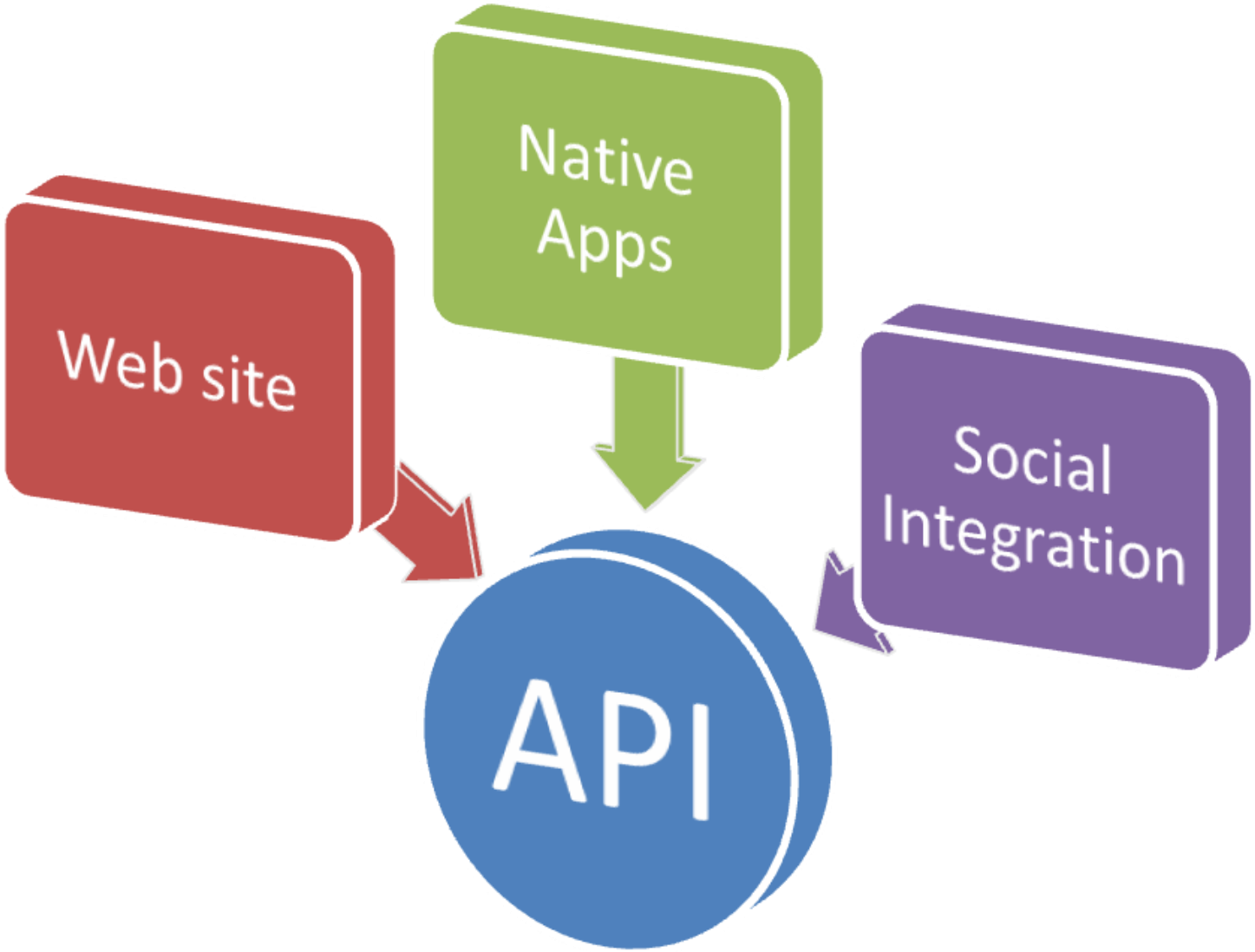
Rest API

Fabrício Tonetto Londero
Ricardo Frohlich da Silva

API, REST e RESTful

- Por vezes, diferentes aplicativos em diferentes plataformas precisam acessar os mesmos conjuntos de dados
  - Uma alternativa para a não reimplementação para diferentes plataformas, é a utilização de APIs
- Uma API é um conjunto de definições e protocolos para construir e integrar software de aplicativo.
- Podemos pensar em uma API como um mediador entre os usuários ou clientes e os recursos ou serviços da web que desejam obter
- *Application Programming Interface* (Em português, significa Interface de Programação de Aplicações


Native Apps
Web site
Social Integration
API

API

- Desta forma, entendemos que as APIs permitem uma interoperabilidade entre aplicações.
- Em outras palavras, a comunicação entre aplicações e entre os usuários.
- Também pode aparecer com o nome WebServices

Representações

- Existem atualmente, três formas de representação
  - XML
  - JSON
  - YAML

Representação XML

```xml
<endereco>
  <rua>
    Rua Recife
  </rua>
  <cidade>
    Paulo Afonso
  </cidade>
</endereco>
```

Representação JSON

```
{ endereco:
  {
  rua: Rua Recife,
  cidade: Paulo Afonso
  }
}
```

Representação YAML

```
endereco:
rua: rua Recife
cidade: Paulo Afonso
```

REST

- API REST é uma forma de implementação de APIs que seguem algumas restrições arquitetônicas.
- REST significa **Representational State Transfer**. Em português, **Transferência de Estado Representacional**
- Uma API transfere uma representação do estado do recurso para o solicitante ou terminal. Essas informações, ou representação, são fornecidas em um dos vários formatos via HTTP (json, xml...)
- cabeçalhos e parâmetros também são importantes nos métodos HTTP de uma solicitação HTTP da API REST
- Existem cabeçalhos de solicitação e cabeçalhos de resposta, cada um com suas próprias informações de conexão HTTP e códigos de status.

RESTful

É apenas uma API que implementa todas as “regras” de uma API REST.

sistemas que utilizam os princípios REST são chamados de RESTful

RESTful - critérios

Uma arquitetura cliente-servidor composta de clientes, servidores e recursos, com solicitações gerenciadas por meio de **HTTP**.

Comunicação ***Stateless*** entre cliente-servidor, ou seja, as informações do cliente não são armazenadas nas solicitações GET e cada pedido é separado e desconectado.

