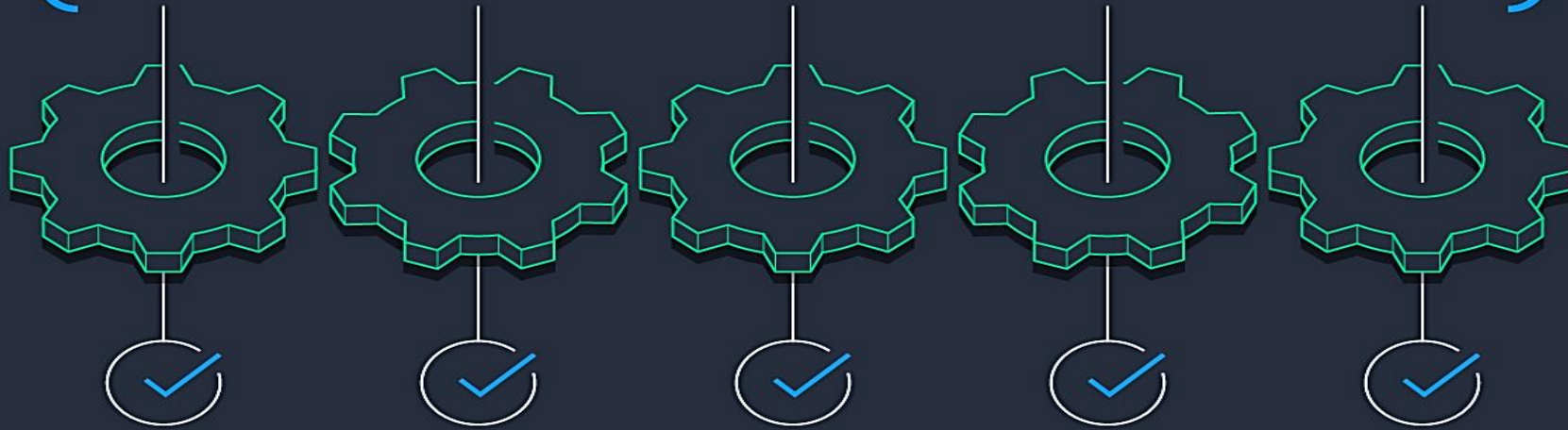


{ RESTful API }



Camada de Serviços (REST – RESTFULL)

Programação III

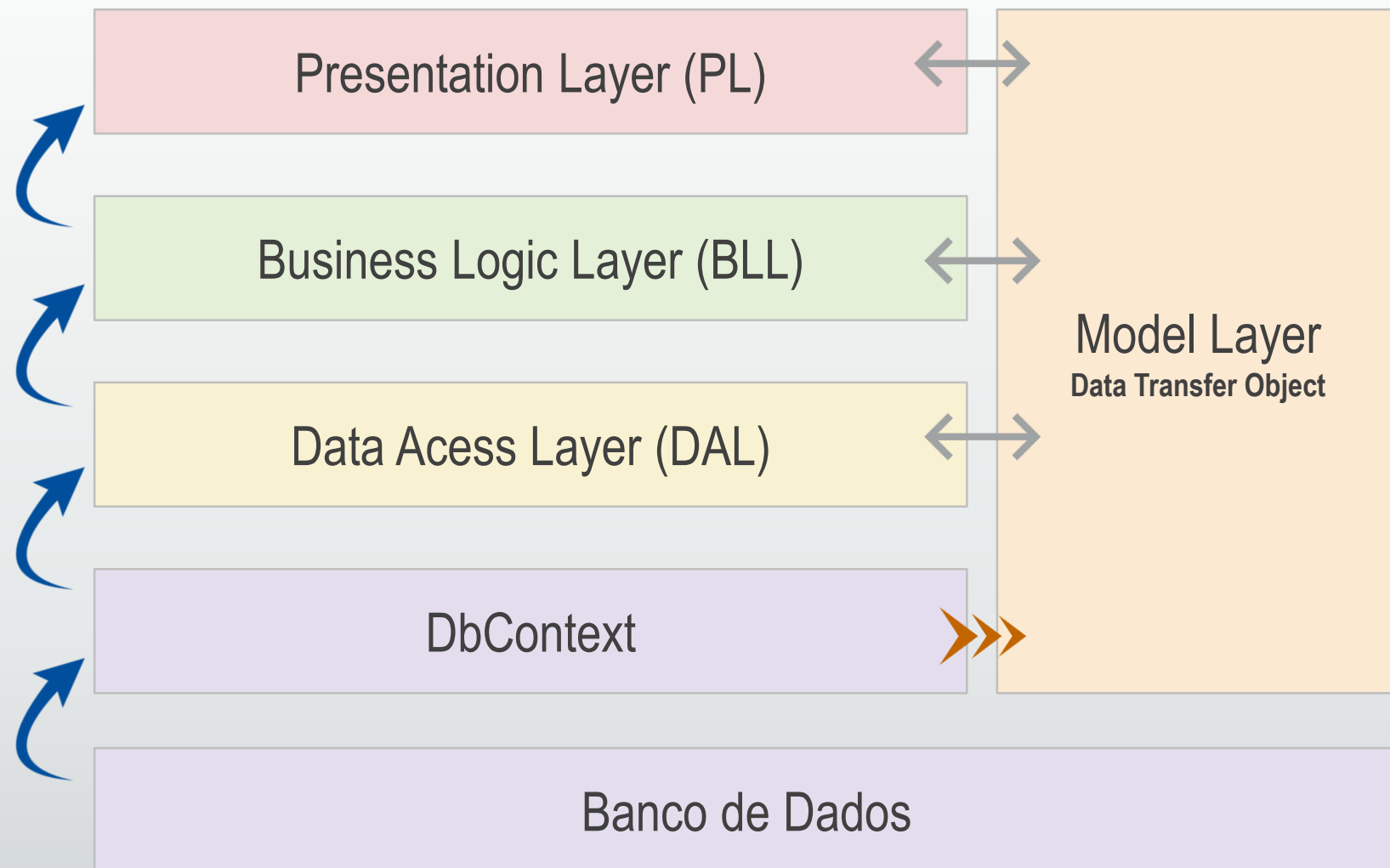
Prof. Edson Mota, PhD, MSc, PMP

Quais benefícios uma camada
de serviços pode trazer?



Implementação Multicamada Convencional

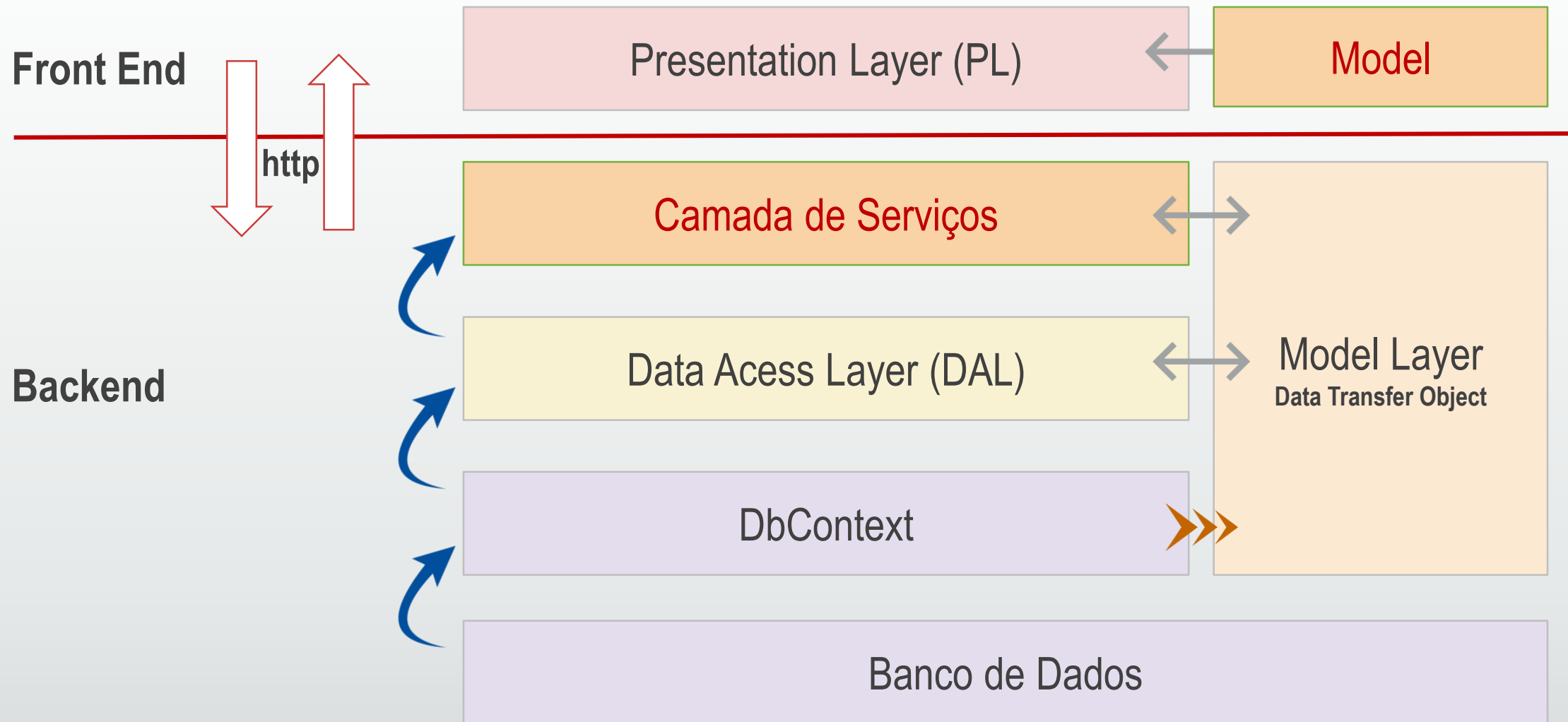
Como uma **camada de serviço** se integraria a essa estrutura?



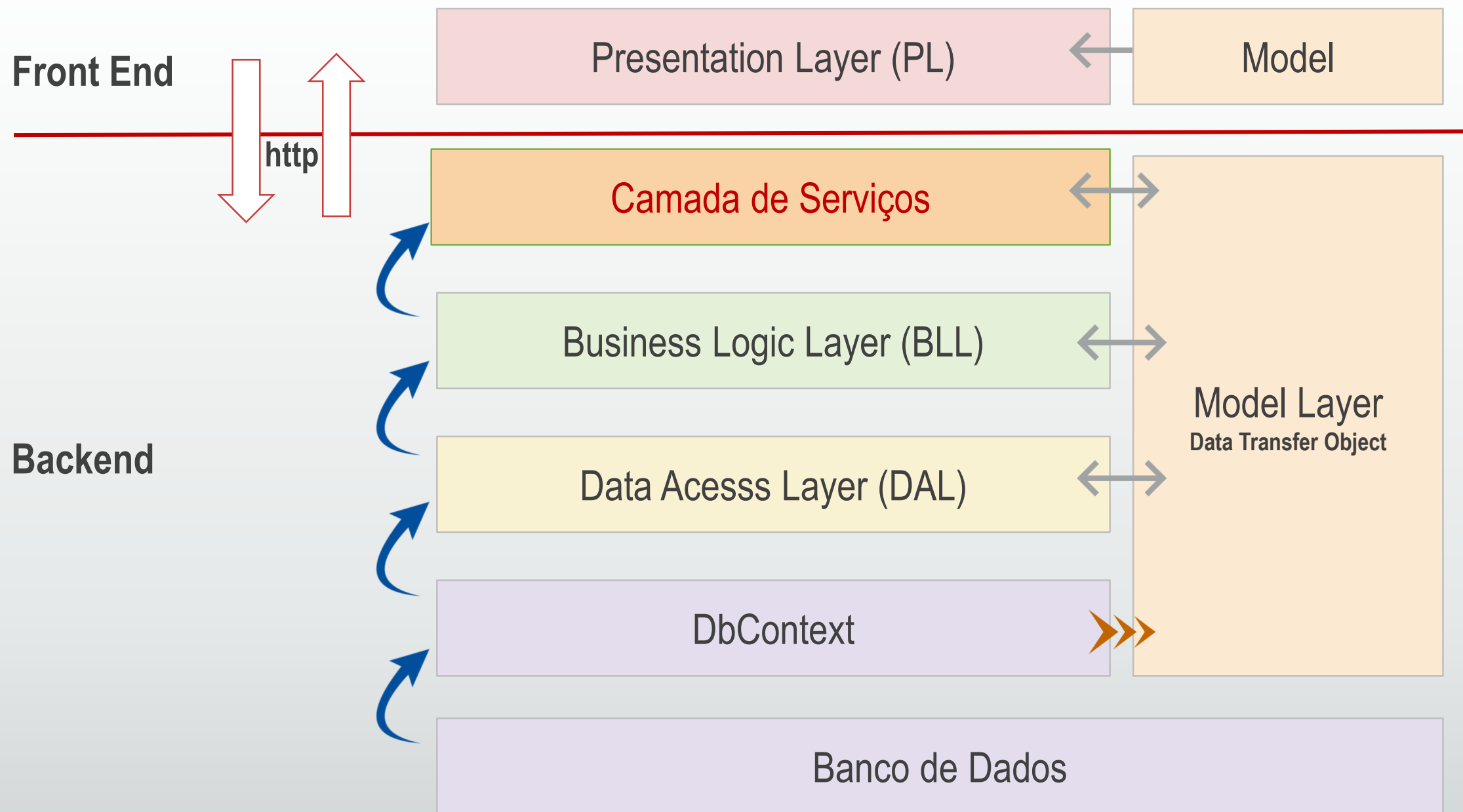
Camada de Serviço

- Existem diferentes formas de integrar uma camada de serviços a infraestrutura multicamada, mas em geral, a decisão está relacionada a dois fatores:
 1. Substituir a camada de negócio por uma camada baseada em serviços.
 2. Incluir uma nova camada de serviço como fachada para a integração entre aplicações.

Utilizando a camada de
negócio para expor serviços



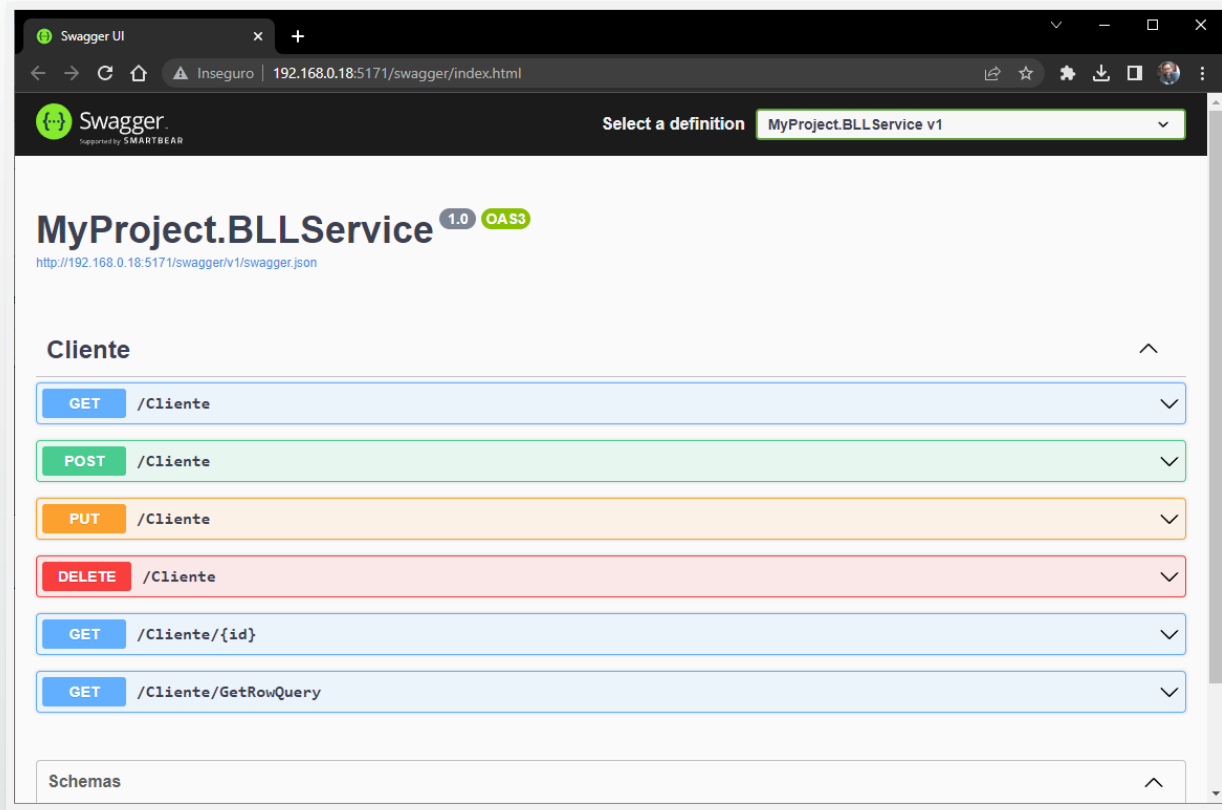
Incluindo uma nova
camada de serviços



Alguns detalhes da Implementação

Onde queremos chegar...

Serviços Expostos



HTTP



Windows Form



Console Application



Application Mobile



Outras Aplicações

Desenvolvimento de APIs

API - Application Programming Interface

- API significa Interface de Programação de Aplicativos e se refere a um conjunto de rotinas, protocolos e ferramentas que permitem a comunicação entre diferentes sistemas.
- Algumas características importantes são:
 - Possibilitam a integração entre sistemas diferentes, permitindo a comunicação e troca de informações;
 - São baseadas em padrões e protocolos bem definidos, como HTTP, JSON, XML, entre outros;
 - Permitem que desenvolvedores criem novas aplicações, produtos e serviços usando recursos de outros sistemas, sem precisar reinventar a roda.

Mas então, existe **apenas** um
tipo de **implementação** para
serviços web?



Não, existem várias - e aumentando... (1)

- SOAP (Simple Object Access Protocol)
 - Protocolo aberto, desenvolvido pela W3C, baseado em XML que utiliza o HTTP ou outros protocolos para enviar mensagens entre aplicativos.
- XML-RPC, JSON-RPC, gRPC
 - São protocolos de chamada a procedimento remoto (RPC - Remote Procedure Call), ou seja, permitem que um aplicativo chame um procedimento em outro aplicativo em um sistema distribuído.
 - O mais tradicional, atualmente, é o gRPC. Produzido pelo Google, este protocolo tem se tornado cada vez mais relevante na disponibilização de serviços distribuídos.

Não, existem várias - e aumentando... (2)

- GraphQL

- Consiste em uma linguagem de consulta de dados para APIs, desenvolvida pelo Facebook.
- Permite que o cliente especifique quais dados ele precisa e como eles devem ser formatados, em vez de receber todas as informações de uma só vez.
- Ganhando relevância nos últimos anos, especialmente em função do crescimento de aplicações baseadas em microserviços.

RESTful

- O RESTful é uma abordagem para construir APIs que baseia-se nos princípios REST.
- REST (Representational State Transfer) é um estilo arquitetural para o desenvolvimento de APIs baseado em princípios simples, como utilizar os métodos HTTP de forma adequada e trabalhar com recursos identificáveis (verbos).
- Algumas características do RESTful são:
 - Utiliza os verbos **HTTP** (GET, POST, PUT, DELETE, etc.) para definir as operações que podem ser realizadas sobre os recursos;
 - Os recursos são identificados por URLs (Uniform Resource Locator);
 - Utiliza o formato JSON ou XML para a representação dos dados.

HTTP

VERBO: GET, PUT, POST ...

Curl

```
curl -X 'GET' \  
  'http://192.168.0.18:5171/Cliente' \  
  -H 'accept: text/plain'
```

Request URL

```
http://192.168.0.18:5171/Cliente
```

URL

Response headers

```
content-type: application/json; charset=utf-8  
date: Mon,01 May 2023 16:29:12 GMT  
server: Kestrel  
transfer-encoding: chunked
```

HTTP
HEADER

Code Status

Responses	
Code	Description
200	Success

Status Code



1xx Informational 100 Continue 101 Switching Protocols 102 Processing	3xx Redirection 300 Multiple Choices 301 Moved Permanently 302 Found 303 See Other 304 Not Modified 305 Use Proxy 307 Temporary Redirect 308 Permanent Redirect	4xx Client Error 400 Bad Request 401 Unauthorized 402 Payment Required 403 Forbidden 404 Not Found 405 Method Not Allowed 406 Not Acceptable 407 Proxy Authentication Required 408 Request Timeout 409 Conflict	5xx Server Error 500 Internal Server Error 501 Not Implemented 502 Bad Gateway 503 Service Unavailable 504 Gateway Timeout 505 HTTP Version Not Supported 506 Variant Also Negotiates 507 Insufficient Storage 508 Loop Detected 510 Not Extended 511 Network Authentication Required 599 Network Connect Timeout Error
2xx Success 200 OK 201 Created 202 Accepted 203 Non-authoritative Information 204 No Content 205 Reset Content 206 Partial Content 207 Multi-Status 208 Already Reported 226 IM Used	410 Gone 411 Length Required 412 Precondition Failed 413 Payload Too Large 414 Request-URI Too Long 415 Unsupported Media Type 416 Requested Range Not Satisfiable 417 Expectation Failed 418 I'm a teapot — 421 Misdirected Request 422 Unprocessable Entity 423 Locked 424 Failed Dependency 426 Upgrade Required 428 Precondition Required 429 Too Many Requests 431 Request Header Fields Too Large 444 Connection Closed Without Response 451 Unavailable For Legal Reasons 499 Client Closed Request		

Requisições x Verbos HTTP

Verbos

- Os verbos HTTP são usados para indicar a ação que deve ser executada em um recurso.
- Na arquitetura RESTful, cada verbo HTTP é usado de forma específica para operar em recursos e executar ações, seguindo o princípio da uniformidade.
- Os verbos HTTP são padronizados e bem definidos, permitindo uma comunicação clara e consistente entre consumidores e fornecedores em uma aplicação RESTful.

