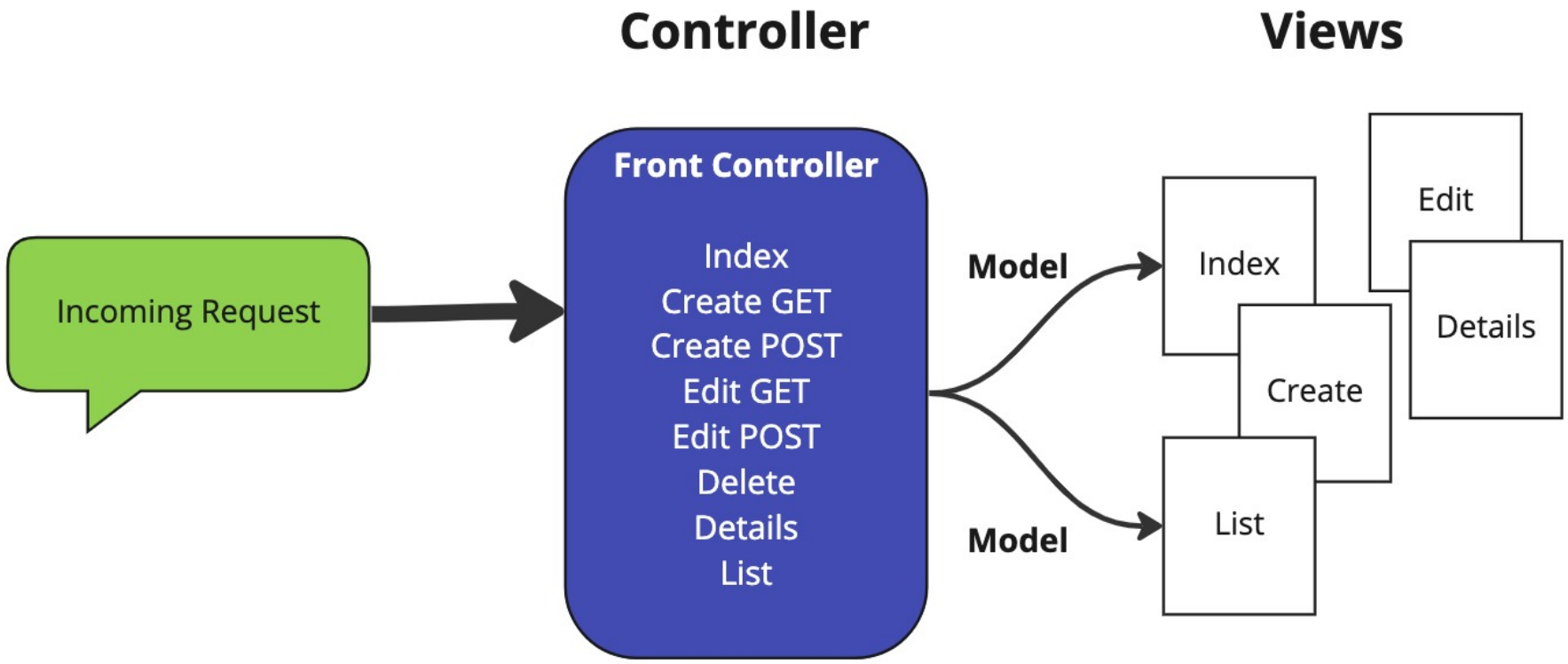
ASP.NET started as a successor to Active Server Pages (ASP) == Server Side Scripting

- ASP.NET allowed development of Web Forms
- ASP.NET MVC added the MVC design pattern
  - Implements the Model View Controller pattern = Separation of concerns
  - Made life easier for many developers to collaborate
  - Improved unit testing
- ASP.NET Razor Pages
  - Sits on top of MVC and provides a simpler implementation of MVC design pattern
  - Allows a separation of concerns
  - Reduces complexity

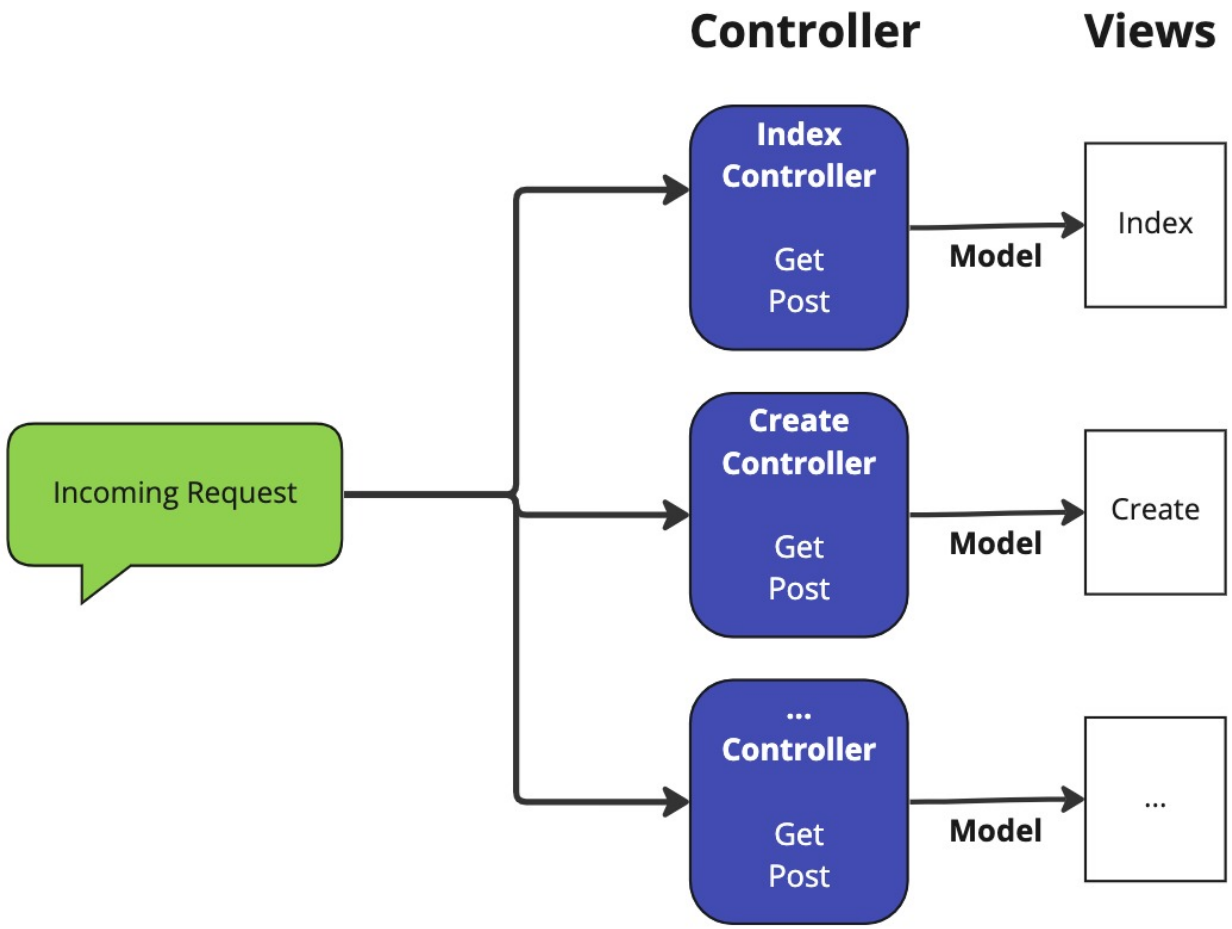
ASP.NET Technologies



Model View Controller - ASP.NET MVC

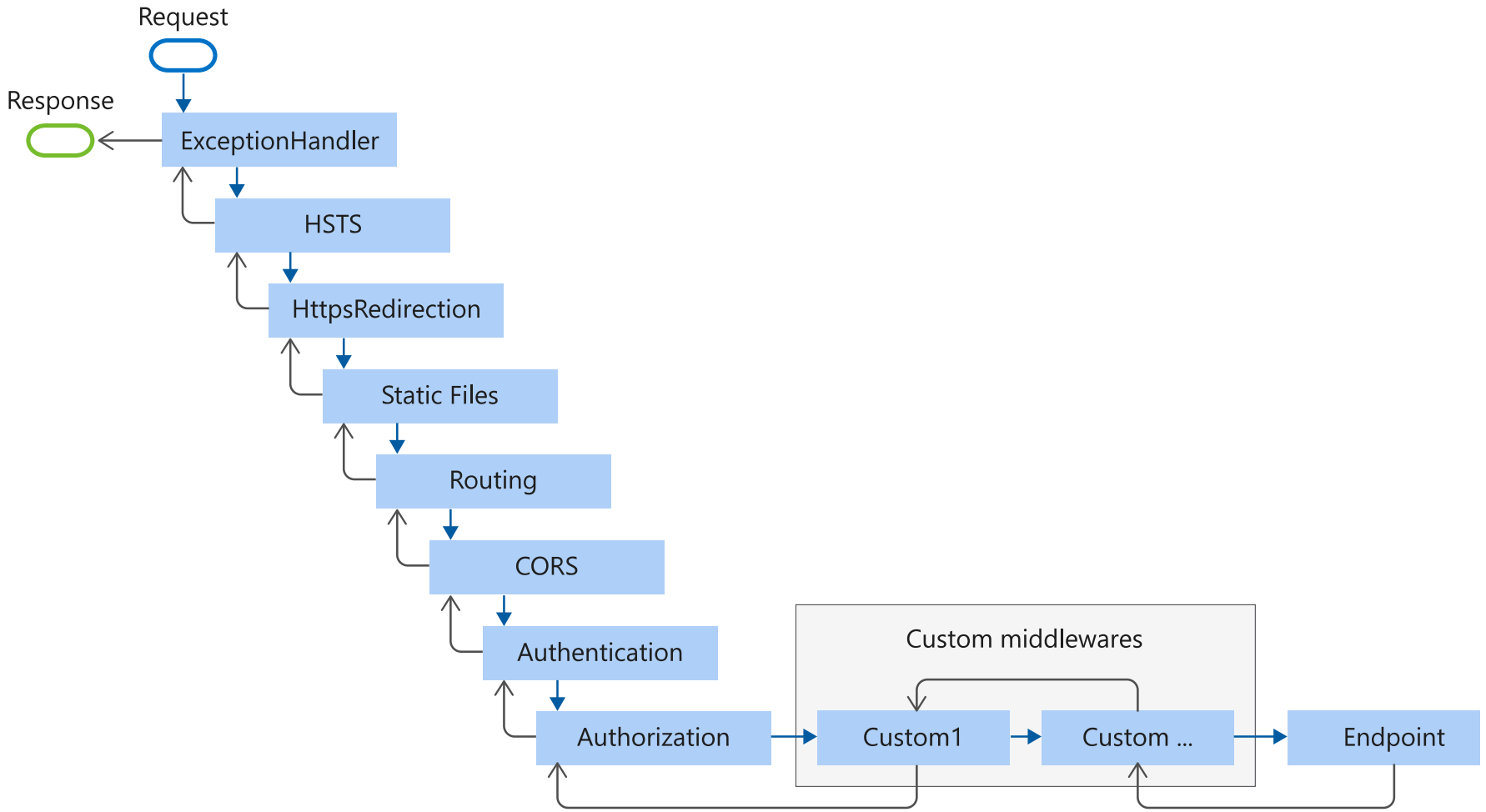


Model View Controller - Razor Pages



ASP.NET Pipeline

- ASP.NET implements a Chain of Responsibility Design pattern in a Pipeline



Creating and ASP.NET Core Web App

Using Visual Studio 2022:

- Create a new Project
- Choose (or search for) ASP.NET Core Web Application Template
- Give the project and solution suitable names
    - Choose .NET 6.0 (Long Term Support)

Using Visual Studio Code (better for cross platform)

First we need to trust the self-signed certificate we will use for HTTPS browsing

```
dotnet dev-certs https --trust
```

Next create the app

```
dotnet new sln
dotnet new webapp -o FirstApp
dotnet sln add FirstApp\FirstApp.csproj
```

- Line 1 - Create a solutions with the same name as the containing directory
- Line 2 - Create ASP.NET Core WebApp "FirstApp" (created in subfolder FirstApp)
- Line 3 - Adds the FirstApp.csproj Project file to the solution file

Running Web App

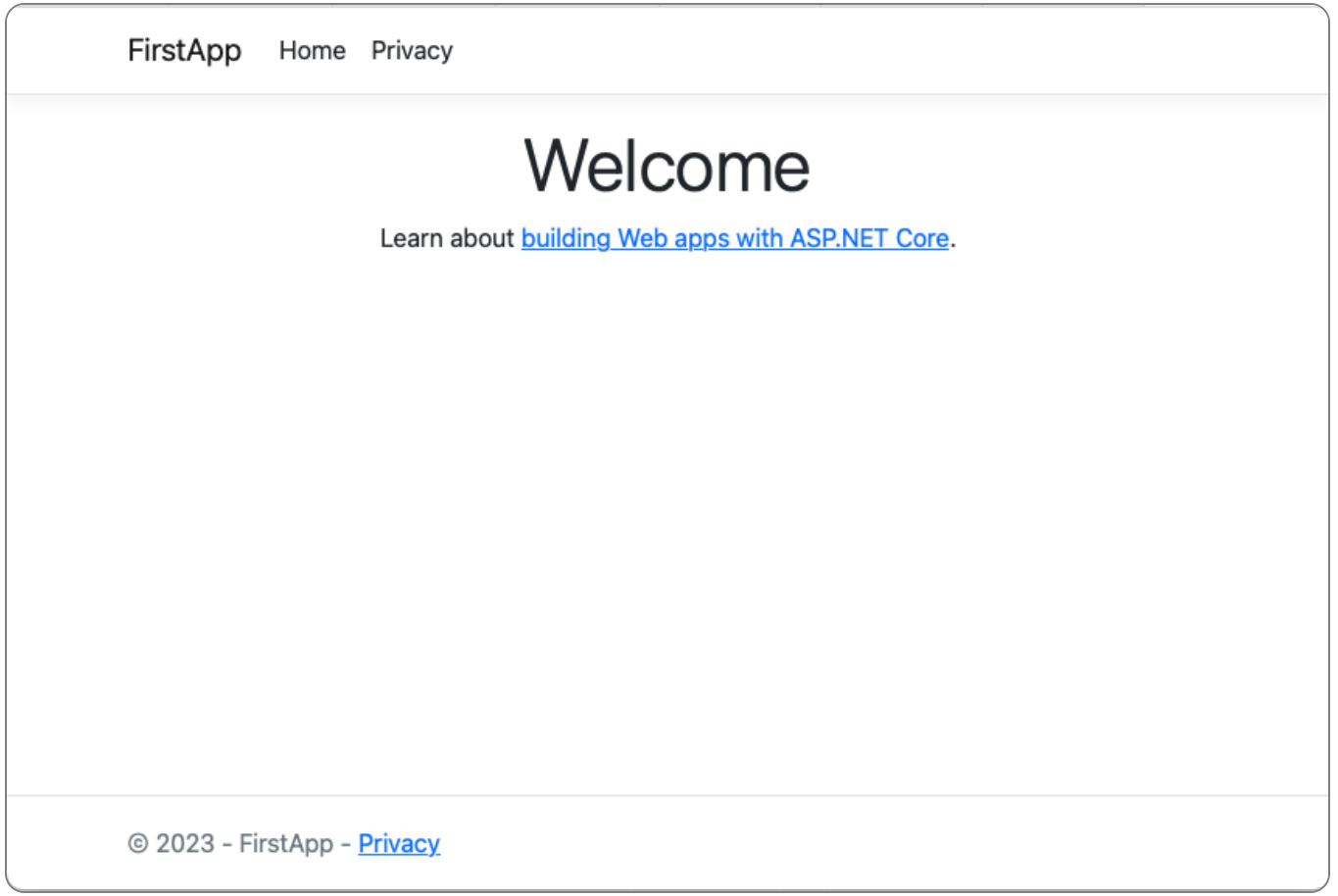
From the solution folder run the following in a terminal

```
dotnet run --project FirstApp/FirstApp.csproj
```

```
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5169
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /Users/someuser/Documents/dotnet_core/apps/FirstApp
```

The URL listed can be navigated to by press CTRL (CMD on Mac) and clicking with mouse

The Basic Web App



Hot Reload

To reload the browser with changes run the app using watch

```
dotnet watch run
Changes will be reflected in the browser live
```

What's created - Files and Folders

- Properties - Contains launchSettings.json allowing for control over how app is run
- wwwroot - Contains the static files including css, javascript and images
- Pages containing
    - Pages - Act as Views - .cshtml containing HTML + content generating code
    - PageModel - Act as ViewControllers handling Get and Post requests and supplying Model
    - Shared Folder containing shared re-usable layouts for some/all pages
- appsettings.json - Configuration settings eg logging, database connections
- Program.cs - Bootstraps application. Allows setup of features like Logging + Dependency injection. Sets up the Request Pipeline and registers middleware.

