ATIVIDADE INTEGRADORA

Programação WEB III, Banco de Dados III e Desenvolvimento do TCC

1. Criar um Repositório com o nome **LojinhaMongoDB** no seu **GitHub.**

2. Clonar o Repositório **LojinhaMongoDB** na **Área de Trabalho**.

3. Abrir a pasta do Repositório **LojinhaMongoDB** no **Visual Studio Code**.

4. Abrir o terminal, conferir se está na pasta do Repositório e executar os comandos abaixo:

```
dotnet new sln --name Lojinha
dotnet new webapi -o LojinhaServer
dotnet sln add LojinhaServer\LojinhaServer.csproj
```

5. Criar dentro da pasta do projeto **LojinhaServer** as pastas:

- Collections
- Extensions
- Models
- Repositories
- Services

6. No terminal use o comando abaixo para acessar a pasta **LojinhaServer:**

```
cd LojinhaServer
```

7. Executar o comando abaixo no terminal, para instalação do pacote do driver de conexão do **MongoDB**:

```
dotnet add package MongoDB.Driver --version 2.19.0
```

8. Abra o arquivo **appSettings.json,** e inclua as alterações abaixo:

```
{
   "MongoDBSettings":{
       "ConnectionString": "copiar sua ConnectionString",
       "DatabaseName": "nome do database"
   },
   "Logging": {
       "LogLevel": {
       "Default": "Information",
       "Microsoft.AspNetCore": "Warning"
       }
   },
   "AllowedHosts": "*"
}
```

9. Abra o arquivo **LojinhaServer.csproj** e altere a linha 5, conforme código abaixo:

```
<Nullable>disable</Nullable>
```

10. Crie uma classe com o nome **MongoDBSettings** na pasta **Models** e faça as alterações abaixo:

```
namespace LojinhaServer.Models;

public class MongoDBSettings
{
   public string ConnectionString { get; set; }
   public string DatabaseName { get; set; }
}
```

11. Crie uma classe com o nome **ServiceExtensions** na pasta **Extensions** e faça as alterações abaixo:

```
using LojinhaServer.Models;
using MongoDB.Driver;

namespace LojinhaServer.Extensions;

public static class ServiceExtensions
{
   public static void ConfigureCors(this IServiceCollection services)
   {
       services.AddCors(options =>
       {
           options.AddPolicy("CorsPolicy",
               builder => builder.AllowAnyOrigin()
               .AllowAnyMethod()
               .AllowAnyHeader());
       });
   }

   public static void ConfigureMongoDBSettings(this IServiceCollection services,
IConfiguration config)
   {
       services.Configure<MongoDBSettings>(
           config.GetSection("MongoDBSettings")
       );

       services.AddSingleton<IMongoDatabase>(options => {
           var settings =
config.GetSection("MongoDBSettings").Get<MongoDBSettings>();
           var client = new MongoClient(settings.ConnectionString);
           return client.GetDatabase(settings.DatabaseName);
       });
   }
}
```

12. Adicionar o código abaixo no **Program.cs** na linha 4:

```
builder.Services.ConfigureMongoDBSettings(builder.Configuration);
```

OBS: É necessário adicionar um **using** no **Program** para não ter erros no código acima:

```
using LojinhaServer.Extensions;
```

13. Crie a classe **Product** na pasta **Collections** e faça as alterações abaixo:

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.Json.Serialization;

namespace LojinhaServer.Collections;

[Table("products")]
[BsonIgnoreExtraElements]
public class Product
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string Id { get; set; }

    [BsonElement("name")]
    [JsonPropertyName("Nome")]
    public string Name { get; set; }

    [BsonElement("description")]
    [JsonPropertyName("Descrição")]
    public string Description { get; set; }

    [BsonElement("price")]
    public decimal Price { get; set; }

    [BsonElement("offPrice")]
    public decimal OffPrice { get; set; }

    [BsonElement("categories")]
    public List<string> Categories { get; set; }

    [BsonElement("tags")]
    public List<string> Tags { get; set; }

    [BsonElement("brand")]
    public string Brand { get; set; }
}
```

14. Crie na pasta **Repositories** a interface **IProductRepository** e faça as alterações abaixo:

```
using LojinhaServer.Collections;
namespace LojinhaServer.Repositories;

public interface IProductRepository
{
   Task<List<Product>> GetAllAsync();
   Task<Product> GetByIdAsync(string id);
   Task CreateAsync(Product product);
   Task UpdateAsync(Product product);
   Task DeleteAsync(string id);
}
```

15. Crie na pasta **Repositories** a classe **ProductRepository** e faça as alterações abaixo:

```
using LojinhaServer.Collections;
using MongoDB.Driver;

namespace LojinhaServer.Repositories;

public class ProductRepository : IProductRepository
{
   private readonly IMongoCollection<Product> _collection;

   public ProductRepository(IMongoDatabase mongoDatabase)
   {
       _collection = mongoDatabase.GetCollection<Product>("products");
   }

   public async Task<List<Product>> GetAllAsync() =>
       await _collection.Find(_ => true).ToListAsync();

   public async Task<Product> GetByIdAsync(string id) =>
       await _collection.Find(_ => _.Id == id).FirstOrDefaultAsync();

   public async Task CreateAsync(Product product) =>
       await _collection.InsertOneAsync(product);

   public async Task UpdateAsync(Product product) =>
       await _collection.ReplaceOneAsync(x => x.Id == product.Id, product);

   public async Task DeleteAsync(string id) =>
       await _collection.DeleteOneAsync(x => x.Id == id);
}
```

16. Agora adicione na classe **ServiceExtensions** o método abaixo logo antes do fim da classe:

```
public static void ConfigureProductRepository(this IServiceCollection services)
{
    services.AddSingleton<IProductRepository, ProductRepository>();
}
```

OBS: É necessário adicionar um using para não ter erros no código acima:

```
using LojinhaServer.Repositories;
```

17. Agora adicione no **Program.cs**, abaixo da linha 6 a chamada ao método de configuração criado acima:

```
// Add services to the container.
builder.Services.ConfigureMongoDBSettings(builder.Configuration);
builder.Services.ConfigureProductRepository();
```

18. Agora exclua o arquivo **WeatherForecast.cs** e o **controller WeatherForecastController.cs**

19. Agora clique com o botão direito do mouse sobre a pasta **Controllers** e selecione a opção "**Api Controller**" e digite o nome **ProductsController**

20. Altere o código do **ProductsController** conforme demonstrado abaixo:

```
using LojinhaServer.Collections;
using LojinhaServer.Repositories;
using Microsoft.AspNetCore.Mvc;

namespace LojinhaServer.Controllers;

[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
    private readonly IProductRepository _repo;
    public ProductsController(IProductRepository repo)
    {
        _repo = repo;
    }

    [HttpGet]
    public async Task<IActionResult> Get()
    {
        var product = await _repo.GetAllAsync();
        return Ok(product);
    }

    [HttpGet]
    [Route("{id}")]
    public async Task<IActionResult> Get(string id)
    {
        var product = await _repo.GetByIdAsync(id);
        if (product == null)
        {
            return NotFound();
        }
            return Ok(product);
    }
```

```
    [HttpPost]
    public async Task<IActionResult> Post(Product product)
    {
        await _repo.CreateAsync(product);
        return CreatedAtAction(nameof(Get), new { id = product.Id }, product);
    }

    [HttpPut]
    public async Task<IActionResult> Put(Product product)
    {
        var oldProduct = await _repo.GetByIdAsync(product.Id);
        if (oldProduct == null)
        {
            return NotFound();
        }

        await _repo.UpdateAsync(product);
        return NoContent();
    }

    [HttpDelete]
    public async Task<IActionResult> Delete(string id)
    {
        var product = await _repo.GetByIdAsync(id);
        if (product == null)
        {
            return NotFound();
        }

        await _repo.DeleteAsync(id);
        return NoContent();
    }
}
```