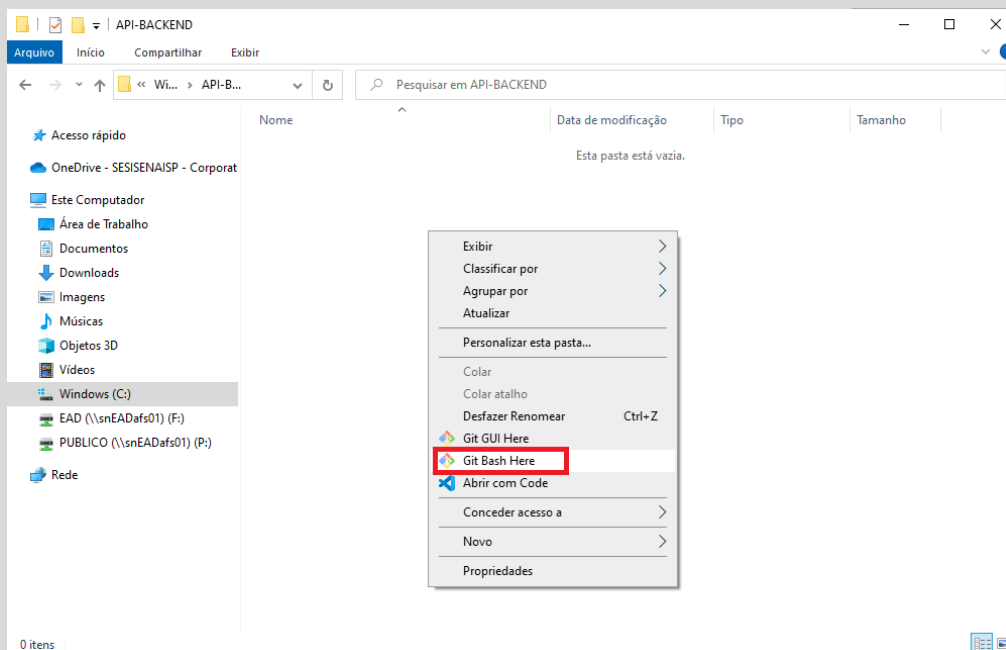


Projeto API Chapter

Criando o projeto API Chapter e instalando os pacotes

1. Crie uma pasta e dê a ela o nome **PROJETO-API-Chapter**. Dentro dela, clique com o botão direito para acessar o menu de opções. Selecione **Git bash Here** para acessar o terminal.



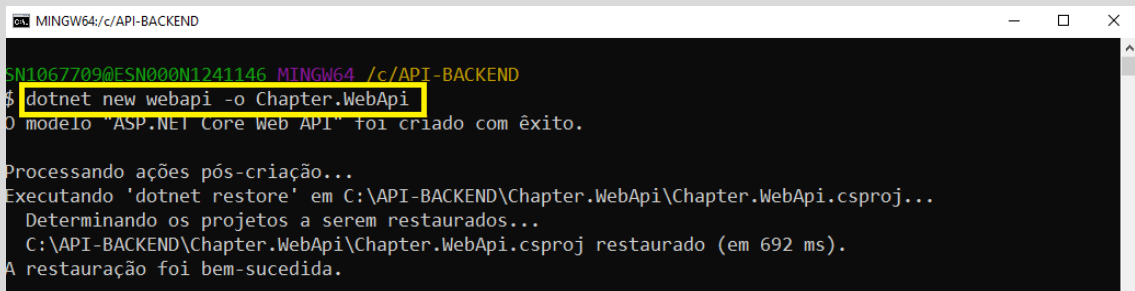
2. No terminal que será aberto, digite o comando **dotnet --version** e dê **Enter** para verificar a versão do dotnet instalada em sua máquina; certifique-se de é a 6 ou superior.



```
MINGW64/c/API-BACKEND
$ dotnet --version
6.0.404
```

3. Para criar o projeto de API, digite o comando abaixo.

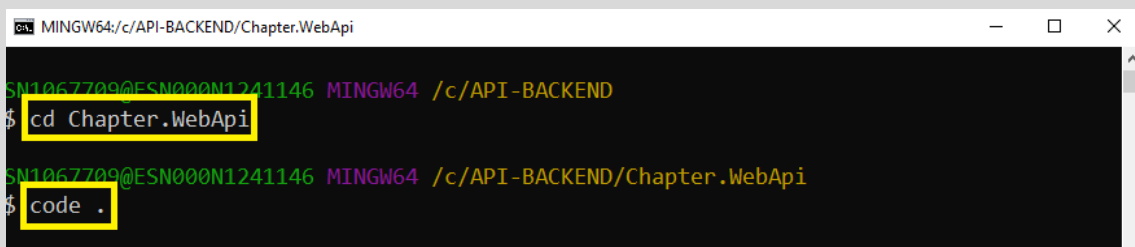
```
dotnet new webapi -o Chapter.WebApi
```



```
MINGW64/c/API-BACKEND
$ dotnet new webapi -o Chapter.WebApi
O modelo "ASP.NET Core Web API" foi criado com êxito.

Processando ações pós-criação...
Executando 'dotnet restore' em C:\API-BACKEND\Chapter.WebApi\Chapter.WebApi.csproj...
Determinando os projetos a serem restaurados...
C:\API-BACKEND\Chapter.WebApi\Chapter.WebApi.csproj restaurado (em 692 ms).
A restauração foi bem-sucedida.
```

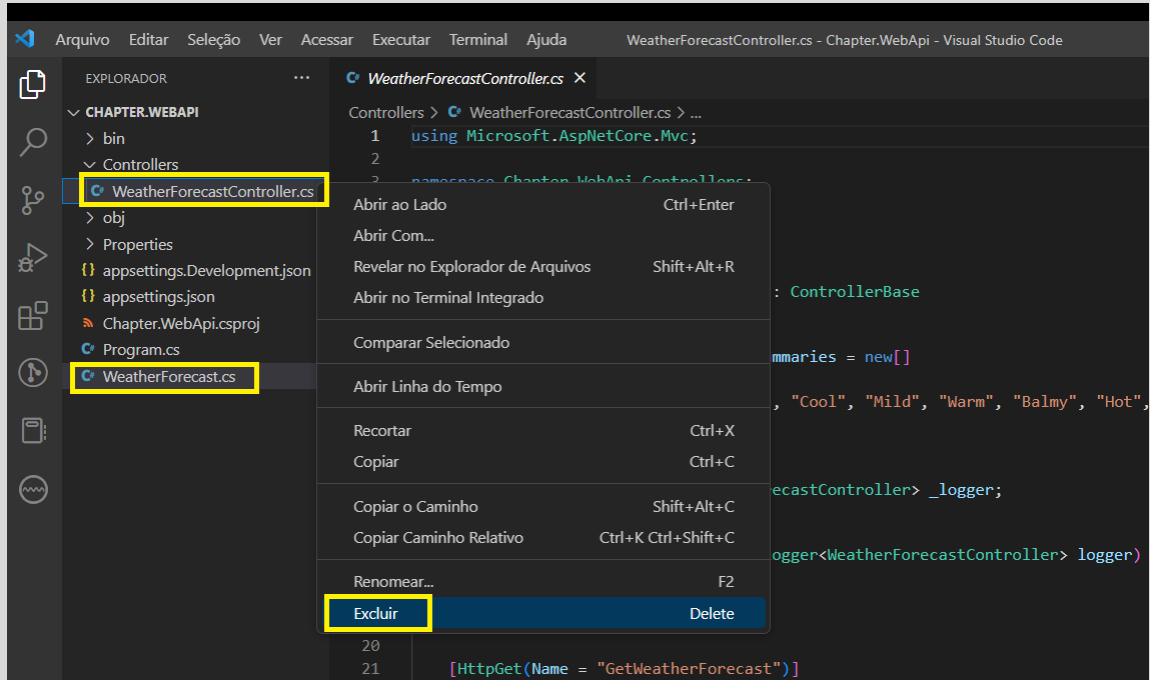
4. Para facilitar o desenvolvimento, acesse a pasta do seu projeto digitando **cd Chapter.WebApi**. Depois, abra o editor de códigos VS Code pelo terminal digitando o comando **code .** conforme a imagem.



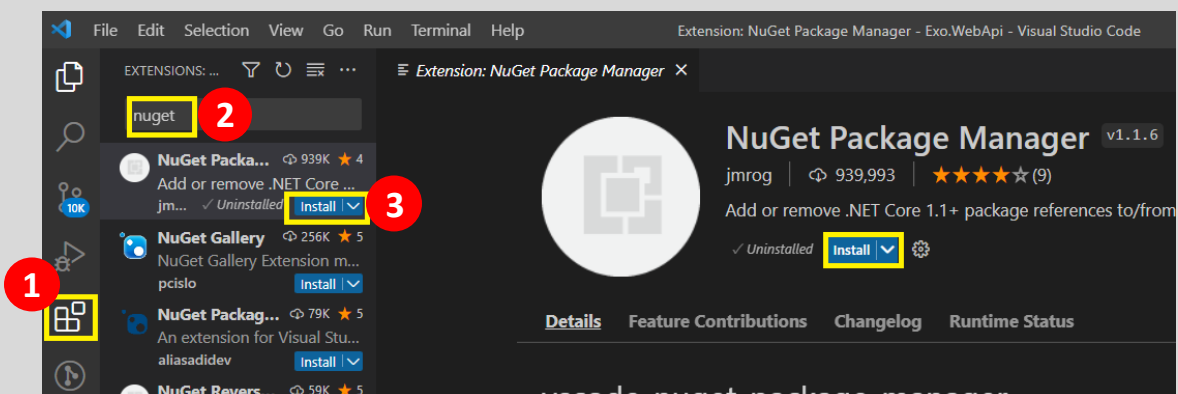
```
MINGW64/c/API-BACKEND/Chapter.WebApi
$ cd Chapter.WebApi

MINGW64/c/API-BACKEND/Chapter.WebApi
$ code .
```

