Bootcamp IGTI: Banco WIZ – Speed Wiz Dev

Desafio

Módulo 3	ASP.NET Core

Objetivos

Exercitar os seguintes conceitos trabalhados no módulo:

- ✔ Criar um projeto no Visual Studio Community 2019.
- ✔ Criar controllers, rotas e actions.
- ✔ Criar um contexto de banco de dados em memória com Entity Framework Core (EFC).
- ✔ Aplicar conceitos de Autenticação e Autorização com JWT Tokens.
- ✔ Documentar a API utilizando swagger.
- ✔ Criar uma arquitetura simples e bem definida.

Enunciado

O trabalho prático consiste em construir uma aplicação ASP.NET Core Web API em C# que simule uma loja de carros (**CarroShop**). O objetivo é criar uma aplicação que forneça um serviço de cadastro de carros em um banco de dados em memória. A aplicação também contará com um cadastro de usuários, sendo que estes poderão ser associados a 2 tipos de papéis (administrador e usuário). Os usuários do tipo administrador poderão cadastrar, atualizar e deletar carros, já os usuários comuns somente poderão visualizá-los. Todo este processo será realizado através de autenticação e autorização dos serviços através da utilização de Tokens JWT. As operações envolvidas no cadastro

de usuário, carros e autenticação são baseadas nos verbos HTTP representando as operações CRUD. Documente a API para visualizar o funcionamento e realizar testes.

Atividades

Os alunos deverão desempenhar as seguintes atividades:

1. Crie um projeto ASP.NET Core Web API no Visual Studio Community 2019.
2. Crie 2 classes representando Usuario e Carro.
   a. Para a classe Carro, crie propriedades como Id, Marca, Modelo e Ano.
   b. Para a classe Usuario, crie propriedades como Id, Email, Senha, Role:

i.Role representa o papel do usuário para Administrador e Usuário comum, um exemplo seria Role = "***admin"*** ou Role = ***"usuario***".

3. Adicione um contexto (**CarrosBdContexto**) que represente o banco de dados.
   a. Utilize o Entity Framework Core:

i.Adicione o pacote **Microsoft.EntityFrameworkCore**.

   b. Utilize o banco de dados em memória.

i.Adicione o pacote **Microsoft.EntityFrameworkCore.InMemory**.

   c. Caso ache necessário, crie um gerador de dados para inicializar a aplicação com dados em memória.
4. Para os processos de autenticação e autorização com JWT, utilize o pacote **Microsoft.AspNetCore.Authentication.JwtBearer**.
5. Utilize o arquivo appSettings.json para armazenar qualquer informação referente às configurações do JWT.
6. Adicione uma controller para gerenciar usuários, utilize os verbos GET e POST, para representar 2 operações de leitura e escrita (utilize o Entity Framework Core (EFC) para realizar estas operações).
7. Adicione uma controller para gerenciar carros, utilize os verbos, GET, POST, PUT e DELETE (utilize o Entity Framework Core (EFC) para realizar estas operações).
8. Adicione uma controller para realizar a autenticação do usuário e emitir o token JWT, guarde o token e o utilize nas chamadas subjacentes para os endpoints que requerem autorização. Para essa controller, crie uma única action, utilize POST como

verbo, e envie as credenciais do usuário para obter o token, essa action é pública e pode ser acessada por qualquer usuário.

a. Lembre-se de que o token é passado no header da requisição através da chave **Authorization,** com o valor **Bearer + JWT Token.**

9. Para os resultados (Action Results), retorne os códigos HTTP de acordo com cada tipo de ação/verbo HTTP, como 200, 404, 201 etc. Trabalharemos com o formato JSON tanto para a requisição (request) quanto para a resposta (response).

10. Decore as actions (métodos da controller) com o atributo [**Authorize**] para aplicar o processo de autorização.

a. Aplique os papéis (Roles) do usuário junto ao atributo [**Authorize**].

b. Para administradores, você pode utilizar algo dessa forma **[Authorize(Roles = RolesUsuario.Admin)]**.

c. Para usuários comuns, você pode utilizar algo dessa forma **[Authorize(Roles = RolesUsuario.Admin)]**.

11. Documente a API utilizando Swagger e o utilize para testá-la.

a. Instale o swagger com o pacote **Swashbuckle.AspNetCore**, utilize o **Configure** para o adicionar ao container e **ConfigureServices** ao middleware.

Respostas Finais

Os alunos deverão desenvolver a prática e, depois, responder as seguintes questões objetivas: