C# MVC REST API

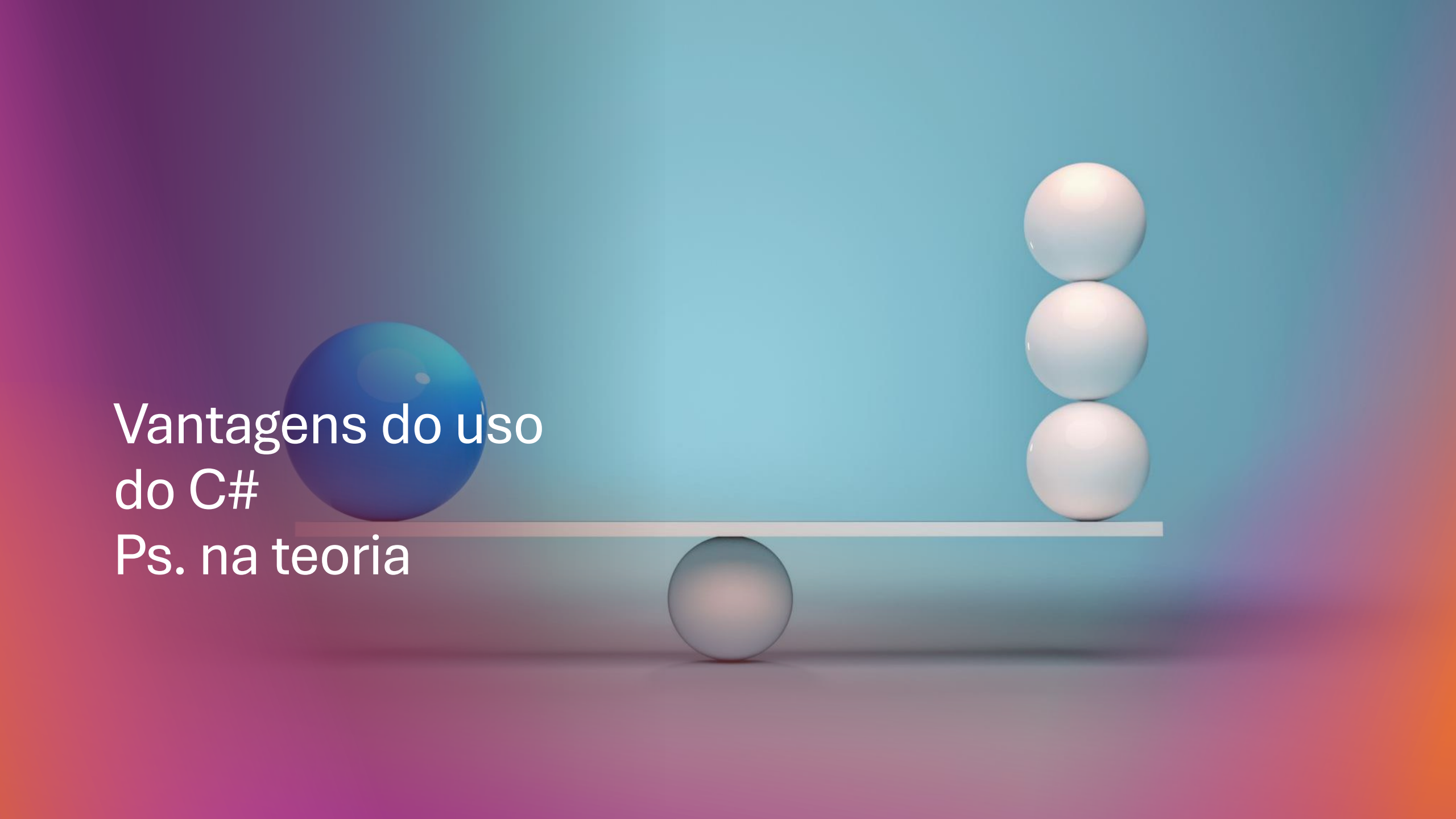
Prof. Me. Hélio Esperidião

Kestrel

- O **Kestrel** é o servidor web interno usado pelo **ASP.NET Core** para lidar com requisições HTTP. Ele é um servidor web multiplataforma e de alto desempenho, sendo o servidor web padrão para aplicações ASP.NET Core. O Kestrel é utilizado principalmente para hospedar a aplicação web e gerenciar a comunicação entre a aplicação e os clientes, como navegadores ou APIs externas.
- **Características do Kestrel:**

1. **Alta performance**: O Kestrel é projetado para ser rápido e eficiente, sendo capaz de lidar com milhares de conexões simultâneas.
2. **Multiplataforma**: Ele funciona em várias plataformas, incluindo Windows, Linux e macOS.
3. **Escuta requisições HTTP/HTTPS**: O Kestrel gerencia as requisições HTTP e HTTPS que chegam à aplicação, repassando essas requisições para o pipeline de middleware do ASP.NET Core.
4. **Suporte a SSL/TLS**: Ele pode ser configurado para servir conteúdos via HTTPS usando certificados SSL/TLS, oferecendo segurança nas comunicações.
5. **Configuração de portas e IPs**: O Kestrel permite configurar em quais portas e endereços IP a aplicação deve escutar

Vantagens do uso do C#
Ps. na teoria



Ambiente de Desenvolvimento Robusto

- **Visual Studio**: O C# possui um dos IDEs mais avançados, o Visual Studio, que oferece ferramentas poderosas para depuração, testes e desenvolvimento.

- **IntelliSense**: O suporte à autocompletação de código facilita a escrita e a manutenção do código.

Ecossistema Amplo

- **Bibliotecas e Pacotes**: O C# e o ASP.NET Core têm um ecossistema rico com bibliotecas e pacotes disponíveis através do NuGet, facilitando a integração de funcionalidades adicionais.
- **Suporte a Microserviços**: O ASP.NET Core oferece suporte robusto para a criação de microserviços, permitindo que desenvolvedores construam sistemas distribuídos.

Segurança

- **Recursos de Segurança Integrados**: O ASP.NET Core possui recursos integrados para autenticação e autorização, como suporte para JWT (JSON Web Tokens) e OAuth.

- **Proteção Contra Vulnerabilidades**: O framework possui proteções contra as principais vulnerabilidades de segurança, como injeções de SQL e XSS.

Integração com Tecnologias Microsoft

- **Compatibilidade com Azure**: C# se integra facilmente com serviços da nuvem da Microsoft, como o Azure, facilitando a implantação e escalabilidade de aplicações.
- **Interoperabilidade**: O C# pode interagir facilmente com outras tecnologias da Microsoft, como SQL Server e serviços de mensageria.

Suporte a Programação Assíncrona

- **Programação Assíncrona**: O C# oferece suporte nativo para programação assíncrona, permitindo que aplicações lidem com operações de entrada/saída sem bloquear threads, o que é fundamental para aplicações de rede.

Padrões de Design

- **Suporte a Padrões de Projeto**: C# e ASP.NET Core incentivam o uso de padrões de projeto como MVC (Model-View-Controller) e Repository, que ajudam a manter o código organizado e fácil de manter.

Comunitário e Suporte

- **Comunidade Ativa**: C# tem uma grande comunidade de desenvolvedores que oferecem suporte, documentação e recursos para ajudar novos desenvolvedores.
- **Documentação**: A documentação oficial da Microsoft é abrangente e detalhada, tornando mais fácil para desenvolvedores aprenderem e resolverem problemas.

Versatilidade

- **Desenvolvimento Multi-Plataforma**: Com o ASP.NET Core, é possível desenvolver aplicações que funcionam em diferentes plataformas, como Windows, macOS e Linux.

Teste e Manutenção

- **Facilidade de Testes**: C# suporta testes unitários e de integração, facilitando a manutenção e a qualidade do código através de práticas de TDD (Test-Driven Development).

Desvantagens do uso do C# na web

Curva de Aprendizado

- **Complexidade**: C# pode ser considerado mais complexo para iniciantes em comparação com linguagens como Python ou JavaScript, especialmente devido à sua vasta gama de funcionalidades e conceitos avançados (como programação assíncrona, LINQ, etc.).
- **Frameworks**: A variedade de frameworks e bibliotecas disponíveis pode ser confusa para novos desenvolvedores que estão começando.

Custo de Licenciamento

- **Ambiente de Desenvolvimento**: Embora o .NET Core seja gratuito e de código aberto, algumas ferramentas de desenvolvimento e ambientes, como o Visual Studio (especialmente as versões mais completas), podem ter custos associados. Isso pode ser uma preocupação para pequenas empresas ou projetos de baixo orçamento.

Dependência de Plataforma

- Windows: C# e .NET tradicionalmente têm sido mais associados ao ecossistema Windows, o que pode ser uma limitação em ambientes onde outras plataformas (como Linux ou macOS) são preferidas. No entanto, o .NET Core e o .NET 5+ têm abordado essa questão, permitindo que aplicativos sejam executados em várias plataformas.

Performance em Cargas Altas

- **Overhead**: O C# e o ambiente .NET têm algum overhead em comparação com linguagens compiladas diretamente para código nativo, como C ou C++. Embora o desempenho do C# seja excelente na maioria dos cenários, para aplicações que exigem o máximo de eficiência em recursos, outras linguagens podem ser mais adequadas.

Bibliotecas e Suporte de Terceiros

- Menor Diversidade em Algumas Áreas: Embora o C# tenha uma boa quantidade de bibliotecas disponíveis, algumas áreas específicas podem ter menos opções em comparação com outras linguagens populares, como JavaScript, que possui um ecossistema muito rico, especialmente para desenvolvimento web.

Desenvolvimento em Tempo Real

- Complexidade Adicional: Para aplicativos que exigem atualizações em tempo real (como chats ou jogos), a implementação pode ser mais complexa em C# do que em Node.js, que foi projetado com esse tipo de operação em mente.

Menos Popularidade em Algumas Áreas

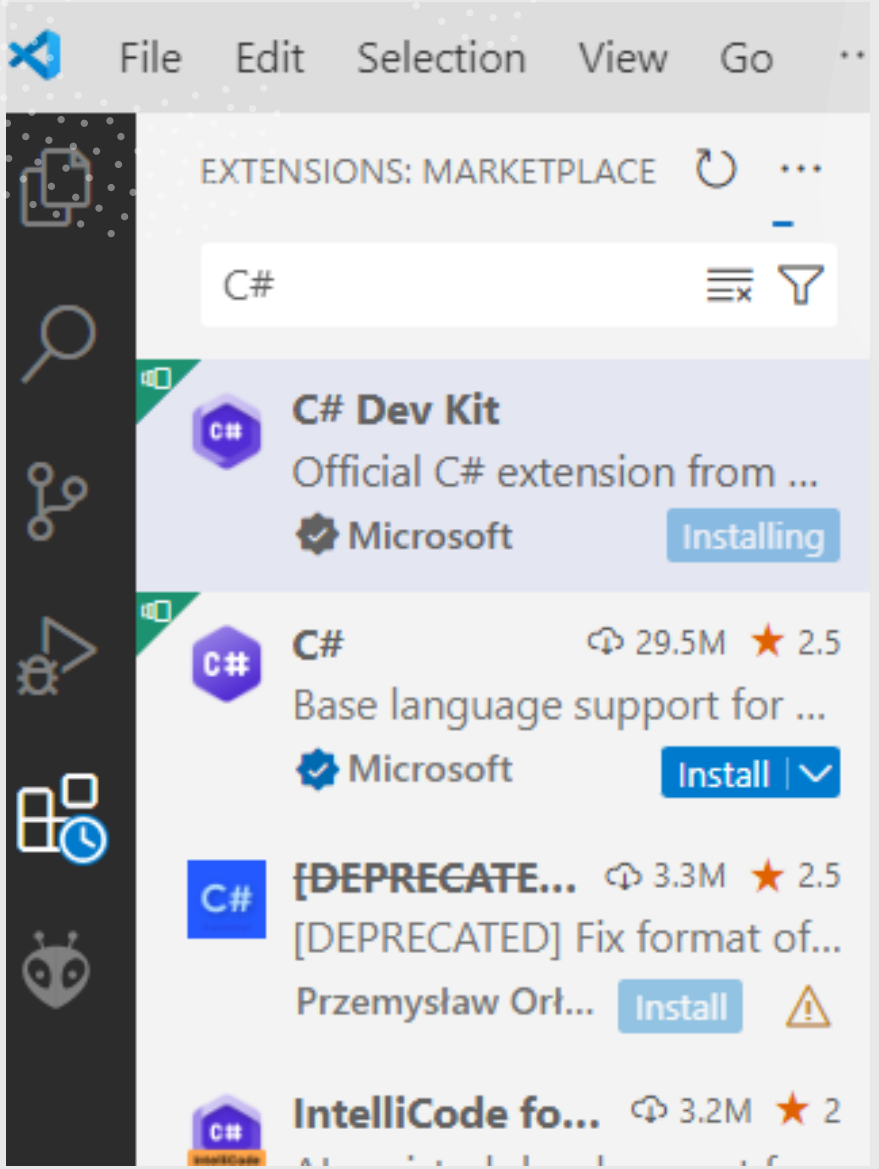
- Adoção em Web e Mobile: Embora C# e ASP.NET sejam populares para desenvolvimento de back-end, outras linguagens como php e JavaScript (com Node.js) têm dominado o desenvolvimento web, enquanto linguagens como Java e Swift são mais comuns no desenvolvimento mobile. Isso pode resultar em uma comunidade menor e menos recursos disponíveis em algumas áreas.

Atualizações e Mudanças de Versão

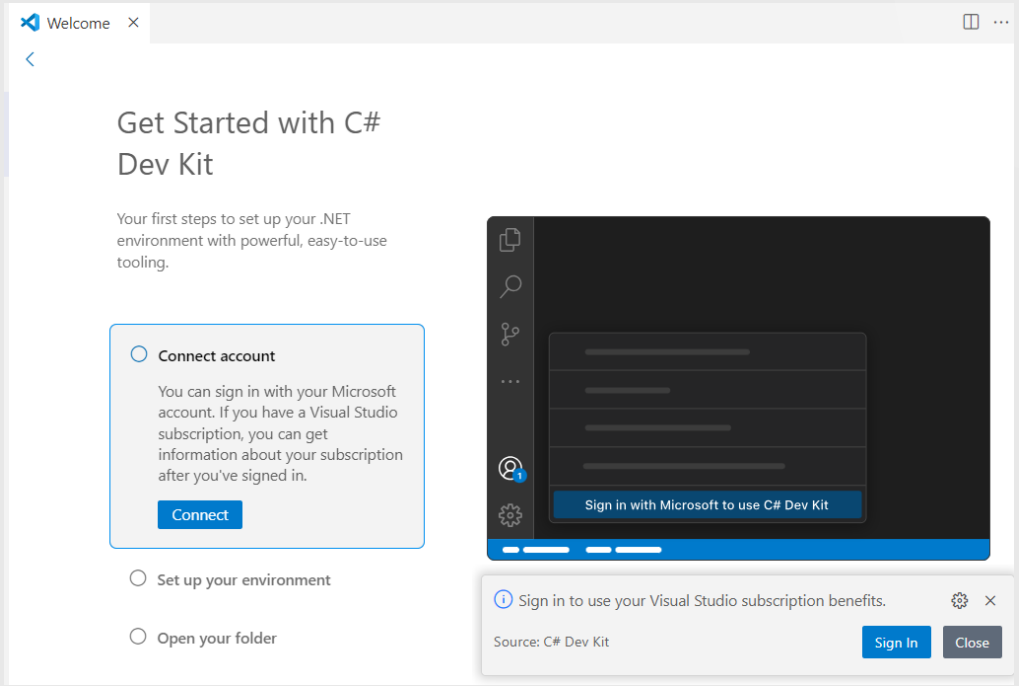
- **Mudanças de API**: O C# e o .NET têm passado por mudanças significativas ao longo dos anos, o que pode levar a problemas de compatibilidade entre versões e requer aprendizado adicional quando novas funcionalidades são introduzidas.

C# Dev Kit extension

- Instale a extensão:
  - C# dev Kit



Fazer login após a instalação é opcional.



dotnet new webapi -n RetanguloApi

- Crie uma pasta em branco e abra a pasta no visual studio code.
- Abra o terminal dentro do visual studio code e digite:
  - dotnet new webapi -n `RetanguloApi`
- Esse é um comando do ambiente de desenvolvimento do .net que permite criar um novo projeto do tipo web api, e o nome do projeto será: “RetanguloApi”

Crie as pastas

- Após a criação do projeto crie as pastas adicionais: Modelo e Controle.
- Crie utilizando o ambiente gráfico do visual studio.
- Também é possível criar utilizando o terminal:
  - Md Modelo
  - Md Controle

Modelo/Retangulo.cs

```
// Declaração do namespace para organizar o código e evitar conflitos de nomes
namespace RetanguloApi.Modelo{
    // Definição da classe Retangulo
    public class Retangulo    {
        public double Largura { get; set; } // Propriedade que representa a largura do retângulo
        public double Altura { get; set; }  // Propriedade que representa a altura do retângulo

        // Construtor da classe Retangulo que inicializa as propriedades Largura e Altura
        public Retangulo(double largura, double altura) {
           Largura = largura; // Atribui o valor de largura passado como argumento à propriedade Largura
           Altura = altura;   // Atribui o valor de altura passado como argumento à propriedade Altura
        }

        // Método que calcula e retorna a área do retângulo
        public double CalcularArea() {
            return Largura * Altura; // Retorna o produto da largura pela altura
        }

        // Método que calcula e retorna o perímetro do retângulo
        public double CalcularPerimetro() {
            return 2 * (Largura + Altura); // Retorna duas vezes a soma da largura e altura
        }

        // Método que calcula e retorna a diagonal do retângulo
        public double CalcularDiagonal()  {
            return Math.Sqrt(Largura * Largura + Altura * Altura); // Aplica o teorema de Pitágoras para
calcular a diagonal
        }
    }
}
```

Controle/RetangulosController

```
// Importa o namespace necessário para trabalhar com controladores MVC no ASP.NET Core
using Microsoft.AspNetCore.Mvc;
// Importa o namespace que contém a definição da classe Retangulo
using RetanguloApi.Modelo;
// Declaração do namespace para organizar o código e evitar conflitos de nomes
namespace RetanguloApi.Controle{
    [ApiController] // Indica que esta classe é um controlador de API
    [Route("/[controller]")] // Define a rota base para este controlador
    public class RetangulosController : ControllerBase {
         // A rota será "/retangulos/areas/{largura}/{altura}"
        [HttpGet("/retangulos/areas/{largura}/{altura}")]
        public IActionResult CalcularArea(double largura, double altura) { // recebe largura e altura
            Retangulo retangulo = new Retangulo(largura, altura);
            double area = retangulo.CalcularArea();
            return Ok(new { Area = area });
        }
        // Método que lida com requisições GET para calcular a diagonal do retângulo
        // A rota será "/retangulos/diagonais/{largura}/{altura}"
        [HttpGet("/retangulos/diagonais/{largura}/{altura}")]
        public IActionResult CalcularDiagonal(double largura, double altura) { // recebe largura e altura
            if (largura <= 0 || altura <= 0) {
                // Retorna um resultado HTTP 400 Bad Request se as dimensões não forem válidas
                return BadRequest(new { Message = "Largura e altura devem ser maiores que zero.", status = false
});
            }
            Retangulo retangulo = new Retangulo(largura, altura);
            double diagonal = retangulo.CalcularDiagonal();
            return Ok(new { Diagonal = diagonal, status = true });
        }
    }
}
```

Program.cs

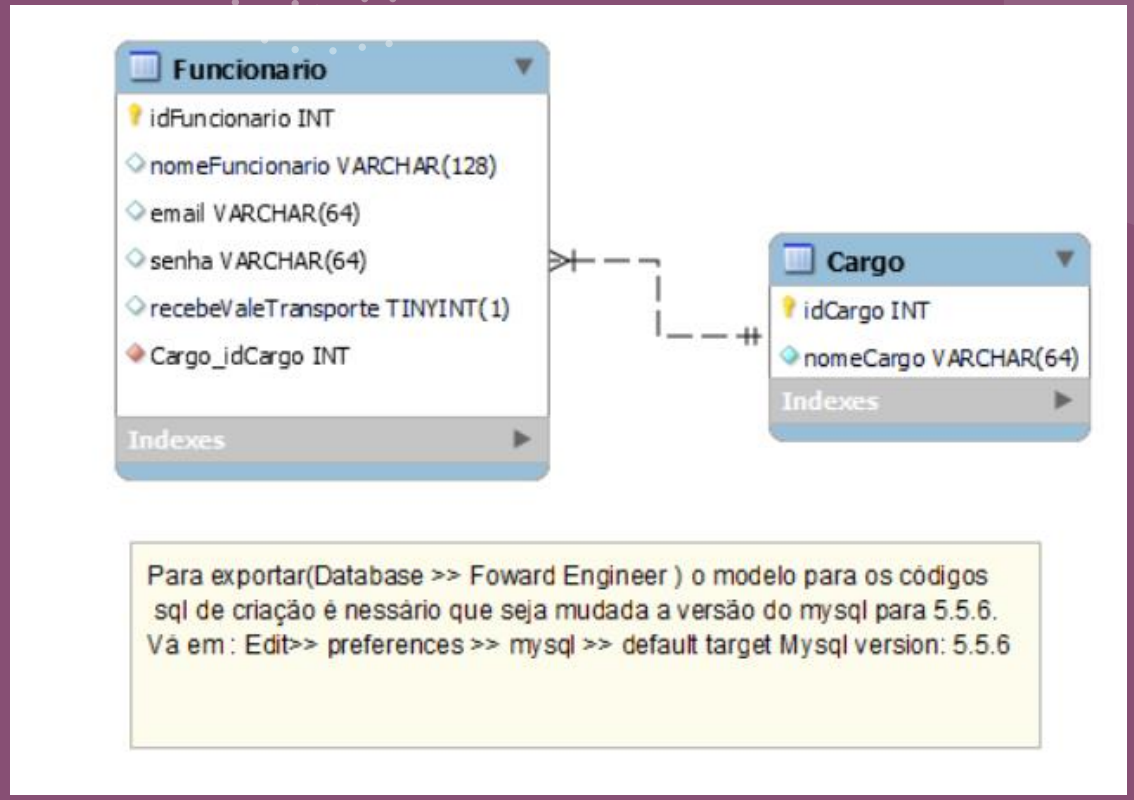
```
// Cria um novo construtor para a aplicação com os argumentos da linha de comando
WebApplicationBuilder construtor = WebApplication.CreateBuilder(args);
// Obtém a coleção de serviços onde os serviços serão registrados
IServiceCollection servicos = construtor.Services;
// Adiciona os serviços de controle ao contêiner de serviços
servicos.AddControllers();
// Adiciona a exploração de endpoints API para a documentação automática
servicos.AddEndpointsApiExplorer();
// Configura o Kestrel, o servidor web que será usado pela aplicação
construtor.WebHost.ConfigureKestrel(serverOptions => {
    // Altera a porta para 8080 e permite conexões de qualquer IP
    serverOptions.ListenAnyIP(8080);
});
// Constrói a aplicação web a partir do construtor
WebApplication app = construtor.Build();
// Obtém o ambiente web atual (desenvolvimento, produção, etc.)
IWebHostEnvironment ambiente = app.Environment;
// Redireciona requisições HTTP para HTTPS
app.UseHttpsRedirection();
app.UseStaticFiles(); // Isso faz com que arquivos em wwwroot sejam servidos
// Habilita a autorização na aplicação
app.UseAuthorization();
// Mapeia os controladores para que as rotas possam ser acessadas
app.MapControllers();
// Inicia a execução da aplicação
app.Run();
```

Runing

- Para rodar o programa:
  - Digite no terminal:
    - dotnet run

Exemplo CRUD

Modelo de banco de dados



SQL/Banco

```
DROP SCHEMA  `aula_api_2024`;
CREATE SCHEMA IF NOT EXISTS `aula_api_2024` DEFAULT CHARACTER SET utf8 ;
USE `aula_api_2024` ;
CREATE TABLE IF NOT EXISTS `aula_api_2024`.`Cargo` (
  `idCargo` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `nomeCargo` VARCHAR(64) NOT NULL,
  PRIMARY KEY (`idCargo`),
  UNIQUE INDEX `idCargo_UNIQUE` (`idCargo` ASC),
  UNIQUE INDEX `nomeCargo_UNIQUE` (`nomeCargo` ASC))
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `aula_api_2024`.`Funcionario` (
  `idFuncionario` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `nomeFuncionario` VARCHAR(128) NULL,
  `email` VARCHAR(64) NULL,
  `senha` VARCHAR(64) NULL,
  `recebeValeTransporte` TINYINT(1) NULL,
  `Cargo_idCargo` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`idFuncionario`),
  UNIQUE INDEX `idFuncionario_UNIQUE` (`idFuncionario` ASC),
  INDEX `fk_Funcionario_Cargo_idx` (`Cargo_idCargo` ASC),
  CONSTRAINT `fk_Funcionario_Cargo`
    FOREIGN KEY (`Cargo_idCargo`)
    REFERENCES `aula_api_2024`.`Cargo` (`idCargo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (1, 'Administrador');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (2, 'Técnico em Informática Jr');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (3, 'Técnico em Informática Pleno');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (4, 'Analista de Sistemas Jr');

INSERT INTO `aula_api_2024`.`funcionario` (`nomeFuncionario`, `email`, `senha`, `recebeValeTransporte`, `Cargo_idCargo`) VALUES ('adm', 'adm@adm',
md5(123456), 1, 1);
```

Novo projeto

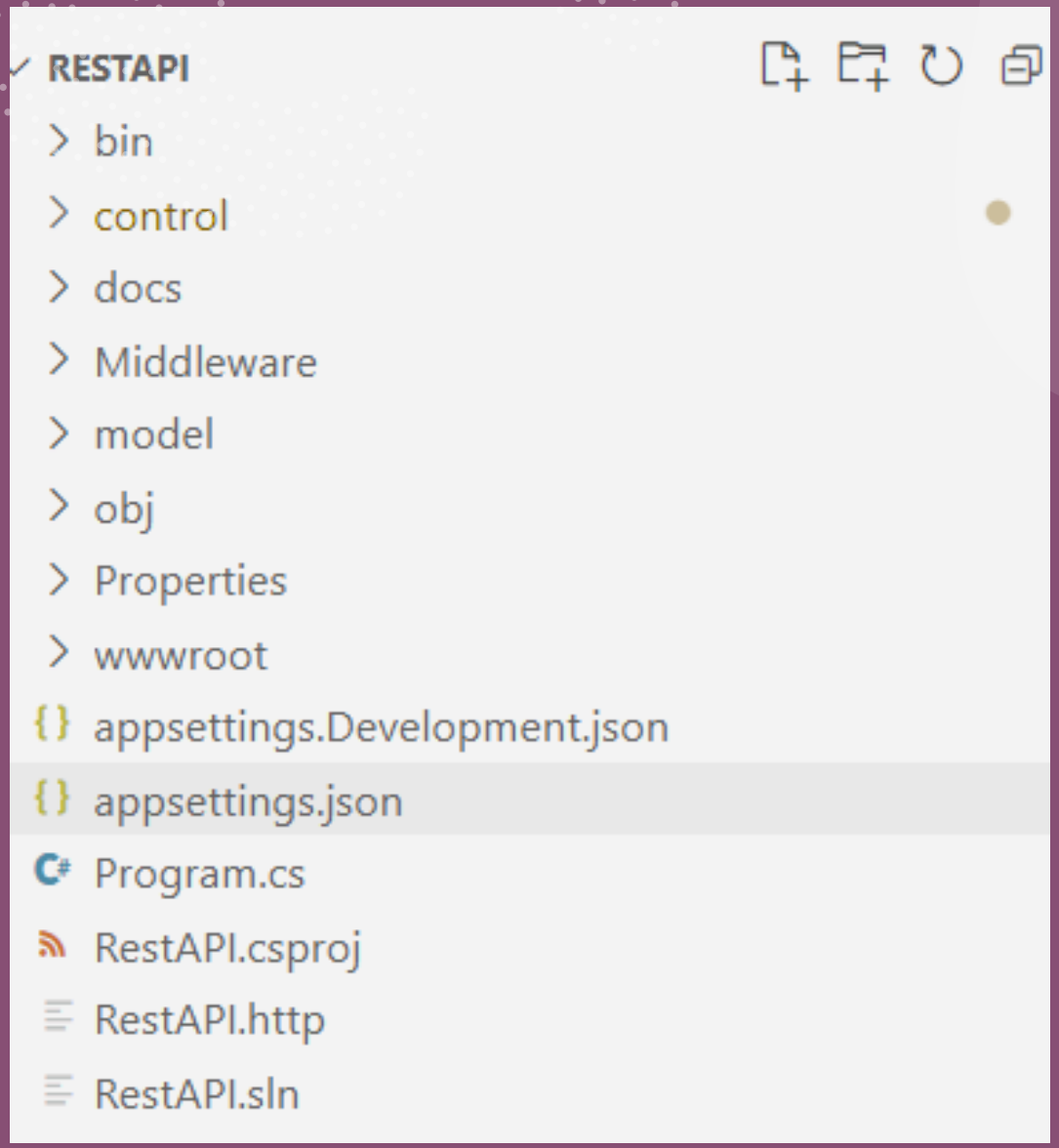
- Crie um novo projeto
- Crie uma pasta no Windows e abra no visual studio code
- Abra o terminal dentro do visual stuido code e digite:
  - dotnet new webapi -n RestAPI
- cd restapi
- Md Modelo
- Md Controle

Arquitetura

- MVC
  - Sérá utilizada a arquitetura
    - Model
    - View
    - Controler

Estrutura de diretórios:

- Aplicação:
  - control
  - docs
  - middleware
  - model
  - wwwroot
    - Esta pasta é a pasta dos arquivos estáticos, ou seja html, css, js, etc



Mysql

- Instalar o pacote de acesso ao mysql
- Adicionar o Pacote MySQL
  - dotnet add package MySql.Data

Iniciando o servidor e a aplicação

Program.cs

```
// Cria um construtor para configurar o ambiente da aplicação web
WebApplicationBuilder construtor = WebApplication.CreateBuilder();
// Obtém a coleção de serviços (Dependency Injection) da aplicação
IServiceCollection servicos = construtor.Services;
// Adiciona o suporte a controladores para a API (MVC sem visão/Views)
servicos.AddControllers();
// Adiciona o suporte para explorar e gerar automaticamente endpoints da API
servicos.AddEndpointsApiExplorer(); // Para a geração de endpoints de API
// Configura o servidor Kestrel para escutar na porta 8080
construtor.WebHost.ConfigureKestrel(serverOptions =>
{
    serverOptions.ListenAnyIP(8080); // Altera a porta para 8080 e permite conexões de qualquer IP
});
// Constrói a aplicação web com base nas configurações definidas anteriormente
WebApplication app = construtor.Build();
// Ativa o middleware para servir arquivos estáticos a partir da pasta "wwwroot"
app.UseStaticFiles(); // Isso faz com que arquivos em wwwroot sejam servidos
// Redireciona automaticamente as requisições HTTP para HTTPS
//app.UseHttpsRedirection();
// Configura o middleware de autorização para proteger rotas que exigem autenticação
app.UseAuthorization();
// Mapeia os controladores para suas respectivas rotas, tornando a API acessível
app.MapControllers(); // Mapeia os controladores
// Exibe uma mensagem no console indicando que a API está rodando na porta 8080
Console.WriteLine("API rodando em: http://localhost:8080");
// Inicia a aplicação e começa a escutar as requisições
app.Run();
```

Controle/cargo

CargoController.cs 1/5

```
using RestAPi.Model; // Importa os modelos usados pela API

// Define o namespace do controlador
namespace RestAPi.Control
{
    // Indica que esta classe é um controlador de API
    [ApiController]
    // Define a rota base para este controlador
    [Route("/[controller]")]
    public class CargoController : ControllerBase // Classe que herda de ControllerBase, usada para criar APIs RESTful
    {
```

CargoController.cs 2/5

```
// Define um endpoint HTTP GET para obter cargo por ID
        [HttpGet("/cargos/{idCargo}/")]
        // Método que lida com solicitações GET para "/cargos/{idCargo}/"
        public IActionResult Get_Cargos_by_Id(uint idCargo){
            // Cria uma nova instância do objeto Cargo
            Cargo objCargo = new Cargo();
            // Atribui o ID do cargo que foi passado pela rota
            objCargo.IdCargo = idCargo;
            // Cria uma resposta contendo o cargo retornado pelo método ReadById()
            object resposta = new
            {
                cargos = objCargo.ReadById() // Obtém os dados do cargo pelo ID
            };

            // Retorna a resposta com status 200 OK
            return Ok(resposta);
        }
```

CargoController.cs 3/5

```
    [HttpPost("/cargos/")]
    public IActionResult Post_Cargos([FromBody] Dictionary<string, Dictionary<string, object>> jsonData){
    try {
      string authorization = Request.Headers["Authorization"].ToString(); //autorização da requisição
      JwtMiddleware jwtMiddleware = new JwtMiddleware();
      CursoMiddleware CursoMiddleware = new CursoMiddleware();
      jwtMiddleware.Validar(authorization);
      CursoMiddleware.ValidarNomeCurso(jsonData);
      Dictionary<string, object> cargoData = jsonData["cargo"];
      string nomeCargo = cargoData["nomeCargo"].ToString();
      CursoMiddleware.IsNotCargo(nomeCargo);
      Cargo cargo = new Cargo();
      cargo.NomeCargo = nomeCargo;
      if (cargo.Create() == false){
         object resposta = new {
           mensagem = "Erro ao salvar cargo." // Mensagem de erro
           };
           return StatusCode(500, resposta);
       }else{
           object resposta = new{
                    mensagem = "Cargo criado com sucesso", // Mensagem de sucesso
                    cargos = cargo // Inclui o cargo criado na resposta
                };
                return Ok(resposta);
            }
        }catch (Exception e) { // Captura qualquer exceção que possa ocorrer
            // Cria uma resposta de erro com a mensagem da exceção
            object resposta = new {
                mensagem = e.Message // Mensagem de erro
            };
            // Retorna uma resposta 400 Bad Request
            return BadRequest(resposta);
        }
    }
}
```

```
{
  "cargo":{
    "nomeCargo": "Novo Cargo"
  }
}
```

CargoController.cs 4/5

```
[HttpPut("/cargos/{id}")]
  public IActionResult Put_Cargos(uint id, [FromBody] Dictionary<string, Dictionary<string, object>> jsonData){
     try {
        string authorization = Request.Headers["Authorization"].ToString();
        JwtMiddleware jwtMiddleware = new JwtMiddleware();
        CursoMiddleware cursoMiddleware = new CursoMiddleware();
        jwtMiddleware.Validar(authorization);
        cursoMiddleware.ValidarNomeCurso(jsonData);
        Dictionary<string, object> cargoData = jsonData["cargo"];
        string nomeCargo = cargoData["nomeCargo"].ToString();
        cursoMiddleware.IsNotCargo(nomeCargo);
        Cargo cargo = new Cargo();
        cargo.IdCargo = id; // Define o ID do cargo a ser atualizado
        cargo.NomeCargo = nomeCargo; // Atribui o nome do cargo
                if (cargo.Update() == false) { // Verifica se o método Update() falhou
                    object resposta = new {
                        mensagem = "Erro ao atualizar cargo." // Mensagem de erro
                    };
                    // Retorna uma resposta de erro 500 Internal Server Error
                    return StatusCode(500, resposta);
                }else {
                    // Cria uma resposta de sucesso
                    object resposta = new {
                        mensagem = "Cargo atualizado com sucesso", // Mensagem de sucesso
                        cargo = cargo // Inclui o cargo atualizado na resposta
                    };
                    // Retorna a resposta com status 200 OK
                    return Ok(resposta);
                }
            }
            catch (Exception e) { // Captura qualquer exceção que possa ocorrer

                // Cria uma resposta de erro com a mensagem da exceção
                object resposta = new {
                    mensagem = e.Message // Mensagem de erro
                };
                // Retorna uma resposta 400 Bad Request
                return BadRequest(resposta);
            }
        }
```

```
{
  "cargo": {
    "nomeCargo": "Novo Cargo"
  }
}
```