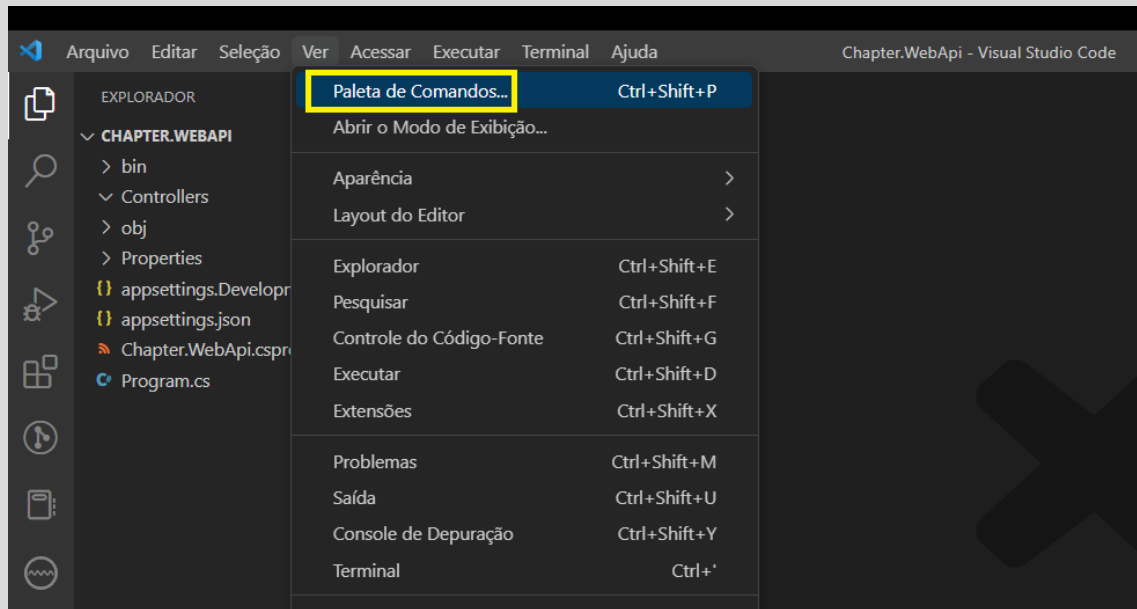
5. Quando você cria um projeto do tipo webapi, o template fornece dois arquivos de exemplo que não utilizaremos, então, precisamos apagá-los. Para isso, selecione os arquivos na coluna (Explorador) e apague os arquivos “WeatherForecast.cs” e “WeatherFoExplorerrecastController.cs”.



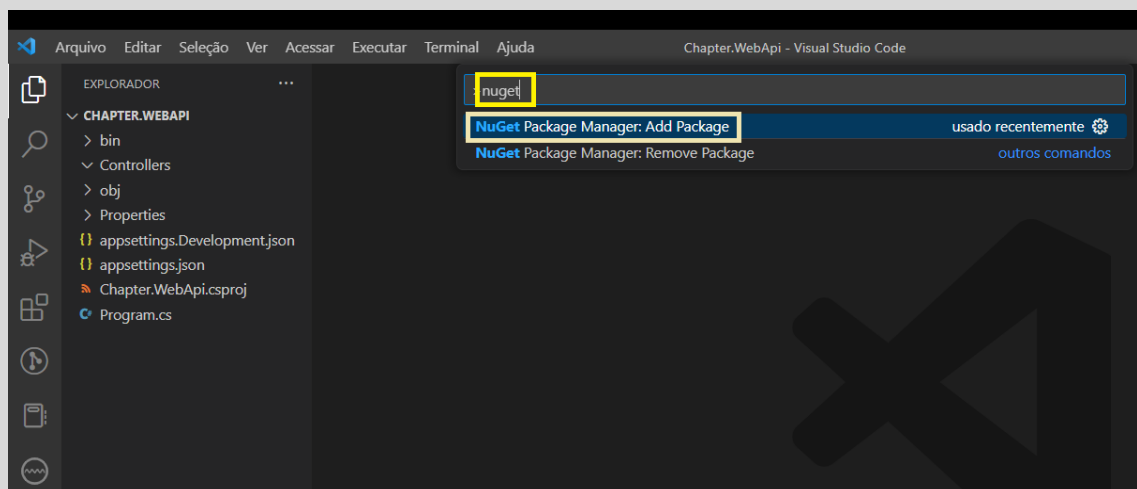
6. O projeto vai precisar da instalação de alguns pacotes e isso será feito através do NuGet Package Manager. Caso você não tenha a extensão instalada em seu VS Code, clique em **Extensões** (1), busque por **nuget** (2) e **instale** (3) o NuGet Package Manager.



7. Durante os tutoriais, usaremos alguns pacotes em nossa aplicação. Então, faremos a instalação de todos os que serão usados neste e nos próximos tutoriais. Para abrir o gerenciador de pacotes do NuGet, acesse o menu **Ver (View) > Paleta de Comandos...** (Command Palette):

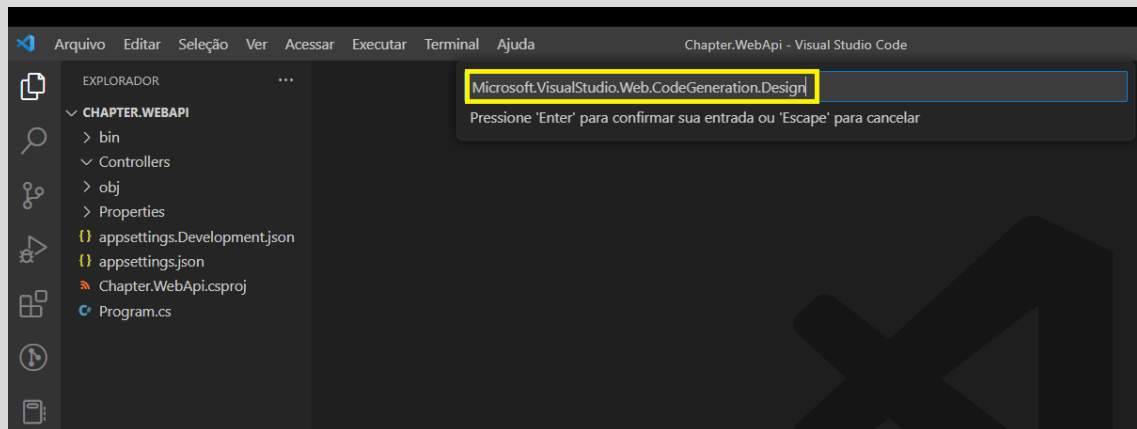


8. No campo da paleta de comandos, digite **nuget** e clique em **NuGet Package Manager: Add Package**.

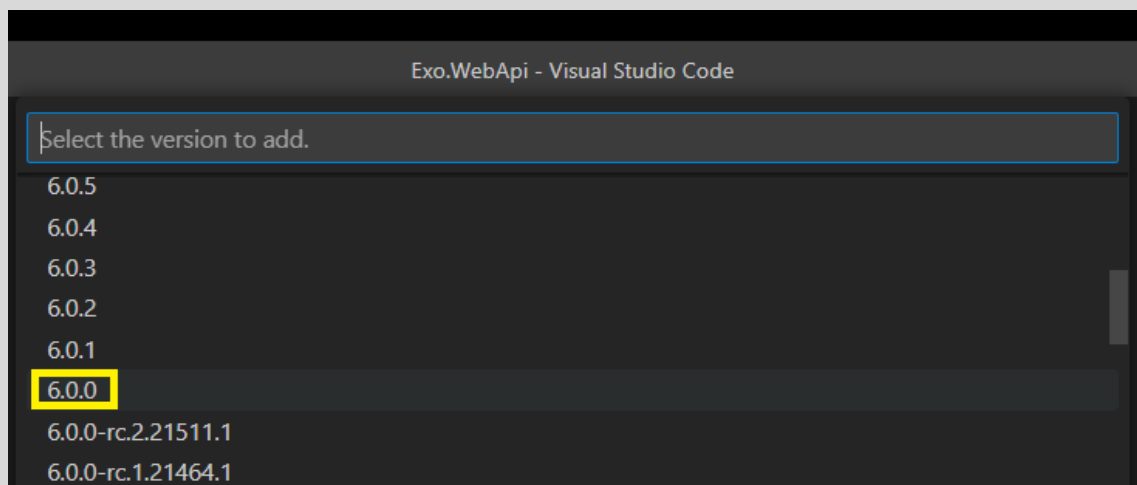


9. Outro campo de busca vai abrir, solicitando que você insira o pacote que deseja instalar. Escolha o pacote abaixo e pressione **Enter**.

Microsoft.VisualStudio.Web.CodeGeneration.Design



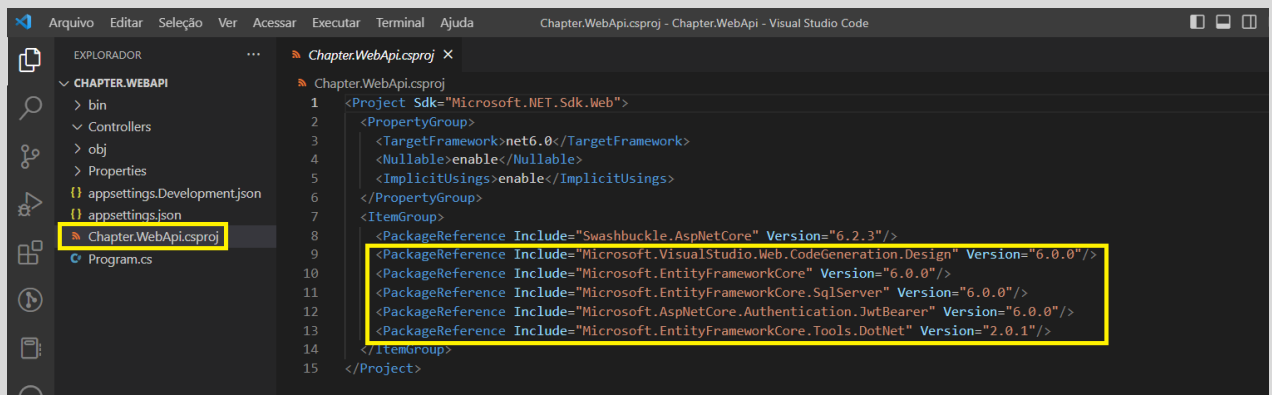
10. Escolha a versão 6.0.0.



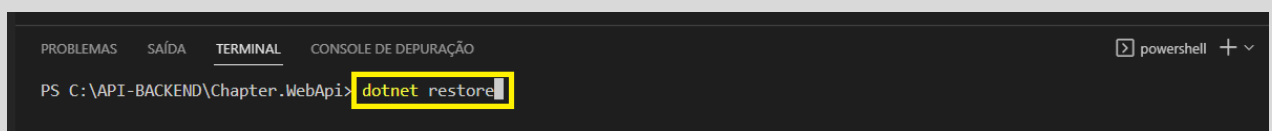
11. Repita o processo anterior e instale, também, os pacotes abaixo, respeitando as versões indicadas:

- **Microsoft.EntityFrameworkCore**
 - **Versão 6.0.0**
- **Microsoft.EntityFrameworkCore.SqlServer**
 - **Versão 6.0.0**
- **Microsoft.AspNetCore.Authentication.JwtBearer**
 - **Versão 6.0.0**
- **Microsoft.EntityFrameworkCore.Tools.DotNet**
 - **Versão 2.0.1**

12. Abra o `Exo.webApi.csproj` e verifique se os pacotes instalados constam no arquivo. Compare seu projeto com o destaque abaixo:

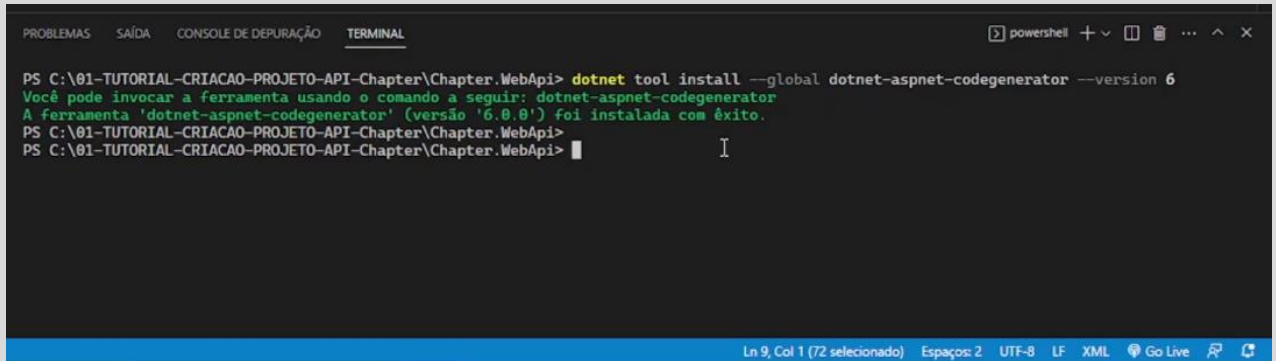


13. Clique em **Terminal/Novo Terminal** e, na janela que abrir na parte inferior, execute o comando **dotnet restore** para consolidar as operações.



14. Para finalizar essa etapa, será necessário instalar o **code generator** para alguns procedimentos em seu projeto. Para isso, execute no terminal o comando abaixo:

```
dotnet tool install --global dotnet-aspnet-codegenerator --version 6
```



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  powershell + v [ ] [ ] ... ^ X
PS C:\01-TUTORIAL-CRIACAO-PROJETO-API-Chapter\Chapter.WebApi> dotnet tool install --global dotnet-aspnet-codegenerator --version 6
Você pode invocar a ferramenta usando o comando a seguir: dotnet-aspnet-codegenerator
A ferramenta 'dotnet-aspnet-codegenerator' (versão '6.0.0') foi instalada com êxito.
PS C:\01-TUTORIAL-CRIACAO-PROJETO-API-Chapter\Chapter.WebApi>
PS C:\01-TUTORIAL-CRIACAO-PROJETO-API-Chapter\Chapter.WebApi> |
```

Ln 9, Col 1 (72 selecionado) Espaços: 2 UTF-8 LF XML Go Live [] []

Separando o projeto em camadas

Ao criarmos aplicações, sejam elas aplicações front-end, back-end ou, até mesmo, aplicativos para dispositivos móveis, muitos desenvolvedores já passaram por problemas. Com isso, ao longo do tempo, padrões de código, ferramentas e técnicas vão se tornando padrões para que os mesmos erros não sejam cometidos novamente ou para que a manutenção e sustentabilidade de uma aplicação sejam viáveis.

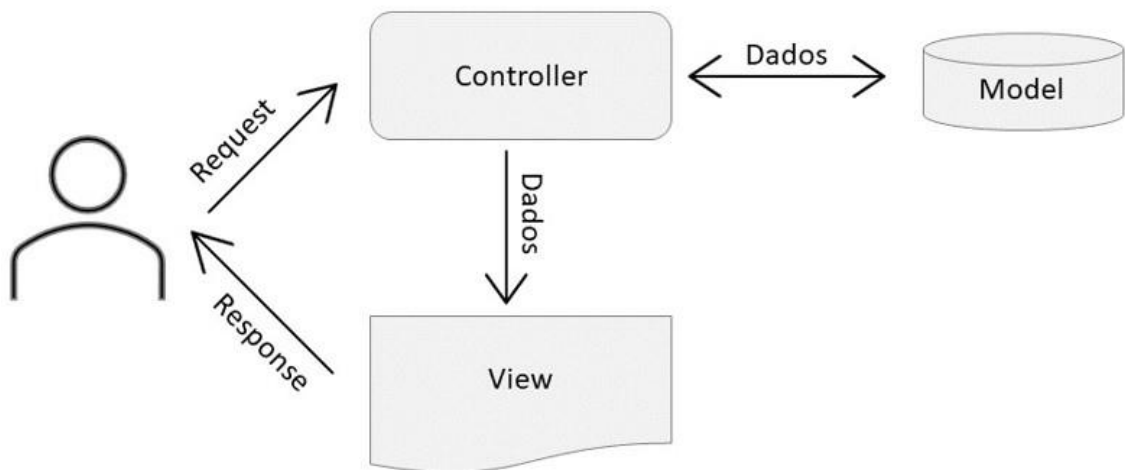
Para facilitar o processo de desenvolvimento, existem padrões que nos auxiliam na demanda de nossas aplicações, os quais são boas soluções para um determinado problema.

MVC

O padrão de arquitetura de software Model-View-Controller (MVC) foi proposto no final da década de 1970, utilizando conceitos de orientação a objetos, que definem que as classes são organizadas em três camadas:

- **view:** responsável por apresentar as informações de maneira visual para o usuário;
- **controller:** responsável por receber as requisições do usuário, processando e repassando as informações entre as camadas; e
- **model:** é a camada do domínio da aplicação e representação dos dados e informações.

O MVC provê uma organização dos nossos recursos, facilidade de colaboração entre os desenvolvedores e segurança entre as camadas.



No projeto de API, não teremos uma camada de visualização (como um HTML ou CSS), visto que devolveremos um JSON e o front-end ficará responsável por renderizar essa informação. Utilizaremos, então, o M (model) e o C (controller). Adicionaremos a camada **Repository**, responsável por trabalhar com a fonte de dados da aplicação, no caso, o banco de dados.

Adicionando o controlador

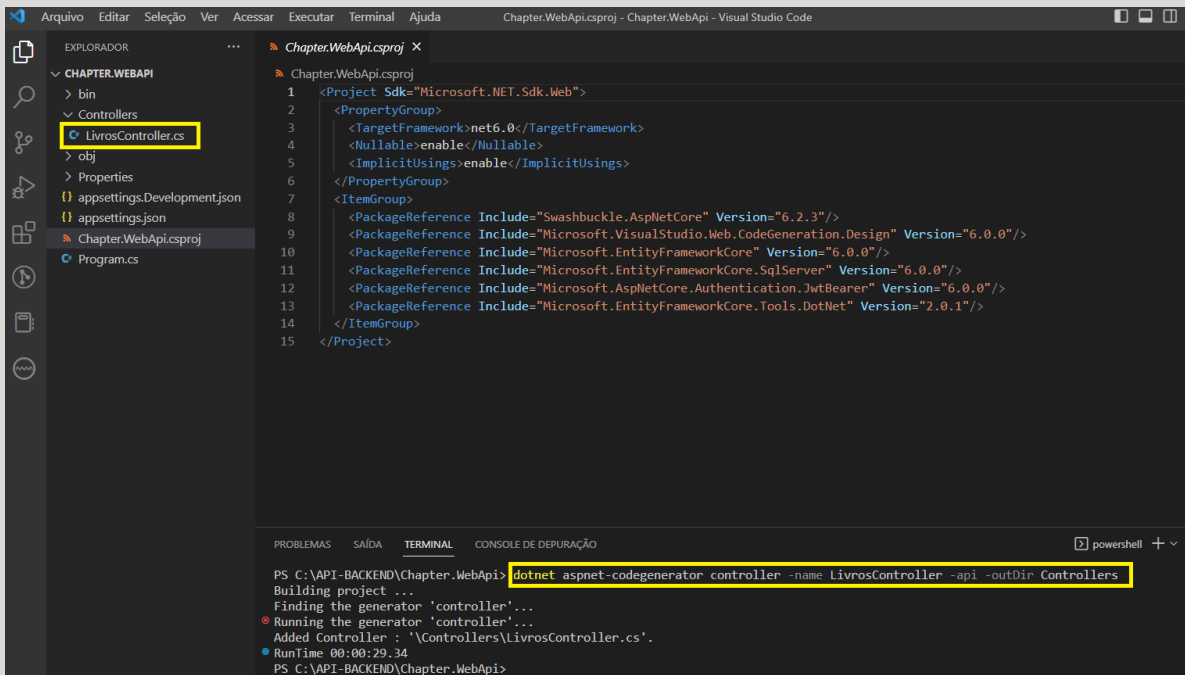
Enquanto a camada de modelo é o domínio da aplicação e a de repositório é responsável pelos dados, a de controlador lida com as solicitações. Quando a aplicação receber uma solicitação HTTP, essa camada é responsável por receber a requisição, realizar os processos necessários e devolver a resposta.

Os controladores derivam da classe **ControllerBase**, que fornece as propriedades e métodos para lidarmos com as solicitações HTTP.

Método	Endpoint	Descrição	Corpo da solicitação	Corpo da resposta
GET	/api/livros	Obter todos os livros	Nenhum	Lista de livros
GET	/api/livros/{id}	Obter um livro por um identificador	Nenhum	Um livro
POST	/api/livros	Adicionar um livro	Dados do livro	Nenhum
PUT	/api/livros/{id}	Atualizar os dados de um livro existente	Dados do livro	Nenhum
DELETE	/api/livros/{id}	Excluir um livro	Nenhum	Nenhum

1. Agora, iremos criar o Controlador API vazio via linha de comando. Para isso, execute no terminal do VS Code o comando abaixo:

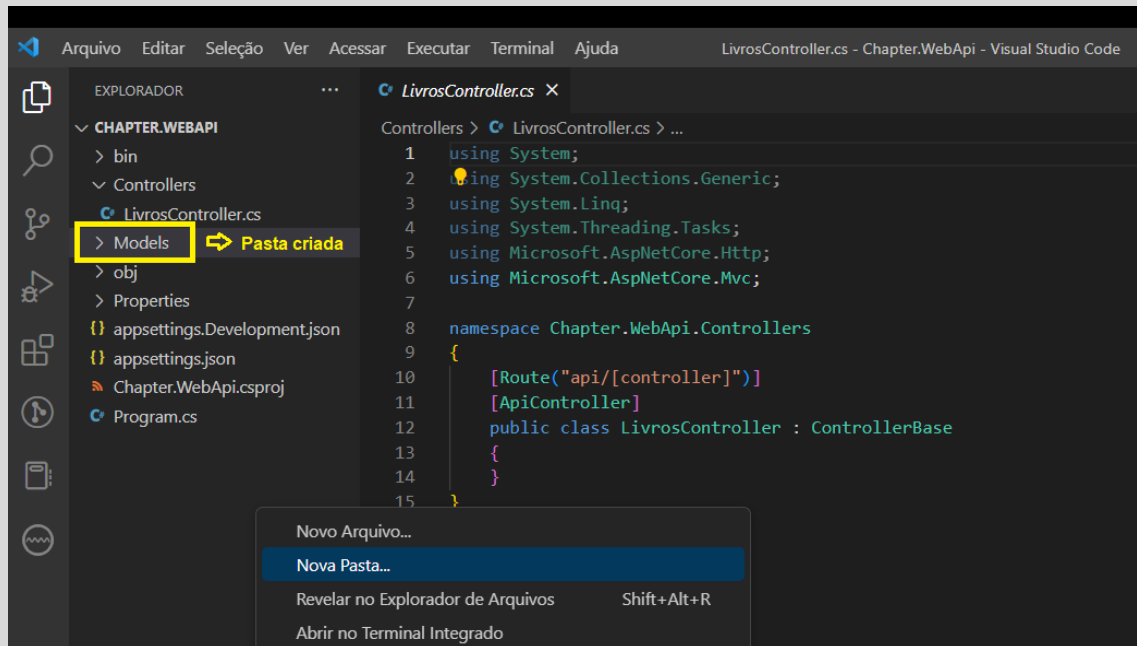
```
dotnet aspnet-codegenerator controller -name LivrosController -api -outDir Controllers
```



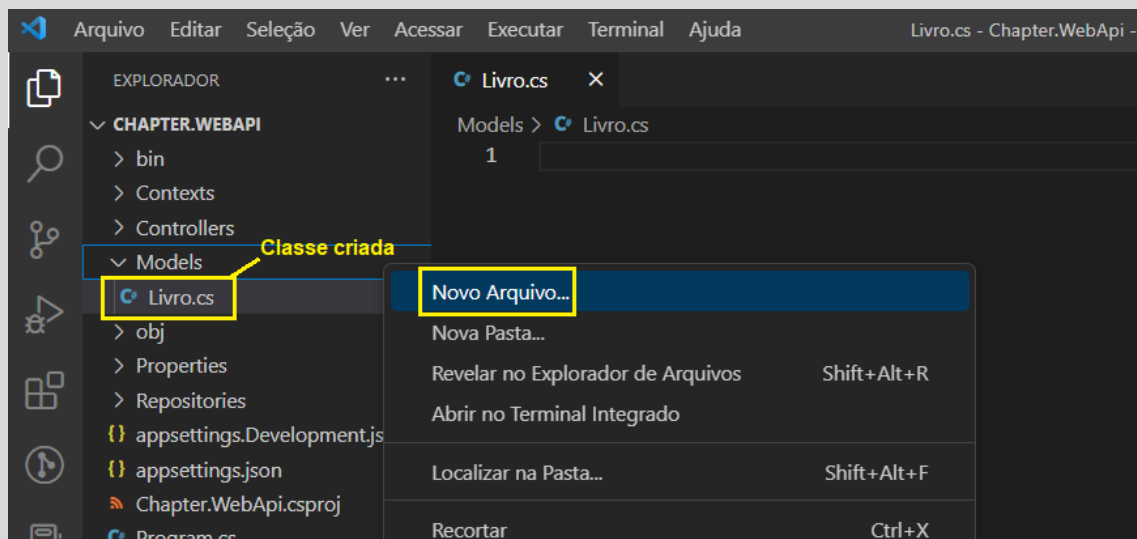
Por enquanto, vamos somente criar a estrutura do projeto com as pastas e os arquivos.

Os parâmetros do comando nomeiam o controlador como **LivrosController**, especifica que se trata de uma API e salva na pasta **Controllers**.

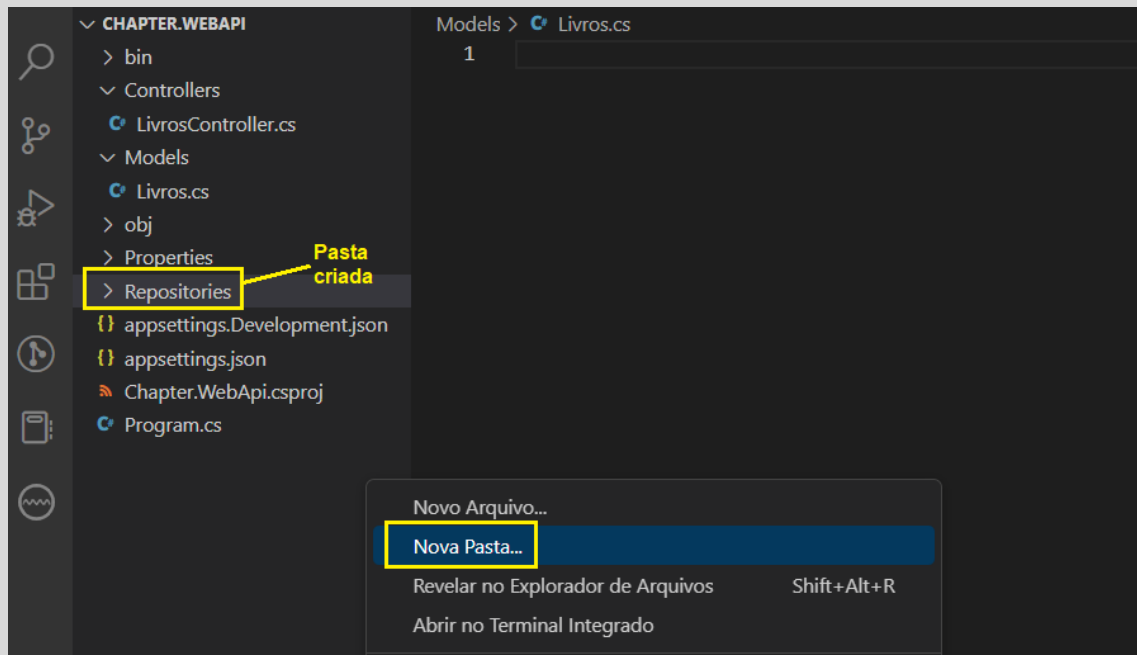
2. Clique com o botão direito dentro do Explorer do VS Code (abaixo do Program.cs) e escolha **Nova Pasta** (New Folder), nomeie-a **Models**.



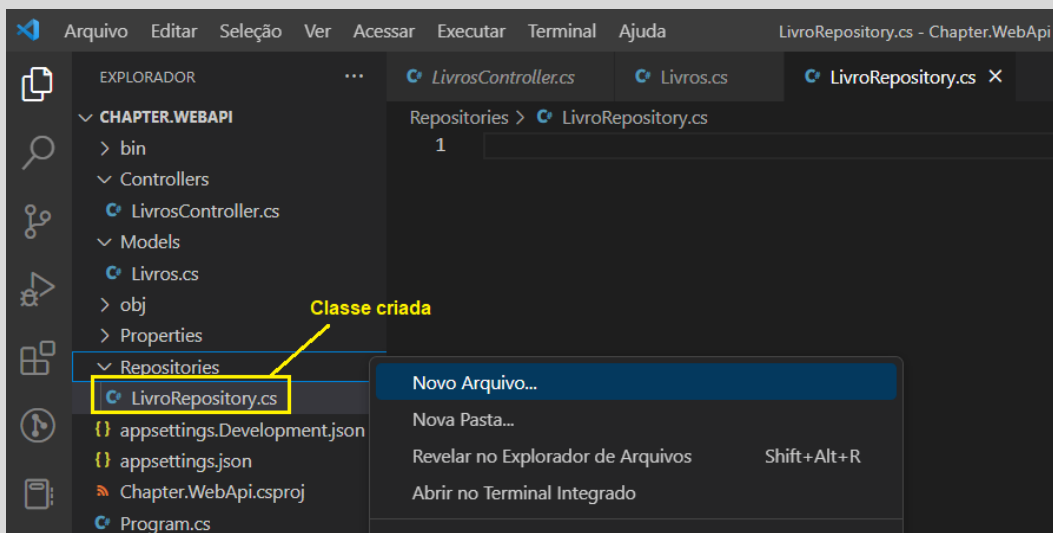
3. Clique com o botão direito em cima da pasta **Models** e crie a classe **Livro.cs**:



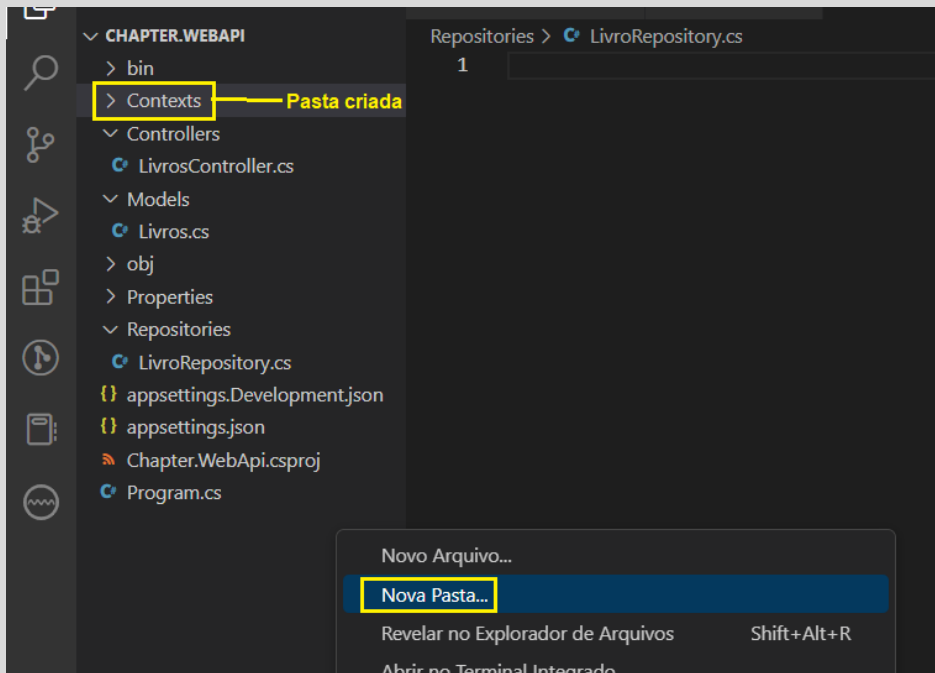
4. Para criar a pasta **Repositories**, clique com o botão direito dentro do Explorer do VS Code (abaixo de **Program.cs**) e escolha **Nova Pasta** (New Folder). Nomeie-a **Repositories**.



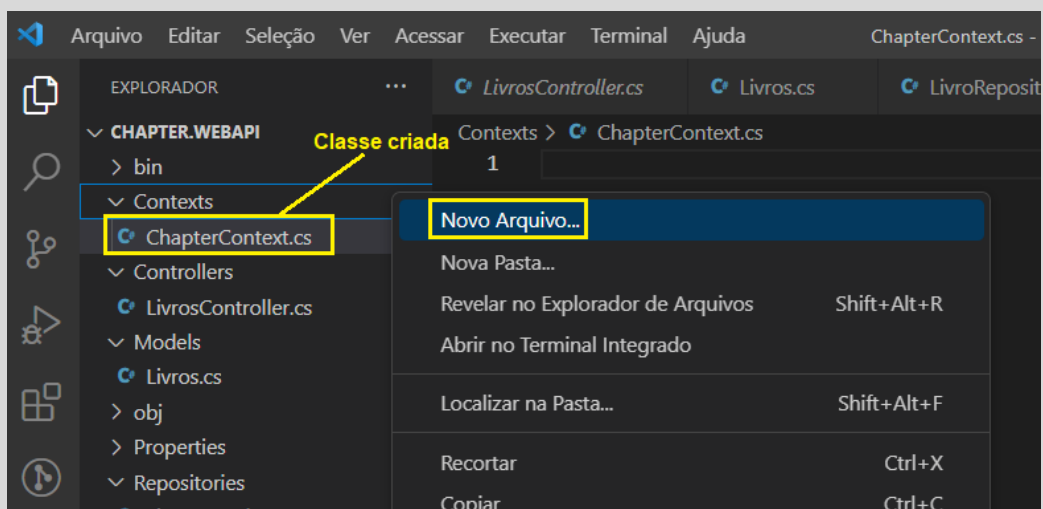
5. Clique com o botão direito em cima da pasta **Repositories** e crie a classe **LivroRepository.cs**:



6. Crie a pasta **Contexts**. Para isso, clique com o botão direito dentro do Explorer do VS Code (abaixo do **Program.cs**) e escolha **Nova Pasta** (New Folder), nomeie-a **Contexts**:



7. Clique com o botão direito em cima da pasta **Contexts** e crie a classe **ChapterContext.cs**:



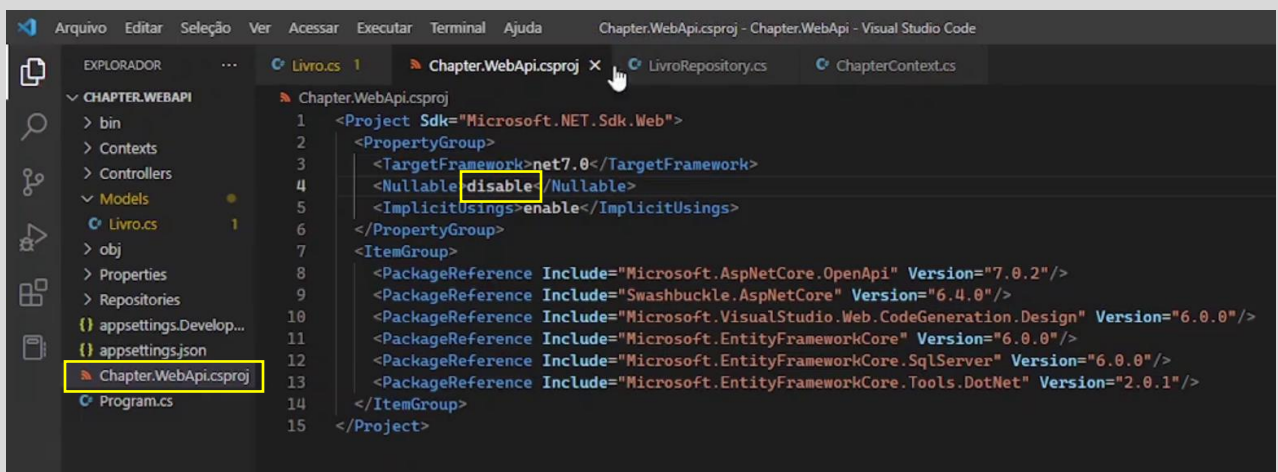
Inserindo os códigos

1. Na classe **Livro.cs**, insira o código abaixo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Chapter.WebApi.Models
{
    public class Livro
    {
        public int Id { get; set; }
        public string Titulo { get; set; }
        public int QuantidadePaginas { get; set; }
        public bool Disponivel { get; set; }
    }
}
```

2. Em **Chapter.WebApi.csproj**, localize a linha **Nullable** e substitua enable por **disable**.



3. Na classe **ChapterContext.cs**, insira o código abaixo.

```
using Chapter.WebApi.Models;
using Microsoft.EntityFrameworkCore;
using System.Data.SqlClient;
using Microsoft.Data.SqlClient;

namespace Chapter.WebApi.Contexts
{
    public class ChapterContext : DbContext
    {
        public ChapterContext()
        {
        }
        public ChapterContext(DbContextOptions<ChapterContext>
options) : base(options)
        {
        }
        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            if(!optionsBuilder.IsConfigured)
            {
                // Essa string de conexão foi depende da SUA máquina.
                optionsBuilder.UseSqlServer("Server=localhost\\SQLEXP
RESS01;"
+
"Database=Chapter;Trusted_Connection=True;");

                // Exemplo 1 de string de conexão:
                // User ID=sa;Password=admin;Server=localhost;Database=Chapter;-
                // + Trusted_Connection=False;

                // Exemplo 2 de string de conexão:
                //
                Server=localhost\\SQLEXPRESS;Database=Chapter;Trusted_Connection=True
;
            }
        }
        public DbSet<Livro> Livros { get; set; }
    }
}
```

Importante

O código anterior possui comentários com informações importantes, caso seja necessária a substituição da string de conexão do servidor SQL.



4. Na classe **LivroRepository.cs**, insira o código abaixo.

```
using Chapter.WebApi.Contexts;
using Chapter.WebApi.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Chapter.WebApi.Repositories
{
    public class LivroRepository
    {
        private readonly ChapterContext _context;
        public LivroRepository(ChapterContext context)
        {
            _context = context;
        }
        public List<Livro> Listar()
        {
            return _context.Livros.ToList();
        }
    }
}
```


5. Na classe **LivrosController.cs**, insira o código abaixo.

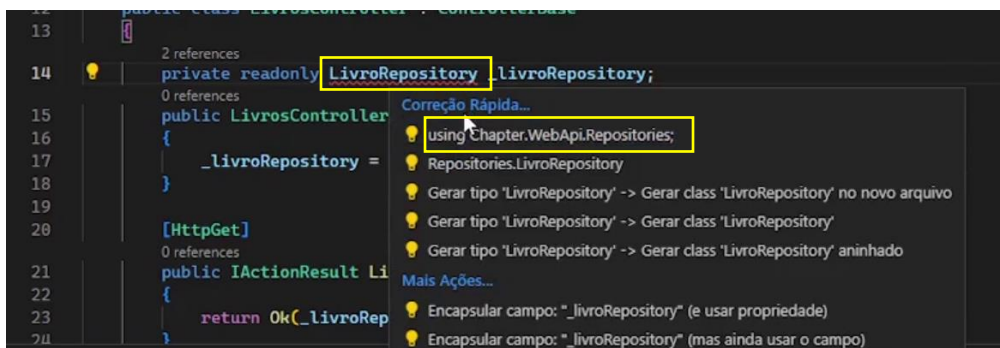
```
using Chapter.WebApi.Models;
using Chapter.WebApi.Repositories;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System;

namespace Chapter.WebApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class LivrosController : ControllerBase
    {
        private readonly LivroRepository _livroRepository;
        public LivrosController(LivroRepository livroRepository)
        {
            _livroRepository = livroRepository;
        }

        [HttpGet]
        public IActionResult Listar()
        {
            return Ok(_livroRepository.Listar());
        }
    }
}
```

Dica!

Para corrigir um erro, clique em cima do trecho sublinhado em vermelho, pressione **Ctrl + .** e selecione uma opção da lista.



6. Na classe **Program.cs**, insira o código abaixo. Você pode substituir o código inteiro ou inserir somente os trechos em destaque.

```
using Chapter.WebApi.Contexts;
using Chapter.WebApi.Repositories;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddScoped<ChapterContext,
ChapterContext>();
builder.Services.AddControllers();
builder.Services.AddTransient<LivroRepository,
LivroRepository>();

var app = builder.Build();

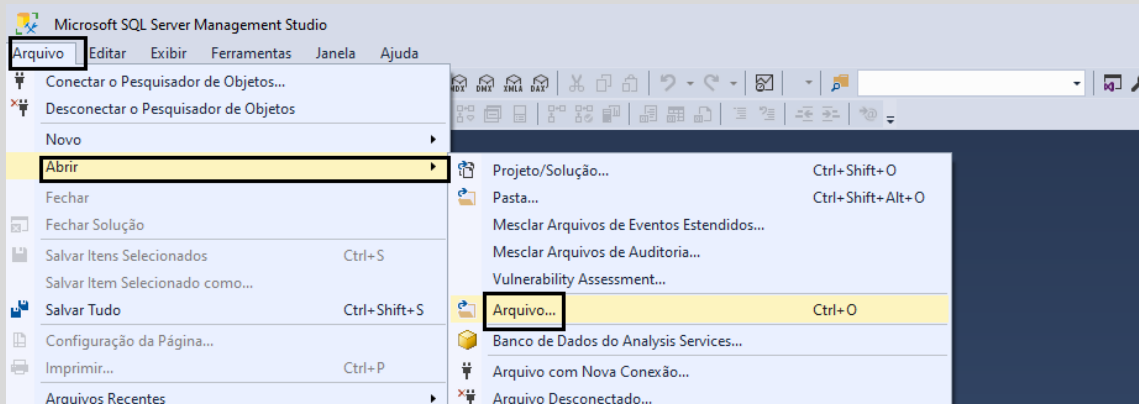
app.UseRouting();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});

app.Run();
```

Criando o banco de dados

1. Baixe o arquivo disponibilizado no material digital (Desafio 1 > Pré-requisito e roteiro de configuração). Acesse o Microsoft SQL Server Management Studio (SSMS), acesse **Arquivo** > **Abrir** > **Arquivo...** e escolha o arquivo baixado.

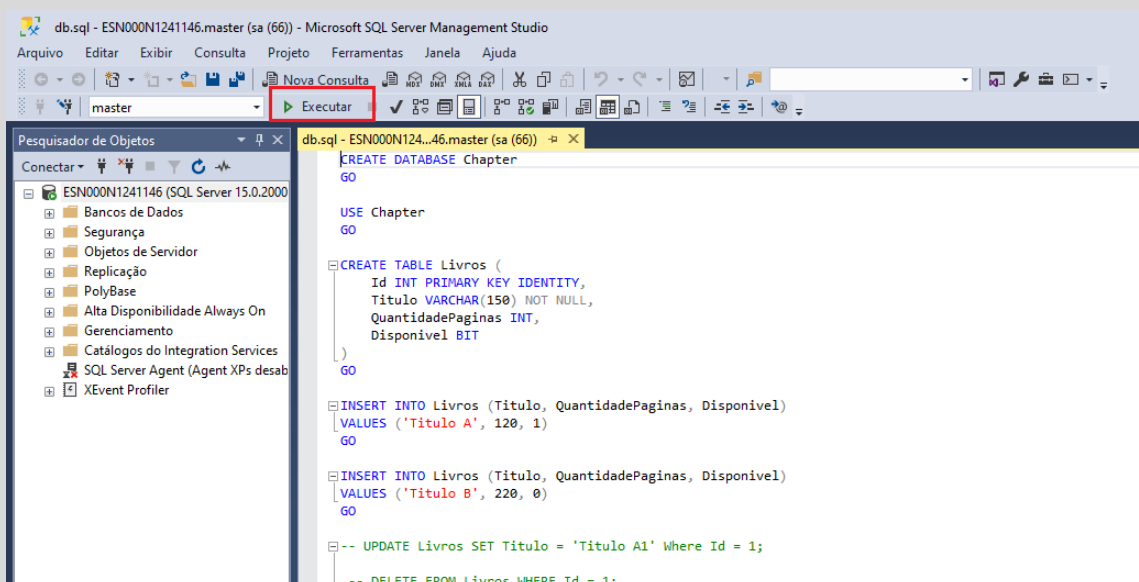


Importante

Você também pode baixar o arquivo do passo acima clicando [aqui](#).



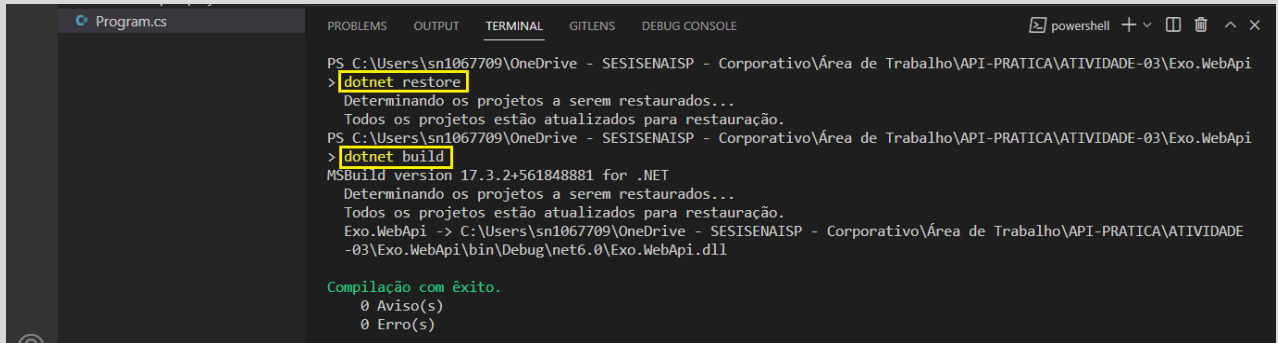
2. Clique em **Abrir** e depois em **Executar** para que o banco seja criado.



Executando o projeto

1. No terminal do VS Code, execute os códigos abaixo, um após o outro.

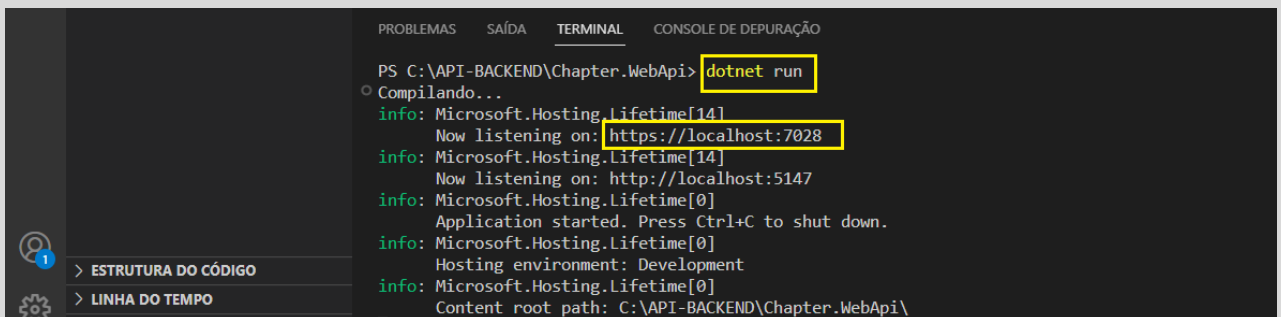
```
dotnet restore
dotnet build
dotnet run
```



```
PS C:\Users\sn1067709\OneDrive - SESISENAISP - Corporativo\Área de Trabalho\API-PRATICA\ATIVIDADE-03\Exo.WebApi> dotnet restore
Determinando os projetos a serem restaurados...
Todos os projetos estão atualizados para restauração.
PS C:\Users\sn1067709\OneDrive - SESISENAISP - Corporativo\Área de Trabalho\API-PRATICA\ATIVIDADE-03\Exo.WebApi> dotnet build
MSBuild version 17.3.2+561848881 for .NET
Determinando os projetos a serem restaurados...
Todos os projetos estão atualizados para restauração.
Exo.WebApi -> C:\Users\sn1067709\OneDrive - SESISENAISP - Corporativo\Área de Trabalho\API-PRATICA\ATIVIDADE-03\Exo.WebApi\bin\Debug\net6.0\Exo.WebApi.dll

Compilação com êxito.
0 Aviso(s)
0 Erro(s)
```

2. Ao executar o último comando, **dotnet run**, o servidor será inicializado e disponibilizará a url de acesso ao projeto. Será com esse endereço, sucedido do sufixo **api/livros**, que você acessará a API desenvolvida:



```
PS C:\API-BACKEND\Chapter.WebApi> dotnet run
Compilando...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7028
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5147
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\API-BACKEND\Chapter.WebApi\
```

Importante

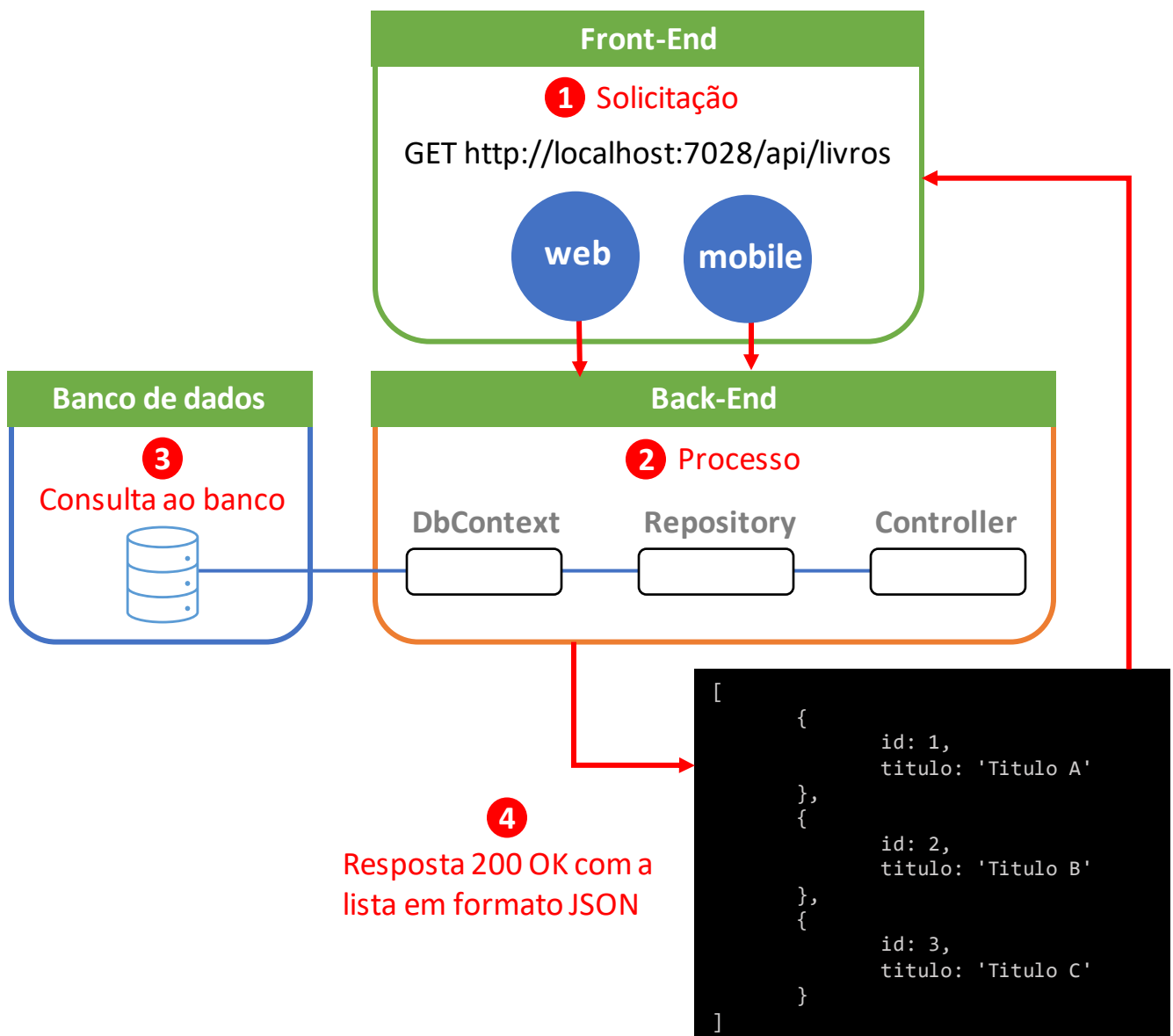
Em nosso exemplo, o número da porta na url gerada é **7020**, porém, poderá ser outra porta. Fique atento para utilizar o link gerado em seu terminal.



Entendendo o funcionamento da API

As informações salvas no banco de dados foram lidas e mapeadas em objetos da aplicação e disponibilizadas em formato JSON para representação.

O diagrama abaixo representa o processo desde a solicitação de um recurso por um dispositivo web ou mobile, processamento da informação no back-end, consulta no banco de dados e retorno da informação.



Uma das vantagens da API é que o aplicativo web e o aplicativo mobile podem construir a interface visual conforme especificação baseada no retorno do JSON. Esse tipo de arquitetura possibilita que o front-end/mobile não conheçam os detalhes de implementação do serviço back-end e renderizem a estrutura visualmente customizada para cada aplicativo, tendo em comum a mesma camada de abstração.

Acompanhe a interface web construída a partir do retorno JSON disponibilizado na imagem abaixo.



The screenshot shows a web browser window with a URL pointing to a file. The page displays a table titled 'LIVROS' (Books). The table has four columns: '#', 'Titulo', 'Quantidade de Páginas', and 'Disponível?'. There are two rows of data: Row 1 with '1', 'Titulo A', '120', and 'Sim'; Row 2 with '2', 'Titulo B', '220', and 'Não'.

#	Titulo	Quantidade de Páginas	Disponível?
1	Titulo A	120	Sim
2	Titulo B	220	Não

Na imagem ao lado, é apresentada a interface mobile a partir do retorno JSON disponibilizado.



Continuação do CRUD com teste no Insomnia

Para buscar as informações de um livro, adicionaremos um endpoint para ser responsável por receber a solicitação de GET/api/livros/{id} e adicionaremos a camada de repositório, um método para buscar as informações de um livro na base.

Método	Endpoint	Descrição	Corpo da solicitação	Corpo da resposta
GET	/api/livros	Obter todos os livros	Nenhum	Lista de livros
GET	/api/livros/{id}	Obter um livro por um identificador	Nenhum	Um livro
POST	/api/livros	Adicionar um livro	Dados do livro	Nenhum
PUT	/api/livros/{id}	Atualizar os dados de um livro existente	Dados do livro	Nenhum
DELETE	/api/livros/{id}	Excluir um livro	Nenhum	Nenhum

Importante

Nessa etapa, vamos transitar entre VS Code e Insomnia. Para aplicar as alterações de código no VS Code, será necessário acessar o **Terminal** e pressionar **Ctrl + C**, parando a aplicação. Depois, digite **dotnet run** para reiniciá-la.



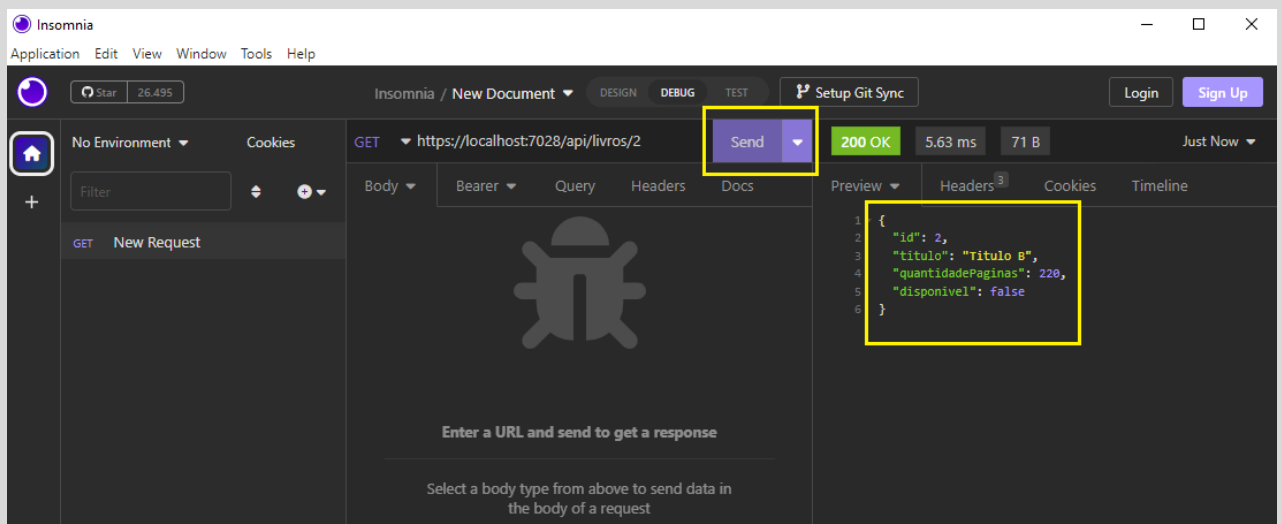
1. Na classe **LivroRepository.cs**, adicione o código abaixo.

```
// buscar as informações de um livro pelo id
public Livro BuscaPorId(int id)
{
    // select where id = id
    return _context.Livros.Find(id);
}
```

2. Na classe **LivrosController.cs**, vamos adicionar o código abaixo para inserir o novo recurso para buscar um livro por id através do método GET.

```
// get /api/livros/{id}
[HttpGet("{id}")] // busca pelo id passado
public IActionResult BuscarPorId(int id)
{
    Livro livro = _livroRepository.BuscaPorId(id);
    if (livro == null)
    {
        return NotFound();
    }
    return Ok(livro);
}
```

3. Inicie a aplicação Insomnia. Escolha o método GET, insira a url **http://localhost:7028/api/livros/2** e clique em **SEND**. Nesse exemplo, o resultado será o segundo item do banco, pois, no id, você inseriu 2:



Para atualizar as informações do livro, adicionaremos um novo endpoint que as receberá a partir de **PUT** `/api/livros/{identificador}` e adicionaremos o método responsável por atualizar as informações de um livro no banco de dados.

Método	Endpoint	Descrição	Corpo da solicitação	Corpo da resposta
GET	<code>/api/livros</code>	Obter todos os livros	Nenhum	Lista de livros
GET	<code>/api/livros/{id}</code>	Obter um livro por um identificador	Nenhum	Um livro
POST	<code>/api/livros</code>	Adicionar um livro	Dados do livro	Nenhum
PUT	<code>/api/livros/{id}</code>	Atualizar os dados de um livro existente	Dados do livro	Nenhum
DELETE	<code>/api/livros/{id}</code>	Excluir um livro	Nenhum	Nenhum

4. Na classe **LivroRepository.cs**, adicione o código abaixo.

```
// atualizar as infos de um livro
public void Atualizar(int id, Livro livro)
{
    // busca o livro pelo id
    Livro livroBuscado = _context.Livros.Find(id);

    if (livroBuscado != null)
    {
        livroBuscado.Titulo = livro.Titulo;
        livroBuscado.QuantidadePaginas =
livro.QuantidadePaginas;
        livroBuscado.Disponivel = livro.Disponivel;
    }

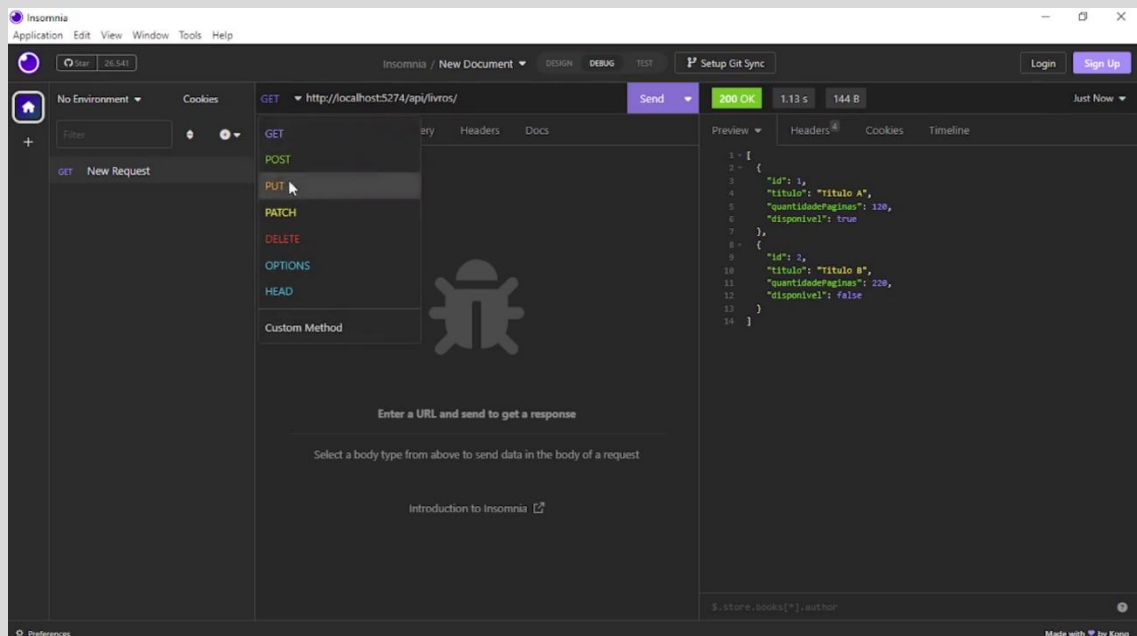
    _context.Livros.Update(livroBuscado);

    _context.SaveChanges();
}
```

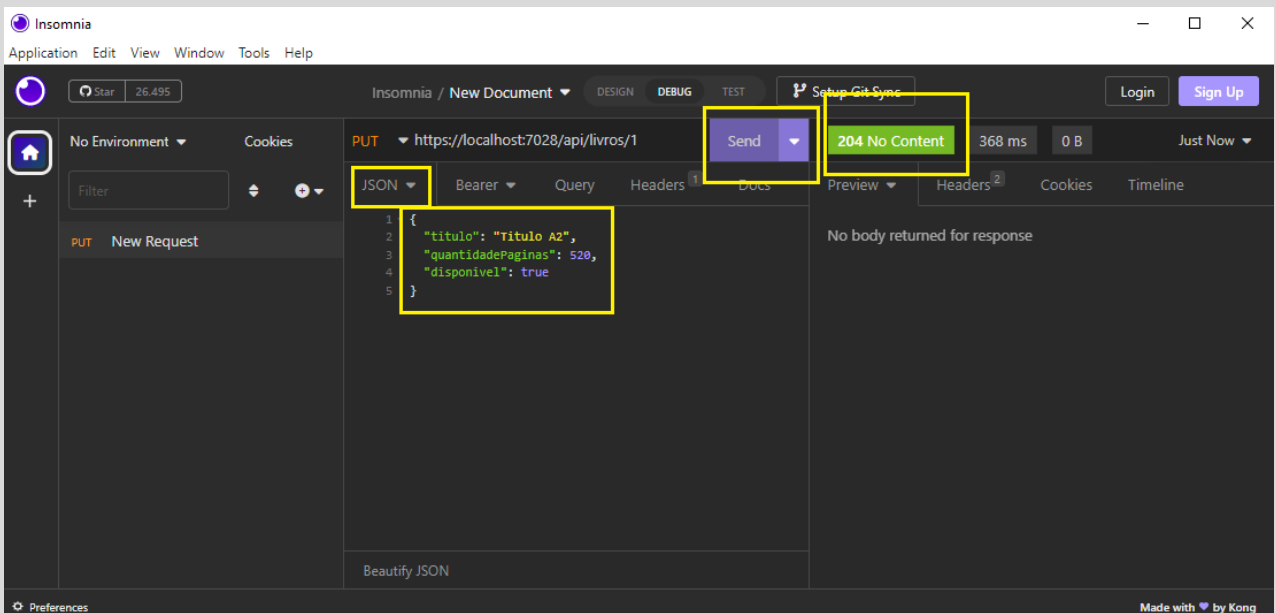
5. Na classe **LivrosController.cs**, adicione o código abaixo.

```
// put /api/livros/{id}
// recebe a informação do livro
// atualiza o corpo da requisição
[HttpPut("{id}")] // o id passado no put
/api/livros/1
public IActionResult Atualizar(int id, Livro livro)
{
    _livroRepository.Atualizar(id, livro);
    return StatusCode(204);
}
```

6. Agora, vamos acessar o Insomnia e testar se o método Atualizar (PUT) está funcional. Execute o projeto com o comando **dotnet run** e inicie a aplicação Insomnia. Escolha o método PUT e insira a url **https://localhost:7028/api/livros/2**:



7. Onde está escrito Body, troque por **JSON** o método da requisição, preencha com os novos dados e clique em **Send**. Nesse exemplo, o resultado será a alteração do primeiro item do banco, pois, no id, você inseriu 2:



```
{
  "titulo": "Titulo D",
  "quantidadePaginas": 820,
  "disponivel": true
}
```

Para adicionar um livro no banco de dados, criaremos um método na camada de repositório para inserirmos um registro no banco de dados e adicionaremos um novo recurso no controlador para ser responsável por receber essa requisição, processar e retornar o resultado.