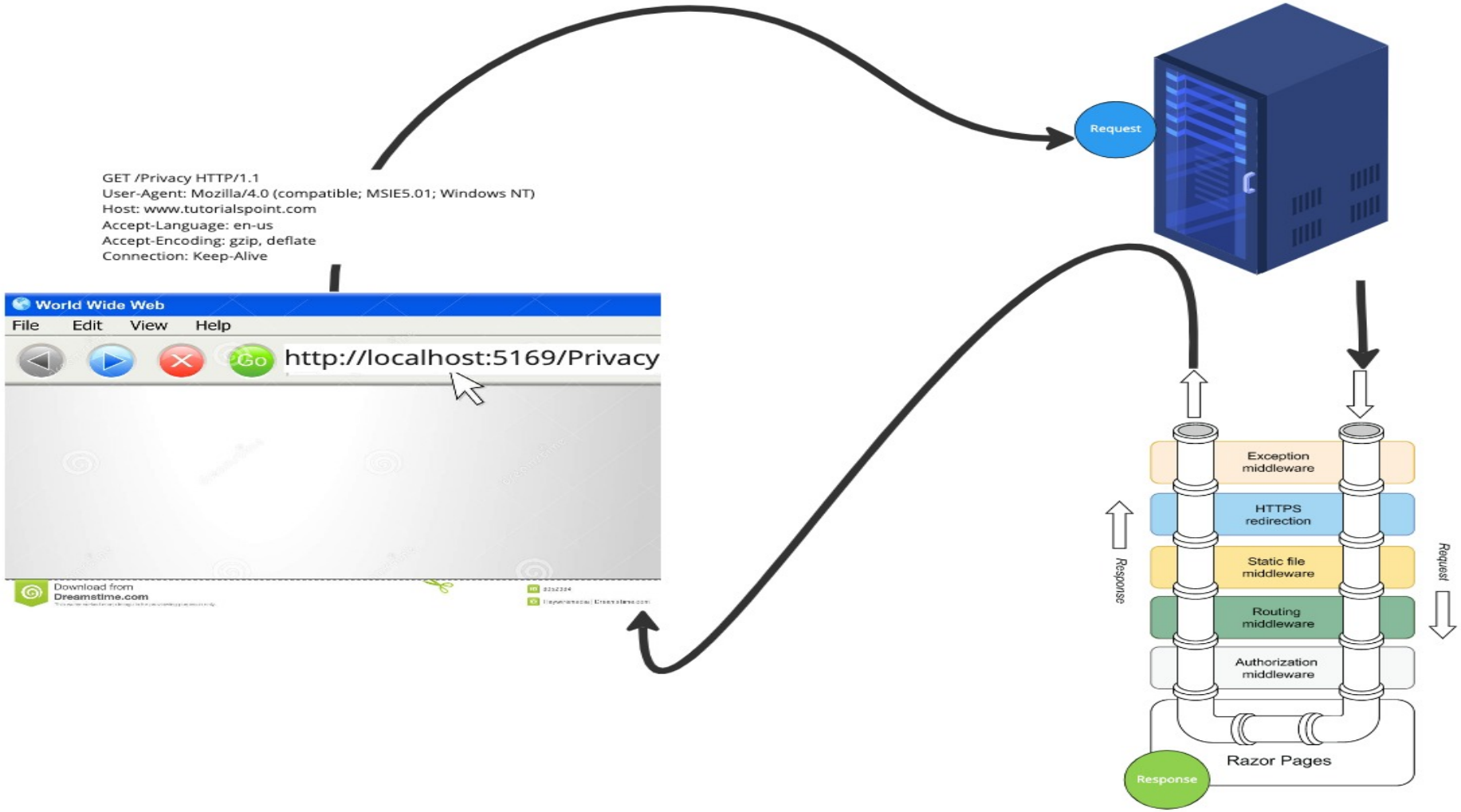
Shared Layouts

By Default Razor Pages use a reusable shared layout

- Located in Shared folder
- Called _Layout.cshtml
- Contains a HTML5 compliant layout
- Includes a _Layout.cshtml.css
- Also uses Bootstrap css and js files

Modifying and dynamically changing layouts is outside of scope for us

HTTP and ASP.NET Core Web Apps



Creating a new Page

In the Pages folder

```
dotnet new page --n Hello
```

- Creates Hello.cshtml and Hello.cshtml.cs
- Navigate to it using http://localhost:5000/Hello

or create a sub-folder of Pages called Hello then inside that folder

```
dotnet new page --n index
```

- Creates index.cshtml and index.cshtml.cs (index acts as a default page for a folder)
- Navigate to it using http://localhost:5000/Hello

Modifying Generated Pages

Here we added a "Hello World" message to Hello.cshtml

```
@page
@model MyApp.Namespace.HelloModel
@{
}
<p>Hello World</p>
```

Hello.cshtml.cs

```
public class HelloModel : PageModel {
    public void OnGet() {
        // retrieve data, setup member variables
    }
}
```

Returning content other than Current Page

We can control what gets returned from OnGet or OnPost

- Set return type to **IActionResult**
- Return one of a number of possible return options

```
public IActionResult OnGet() {
    return RedirectToPageResult("Index");
}
```

Other options include:

- NotFound(); - Not Found exception
- Page(); - Current Page
- Content("Hello World"); - Raw content

Async Pages

When calling async functions we can use

```
public async Task<IActionResult> OnGetAsync()
{
    var accounts = await _webServices.GetAccounstAsync();
    ...

    return Page();
}
```

Getting Data from the PageModel

```
@page
@model MyApp.Namespace.HelloModel
@{
}
<p>Hello @Model.Name</p>
```

```
public class HelloModel : PageModel {
    public string Name { get; set; } = "";
    public void OnGet() {
        this.Name = "World";
    }
}
```

http://localhost:5000/Hello should display Hello World inside our standard layout

Routes and Templates

By Default the Pages Folder structure becomes the URL Path

- Sub Folder path `.\Pages\Hello` becomes route `/pages/hello`
- http://localhost:5000/pages/hello

Changing the route using the `@page` directive allows http://localhost:5000/welcome

```
@page "/welcome"
```

Route Parameters allow passing data http://localhost:5000/welcome/Fred

```
@page "/welcome/{name:alpha}"
```

Passing Data in the URL

```
@page "{name:alpha=World}"
@model MyApp.Namespace.HelloModel
@{
}
<p>Hello @Model.Name</p>
```

```
public class HelloModel : PageModel {
    public string Name { get; set; } = "";
    public void OnGet(string name="World") {
        this.Name = name;
    }
}
```

http://localhost:5000/Hello/Beth

Hello Beth

Working with Razor Pages

Razor Pages can contain HTML and code

- Code is delineated using the @ symbol

```
@{
    // C# goes here
}
```

Define functions

```
@functions{
    int doubleIt(int dataIn){
        return dataIn * 2;
    }
}
<p>Double 2 is @doubleIt(2)</p>
```

Control blocks in Razor Pages

You can embed if, for, foreach, switch, while inside a page - notice @ prefix

```
@if Mode.movies[0].Year < 1980
{
    <p>Old Movie</p>
}
else
{
    <p>Not an Old Movie</p>
}
```

Defining a PageModel with a Data Property

```
public class indexModel : PageModel{

    public List<string> Movies {get;set;} = new() {"Star Wars", "Jaws", "Jurassic Park"};

    public void OnGet()
    {
        // Could use Entity Framework to retrieve data
    }
}
```

Generating HTML from a Model Dataset

Defining a Razor Page to consume a simple Page Model

```
@page
@model MyApp.Namespace.indexModel
@{}
<ul>
@foreach(string movie in Model.Movies){
    <li>@movie</li>
}
</ul>
```

Handling HTML Forms

A HTML Form is defined using:

- form element with a http **method** (get or post)
- Input elements for data entry

```
<form method="post">
    <div class="mb-3">
        <label for="name">Title</label>
        <input class="form-control" type="text" name="Title" />
    </div>
    <div class="mb-3">
        <label for="name">Director</label>
        <input class="form-control" type="text" name="Director" />
    </div>

    <button class="btn btn-primary">Submit</button>
</form>
```

Receiving Data in the PageModel

OnPost can receive an object

- name attribute of input field is mapped to properties of object
- name = "Title" would map to the Title property of movie parameter

```
public class IndexModel : PageModel {
    ...
    public void OnPost(Movie movie){
       // Create a new Movie
    }

}
```

Binding a Form to a Property on Post

ASP.NET is intelligent and can bind fields to PageModel properties

- Note the use of the [BindProperty] attribute to enable two way binding
- [BindProperties] on PageModel will bind all public properties

```
public class IndexModel : PageModel {
    [BindProperty]
    public Person CurrentPerson {get;set;}
    public void OnGet(){
        // Used to retrieve any data for building the form for display
    }
    public void OnPost(){

       // Executes when the form is submitted (as long as method is set to POST)
       // CurrentPerson fields will be filled in with data submitted by form
    }
}
```

Using Tag Helpers

Tag helpers assist with forming HTML elements- Prefixed with **asp-**

- asp-for will generate the correct control based on the data data type

.NET Type	HTML Input Type
string	text
bool	checkbox
int, byte, short, long	number
decimal, double, float	text
DateTime	datetime-local
IFormFile	file

Defining Page Handlers

Page handlers allow you to define the handler method receiving the Post

```
<form method="post" asp-page-handler="Search">
...
</form>
```

```
public void OnPostSearch(Movie movie) {
}
```

Other asp-page-handler uses

Can be placed on a <button> or <a> together with **asp-route-id**

```
<button asp-page-handler="Delete"
        asp-route-id="@Model.Movie.Id" class="btn-danger">Delete</button>
```

```
public void OnPostDelete(int id) {

}
```

Validation

Validation attributes can be placed on the PageModel properties

Compare	Compares to another property.	[Compare(nameof(PasswordRepeat))]
MaxLength	Max chars allowed.	[MaxLength(30)]
Range	Min and max values of a range.	[Range(5,8)]
RegularExpression	Checks against a regex.	[RegularExpression(@"\d{2}-\d{2}-\d{2}")]
Required	Required value.	[Required]
MinLength	Minimum number of chars accepted.	[MinLength(3)]
StringLength	Maximum and the minimum chars	[StringLength(2)], [StringLength(10, MinimumLength=2)]

Validation - Client Side

- Prevents server processing unnecessary requests and acts as a courtesy to the user
- Is easily circumvented

To activate add the following to the page

```
@section scripts{
    <partial name="_ValidationScriptsPartial"/>
}
```

Add an errors summary container at the top of the form

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
```

Add an error message ontainer for each control

```
<span asp-validation-for="Movie.Year" class="text-danger"></span>
```

Scaffolding Data Entities for CRUD Operations

Once an Entity Framework Mode has been setup its possible to generate:

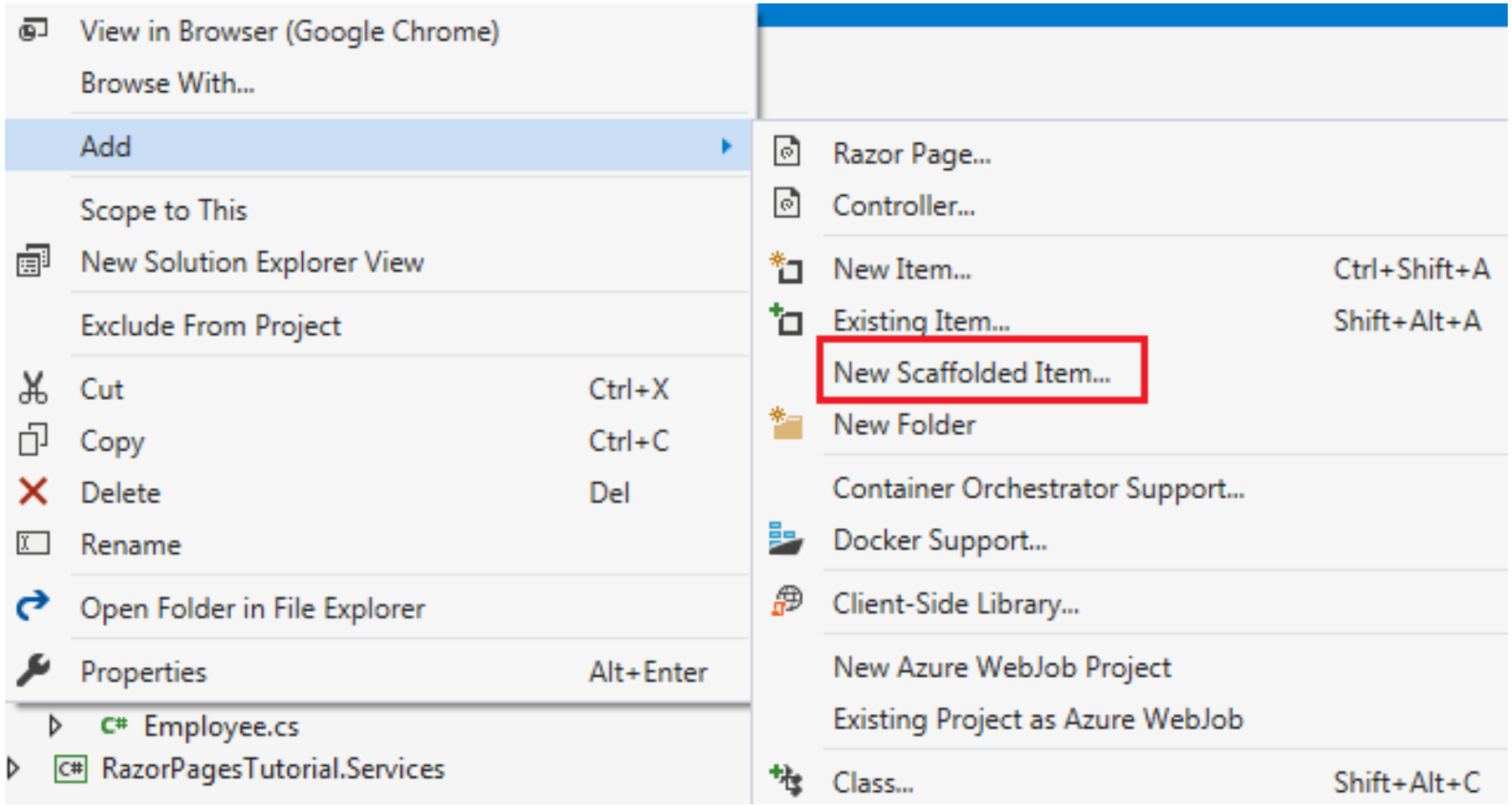
- Razor Page HTML Form UI for performing Create, Read, Update and Delete operations
- PageModel that binds to the HTML Forms
- Validation support

IDE Support

- Visual Studio 2022 has great support from within the IDE
- VS Code requires a console app

Scaffolding an Entity in VS 2022

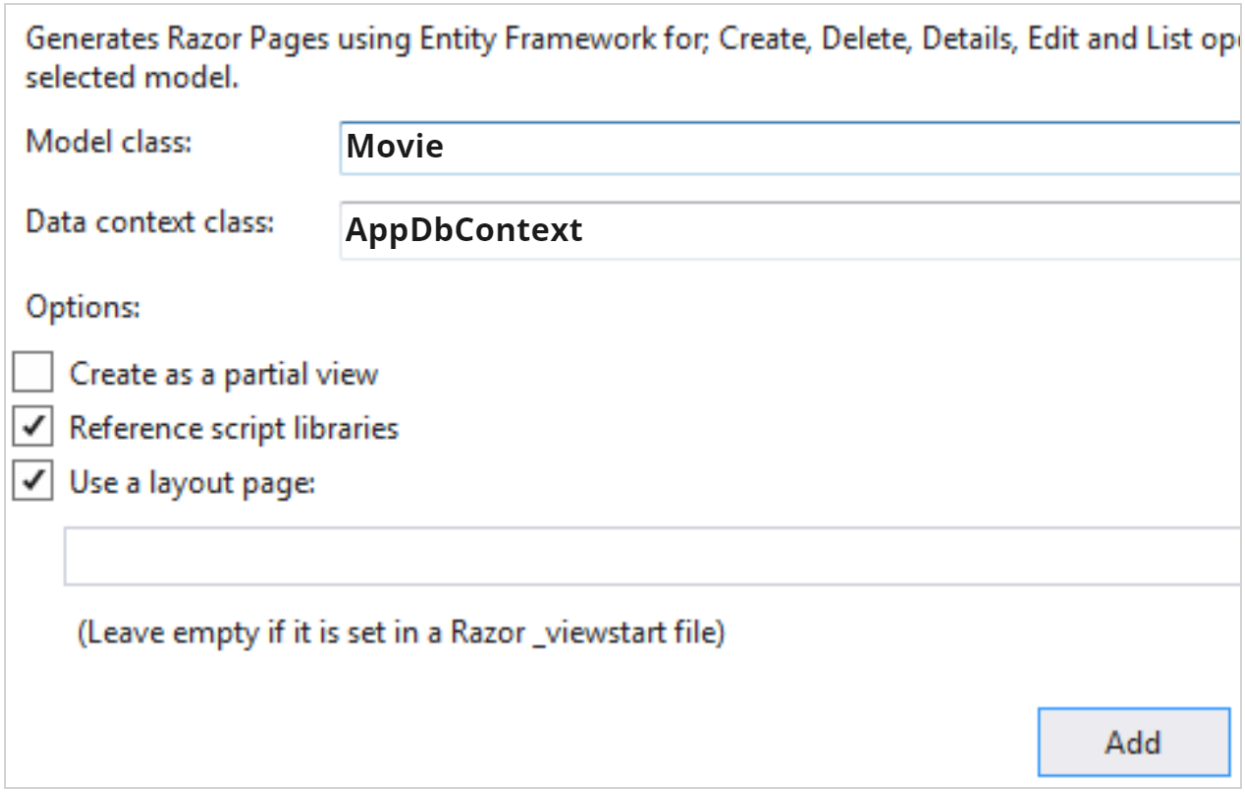
Create a folder within /Pages eg Movies then



Scaffolding an Entity in VS 2022 - Continued

Select the Entity and Entity Framework Application Context to generate

- Create, View, Update and Delete Razor pages



Scaffolding in VSCode

- Ensure the following Nuget Packages are installed

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet add package Microsoft.EntityFrameworkCore.Tools
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design
dotnet add package Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore
```

- Create a Pages/Movies folder
- Install asp.net code generator

```
dotnet tool install --global dotnet-aspnet-codegenerator
```

Scaffolding in VSCode - Continued

Scaffold Movie entity

- On Windows

```
dotnet aspnet-codegenerator razorpage -m Movie
-dc MoviesDemo.AppDbContext
-udl -outDir Pages\Movies --referenceScriptLibraries
```

- On Windows

```
dotnet aspnet-codegenerator razorpage -m Movie
-dc ContosoUniversity.Data.SchoolContext
-udl -outDir Pages/Movies
--referenceScriptLibraries
```

Appendix

In this chapter we learned ...

- What is ASP.NET Core
- How to use write a CRUD application using ASP.NET Core