CargoController.cs 5/5

```
// Define um endpoint HTTP GET para obter todos os cargos
        [HttpGet("/cargos/")]
        public IActionResult Get_Cargos() { // Método que lida com solicitações GET para "/cargos/"
            try{
                string authorization = Request.Headers["Authorization"].ToString();// autorização da requisição
                JwtMiddleware jwtMiddleware = new JwtMiddleware();
                jwtMiddleware.Validar(authorization);
            }catch (Exception e) { // Captura qualquer exceção que possa ocorrer
                object resposta_erro = new {
                    status = false, // Indica que houve um erro
                    mensagem = e.Message // Mensagem da exceção
                };
                // Retorna uma resposta 401 Unauthorized com a mensagem de erro
                return Unauthorized(resposta_erro);
            }
            Cargo objCargo = new Cargo();
            object resposta = new {
                status = true, // Define o status como sucesso
                mensagem = "Executado com sucesso", // Mensagem de sucesso
                cargos = objCargo.Read() // Chama o método Read() que retorna a lista de cargos
            };
            // Retorna a resposta com status 200 OK
            return Ok(resposta);

        }
```

CargoController.cs 5/5

```
// Define um endpoint HTTP GET para obter todos os cargos
        [HttpGet("/cargos/")]
        public IActionResult Get_Cargos() { // Método que lida com solicitações GET para "/cargos/"
            try{
                string authorization = Request.Headers["Authorization"].ToString();// autorização da requisição
                JwtMiddleware jwtMiddleware = new JwtMiddleware();
                jwtMiddleware.Validar(authorization);
            }catch (Exception e) { // Captura qualquer exceção que possa ocorrer
                object resposta_erro = new {
                    status = false, // Indica que houve um erro
                    mensagem = e.Message // Mensagem da exceção
                };
                // Retorna uma resposta 401 Unauthorized com a mensagem de erro
                return Unauthorized(resposta_erro);
            }
            Cargo objCargo = new Cargo();
            object resposta = new {
                status = true, // Define o status como sucesso
                mensagem = "Executado com sucesso", // Mensagem de sucesso
                cargos = objCargo.Read() // Chama o método Read() que retorna a lista de cargos
            };
            // Retorna a resposta com status 200 OK
            return Ok(resposta);

        }
```

Modelo/Banco

Banco.cs 1/3

```
using MySql.Data.MySqlClient;

namespace RestAPi.Model
{
    public static class Banco
    {
        // Constantes privadas que armazenam os dados da conexão com o banco de dados
        private const string Host = "127.0.0.1";  // Endereço do servidor MySQL (neste caso, localhost)
        private const string User = "root";       // Nome de usuário para conectar ao banco de dados
        private const string Password = "";       // Senha para o usuário especificado
        private const string DatabaseName = "aula_api_2024"; // Nome do banco de dados a ser utilizado
        private const string Port = "3306";       // Porta do servidor MySQL (padrão é 3306)

        // Variável estática privada que armazena a conexão MySQL
        private static MySqlConnection? CONEXAO;
```

Banco.cs 2/3

```
 // Método privado responsável por estabelecer a conexão com o banco de dados
        private static void Connect(){
            // Monta a string de conexão usando as constantes definidas acima
            string connectionString = $"Server={Host};Database={DatabaseName};User
ID={User};Password={Password};Port={Port};";

            // Inicializa a conexão MySQL com a string de conexão
            CONEXAO = new MySqlConnection(connectionString);

            // Abre a conexão com o banco de dados
            CONEXAO.Open();
        }
```

Banco.cs 3/3

```
// Método público que retorna a conexão com o banco de dados
public static MySqlConnection GetConnection()
{
    // Verifica se a conexão não foi criada ou se não está aberta
    if (CONEXAO == null || CONEXAO.State != System.Data.ConnectionState.Open)
    {
        // Se a conexão não existir ou estiver fechada, chama o método Connect para abrir a conexão
        Banco.Connect();
    }
    // Retorna a conexão MySQL ativa
    return CONEXAO;
}
```

Modelo/Cargo

Cargo.cs
1/7

```
// Classe Cargo que representa a entidade "Cargo" no banco de dados
public class Cargo
{
    // Propriedades públicas que representam as colunas da tabela Cargo
    public uint IdCargo { get; set; }  // Armazena o ID do cargo
    public string NomeCargo { get; set; }  // Armazena o nome do cargo
```

Cargo.cs 2/7

```
// Método para criar um novo registro na tabela Cargo
        public bool Create() {
            try {
                // Cria um comando SQL para inserir um novo cargo na tabela
                MySqlCommand mysqlCommand = new MySqlCommand("INSERT INTO Cargo (nomeCargo) VALUES
(@nomeCargo)", Banco.GetConnection());

                // Adiciona o valor da propriedade NomeCargo como parâmetro para a consulta SQL
                mysqlCommand.Parameters.AddWithValue("@nomeCargo", NomeCargo);

                // Executa a consulta e retorna o número de linhas afetadas
                int itensInseridos = mysqlCommand.ExecuteNonQuery();

                // Verifica se algum item foi inserido
                if (itensInseridos > 0) {
                    // Recupera o último ID inserido no banco de dados
                    uint idCargoInserido = Convert.ToUInt32(mysqlCommand.LastInsertedId);

                    // Atribui o ID recuperado à propriedade IdCargo do objeto
                    this.IdCargo = idCargoInserido;
                    return true; // Retorna true se a inserção foi bem-sucedida
                }

            } catch (Exception ex) {
                // Captura qualquer exceção e imprime a mensagem de erro no console
                Console.WriteLine("Erro ao criar cargo: " + ex.Message);
                return false;  // Retorna false se ocorrer um erro
            }
            return false;  // Retorna false se não houve inserção
        }
```

