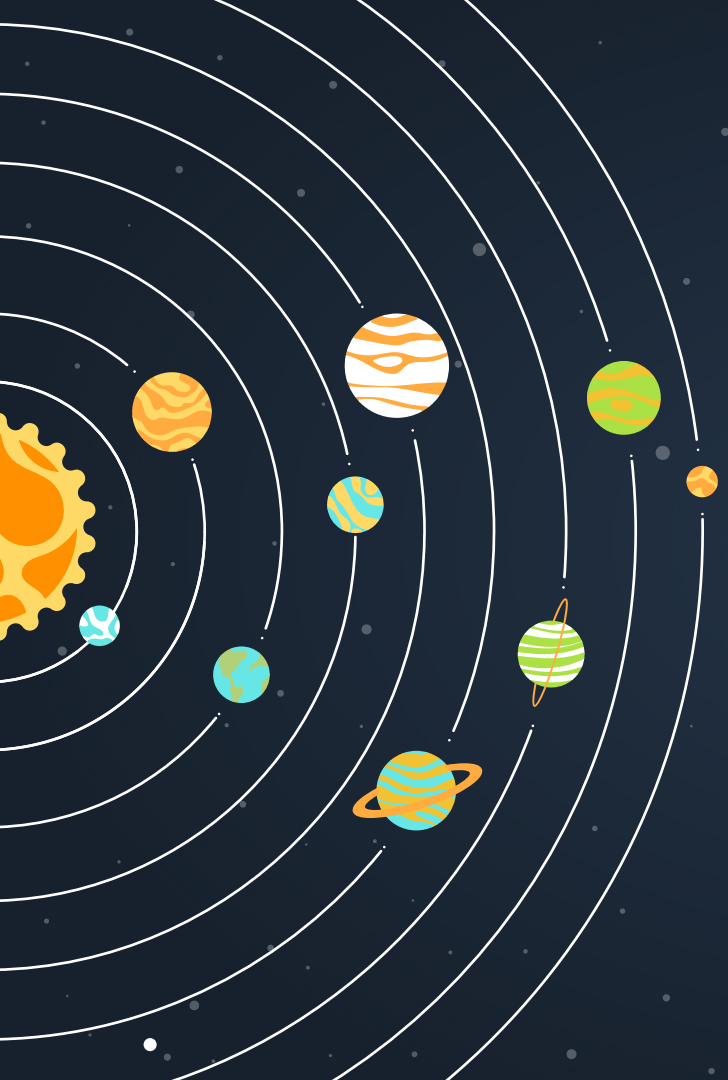


ASP.NET CORE

2020.5.21

TABLE OF CONTENTS

“Before Travel”

- **ASP**: Active Server Pages
  - Microsoft’s server side script engine for dynamically generated web pages
- **.NET**: a software framework to create, run and deploy desktop apps and server based apps
- **ASP.NET**: the extension of the ASP which is part of the .NET framework that simplifies the structure and creation of web apps

01 INTRODUCTION

ASP.NET Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps.

- Build web apps and services, IoT apps, mobile backends
- Can use favorite tools on Windows, macOS, Linux
- Deploy to the cloud
- Run on .NET Core

1. Why choose ASP.NET Core?

- A unified story for building web UI and web APIs.
- Architected for testability.
- Razor Pages: make page-focused scenarios easier and more productive.
- Blazor: make it possible to use C# in the browser alongside JavaScript.

  Ability to develop and run on Windows, macOS, and Linux.
- Open-source and community-focused.
- Support for hosting Remote Procedure Call (RPC) services using gRPC.
- A cloud-ready, environment-based configuration system.
- A lightweight, high-performance, and modular HTTP request pipeline.
- Ability to host on the following: Kestrel, IIS, HTTP.sys, Nginx, Apache, Docker
- Side-by-side versioning.
- Tooling that simplifies modern web development.

2. ASP.NET 4.x vs ASP.NET Core?

	PLATFORM	UI	VERSIONS PER MACHINE	IDE	PERFORMANCE	RUNTIME
ASP.NET CORE	Windows, macOS, Linux	Razor Pages, MVC, Web API, SignalR	Multiple	VS, VS Code	Faster	.net core runtime
ASP.NET 4.X	Windows	Web Forms, SignalR, MVC, Web API, Web Hooks, Web Pages	single	VS	Good	.net framework runtime

3. .NET Framework vs .NET Core for server apps

When **.NET Core?**

1. Cross-platform
2. Microservices
3. Docker container
4. High performance and scalable system
5. Side-by-side .net versions per application

When **.NET Framework?**

1. .NET framework
2. 3rd party libraries not available for .NET Core
3. .NET technologies not available for .NET Core
4. Platform that doesn't support .NET Core

02 FUNDAMENTALS

- Startup class
- Dependency injection (services)
- Middleware
- Host
- Servers: **Kestrel** or **HTTP.sys**
- Environments: **dev, stage, prod**
- Logging: **Console, Debug, Event Source, Event Log**
- Routing
- Making http requests
- Static files

1) STARTUP CLASS

WHAT DOES IT DO?

1. Configure the app's services (**ConfigureServices**)
2. Create request processing pipeline (**Configure**)

SERVICE?

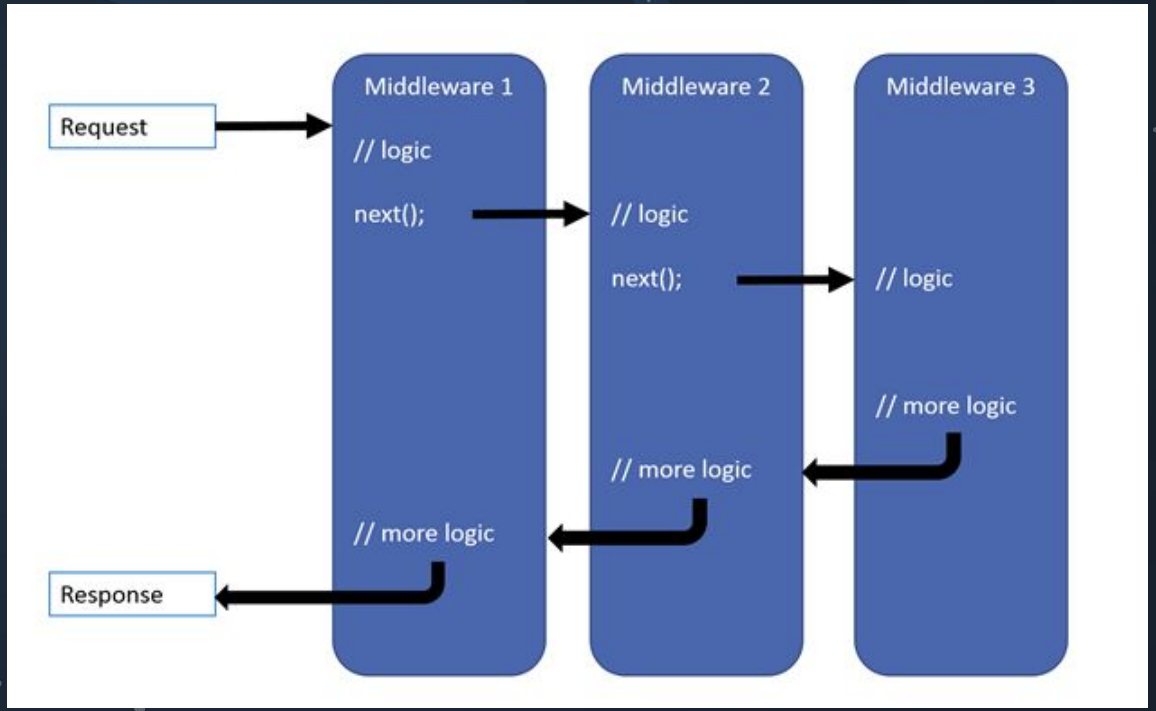
Reusable component that provides app functionality

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddRazorPages();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapRazorPages();
        });
```

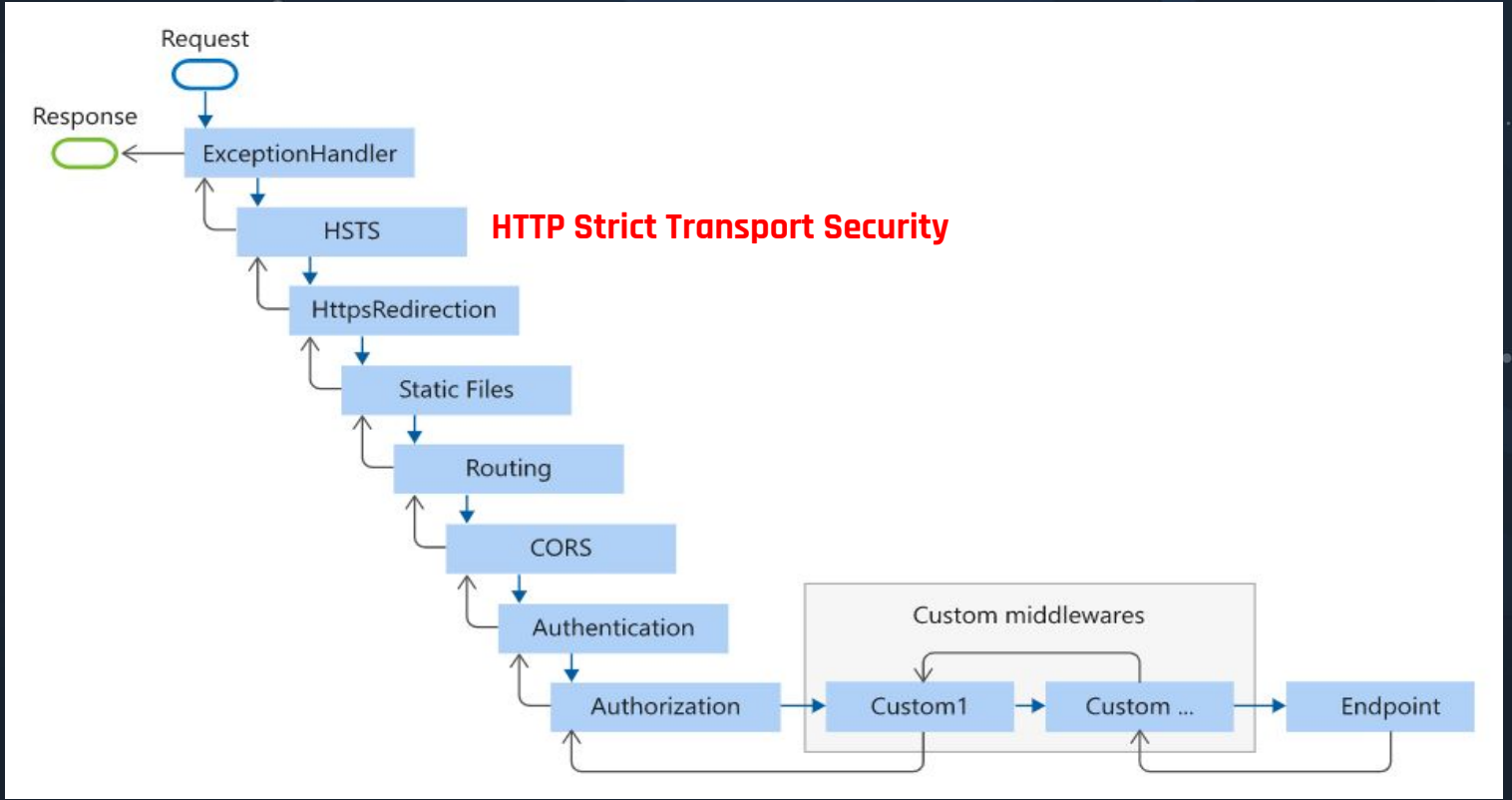
2) MIDDLEWARE

WHAT IS IT?

Software that handles requests and responses



MIDDLEWARE ORDER



3) HOST

WHAT IS IT?

Responsible for app startup and lifetime management

```
using (var host = WebHost.StartWith("http://localhost:8080", app =>
    app.Use(next =>
    {
        return async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        };
    })))
{
    Console.WriteLine("Use Ctrl-C to shut down the host...");
    host.WaitForShutdown();
}
```

4) ROUTING

Matches incoming http requests to endpoints and is based on top of middleware.

HOW TO DEFINE:

1. MapGet
2. MapPost
3. MapPut
4. MapDelete

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGet("/", async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    });
}
```

ENDPOINT

Endpoint is a **functionality**

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/hello/{name:alpha}", async context =>
    {
        var name = context.Request.RouteValues["name"];
        await context.Response.WriteAsync($"Hello {name}!");
    });
});
```

This Route Template Matches:

1. `/hello/Ryan`
2. Any url begins with `/hello/`

:alpha means only accept alphabetic characters

ROUTE CONSTRAINT

- int
- bool
- Datetime
- Decimal
- Double
- Float
- ...

Check stage

Set to use routing

Set authentication & authorization

Define body

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    // Matches request to an endpoint.
    app.UseRouting();

    // Endpoint aware middleware.
    // Middleware can use metadata from the matched endpoint.
    app.UseAuthentication();
    app.UseAuthorization();

    // Execute the matched endpoint.
    app.UseEndpoints(endpoints =>
    {
        // Configure the Health Check endpoint and require an authorized user.
        endpoints.MapHealthChecks("/healthz").RequireAuthorization();

        // Configure another endpoint, no authorization requirements.
        endpoints.MapGet("/", async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    });
}
```

5) HTTP REQUEST

Create request using

IHttpClientFactory

Create request

Create client

call

Process response

```
public class NamedClientModel : PageModel
{
    private readonly IHttpClientFactory _clientFactory;

    public IEnumerable<GitHubPullRequest> PullRequests { get; private set; }

    public bool GetPullRequestsError { get; private set; }

    public bool HasPullRequests => PullRequests.Any();

    public NamedClientModel(IHttpClientFactory clientFactory)
    {
        _clientFactory = clientFactory;
    }

    public async Task OnGet()
    {
        var request = new HttpRequestMessage(HttpMethod.Get,
            "repos/aspnet/AspNetCore.Docs/pulls");

        var client = _clientFactory.CreateClient("github");

        var response = await client.SendAsync(request);

        if (response.IsSuccessStatusCode)
        {
            using var responseStream = await response.Content.ReadAsStreamAsync();
            PullRequests = await JsonSerializer.DeserializeAsync
                    <IEnumerable<GitHubPullRequest>>(responseStream);
        }
        else
        {
            GetPullRequestsError = true;
            PullRequests = Array.Empty<GitHubPullRequest>();
        }
    }
}
```

6) STATIC FILES (Html, Css, Images, JavaScript)

Serve static files

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();
}
```

Static file authorization

```
[Authorize]
public IActionResult BannerImage()
{
    var file = Path.Combine(Directory.GetCurrentDirectory(),
                            "MyStaticFiles", "images", "banner1.svg");

    return PhysicalFile(file, "image/svg+xml");
}
```

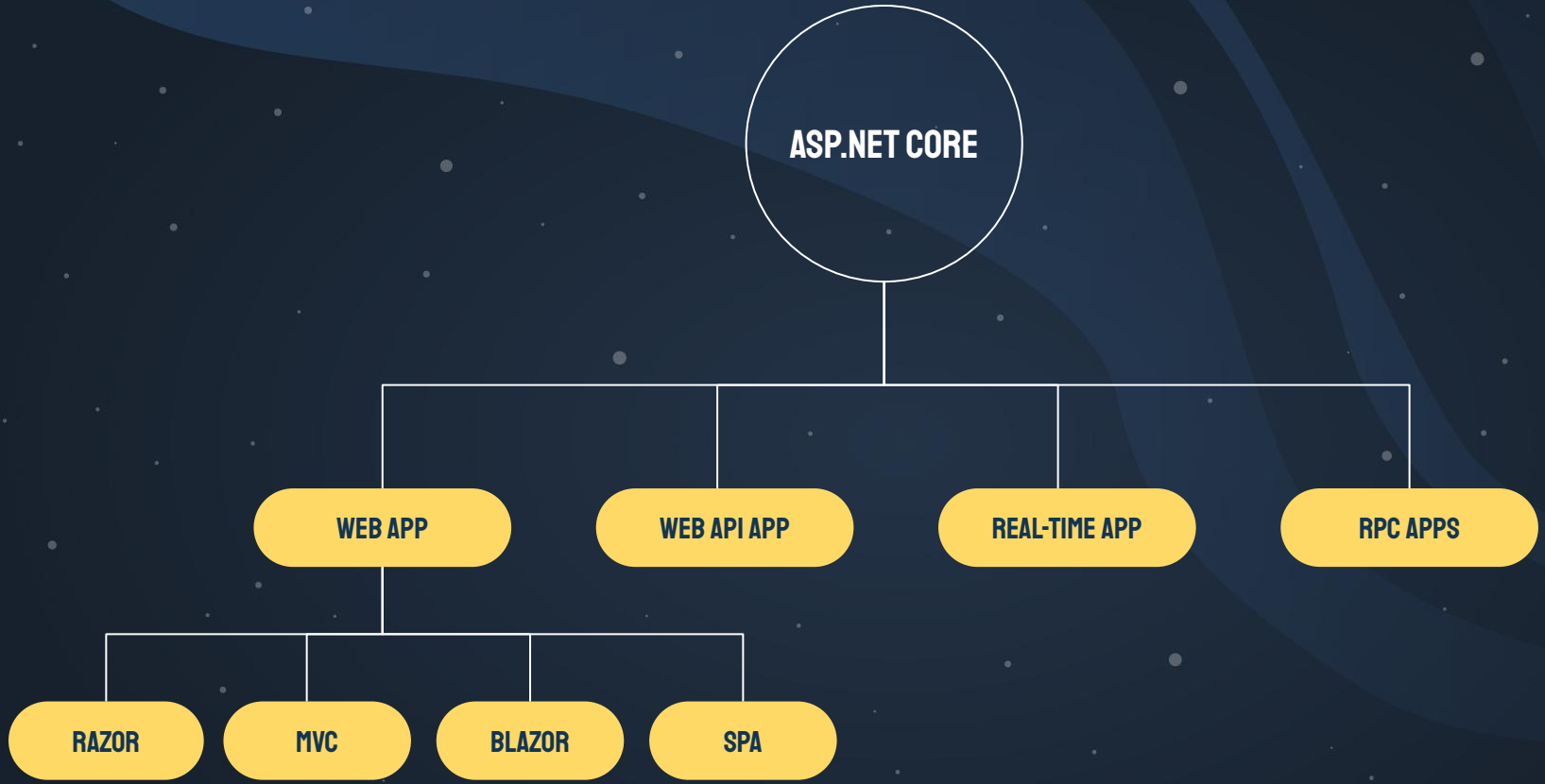
Enable directory browsing

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDirectoryBrowser();
}
```