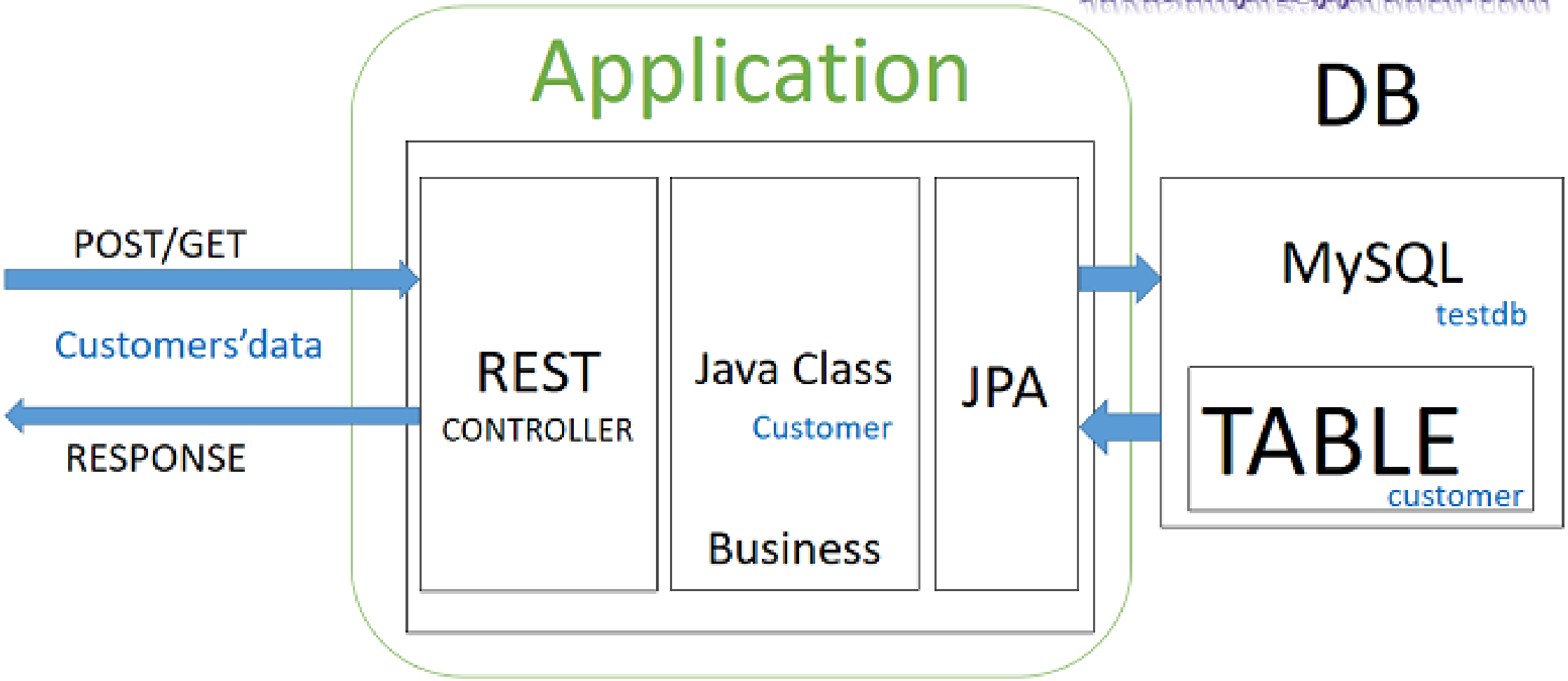
Dados armazenáveis em cache que otimizam as interações cliente-servidor.

informações transferidas em um formato padrão

os recursos solicitados são identificáveis e separados das representações enviadas ao cliente, permitindo que o cliente manipule as informações e recursos

RESTful - critérios

- Informação disponível ao cliente, de como acessar os demais recursos(links) da API
- Um sistema em camadas que organiza cada tipo de servidor (os responsáveis pela segurança, balanceamento de carga, etc.) envolvia a recuperação das informações solicitadas em hierarquias, invisíveis para o cliente.
- Code-on-demand (opcional): a capacidade de enviar código executável do servidor para o cliente quando solicitado, estendendo a funcionalidade do cliente.



HTTP

- O Hypertext Transfer Protocol, sigla HTTP (em português Protocolo de Transferência de Hipertexto) é um protocolo de comunicação.
- é a base para a comunicação de dados da World Wide Web
- O protocolo HTTP faz a comunicação entre o cliente e o servidor por meio de mensagens.
  - O cliente envia uma mensagem de requisição de um recurso e o servidor envia uma mensagem de resposta ao cliente com a solicitação.
- Uma mensagem, tanto de requisição quanto de resposta, é composta por uma linha inicial, nenhuma ou mais linhas de cabeçalhos, uma linha em branco obrigatória finalizando o cabeçalho e por fim o corpo da mensagem, opcional em determinados casos.

Cabeçalho

O cabeçalho da mensagem (header) é utilizado para transmitir informações adicionais entre o cliente e o servidor.

Existem quatro tipos de cabeçalhos que poderão ser incluídos na mensagem os quais são: general-header, request-header, response-header e entity-header.

Corpo

- . Em uma mensagem de resposta, o corpo da mensagem é o recurso que foi requisitado pelo cliente, ou ainda uma mensagem de erro, caso este recurso não seja possível.

Exemplo	Descrição
text/plain	Arquivo no formato texto (ASCII)
text/html	Arquivo no formato HTML, utilizado como padrão para documentos Web
Image/gif	Imagem com o formato GIF
Image/jpeg	Imagem com o formato JPEG
application/zip	Arquivo compactado
application/json	Arquivo no formato JSON
application/xml (ou text/xml)	Arquivo no formato XML

Métodos HTTP

- O protocolo HTTP define oito métodos que indicam a ação a ser realizada no recurso especificado. Também são conhecidos como Verbos HTTP.
  - GET
  - HEAD
  - POST
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
  - CONNECT

Métodos HTTP em API RESTFul

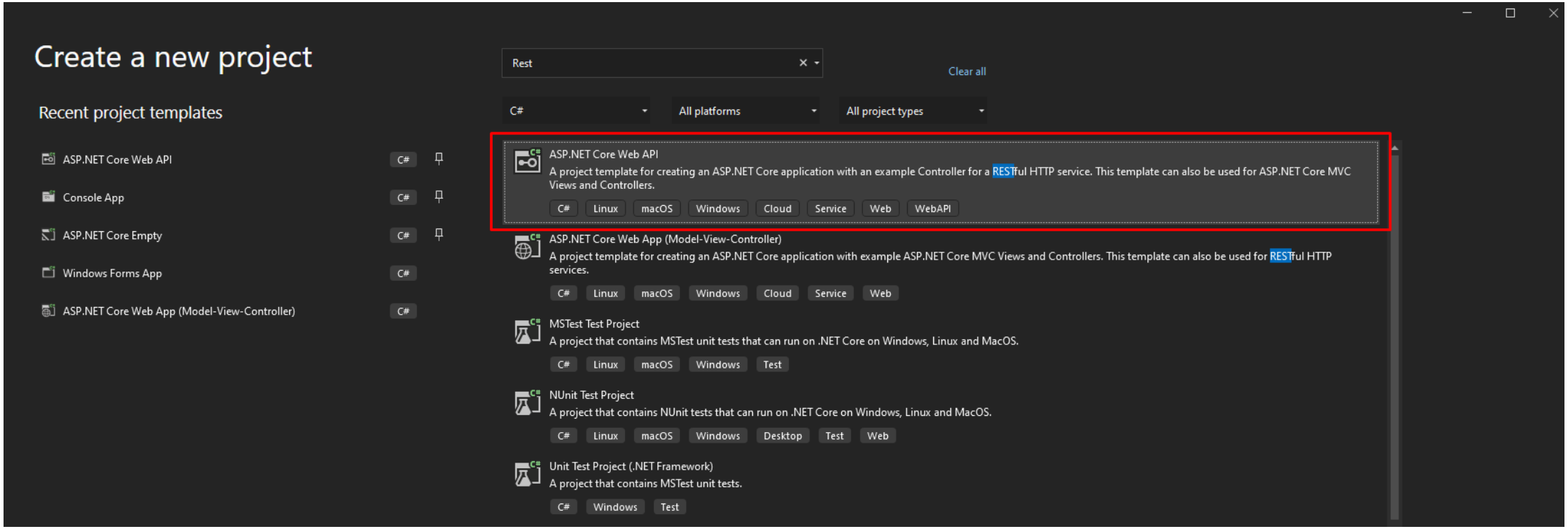
- Os métodos HTTP mais comuns e utilizados são:
  - POST, GET, PUT, PATCH e DELETE
  - Eles correspondem às operações de criação, leitura, atualização e exclusão (ou CRUD), respectivamente.

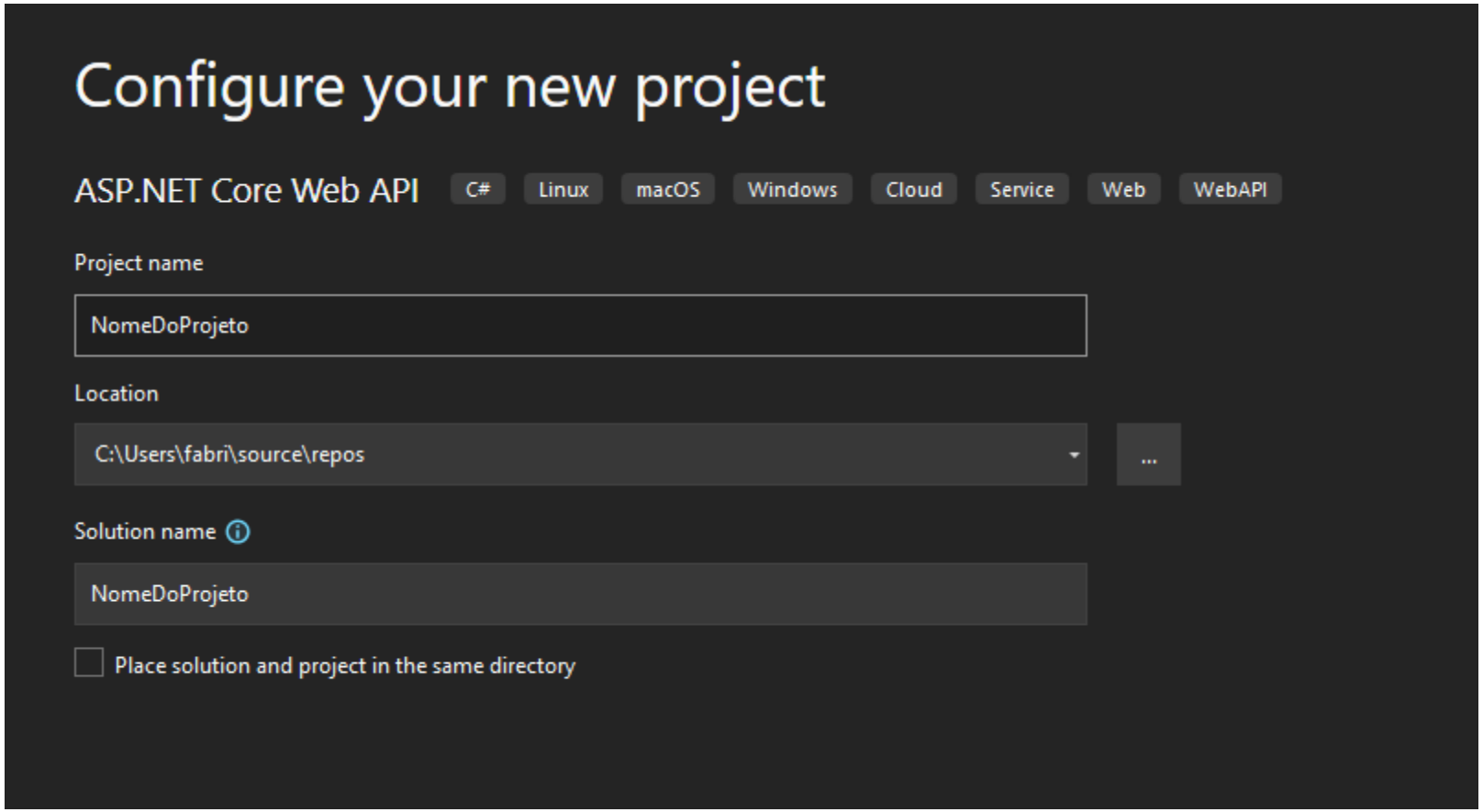
HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