Cargo.cs 3/7

```
// Método para ler todos os registros da tabela Cargo
        public List<Cargo> Read(){
            // Cria uma lista para armazenar os objetos Cargo
            List<Cargo> cargos = new List<Cargo>();

            try {
                // Cria um comando SQL para selecionar todos os registros da tabela Cargo
                MySqlCommand mysqlCommand = new MySqlCommand("SELECT * FROM Cargo", Banco.GetConnection());
                // Executa a consulta e obtém o resultado na forma de um MySqlDataReader
                MySqlDataReader matrizRegistros = mysqlCommand.ExecuteReader();
                // Percorre todos os registros obtidos
                while (matrizRegistros.Read()){
                    // Cria um novo objeto Cargo e popula suas propriedades
                    Cargo cargo = new Cargo();
                    cargo.IdCargo = matrizRegistros.GetUInt32("idCargo");
                    cargo.NomeCargo = matrizRegistros.GetString("nomeCargo");

                    // Adiciona o objeto Cargo à lista
                    cargos.Add(cargo);
                }
                // Fecha o leitor de dados
                matrizRegistros.Close();
            }catch (Exception ex){
                // Captura qualquer exceção e imprime a mensagem de erro no console
                Console.WriteLine("Erro ao ler cargos: " + ex.Message);
            }
            return cargos;  // Retorna a lista de cargos
        }
```

Cargo.cs 4/7

```
 // Método para ler um registro específico pelo Id
        public Cargo ReadById(){
            Cargo cargo = new Cargo();  // Cria um novo objeto Cargo
            try {
                // Cria um comando SQL para selecionar um cargo específico pelo Id
                MySqlCommand mySqlCommand = new MySqlCommand("SELECT * FROM Cargo WHERE idCargo = @idCargo",
Banco.GetConnection());

                // Adiciona o IdCargo como parâmetro para a consulta
                mySqlCommand.Parameters.AddWithValue("@idCargo", this.IdCargo);

                // Executa a consulta e obtém o resultado na forma de um MySqlDataReader
                MySqlDataReader matrizRegistros = mySqlCommand.ExecuteReader();

                // Verifica se algum registro foi encontrado
                if (matrizRegistros.Read()){
                    // Popula o objeto Cargo com os dados do registro
                    cargo = new Cargo();
                    cargo.IdCargo = matrizRegistros.GetUInt32("idCargo");
                    cargo.NomeCargo = matrizRegistros.GetString("nomeCargo");
                }
                // Fecha o leitor de dados
                matrizRegistros.Close();
            }catch (Exception ex){
                // Captura qualquer exceção e imprime a mensagem de erro no console
                Console.WriteLine("Erro ao ler cargo por ID: " + ex.Message);
            }
            return cargo;  // Retorna o objeto Cargo
        }
```

Cargo.cs 5/7

```
  // Método para atualizar um registro existente
        public bool Update(){
            try{
                // Cria um comando SQL para atualizar um registro na tabela Cargo
                MySqlCommand mySqlCommand = new MySqlCommand("UPDATE Cargo SET nomeCargo = @nomeCargo WHERE idCargo
= @idCargo", Banco.GetConnection());

                // Adiciona os parâmetros IdCargo e NomeCargo à consulta
                mySqlCommand.Parameters.AddWithValue("@idCargo", this.IdCargo);
                mySqlCommand.Parameters.AddWithValue("@nomeCargo", this.NomeCargo);

                // Executa a consulta e retorna o número de linhas afetadas
                int qtdCargosAtualizados = mySqlCommand.ExecuteNonQuery();

                // Retorna true se pelo menos um registro foi atualizado
                return qtdCargosAtualizados > 0;
            }catch (Exception ex){
                // Captura qualquer exceção e imprime a mensagem de erro no console
                Console.WriteLine("Erro ao atualizar cargo: " + ex.Message);
                return false;  // Retorna false se ocorrer um erro
            }
        }
```

Cargo.cs 6/7

```
// Método para excluir um registro existente
        public bool Delete(){
            try{
                // Cria um comando SQL para excluir um cargo pelo Id
                MySqlCommand command = new MySqlCommand("DELETE FROM Cargo WHERE idCargo = @idCargo",
Banco.GetConnection());

                // Adiciona o IdCargo como parâmetro para a consulta
                command.Parameters.AddWithValue("@idCargo", this.IdCargo);

                // Executa a consulta e retorna o número de linhas afetadas
                int qtdCargosExcluidos = command.ExecuteNonQuery();

                // Retorna true se pelo menos um registro foi excluído
                return qtdCargosExcluidos > 0;
            }catch (Exception ex){
                // Captura qualquer exceção e imprime a mensagem de erro no console
                Console.WriteLine("Erro ao excluir cargo: " + ex.Message);
                return false;  // Retorna false se ocorrer um erro
            }
        }
```

Cargo.cs 7/7

```
// Método para verificar se um cargo com o mesmo nome já existe no banco de dados
        public bool IsCargoByNomeCargo(string nomeCargo){
            bool existe = false;  // Variável que indica se o cargo já existe
            try{
                // Cria um comando SQL para contar quantos cargos existem com o nome fornecido
                MySqlCommand mySqlCommand = new MySqlCommand("SELECT COUNT(*) as qtd FROM Cargo WHERE nomeCargo =
@nomeCargo", Banco.GetConnection());
                // Adiciona o nome do cargo como parâmetro para a consulta
                mySqlCommand.Parameters.AddWithValue("@nomeCargo", nomeCargo);
                // Executa a consulta e obtém o resultado (campo qtd do SELECT)
                object resultado = mySqlCommand.ExecuteScalar();  // ExecuteScalar retorna o primeiro valor da
primeira linha do resultado
                // Verifica se o resultado não é nulo e o converte para inteiro
                if (resultado != null){
                    int qtd = Convert.ToInt32(resultado);  // Converte o valor retornado para inteiro

                    // Se o número de registros for maior que 0, o cargo já existe
                    existe = qtd > 0;
                }
            }catch (Exception ex){
                // Captura qualquer exceção e imprime a mensagem de erro no console
                Console.WriteLine("Erro ao verificar se o cargo já existe: " + ex.Message);
            }
            return existe;  // Retorna true se o cargo já existe, false caso contrário
        }

    }
}
```

Runing

- Para rodar o programa:
  - Digite no terminal:
    - dotnet run