03 APPS PROVIDED BY .NET CORE

ASP.NET Core provides several different types of apps:

- Web apps
- Web api apps
- Real-time apps
- Remote procedure call apps


ASP.NET CORE
WEB APP
WEB API APP
REAL-TIME APP
RPC APPS
RAZOR
MVC
BLAZOR
SPA

Create a new project

Recent project templates

A list of your recently accessed templates will be displayed here.

All languages | All platforms | All project types

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic Windows Linux macOS Console

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# Linux macOS Windows Cloud Service Web

Blazor App
Project templates for creating Blazor apps that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
Visual Basic Windows Cloud Web

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
C# Android iOS Linux macOS Windows Library

Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
Visual Basic Android iOS Linux macOS Windows Library

Azure Functions
A template to create an Azure Function project.
C# Azure Cloud

gRPC Service
A project template for creating a gRPC ASP.NET Core service using .NET Core.
C# Linux macOS Windows Cloud Service Web

Razor Class Library
A project template for creating a Razor class library.

Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name

RazorPagesMovie

Location

E:\web\ASP\ ...

Solution name

RazorPagesMovie

Place solution and project in the same directory

Create a new ASP.NET Core web application

.NET Core ASP.NET Core 3.1

Empty

An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application

A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular

A project template for creating an ASP.NET Core application with Angular

React.js

Get additional project templates

Authentication

No Authentication

Change

Advanced

Configure for HTTPS

Enable Docker Support

(Requires Docker Desktop)

Linux

Enable Razor runtime compilation

Author: Microsoft

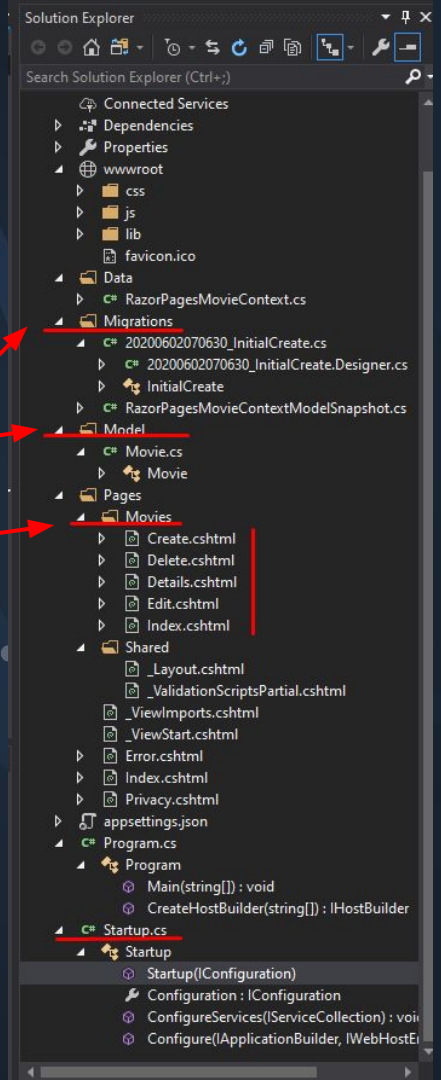
Source: .NET Core 3.1.4

Back Create

D WEB APPS (Razor pages, MVC, Blazor, SPA)

- **Razor Pages** can make coding page-focused scenarios easier and more productive than using controllers and views.

- Adding a data model
- Scaffold the model (CRUD)
- Initial migration (Add-Migration InitialCreate)
- Update the db with initial migration (Update-database)



- ASP.NET Core MVC framework is a lightweight, open source, highly testable presentation framework optimized for use with ASP.NET Core.

Routing

```
routes.MapRoute(name: "Default", template: "{controller=Home}/{action=Index}/{id?}");
```

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
      ...
    }
}
```

Model binding

converts client request data (form values, route data, query string parameters, HTTP headers) into objects that the controller can handle

Model validation

```
using System.ComponentModel.DataAnnotations;
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }
}
```

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    if (ModelState.IsValid)
    {
      // work with the model
    }
    // At this point, something failed, redisplay form
    return View(model);
}
```

Dependency Injection

```
@inject SomeService ServiceName

<!DOCTYPE html>
<html lang="en">
<head>
    <title>@ServiceName.GetTitle</title>
</head>
<body>
    <h1>@ServiceName.GetTitle</h1>
</body>
</html>
```

Filter

```
[Authorize]
public class AccountController : Controller
```

- **ASP.NET Core Blazor** is a framework for building interactive client-side web UI.

It enables to:

- Create rich interactive UIs using C# instead of JavaScript
- Share server-side and client-side app logic written in .NET
- Render the UI as HTML and CSS for wide browser support, including mobile browsers
- Integrate with modern hosting platforms, such as Docker

Advantages of using .NET for client-side web development:

- Write code in C# instead of JavaScript
- Leverage the existing .NET ecosystem of .NET libraries
- Share app logic across server and client
- Benefit from .NET's performance, reliability, and security
- Stay productive with Visual Studio on Windows, Linux, and macOS
- Build on a common set of languages, frameworks, and tools that are stable, feature-rich, and easy to use

Component

```razor
<h1 style="font-style:@headingFontStyle">@headingText</h1>

@code {
    private string headingFontStyle = "italic";
    private string headingText = "Put on your new Blazor!";
}
```

```
@using BlazorApp.Components

...

<BlazorApp.Components.MyCoolComponent />
```

Data binding

```
<input value="@CurrentValue"
    @onchange="@((ChangeEventArgs __e) => CurrentValue =
        __e.Value.ToString())" />

@code {
    private string CurrentValue { get; set; }
}
```

Event handling

```razor
<input type="checkbox" class="form-check-input" @onchange="CheckChanged" />

@code {
    private void CheckChanged()
    {
        ...
    }
}
```

Lifecycle

OnInitialized: when the component is initialized after getting params

SetParameters: Before params are set

OnParametersSet: After params are set

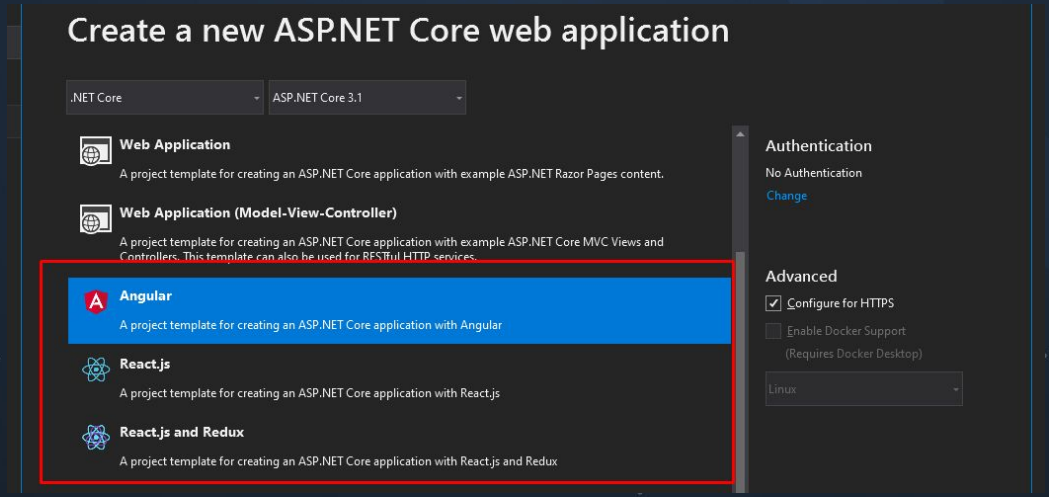
OnAfterRender: After component render

ShouldRender: Force refresh

- Single Page Apps (Angular, React, React with Redux)

JavaScript Services: to make ASP.NET Core as developers' preferred server side platform for building SPAs.

