Web API com ASP .NET Core

API - Conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação para que outras aplicações consigam utilizar suas funcionalidades sem precisar conhecer detalhes da sua implementação.

- Integração entre sistemas;
- Intercâmbio de informações;
- Maior segurança nos dados;
- Controle de acesso;
- Fáceis de implementar e usar.

SOAP - É um protocolo (W3C) baseado em XML, usado para trocar informações entre aplicações no mesmo ou em diferentes sistemas operacionais

- Todas as mensagens SOAP usam o mesmo formato;
- É independente de protocolo (HTTP, SMTP, etc) e devem retornar XML;
- O Navegador não pode armazenar em cache uma solicitação concluída a uma API SOAP.

REST - É um estilo arquitetural sem estado que consiste de um conjunto coordenado de restrições arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema distribuído.

- Tratam cada entidade como um recurso que pode ser acessado por uma interface comum;
- O acesso é baseado no protocolo HTTP (GET, POST, PUT, DELETE);
- Trabalha com texto, XML, JSON, HTML;
- Invoca os serviços via URI.

ASP .NET é o framework da Microsoft para criar Web APIs aderentes ao estilo REST (RESTful) na plataforma .NET

REST - Conjunto de restrições

1. Ter um relacionamento cliente/servidor;
2. Não possuir monitoração de estado (stateless);
3. Ter uma interface uniforme;
4. Suportar o cache de dados e respostas;
5. Suportar um sistema em camadas.

Recurso - Tipo de informação que uma aplicação gerência que possui uma identificação única (URI)

Representação - Instantâneo do estado de um recurso em um ponto no tempo.

Interação sem estado (Stateless) - Comunicação entre cliente e o servidor sempre contém todas as informações necessárias para executar a solicitação.

Mensagens - as mensagens podem ser autodescritivas.

Endpoint é a URL na qual seu serviço pode ser acessado por uma aplicação cliente.

Exemplo Prático:

Sistema: Biblioteca.

Recursos:
- Todos os livros cadastrados
- Um livro cadastrado
- Todos os livros de um autor

URIs ou Endpoints:
- http://api/leonardo/livros
- http://api/leonardo/livros/1
- http://api/leonardo/autor/1/livros

Requisição HTTP:
- Request line: método HTTP + URI + protocolo HTTP (GET /api/livros HTTP/1.1)
- Headers: metadados que se enviam na requisição com informação (Accept: text/*)
- Body (opcional): Informação adicional enviada ao servidor (Dados de texto, dados JSON/ XML)

Resposta HTTP:
- Status line: indica o status da requisição (HTTP/1.1 200 OK)
- Headers: metadados que se enviam na resposta (Content-type: text-plain)
- Body (opcional): dados enviados pelo servidor (Dados de texto, dados JSON)

HTTPS é uma extensão segura do HTTP.

Transport Layer Security (TLS) - permite a comunicação criptografada entre um site e um navegador e substitui o protocolo SSL. Os sites que configuram um **certificado TLS** podem utilizar o **protocolo HTTPS** para estabelecer uma comunicação segura com o servidor. O objetivo do TLS é **tornar segura a transmissão de informações sensíveis** como dados pessoais, de pagamento ou de login.

ASP .NET e HTTPS
Ao criar o projeto ASP .NET são definidos os middlewares:
1. **HTTP de redirecionamento** - app.UseHttpsRedirection - redireciona uma requisição HTTP para HTTPS.
2. **HSTS** - app.UseHsts - Envia ao cliente um header Strict-Transport-Security(HSTS) - indica ao navegador que API deve ser acessada via https e não via http.

JSON é um formato compacto e leve de troca de dados simples e rápida entre sistemas, que utiliza texto legível a humanos, no formato atributo-valor.
Trabalhando com JSON
- Converter uma String (texto) para um objeto JSON
- Converter um objeto JSON para String
- Ler os dados dos atributos de um JSON
- Inserir e alterar os dados dos atributos de um JSON

JSON - ASP .NET

- A classe **HttpClient** é usada para enviar uma requisição **HTTP** e receber a resposta da solicitação (namespace **System.Net.Http**).
- Permite interagir com recursos em servidores remotos por meio dos principais métodos HTTP, como GET, POST, PUT, DELETE.
- Contém métodos de extensão para enviar e receber **conteúdo HTTP como JSON** (namespace **System.Net.Http.Json**).
- O namespace **System.Text.Json** contém classes para trabalhar com serialização e desserialização de dados no formato JSON.
    - JsonSerializer.Serialize(pessoa);
    - JsonSerializer.Deserialize<Pessoa>(json);

Modelo de maturidade de Richardson

Injeção de Dependências

- É uma técnica de programação usada para tornar uma classe independente de suas **dependências**
- A injeção de dependência **(DI)** é um padrão usado para implementar a **Inversão de Controle (IoC)** e assim reduzir o **acoplamento** entre os objetos.
- Ao aplicar a injeção de dependências fazemos com que um objeto forneça as dependências de outro objeto.
- Exemplo: uma classe cria instância de outra classe e utiliza métodos dessa classe, ela possui um forte acoplamento com essa outra classe, A classe possui a responsabilidade de saber como criar uma instância da outra classe e, assim, depende dessa instância. Dessa forma, a classe é dependente da outra classe e de suas dependências. Qualquer mudança feita na outra classe vai afetar a classe dependente. Precisamos fazer com que a classe dependa de uma abstração e não de uma implementação

Princípio SOLID SRP - dependência única

Configurar a injeção de dependência no **Container DI do .NET:**

servites.AddTransient<IPedido, Pedido>();

Tempo de vida útil dos serviços

1. Transient - criados cada vez que são solicitados
2. Scoped - criados uma vez por solicitação
3. Singleton - criados uma vez durante a vida útil do aplicativo

Controller

- É uma classe que é responsável por lidar com as requisições HTTP e atua como um intermediário entre as requisições feitas por clientes (como navegadores da web, aplicativos móveis ou outros serviços) e a lógica de negócios que processa essas requisições.

Entity Framework Core

- Tecnologia com qual os dados serão tratados (banco de dados).
- Abordagem **Code-First** - parte das entidades para criar as tabelas e o banco de dados.
- Padrão Repositório - desacoplar o acesso aos dados da aplicação.

O **Swagger UI** é uma interface gráfica gerada a partir de um arquivo OpenAPI que permite aos usuários testar e explorar API diretamente no navegador.

Criando uma Web API - Definições

Escopo:

- Criar **serviço RESTful** que permita que aplicativos clientes gerenciem os recursos;
- Criar **endpoints** para criar, ler, editar e excluir recursos;
- Definir o que será armazenado;
- Definir o formato das respostas e códigos de resposta

Persistência de dados:

- Banco de dados;
- Serviço gerenciador;
- Tecnologia com qual os dados serão tratados (Entity Framework Core);
- Abordagem.

Nomenclatura.

Estrutura do Projeto (**Arquitetura em camadas**):

- **Camada de apresentação**: a camada de nível mais superior da aplicação é a interface com o usuário. A principal função da interface é traduzir tarefas e resultados para alguma coisa que o usuário possa compreender.
- **Camada de negócio**: Coordena a aplicação, processa comandos, toma decisões lógicas, faz avaliações e realiza cálculos. Também transfere e processa dados entre as camadas de dados e de apresentação.
- **Camada de acesso a dados**: A informação é armazenada e retornada de uma fonte de dados. A informação é então passada de volta para a camada de negócios para processamento e então apresentada ao usuário.
    - **Layers**: Separação lógica.
    - **Tiers**: Separação física.
    - n-layers não implica em n-tiers e vice-versa.
    - Criar **projetos distintos** em Class Library (muito complexo ou distribuído remotamente em servidores distintos).
    - Separar por **pastas** no mesmo projeto.

1. **Presentation layer**: Angular, Views, Mobile, Desktop.
2. **Business logic**/**Application core**: repositórios, domínio, serviços.
3. **Data access**/**Persistence**: EF core, MySQL.

O **roteamento** em Web APIs com controladores é baseado em atributos:

- [Route("[controller]")] (rota padrão a todos os endpoints).
- [HttpGet].
- [HttpGet("primeiro")] .
- [HttpPut("{id:int}")].

As **restrições de rota** para validação de entradas nos endpoints devem ser usadas apenas para distinguir entre duas rotas parecidas.

Ao acessar banco de dados, realizar uma operação de IO, efetuar um cálculo complexo, no geral, operações que demandam muito tempo, justifica-se a utilização de um **método action assíncrono.** Já que a perda de desempenho. O ganho é na execução paralela, podendo atender a mais de uma requisição de uma vez, mas a requisição específica não ficará mais rápida em hipótese alguma

Model Binding é o recurso que nos permite mapear dados de uma requisição HTTP para os parâmetros de uma action de um controlador.