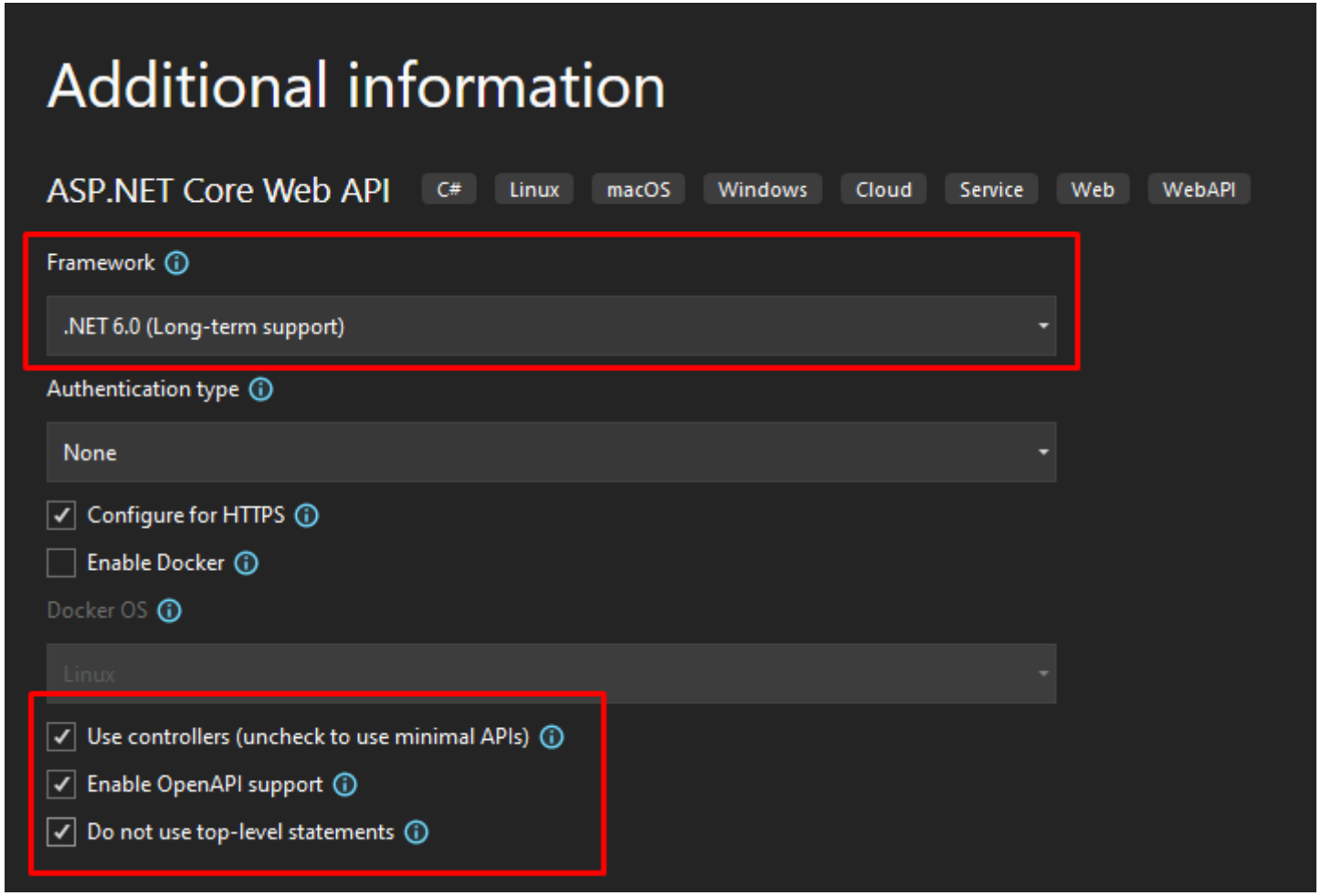
Referencias

- https://becode.com.br/o-que-e-api-rest-e-restful/
- https://www.redhat.com/en/topics/api/what-is-a-rest-api
- https://www.toptal.com/spring/spring-boot-oauth2-jwt-rest-protection
- https://betterprogramming.pub/secure-a-spring-boot-rest-api-with-json-web-token-reference-to-angular-integration-e57a25806c50
- https://www.restapitutorial.com/lessons/httpmethods.html#:~:text=The%20primary%20or%20most%2Dcommonly,but%20are%20utilized%20less%20frequently.

ApiRest com .Net

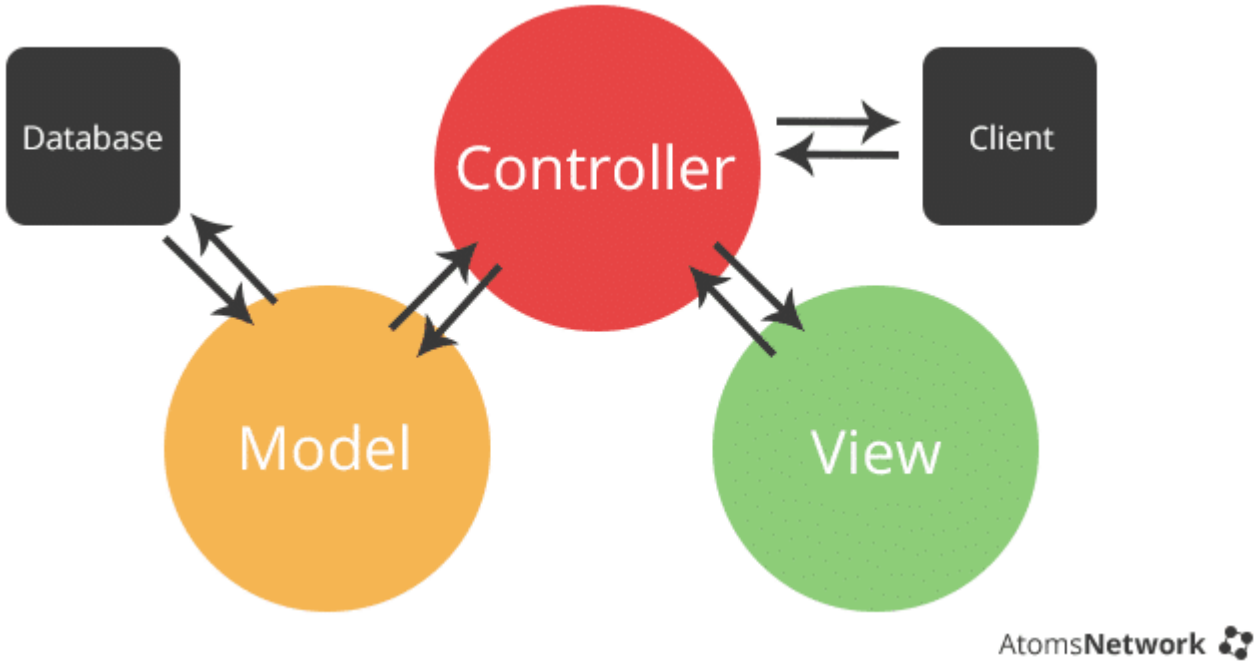


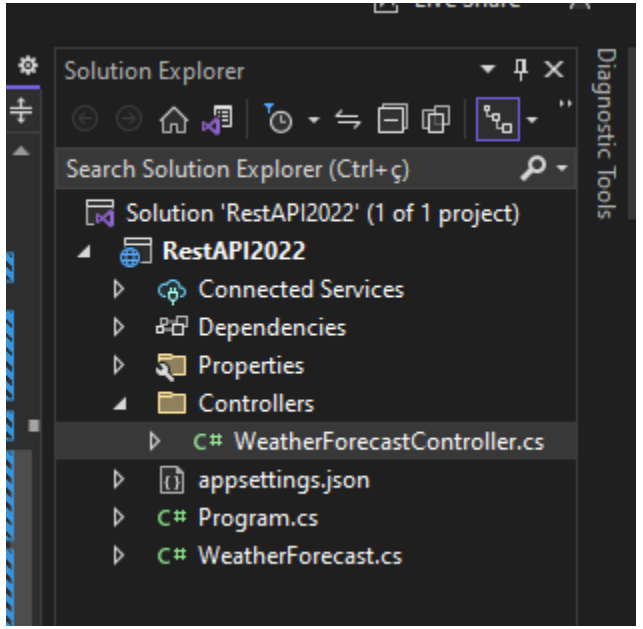

Configure your new project
ASP.NET Core Web API
C#
Linux
macOS
Windows
Cloud
Service
Web
WebAPI
Project name
NomeDoProjeto
Location
C:\Users\fabri\source\repos
...
Solution name
NomeDoProjeto
Place solution and project in the same directory


Additional information
ASP.NET Core Web API
C#
Linux
macOS
Windows
Cloud
Service
Web
WebAPI
Framework
.NET 6.0 (Long-term support)
Authentication type
None
Configure for HTTPS
Enable Docker
Docker OS
Linux
Use controllers (uncheck to use minimal APIs)
Enable OpenAPI support
Do not use top-level statements

.Net RestAPI

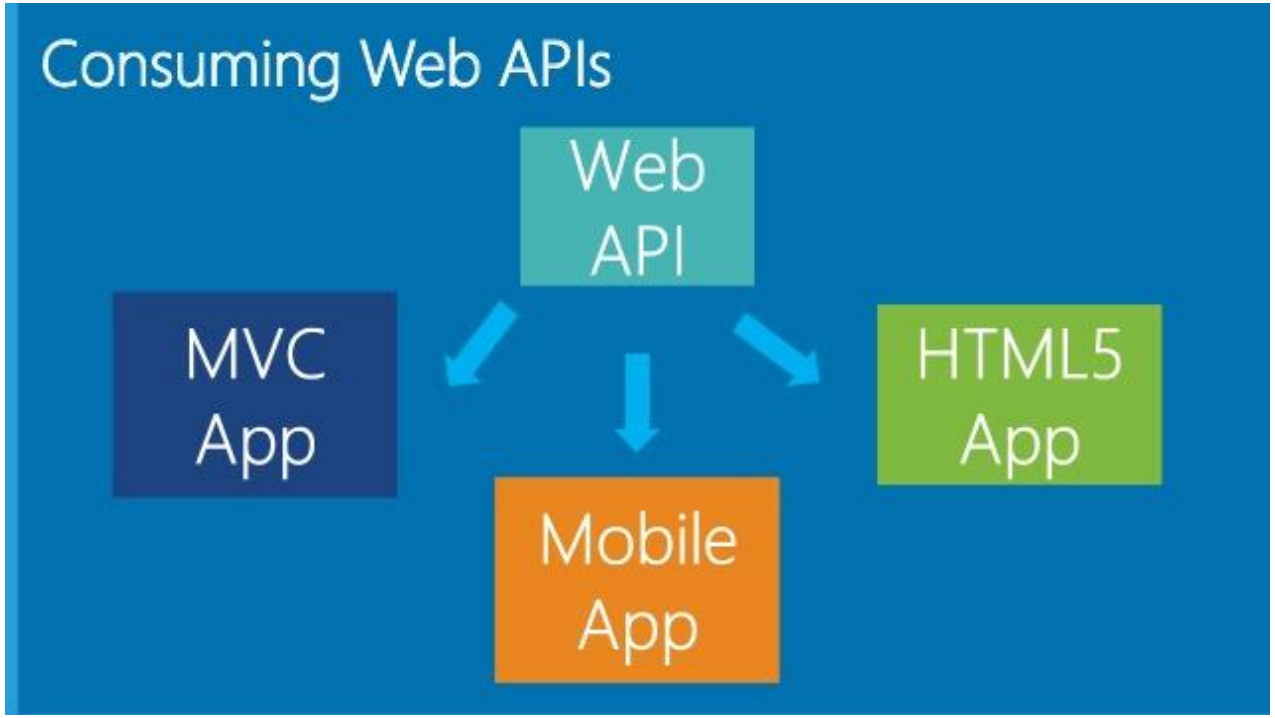
- Segue o padrão MVC





.Net RestAPI

- A Controller é onde fica a regra de negócios
- É por ela que o sistema interage com as Models, que por sua vez, interage com o banco de dados
- A View é a parte visual, que o usuário interage
- Em RestAPI, a View é um outro sistema, podendo ser em qualquer linguagem de programação, e inclusive ser vários projetos diferentes
- Toda a comunicação externa se da pelas Controllers



.Net RestAPI

- O projeto recém criado, possui uma Model simples denominada WeatherForecast, e uma controller.
- Uma controller é uma classe que herda de ControllerBase
- A notação [ApiController] define a classe como uma Controller do tipo API, pois uma controller pode ser do padraõ web com MVC.
- A notação [Route(“”)] define parte do caminho explicito na URL das requisições que serão direcionadas aos métodos (endPoints) desta controller, referente ao a controller em questão. OBS: Não pode se repetir em mais de uma controller.
- O valor entre [ ] da Route se refere ao nome original da controller, pode ser colocado qualquer valor, desde que remova os [ ]

Estrutura da url:
http://localhost:porta/controller/endpoint

```
[ApiController]
[Route("[controller]")]
3 references
public class WeatherForecastController : ControllerBase
{
```

.Net RestAPI - EndPoint

- Dentro da controller, os métodos são denominados EndPoints
- Cada endPoint possui uma notação informando qual o verbo http associado a ele (POST, GET, PUT, DELETE...), como HttpGet no exemplo
- Depois do verbo, segue o nome do endPoint, como sera representado na URL completa
- O nome do EndPoint só pode se repetir se o verbo http foi diferente.

```
[HttpGet("hello")]
0 references
public string HelloWorld()
{
    return "Hello World API Net6";
}
```

.Net RestAPI - EndPoint

- O método tem um retorno, normalmente do tipo string, void ou `async Task<IActionResult>`
- Independente da forma utilizada, o retorno por padrão, é definido como uma string JSON

```
[HttpGet("hello")]
0 references
public string HelloWorld()
{
    return "Hello World API Net6";
}
```

```
[HttpGet]
[Route("pessoas")]
0 references
public async Task<IActionResult> getAllAsync(
    [FromServices] Contexto contexto)
{
```

EndPoints

- Exemplo de notações:
  - [Authorize] -> exige autenticação, caso configurado
  - [AllowAnonymous] -> quando a autenticação esta configurada no projeto, essa notação torna o endPoint livre dela

```
[Authorize]
[HttpGet]
[Route("pessoas/{id}")]
0 references
public async Task<IActionResult> getByIdAsync(
    [FromServices] Contexto contexto,
    [FromRoute] int id
    )
{
    var pessoa = await contexto
        .Pessoas.AsNoTracking()
        .FirstOrDefaultAsync(p => p.id == id);

    return pessoa == null ? NotFound() : Ok(pessoa);
}
```

EndPoints

- Parâmetros na rota:
  - Valores definidos dentro de { } são passados pela URL e devem ser recebidos como parâmetros no endPoint, com a notação [FromRoute]
  - A notação [FromService] refere-se a injeção de dependência, definido na classe Program.cs

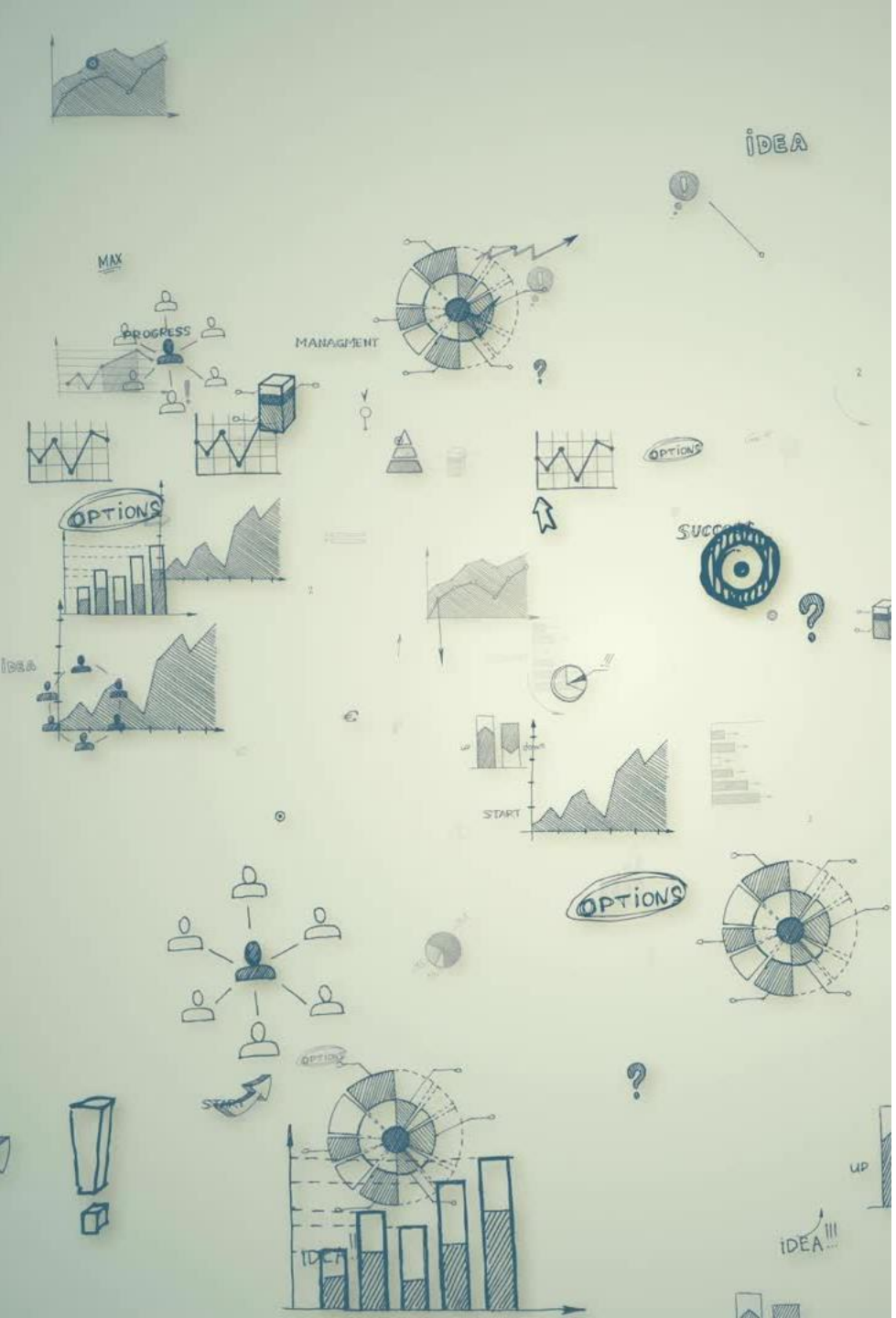
```
[Authorize]
[HttpGet]
[Route("pessoas/{id}")]
0 references
public async Task<IActionResult> getByIdAsync(
    [FromServices] Contexto contexto,
    [FromRoute] int id
    )
{
    var pessoa = await contexto
        .Pessoas.AsNoTracking()
        .FirstOrDefaultAsync(p => p.id == id);

    return pessoa == null ? NotFound() : Ok(pessoa);
}
```

Program.cs

- Na Program.cs, podemos configurar várias propriedades do nosso projeto Rest
- A linha abaixo, utiliza a classe Contexto (do Entity, por exemplo) de forma que o framework fica responsável por controlar a criação e destruição das instancias dessa classe
- Quando precisamos utilizar, criarmos um parâmetro no EndPoint com a notação [FromService]
- Por padrão, o Swagger vem configurado e pronto para ser utilizado

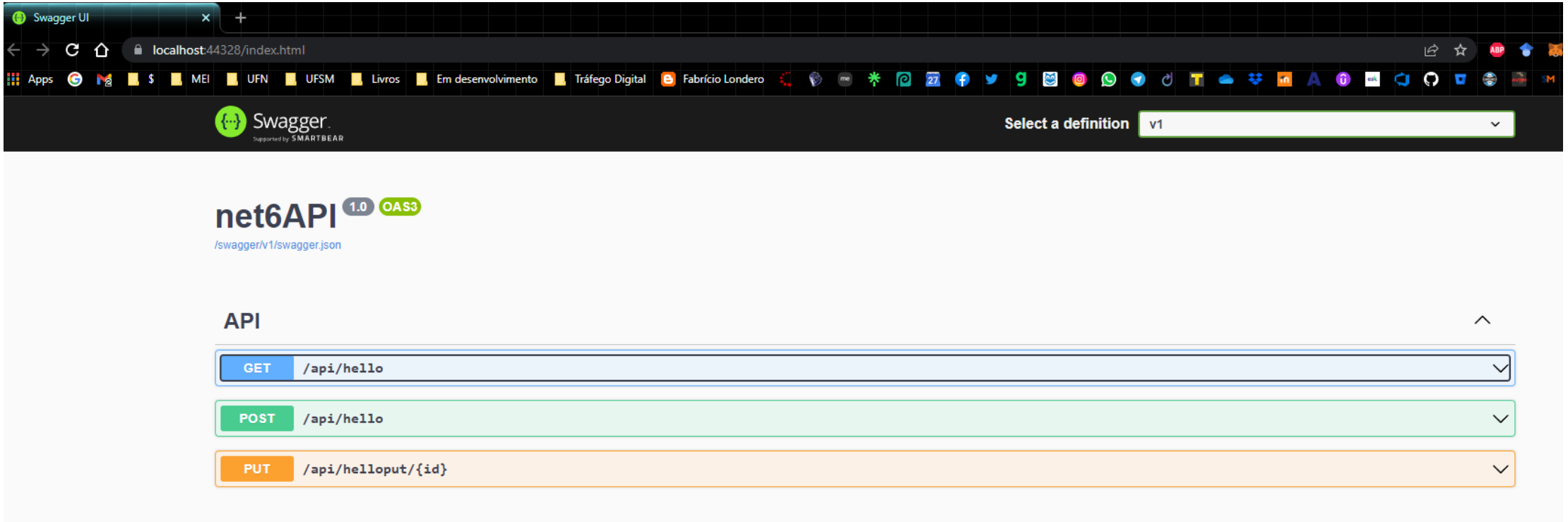
```
builder.Services.AddDbContext<Contexto>();//libera a ijeção de dependencia
```



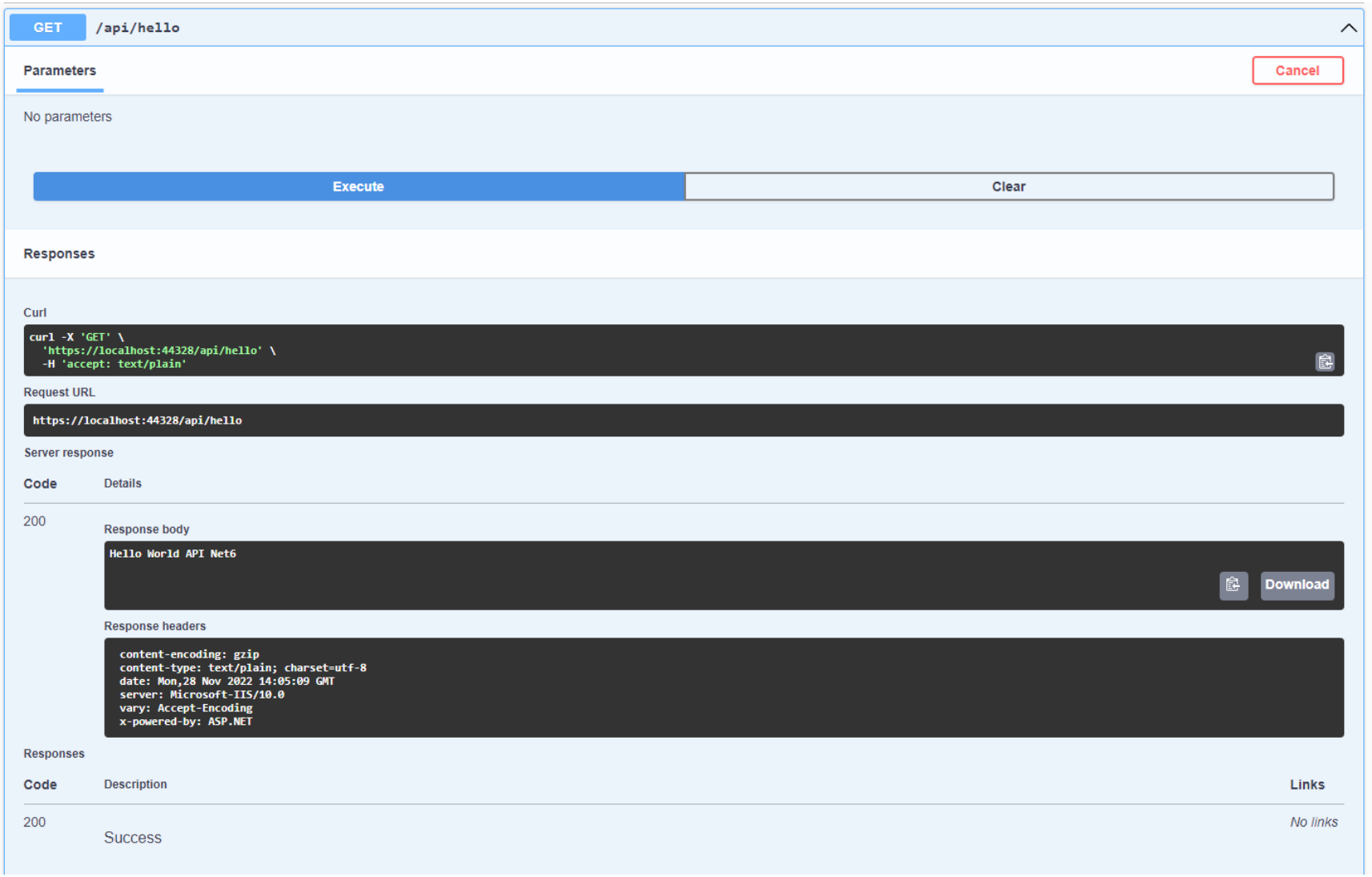
Swagger

- Identifica os EndPoints das Controllers do projeto e cria uma interface gráfica para testar a API
- Funciona do ambiente de desenvolvimento

Swagger


Swagger UI
localhost:44328/index.html
Apps
$
MEI
UFN
UFSM
Livros
Em desenvolvimento
Tráfego Digital
Fabrício Londero
Swagger
Supported by SMARTBEAR
Select a definition
v1
net6API
1.0
OAS3
/swagger/v1/swagger.json
API
GET
/api/hello
POST
/api/hello
PUT
/api/helloput/{id}

Swagger


GET
/api/hello
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'https://localhost:44328/api/hello' \
-H 'accept: text/plain'
Request URL
https://localhost:44328/api/hello
Server response
Code
Details
200
Response body
Hello World API Net6
Download
Response headers
content-encoding: gzip
content-type: text/plain; charset=utf-8
date: Mon,28 Nov 2022 14:05:09 GMT
server: Microsoft-IIS/10.0
vary: Accept-Encoding
x-powered-by: ASP.NET
Responses
Code
Description
Links
200
Success
No links

Exercício 1

- Crie uma API que com diferentes EndPoints
  - Um que retorne o seu nome
  - Outro que retorne a sua idade
  - Outro que receba o nome e o retorne
  - Outro que receba o nome e a idade, e mostre a frase: “Fulano é maior de idade”
  - Teste no Swagger

ApiRest + Entity: Consulta

```
[HttpGet]
[Route("pessoas")]
0 references
public async Task<IActionResult> getAllAsync(
    [FromServices] Contexto contexto)
{
    var pessoas = await contexto
        .Pessoas
        .AsNoTracking()//só pode ser utilizado em consultas - altamente recomendado por questões de desempenho
        .ToListAsync();

    return pessoas == null ? NotFound() : Ok(pessoas);
}
```

ApiRest + Entity: Consulta com filtro

```
[HttpGet]
[Route("pessoas/{id}")]
0 references
public async Task<IActionResult> getByIdAsync(
    [FromServices] Contexto contexto,
    [FromRoute] int id
    )
{
    var pessoa = await contexto
        .Pessoas.AsNoTracking()
        .FirstOrDefaultAsync(p => p.id == id);

    return pessoa == null ? NotFound() : Ok(pessoa);
}
```

ApiRest + Entity: Cadastro

```
[HttpPost]
[Route("pessoas")]
0 references
public async Task<IActionResult> PostAsync(
    [FromServices] Contexto contexto,
    [FromBody] Pessoa pessoa
    )
{
    if (!ModelState.IsValid)
    {
        return BadRequest();
    }

    try
    {
        await contexto.Pessoas.AddAsync(pessoa);
        await contexto.SaveChangesAsync();
        return Created($"api/pessoas/{pessoa.id}", pessoa);
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

```
[HttpPut]
[Route("pessoas/{id}")]
0 references
public async Task<IActionResult> PutAsync
    (
        [FromServices] Contexto contexto,
        [FromBody] Pessoa pessoa,
        [FromRoute] int id
    )
{
    if (!ModelState.IsValid)
        return BadRequest("Model inválida");

    var p = await contexto.Pessoas
        .FirstOrDefaultAsync(x => x.id == id);

    if(p == null)
        return NotFound("Pessoa não encontrada!"

    try
    {
        p.nome = pessoa.nome;

        contexto.Pessoas.Update(p);
        await contexto.SaveChangesAsync();
        return Ok(p);
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

ApiRest + Entity: Alteração

ApiRest + Entity: Remoção

```
[HttpDelete]
[Route("pessoas/{id}")]
0 references
public async Task<IActionResult> DeleteAsync(
        [FromServices] Contexto contexto,
        [FromRoute] int id)
{
    var p = await contexto.Pessoas.FirstOrDefaultAsync(x => x.id == id);

    if (p == null)
        return BadRequest("Pessoa não encontrada");

    try
    {
        contexto.Pessoas.Remove(p);
        await contexto.SaveChangesAsync();

        return Ok();
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

Exercício 2

- Faça o CRUD completo para o cadastro de alunos, cursos e matrícula

Exercício 3

- Pesquise e implemente autenticação via JWT