- Microsoft.AspNetCore.NodeServices (NodeServices)
- Microsoft.AspNetCore.SpaServices (SpaServices)



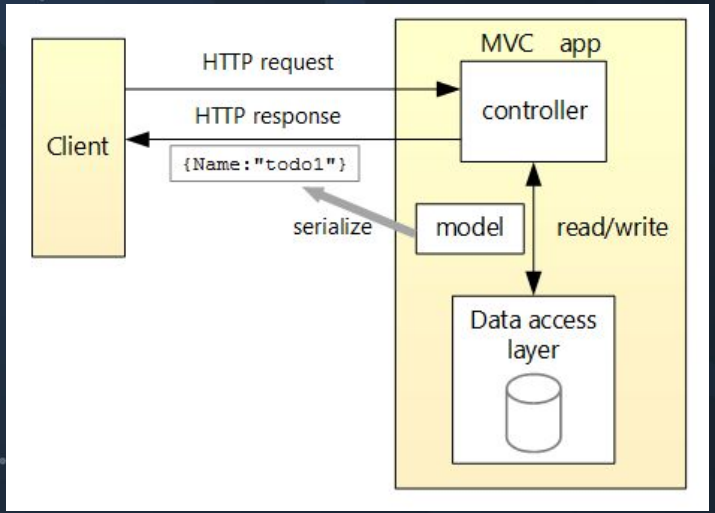
- Session and State management

State management approaches:

- Cookies : Http Cookies
- Session state : Http Cookies and Server-side app code (store user data)
- TempData : Http Cookies or session state
- Query strings : Http Query strings
- Hidden fields : Http form fields
- HttpContext.Items : Server-side app code
- Cache : Server-side app code

2) WEB API APPS (RESTful services by C#)

- ASP.NET Core supports creating RESTful services, also known as web APIs, using C#.



Steps to create Web API with ASP.NET Core:

- Create a Web API project
- Add a model class and a database context
- Scaffold a controller with CRUD methods
- Configure routing, URL paths, and return values

Web API additional features:

- Working with MongoDB
- Documentation using Swagger/OpenAPI (Swashbuckle.Asp.NetCore)

3) REAL-TIME APPS

- **ASP.NET Core SignalR ?**

  open-source library that simplifies adding real-time web functionality to apps.

  Real-time web functionality enables server-side code to push content to clients instantly.

When to use SignalR:

- High frequency updates from the server
- Dashboards and monitoring apps
- Collaborative apps
- Apps with notifications

Features of ASP.NET Core SignalR:

- Handles connection management automatically
- Sends messages to all connected clients simultaneously. For example, a chat room
- Sends messages to specific clients or groups of clients
- Scales to handle increasing traffic

- Transports & Hubs

Transport methods between server and client:

- WebSockets
- Server-Sent Events
- Long Polling
- Scales to handle increasing traffic

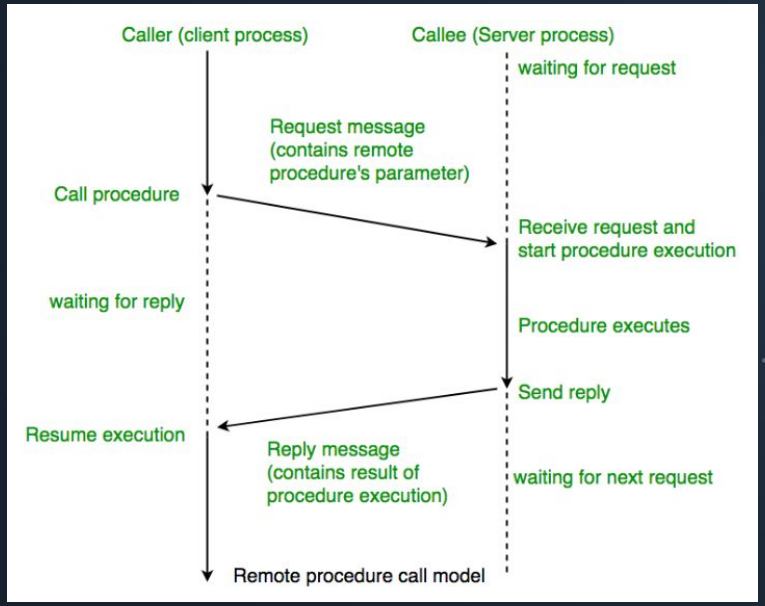
SignalR uses hubs to communicate between servers and clients.

Hub: a high-level pipeline that allows a client and server to call methods on each other.

4) REMOTE PROCEDURE CALL APPS

- **gRPC:** high-performance Remote Procedure Call (RPC) framework.

R
P
C



Main benefits of gRPC:

- **Modern, high-performance, lightweight RPC framework**
- **Contract-first API development, using Protocol Buffers by default, allowing for language agnostic implementations**
- **Tooling available for many languages to generate strongly-typed servers and clients**
- **Supports client, server, and bi-directional streaming calls**
- **Reduced network usage with Protobuf binary serialization**

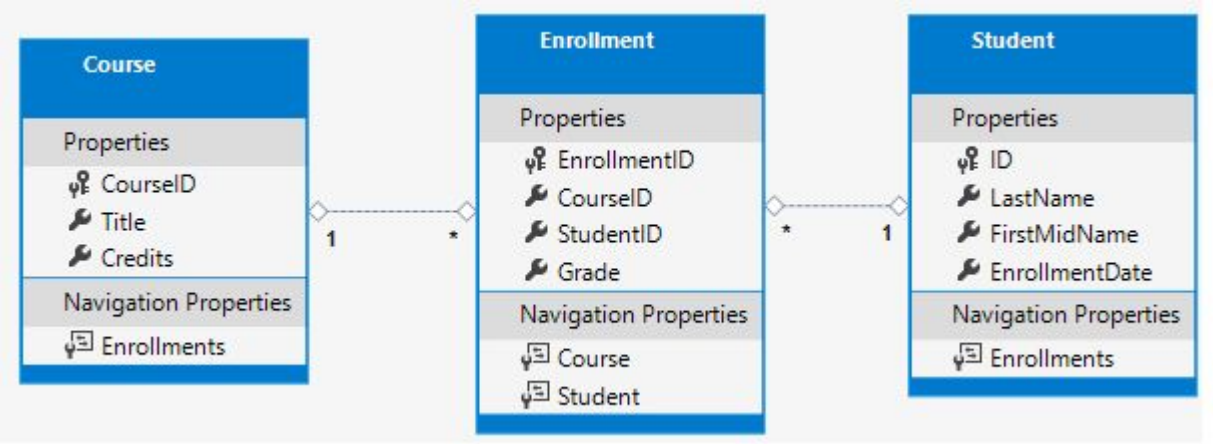
04 MAIN FEATURES

ASP.NET Core also provides other main features:

- Data access
- Host & deploy
- Security and Identity

ID DATA ACCESS

- EF Core(Entity Framework Core): lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology.



2) HOST AND DEPLOY

General steps:

- Deploy the published app to a folder on the hosting server
- Set up a process manager that starts the app when requests arrive and restarts the app after it crashes or the server reboots
  Linux: Nginx, Apache
  Windows: IIS, Windows Service
- For configuration of a reverse proxy, set up a reverse proxy to forward requests to the app

Reverse proxy: a special type of proxy server that hides the target server to the client.

How to publish to Azure app service:

- With Visual Studio
- With the CLI
- Visual Studio and Git
- Continuous integration and deployment with Azure pipelines

3) SECURITY AND IDENTITY

Identity methods:

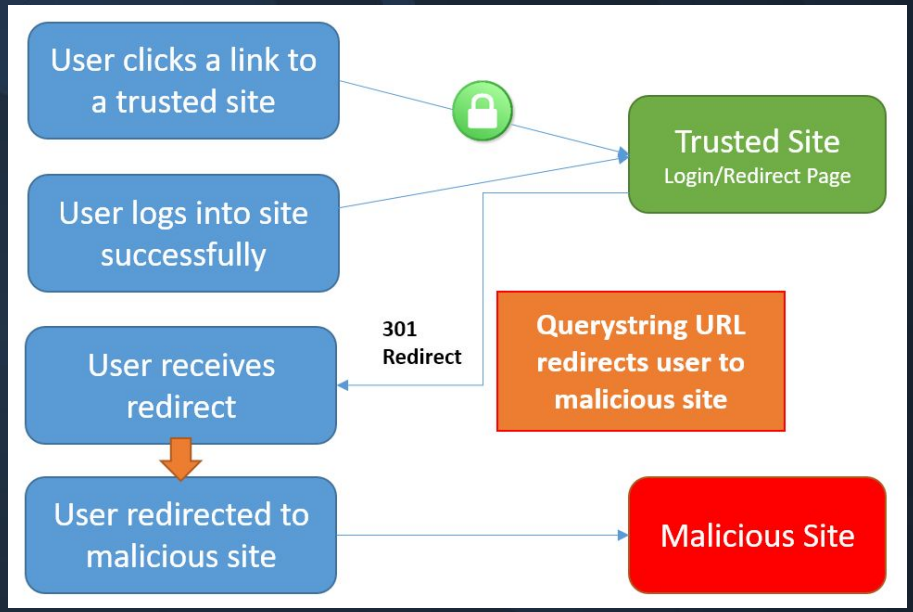
- **Built-in identity providers**
- **Third-party identity services: Facebook, Twitter, LinkedIn**

- Authentication & Authorization

authentication: a process in which a user provides credentials that are then compared to those stored in an operating system, database, app or resource

authorization: actions the user can perform to which objects inside that space (server, database, or app)

- **Security issues:**
  - Cross-Site Request Forgery (XSRF/CSRF)
  - open redirection attacks in ASP.NET Core
  - cross-Site Scripting (XSS) in ASP.NET Core



Open redirection attack

Thank you!