URIs Individuais por Recurso

- Exemplos de URIs
 - `http://192.168.0.18:5171/Cliente`
 - `http://192.168.0.18:5171/Cliente/18`
 - `http://192.168.0.18:5171/Cliente/all`

Requisições GET

Curl

```
curl -X 'GET' \  
  'http://192.168.0.18:5171/Cliente' \  
  -H 'accept: text/plain'
```

Request URL

```
http://192.168.0.18:5171/Cliente
```

- Na arquitetura RESTful, o verbo GET é usado para recuperar dados de um recurso específico no servidor.
- O retorno de uma chamada GET, em geral, consiste no conjunto de dados que representa o recurso solicitado.

Requisição POST

Curl

```
curl -X 'POST' \
  'http://192.168.0.18:5171/Cliente' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 0,
    "nome": "André Santos",
    "telefone": "53453453",
    "email": "andre@teste.com"
  }'
```

Request URL

```
http://192.168.0.18:5171/Cliente
```

- Requisições POST são usadas para enviar dados de uma fonte para um servidor para criar ou atualizar um recurso identificado por uma URI.
- O corpo da solicitação contém os dados (*payload*) a serem criados ou atualizados, e o servidor geralmente responde com a identificação do recurso criado ou atualizado.

Requisição PUT

Curl

```
curl -X 'PUT' \
  'http://192.168.0.18:5171/Cliente' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 18, ←
    "nome": "André Santos Rocha",
    "telefone": "53453453",
    "email": "andre@teste.com"
  }'
```

Request URL

```
http://192.168.0.18:5171/Cliente
```

- As requisições PUT são usadas para atualizar um recurso existente no servidor.
- A solicitação deve incluir o URI do recurso que está sendo atualizado e os novos dados (payload) que serão armazenados no recurso.
- O verbo PUT pode eventualmente ser utilizado para incluir um registro, caso o recurso solicitado não seja encontrado.

Requisição DELETE

Curl

```
curl -X 'DELETE' \  
  'http://192.168.0.18:5171/Cliente?id=18' \  
  -H 'accept: */*'
```

Request URL

```
http://192.168.0.18:5171/Cliente?id=18
```

- As requisições DELETE são utilizadas para remover um recurso identificado pela URI.
- Em geral, não há um payload associado à requisição.
- A execução dessa operação é geralmente irreversível e pode resultar na remoção de informações importantes ou críticas.
- Em geral, requisições DELETE não retornam nenhum corpo de resposta.

Outros Verbos

- **PATCH:**

- Usado para modificar parcialmente um recurso existente.
 - Exemplo: modificar apenas o nome de um cliente em um serviço
 - Na requisição, deve constar um payload contendo as mudanças requeridas.

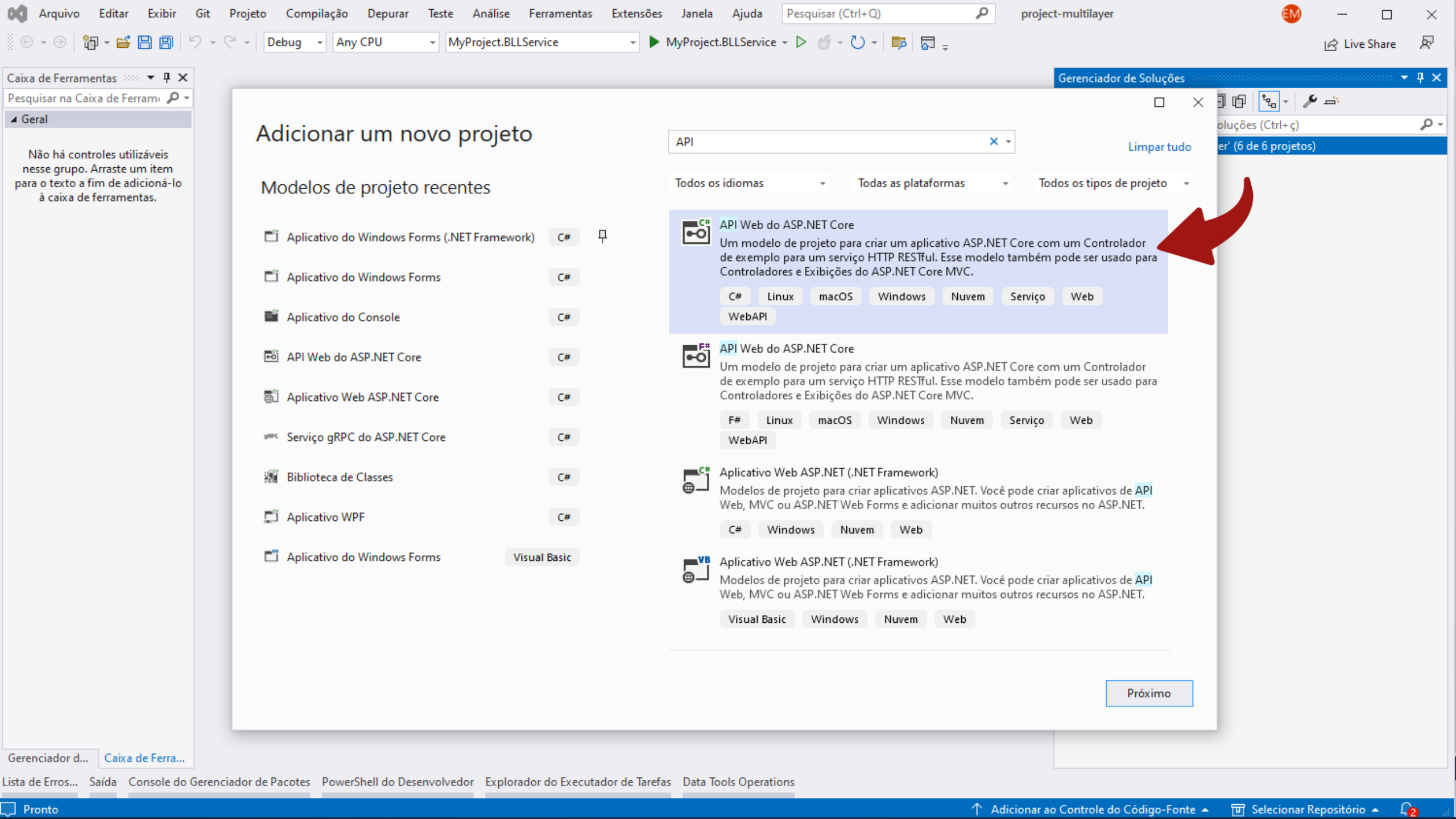
- **HEAD:**

- Usado para recuperar apenas os metadados de um recurso, sem recuperar o corpo da resposta.

- **OPTIONS:**

- Usado para obter informações sobre as opções disponíveis para um recurso.

Vamos Implementar
um Serviço?



Configurar seu novo projeto

API Web do ASP.NET Core

C#

Linux

macOS

Windows

Nuvem

Serviço

Web

WebAPI

Nome do projeto

MyProject.BLLService

Local

D:\project-multilayer-replicar

Voltar

Próximo

Informações adicionais

API Web do ASP.NET Core

C#

Linux

macOS


Windows

Nuvem


Serviço

Web

WebAPI


Estrutura 

.NET 6.0 (Suporte de Longo Prazo)

Tipo de autenticação 


Nenhum

☐ Configurar para HTTPS 


☐ Habilitar o Docker 

Sistema Operacional do Docker 

Linux

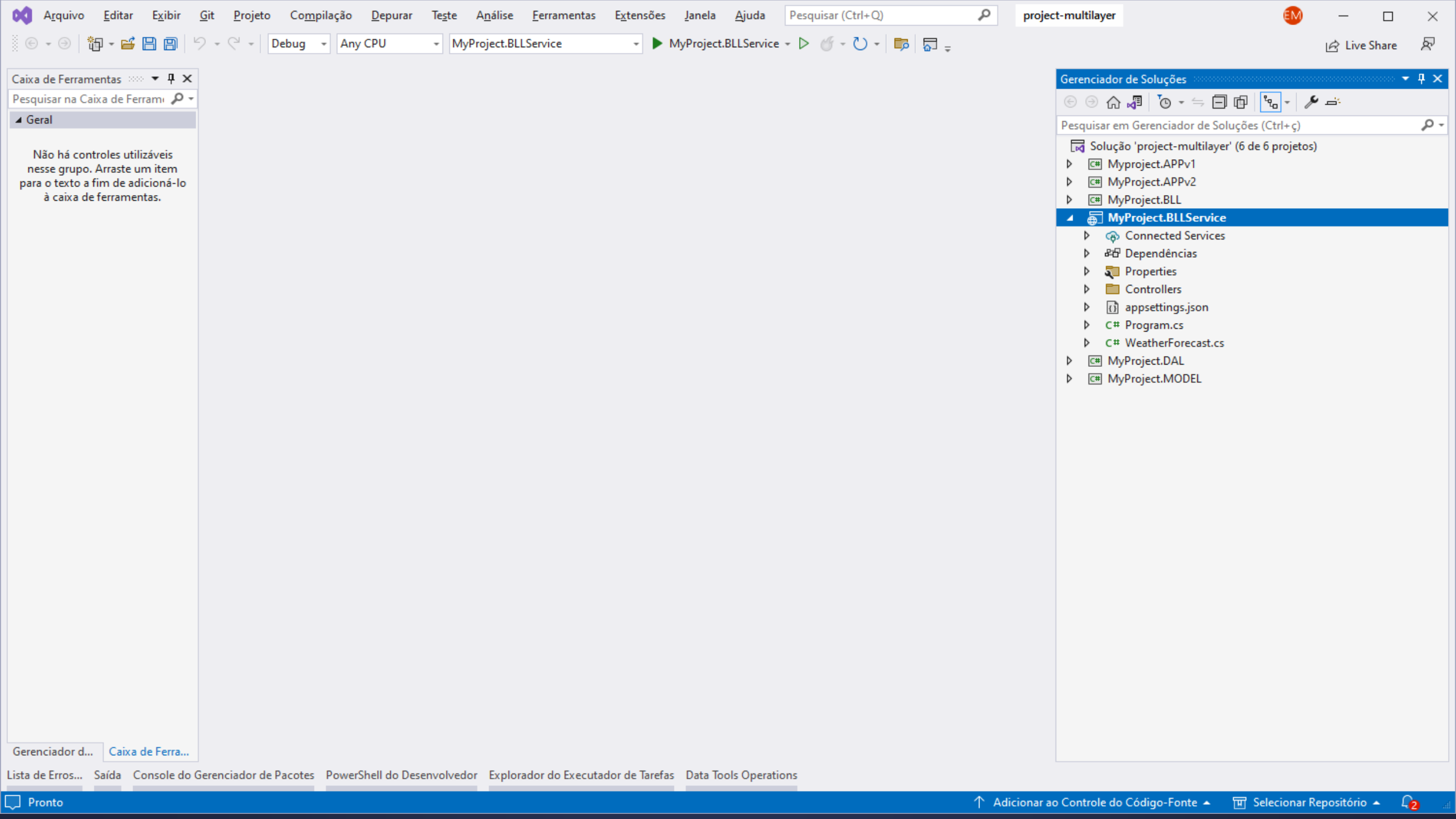
☒ Usar controladores (desmarque para usar APIs mínimas) 

☒ Habilitar o suporte a OpenAPI 

☐ Não use instruções de nível superior 

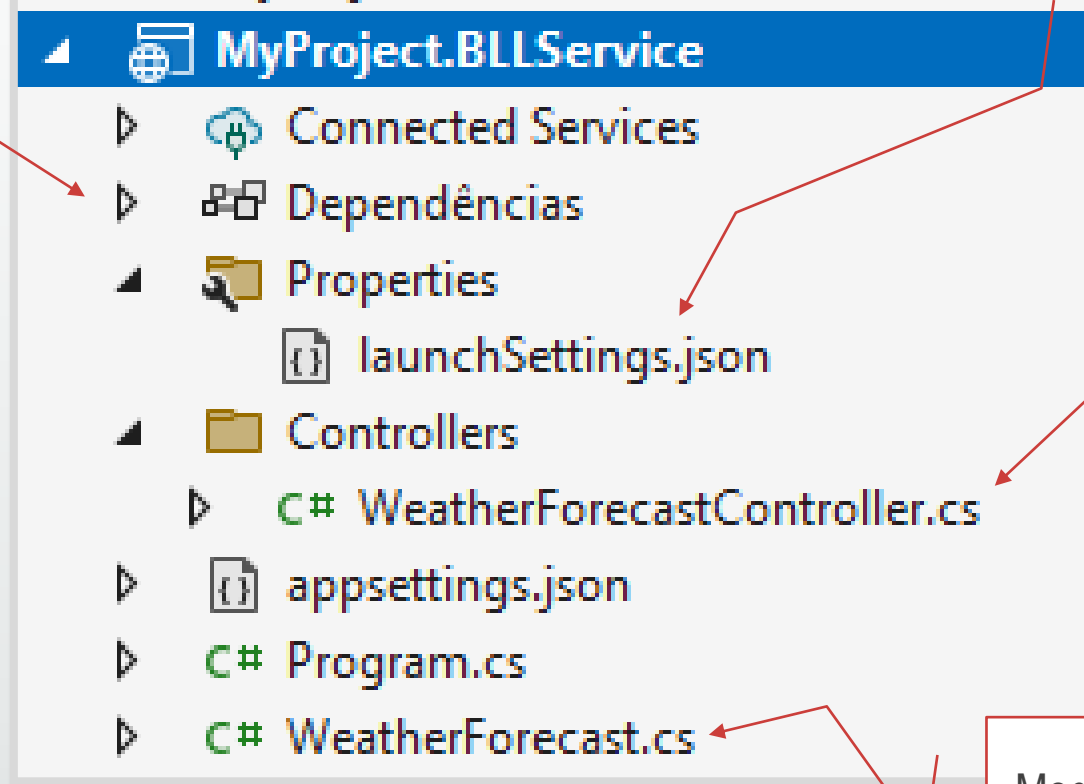
Voltar

Criar



Estrutura do Projeto

Dependências do Projeto



Configura os serviços que devem ser lançados na inicialização da aplicação.

Nessa pasta ficam as classes de controle (Rotas, Verbos, ações).

Model (Utilizado para persistir os objetos)

Torne o projeto como principal e execute



MyProject.BLLService

1.0 OAS3

<http://localhost:5235/swagger/v1/swagger.json>

WeatherForecast

GET /WeatherForecast

Schemas

WeatherForecast >



Swagger é uma estrutura de software de código aberto que ajuda a projetar, criar, documentar e consumir serviços da web RESTful

Select a definition

MyProject.BLL Service v1

MyProject.BLLService 1.0 OAS3

http://localhost:5235/swagger/v1/swagger.json

WeatherForecast

GET /WeatherForecast

Parameters

No parameters

Try it out

Responses

Code	Description	Links
200	Success	No links
Media type		

WeatherForecast

GET /WeatherForecast

Parameters

No parameters

Execute

Responses

Code	Description	Links
200	Success	No links

Media type

text/plain

Controls Accept: header.

Example Value | Schema

```
[ {
  "date": "2023-05-02T00:46:20.557Z",
  "temperatureC": 0,
  "temperatureF": 0,
  "summary": "string"
}]
```

Execute

Clear

Responses

Curl

```
curl -X 'GET' \  
  'http://localhost:5235/WeatherForecast' \  
  -H 'accept: text/plain'
```

Request URL

```
http://localhost:5235/WeatherForecast
```

Server response

Code

Details

200

Response body

```
[  
  {  
    "date": "2023-05-02T21:47:11.1382503-03:00",  
    "temperatureC": 40,  
    "temperatureF": 103,  
    "summary": "Warm"  
  },  
  {  
    "date": "2023-05-03T21:47:11.1393263-03:00",  
    "temperatureC": 38,  
    "temperatureF": 100,  
    "summary": "Chilly"  
  },  
  {  
    "date": "2023-05-04T21:47:11.1393271-03:00",  
    "temperatureC": -17,  
    "temperatureF": 2,  
    "summary": "Hot"  
  },  
  {  
    "date": "2023-05-05T21:47:11.1393273-03:00",  
    "temperatureC": 8,  
    "temperatureF": 46,  
    "summary": "Cool"  
  },  
  {  
    "date": "2023-05-06T21:47:11.1393274-03:00",  
    "temperatureC": 14,  
    "temperatureF": 57,  
    "summary": "Hot"  
  }  
]
```

Response headers

Emissão da Requisição

Localização do
Recurso

Dados Retornados

Download

```
{
  "date": "2023-05-05T21:47:11.1393273-03:00",
  "temperatureC": 8,
  "temperatureF": 46,
  "summary": "Cool"
},
{
  "date": "2023-05-06T21:47:11.1393274-03:00",
  "temperatureC": 14,
  "temperatureF": 57,
  "summary": "Sweltering"
}
}
```

Response headers

```
content-type: application/json; charset=utf-8
date: Tue, 02 May 2023 00:47:10 GMT
server: Kestrel
transfer-encoding: chunked
```

Cabeçalho HTTP

Code	Description	Links
200	Success	No links

Código de Status

Media type

text/plain

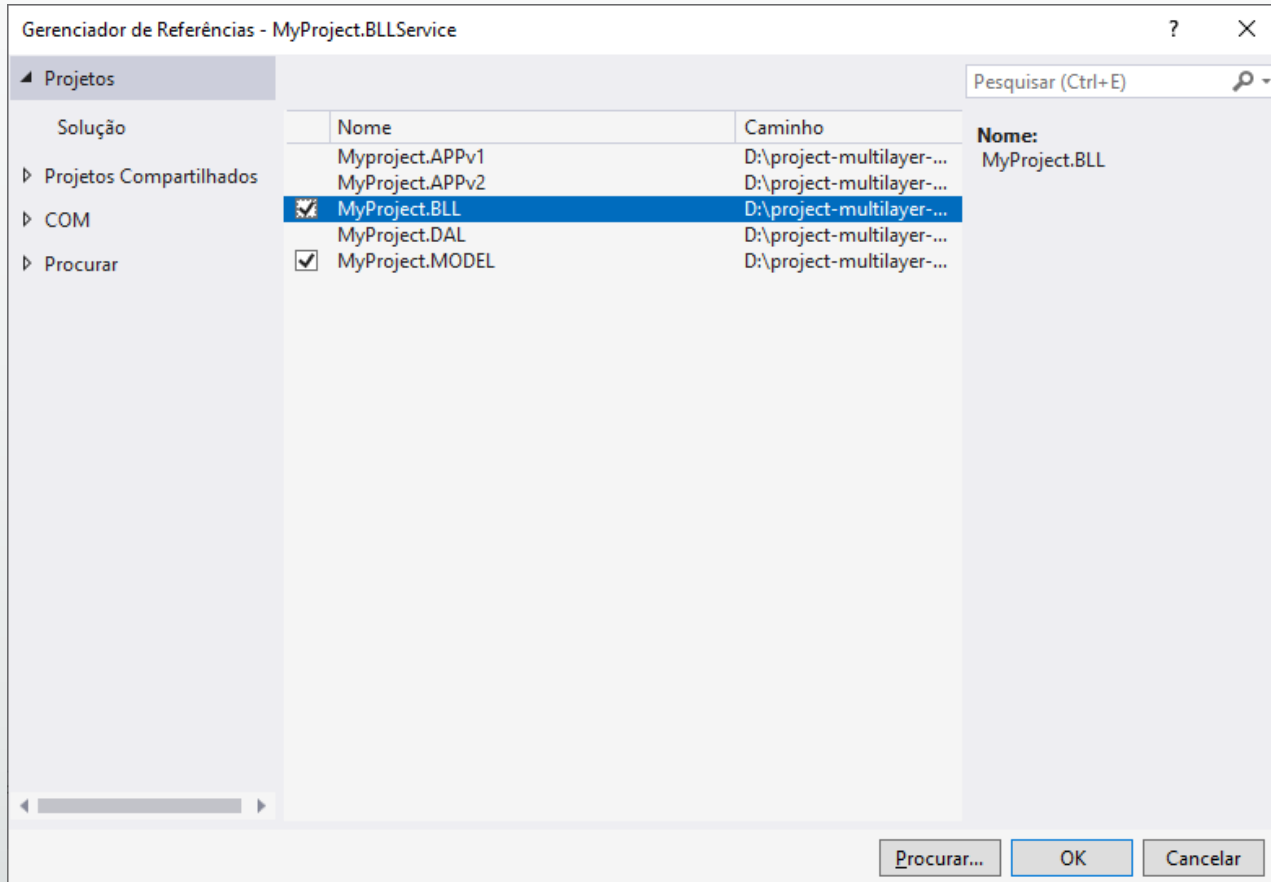
Example Value | Schema

```
[
  {
    "date": "2023-05-02T00:47:11.185Z",
    "temperatureC": 0,
    "temperatureF": 0,
    "summary": "string"
  }
]
```

Documentação da API

Vamos Personalizar
esse Serviço?

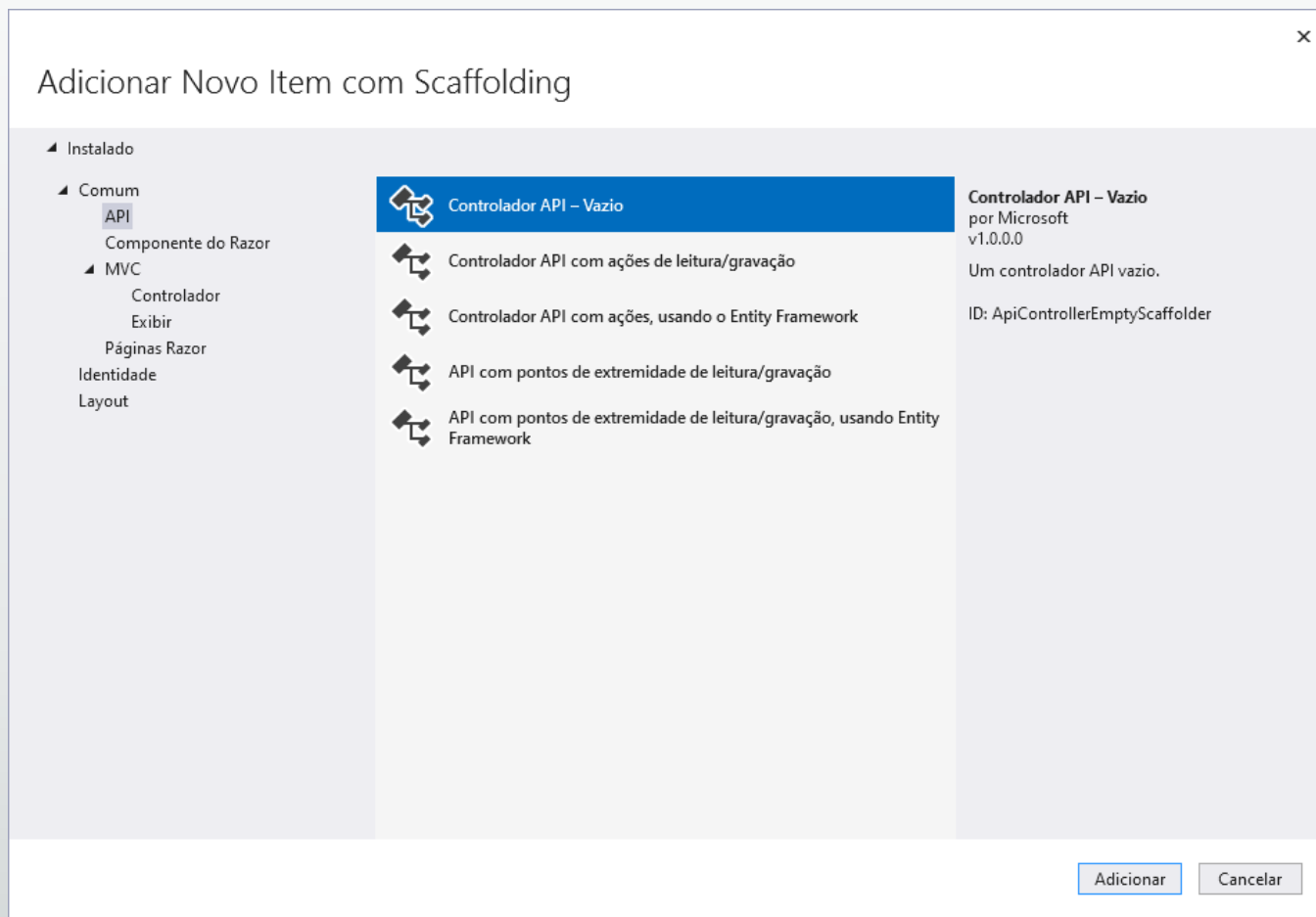
Utilizando o Model



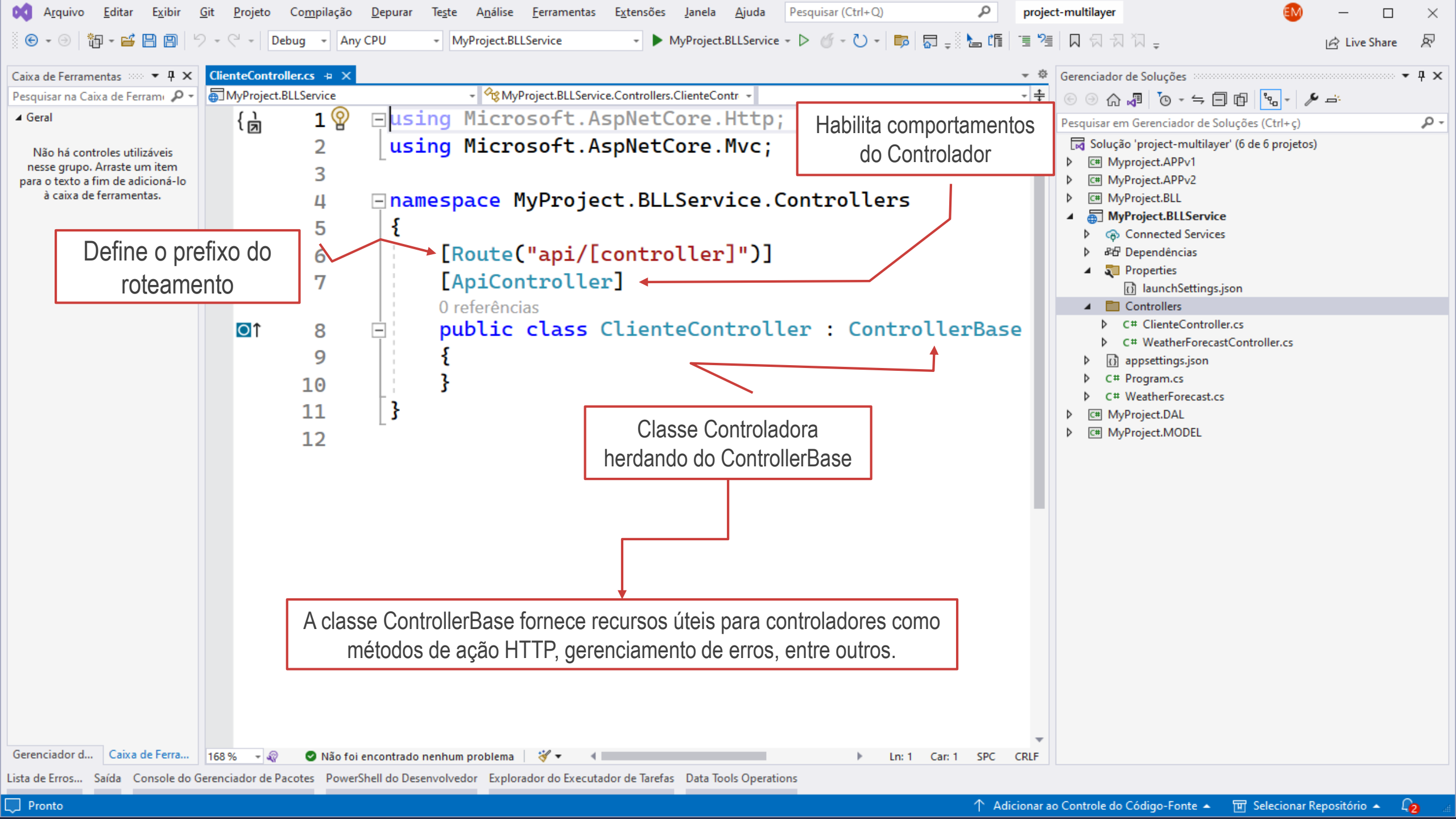
- Para implementar uma nova camada de serviços ao nosso projeto Multicamadas, precisamos:
- Referenciar as camadas MODEL e BLL no Serviço

Criando nossa primeira Controller

- Botão direito na pasta Controller → Adicionar novo item → Controller




- Adicione o novo Controller com o nome:
 - “ClienteController.cs”



ClienteController.cs

MyProject.BLLService.Contr

▼  GetCliente()

Import das Camadas

Raiz da API

Definição do Verbo

Retorno de uma lista
encapsulada em um objeto
ActionResult (Status, header)

Retorno das camadas anteriores e testes

0 referências

3

3

```
if (list != null) { return Ok(list); }
return NotFound();
```

3

GET /Cliente

Parameters

Cancel

No parameters

Execute

Clear

Responses

Curl

```
curl -X 'GET' \  
  'http://localhost:5235/Cliente' \  
  -H 'accept: text/plain'
```

Request URL

```
http://localhost:5235/Cliente
```

Server response

Code

Details

200

Response body

```
[  
  {  
    "id": 12,  
    "nome": "Jose dos Santos",  
    "telefone": "23423",  
    "email": "jose@teste.com"  
  },  
  {  
    "id": 13,  
    "nome": "Marcos Santos",  
    "telefone": "4564565",  
    "email": "marcos@teste.com"  
  },  
  {  
    "id": 14,  
    "nome": "Marcelo Castro",  
    "telefone": "456456",  
    "email": "marcelo@teste.com"  
  },  
  {  
    "id": 19,  
    "nome": "Edson Mota222222",  
    "telefone": "456456",  
    "email": "edson@teste.com"  
  }  
]
```

Utilizando Parâmetros nas Requisições GET

```
[HttpGet("{id}", Name = "GetClienteById")]
```

Definição do Verbo indicando um parâmetro na rota

0 referências

```
public ActionResult<Cliente> GetClienteById(int id)
```

Recebimento do parâmetro

```
{
```

```
    Cliente _cliente = ClienteRepository.GetById(id);
```

Utilização do Parâmetro recebido

```
try
```

```
{
```

```
    if (_cliente != null)
```

```
    {
```

```
        return Ok(_cliente);
```

```
    }
```

200 se a operação dor bem sucedida

```
    return NotFound();
```

Código de erro (404)

```
}
```

```
catch (Exception ex)
```

```
{
```

```
    return StatusCode(500, ex.Message);
```

Código de erro (500)

```
}
```

```
}
```

Name	Description
id * required	
integer(\$int32)	12
(path)	

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://192.168.0.18:5171/Ciente/12' \
-H 'accept: text/plain'
```



Request URL

```
http://192.168.0.18:5171/Ciente/12
```

Server response

Code	Details
------	---------

200

Response body

```
{
  "id": 12,
  "nome": "Jose dos Santos",
  "telefone": "23423",
  "email": "jose@teste.com"
}
```



Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon, 15 May 2023 13:38:08 GMT
server: Kestrel
transfer-encoding: chunked
```


Requisições POST

Definição do Verbo indicando
uma requisição POST

```
[HttpPost(Name = "AddCliente")]
```

0 referências

```
public ActionResult<Cliente> AddCliente(Cliente cliente)
```

Payload recebido na
requisição

```
{
```

```
try
```

```
{
```

```
    Cliente _cliente = ClienteRepository.Add(cliente);
```

```
    return _cliente == null ? NotFound() : Ok(_cliente);
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
    return StatusCode(500, ex.Message);
```

```
}
```

```
}
```

Código de erro

Execução do método
de inclusão e retorno
do objeto

POST

/Cliente



Parameters

Cancel

Reset

No parameters

Request body

application/json



```
{  
  "nome": "Bob Dylan",  
  "telefone": "714567864",  
  "email": "bob@teste.com"  
}
```

Execute

Clear

2



Request URL

<http://192.168.0.18:5171/Cliente>

Server response

Code

Details

200

Response body

```
{
  "id": 20,
  "nome": "Bob Dylan",
  "telefone": "714567864",
  "email": "bob@teste.com"
}
```



Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon, 15 May 2023 13:42:07 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code

Description

Links

200

Success

No links

Media type

text/plain



Controls Accept header.

Example Value | Schema

```
{
  "id": 0,
  "nome": "string",
  "telefone": "string",
  "email": "string"
}
```

Requisições PUT

[HttpPut(Name = "UpdateCliente")]

0 referências

```
public ActionResult<Cliente> UpdateCliente(Cliente cliente)
```

```
{
```

```
try
```

```
{
```

```
    if (cliente != null)
```

```
    {
```

```
        Cliente _cliente = new Cliente();
```

```
        _cliente.Id = cliente.Id;
```

```
        _cliente.Nome = cliente.Nome;
```

```
        _cliente.Email = cliente.Email;
```

```
        _cliente.Telefone = cliente.Telefone;
```

```
        ClienteRepository.Update(_cliente);
```

```
        return Ok(_cliente);
```

```
    }
```

```
    return NotFound();
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
    return StatusCode(500, ex.Message);
```

```
}
```

```
}
```

Definição do Verbo indicando uma requisição POST

Definição do Verbo indicando uma requisição POST

Definição do Verbo indicando uma requisição POST

PUT**/Cliente****Parameters**

Cancel

Reset

No parameters

Request body

application/json



```
{
  "id": 20,
  "nome": "Bob Dylan 2",
  "telefone": "714567864",
  "email": "bob@teste.com"
}
```

**Execute**

Responses

Curl

```
curl -X 'PUT' \
'http://192.168.0.18:5171/Cliente' \
-H 'accept: text/plain' \
-H 'Content-Type: application/json' \
-d '{
  "id": 20,
  "nome": "Bob Dylan 2",
  "telefone": "714567864",
  "email": "bob@teste.com"
}'
```

Request URL

http://192.168.0.18:5171/Cliente

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{ "id": 20, "nome": "Bob Dylan 2", "telefone": "714567864", "email": "bob@teste.com" }</pre></div><div><div>Download</div></div></div>

Response headers

```
content-type: application/json; charset=utf-8
date: Mon,15 May 2023 13:47:52 GMT
server: Kestrel
transfer-encoding: chunked
```


Requisições DELETE

```
[HttpDelete(Name = "DeleteCliente")]
```

0 referências

Definição do Verbo indicando
uma requisição POST

```
public ActionResult DeleteCliente(int id)
```

```
{
```

```
try
```

```
{
```

```
var cliente= ClienteRepository.GetById(id);
```

```
ClienteRepository.Excluir(cliente);
```

```
return Ok();
```

Carregando objeto a
se excluído

```
}
```

```
catch (Exception ex)
```

```
{
```

```
return StatusCode(500, ex.Message);
```

Código de Erro

```
}
```

```
}
```

DELETE

/Cliente

Parameters

Cancel

Name	Description
id	
integer(\$int32)	20
(query)	


Execute

Responses

Code	Description	Links
200	Success	No links

Responses

Curl

```
curl -X 'DELETE' \
'http://192.168.0.18:5171/Cliente?id=20' \
-H 'accept: */*' 
```

Request URL

```
http://192.168.0.18:5171/Cliente?id=20
```

Server response

Code

Details

200

Response headers

```
content-length: 0
date: Mon, 15 May 2023 13:49:12 GMT
server: Kestrel
```

Responses

Code

Description

Links

200

Success

No links

Revisão

- Construa um endpoint capaz de receber um objeto cliente, analisar os dois primeiros dígitos do telefone (DDD) e retornar a cidade de referência.
 - Mantenha uma lista física com pelo menos 5 DDDs
 - Caso o DDD não exista na lista, retorne NtoFound

Bons Estudos!