Método	Endpoint	Descrição	Corpo da solicitação	Corpo da resposta
GET	/api/livros	Obter todos os livros	Nenhum	Lista de livros
GET	/api/livros/{id}	Obter um livro por um identificador	Nenhum	Um livro
POST	/api/livros	Adicionar um livro	Dados do livro	Nenhum
PUT	/api/livros/{id}	Atualizar os dados de um livro existente	Dados do livro	Nenhum
DELETE	/api/livros/{id}	Excluir um livro	Nenhum	Nenhum

8. Na classe **LivroRepository.cs**, adicione o código abaixo.

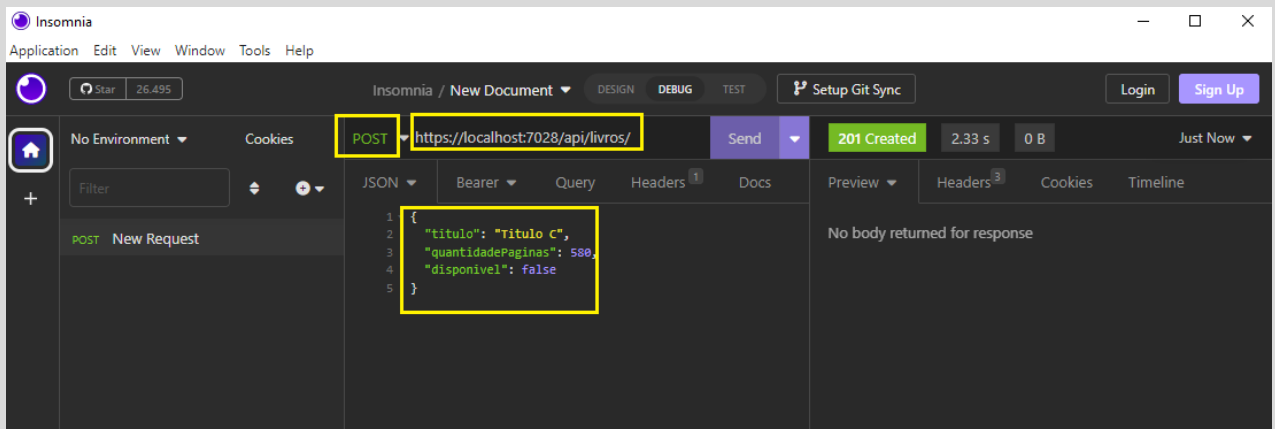
```
// cadastrar livro no bd
public void Cadastrar(Livro livro)
{
    // adiciona o novo livro
    _context.Livros.Add(livro);

    // salva
    _context.SaveChanges();
}
```

9. Na classe **LivrosController.cs**, adicione o código abaixo.

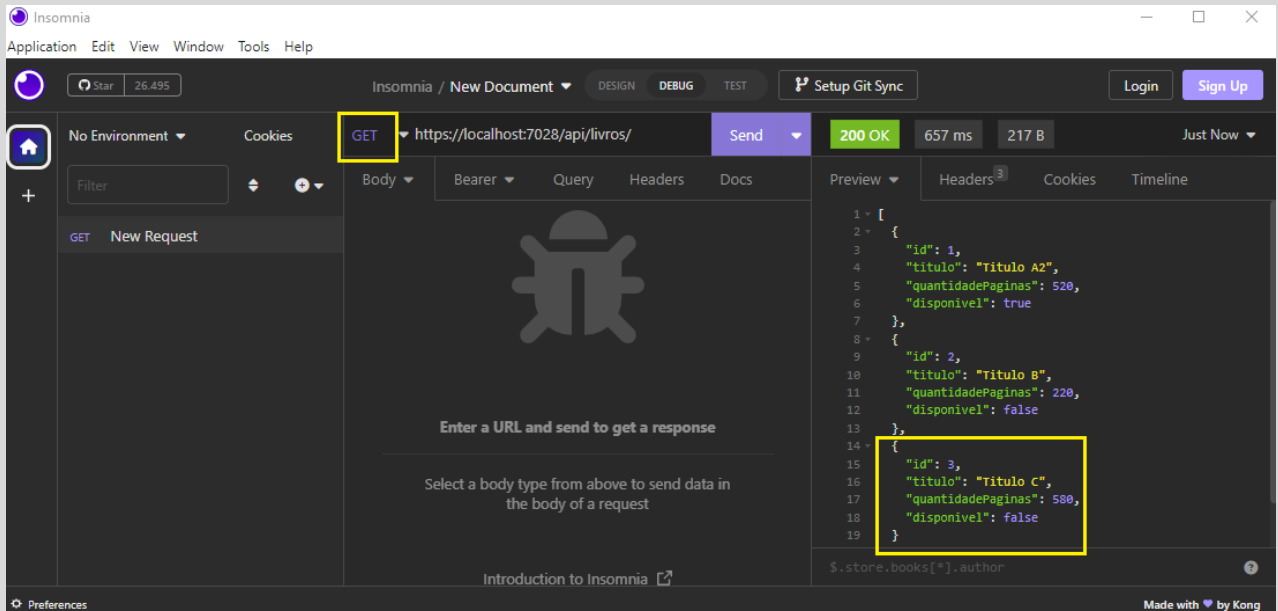
```
// post /api/livros/  
// recebe a info do livro que deseja salvar do  
corpo da requisição  
[HttpPost]  
public IActionResult Cadastrar(Livro livro)  
{  
    _livroRepository.Cadastrar(livro);  
    return StatusCode(201);  
}
```

10. Agora, vamos acessar o **Insomnia** e testar se o método Cadastrar (POST) está funcional. No **Insomnia**, escolha o método POST e insira a url **https://localhost:7028/api/livros/**. Altere a requisição para **JSON**, insira o novo livro que está abaixo e clique em **SEND**.



```
{  
    "titulo": "Titulo C",  
    "quantidadePaginas": 999,  
    "disponivel": false  
}
```

11. Ao fazer uma nova requisição GET para a listagem dos livros, o novo livro aparecerá nos resultados:



Para deletar um registro do livro, adicionaremos um novo endpoint, que receberá as informações a partir de **DELETE**/api/livros/{identificador}, bem como o método responsável por deletar a linha correspondente no banco de dados.

Método	Endpoint	Descrição	Corpo da solicitação	Corpo da resposta
GET	/api/livros	Obter todos os livros	Nenhum	Lista de livros
GET	/api/livros/{id}	Obter um livro por um identificador	Nenhum	Um livro
POST	/api/livros	Adicionar um livro	Dados do livro	Nenhum
PUT	/api/livros/{id}	Atualizar os dados de um livro existente	Dados do livro	Nenhum
DELETE	/api/livros/{id}	Excluir um livro	Nenhum	Nenhum

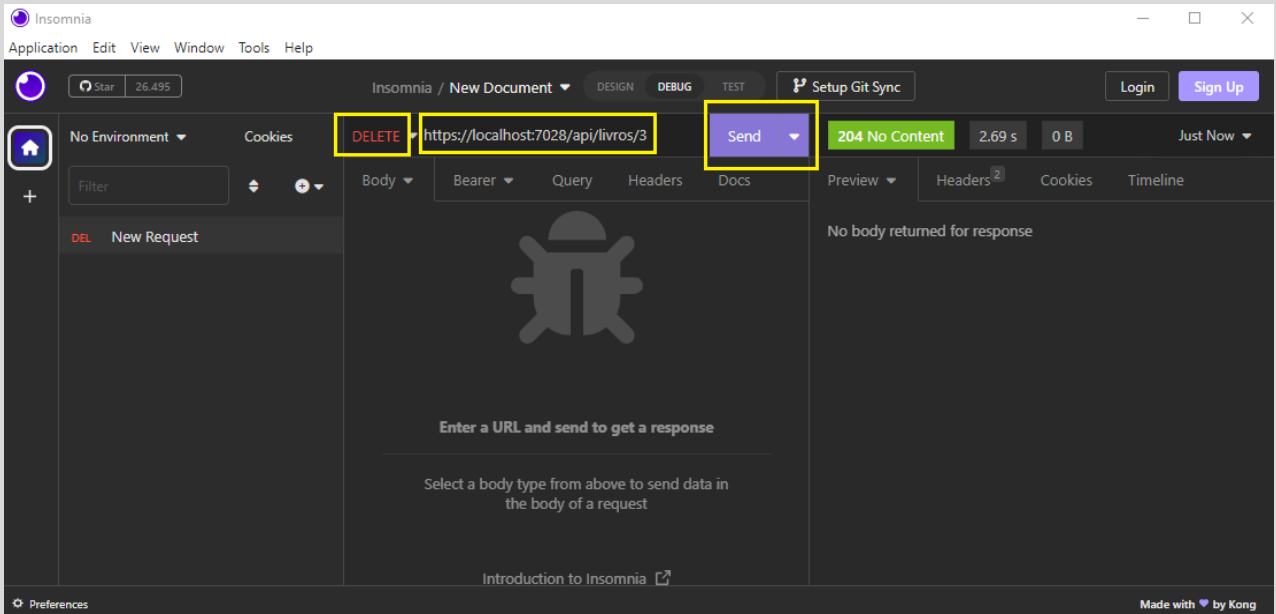
12. Na classe **LivroRepository.cs**, adicione o código abaixo.

```
// deleta o livro a partir de um id
public void Deletar(int id)
{
    // busca
    Livro livroBuscado = _context.Livros.Find(id);
    // remove o livro
    _context.Livros.Remove(livroBuscado);
    //salva
    _context.SaveChanges();
}
```

13. Na classe **LivrosController.cs**, adicione o código abaixo.

```
// delete /api/livros/{id}
[HttpDelete("{id}")] // o id passado no delete
/api/livros/1
public IActionResult Deletar(int id)
{
    _livroRepository.Deletar(id);
    return StatusCode(204);
}
```

14. No **Insomnia**, escolha o método **DELETE** e insira a url **https://localhost:7028/api/livros/3** para apagar o livro com id 3, conforme a imagem abaixo.



15. Ao fazer uma nova requisição GET para a listagem dos livros, o de id 3 não aparecerá mais nos resultados:

