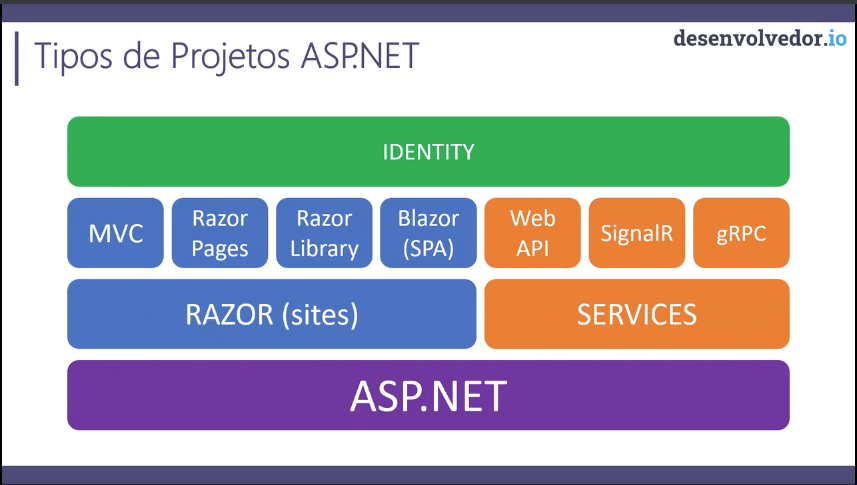
* Escolher versões com suporte estendido LTS
* NET Core 3.1 – 2019... de 2020 para frente uma versão por ano começando da .NET 5.0
* Suporte LTS é de 3 anos e GA (Genreal Avaiability) é de 18 meses
* Verificar sobre as novidades do ASP.NET
  + <https://learn.microsoft.com/pt-br/dotnet/core/whats-new/dotnet-9/overview>
  + <https://github.com/dotnet/core>
* Semântica de versionamento
  + Major: versão principal (breaking changes – o que pode “quebrar” de uma versão para outra)
  + Minor: features novas que não causa quebra
  + Patch: correção de bugs
* ASP.NET desenvolve aplicações fullstack
* .NET é multiplataforma
* .NET é mais rápido que Java Servlets e Nodejs
* .NET é Opensource
* .NET só vai usar o que realmente vai usar (pacotes)
* .NET Foundation: <https://dotnetfoundation.org/projects/current-projects>
* Preparar ambiente de instalação .net Core
  + Baixar e instalar o <https://dot.net>
* Comandos
  + Lista os tipos de projetos do .net: dotnet new --list
  + Opções dentro do projeto que você escolheu (mvc é o nome do projeto que é listado através do dotnet new --list): dotnet new mvc -h
  + Cria aplicação MVC com autenticação: dotnet new mvc -n <NOME\_PROJETO> --auth individual
  + Cria aplicação RAZOR com autenticação: dotnet new webapp -n <NOME\_PROJETO> --auth individual
  + Cria aplicação BLAZOR com autenticação: dotnet new blazorserver -n <NOME\_PROJETO> --auth individual
  + Cria WebAPI com autenticação: dotnet new webapi -n <NOME\_PROJETO> --auth individual
* .NET Standard pode ser usado se precisar de uma compatibilidade com outros Frameworks, caso contrário, utilize a versão normal o .NET
* Tipos de Projetos:

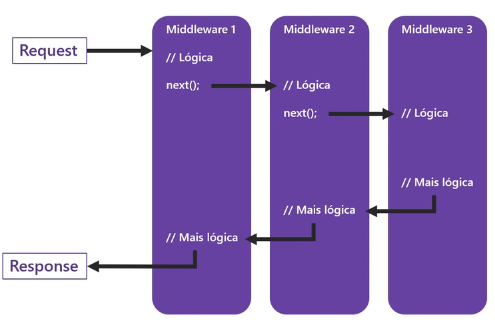


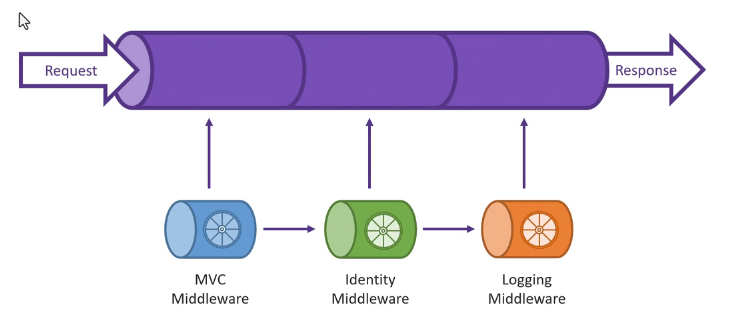
* Verificar a compatibilidade do .NET com o Visual Studio. No caso do .NET 8 é compatível somente com as versões 17 para cima do Visual Studio
* Para iniciar um projeto .NET Core com o Visual Studio Code:
  + Lista tipos de projeto: dotnet new list
  + Criar o projeto web de acordo com o que você quer na lista: dotnet new web -n “NomeProjeto”
  + Executa o projeto: dotnet run
* Para resolver o problema de certificado no Visual Studio 2022
  + dotnet dev-certs https --check
  + dotnet dev-certs https --clean
  + dotnet dev-certs https --check --trust
  + dotnet dev-certs https
  + dotnet dev-certs https --check
  + dotnet dev-certs https --check --trust
  + dotnet dev-certs https --trust
  + dotnet dev-certs https --check –trust
* Talvez seja necessário reiniciar a máquina para que o certificado seja carregado
* No Visual Studio Code não aconteceu porque ele executou via http
* lauchSettings.json: arquivo para “subir” a aplicação
  + Nos perfis pode deixar somente o https
  + Pode ser criado mais de um perfil
* No arquivo de projeto
  + ImplicitUsings: pode ter um arquivo global para ter todos os usings
  + Nullable: é recomendado o uso de desabilitado por conta do tratamento de objetos null que podem diferenciar de uma versão para outra.

Se precisar tratar o objeto, este será tratado no momento do desenvolvimento

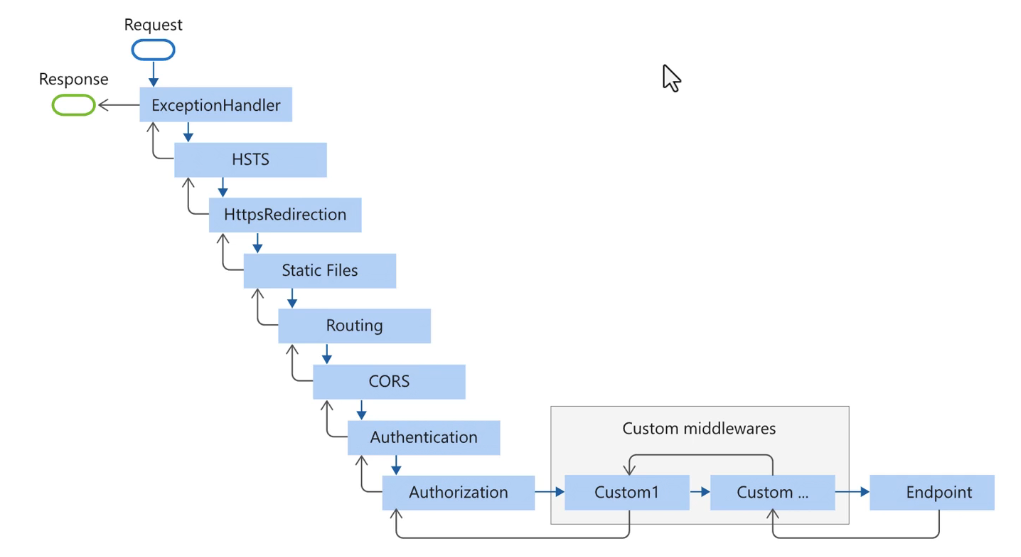
* No arquivo appsettings.json
  + Pode criar parâmetros
  + Configurações de Logging
    - Trace = 0
    - Debug = 1
    - Information = 2
    - Warning = 3
    - Error = 4
    - Critical = 5
    - None = 6
  + Isso tudo pode ser colocado nas configurações afim de saber mais informações
  + A cada opção que você escolhe, vai contemplar tudo o que estiver abaixo dela
  + Se atentar com os dois arquivos de configuração na hora de aplicar as configurações (appsettings.json e appsettings.Development.json)
  + Middlewares são componentes de software em uma aplicação ASP.NET. Estes componentes manipulam dados entre os requests e responses

Um middleware possui uma responsabilidade definida e pode trabalhar lado a lado com outros middlewares. Quando falamos do pipeline do ASP.NET estamos falando basicamente de Middleawares.





* As ordens do Middlewares importam... seguem uma ordem lógica
* Pipeline do ASP.NET (representado também na configuração da classe Program.cs)



* Blazor é um framework da Microsoft para construir aplicativos web interativos utilizando C# e .NET. Existem duas principais variantes do Blazor: Blazor Server e Blazor WebAssembly. Vamos entender as diferenças entre eles:
  + Blazor Server:
    - Modelo de execução: No Blazor Server, a aplicação é executada no servidor. Isso significa que o código C# é executado no servidor, e o HTML é renderizado no lado do cliente.
    - Comunicação: A comunicação entre o cliente e o servidor é realizada por meio de uma conexão em tempo real (WebSocket). As atualizações de UI são enviadas do servidor para o cliente.
    - Requisitos de servidor: Requer um servidor para hospedar a aplicação Blazor Server. O servidor gerencia o estado da aplicação e processa as interações do usuário.
  + Blazor WebAssembly:
    - Modelo de execução: No Blazor WebAssembly, a aplicação é executada diretamente no navegador do cliente. O código C# é compilado para WebAssembly, permitindo a execução de aplicações .NET no navegador.
    - Comunicação: Não há necessidade de uma conexão constante com o servidor para atualizações de UI. A comunicação com o servidor ocorre apenas quando é necessário buscar ou enviar dados.
    - Requisitos de servidor: A aplicação Blazor WebAssembly pode ser hospedada em qualquer servidor HTTP estático, pois o processamento principal ocorre no navegador do cliente.
* Escolha entre Blazor Server e Blazor WebAssembly:
  + Blazor Server: Recomendado quando a latência da rede entre o cliente e o servidor é aceitável e quando se deseja manter a lógica de negócios no lado do servidor.
  + Blazor WebAssembly: Adequado para cenários em que a aplicação precisa ser distribuída para os clientes e executada no navegador, reduzindo a dependência do servidor para interações frequentes.

