### **Exercício Prático: Consumindo uma API com Autenticação utilizando C#**

#### **Objetivo**

Neste exercício, você será desafiado a criar uma aplicação console em **C#** para interagir com uma API em Python (que será fornecida). O foco do exercício é aprender a realizar requisições HTTP, autenticar-se na API, e manipular dados em formato JSON.

### **Descrição do Cenário**

A API em Python oferece três endpoints principais:

1. **/signup** - Endpoint para criar um novo usuário.
2. **/token** - Endpoint para autenticar o usuário e receber um token JWT.
3. **/protected** - Endpoint protegido que retorna uma lista de usuários (somente acessível com um token válido).

A estrutura dos dados dos usuários retornados pelo endpoint /protected segue o seguinte modelo:

{

"id": 1,

"username": "johndoe",

"email": "johndoe@example.com",

"profile": {

"first\_name": "John",

"last\_name": "Doe",

"dob": "1990-01-01",

"avatar": "https://example.com/avatar/johndoe.jpg"

},

"address": {

"street": "100 Main St",

"city": "Metropolis",

"state": "NY",

"zip": "10001"

}

}

### **Tarefas**

1. **Cadastrar Usuário na API**
   1. Crie uma funcionalidade no seu programa para realizar uma requisição POST ao endpoint /signup.
   2. O usuário deverá informar:
      1. **Username**
      2. **Senha**
      3. **Email**
2. **Autenticar o Usuário**
   1. Implemente uma funcionalidade para enviar uma requisição POST ao endpoint /login utilizando as credenciais cadastradas.
   2. O programa deve receber e armazenar o token JWT retornado pela API.
3. **Acessar Dados Protegidos**
   1. Crie uma funcionalidade para realizar uma requisição GET ao endpoint /protected, enviando o token JWT no cabeçalho da requisição.
   2. O programa deve exibir os dados dos usuários no console.

### **Requisitos Técnicos**

1. Utilize a biblioteca **HttpClient** para realizar as requisições HTTP.
2. Os dados deverão ser manipulados utilizando a biblioteca **Newtonsoft.Json**.
3. Organize o código em classes para melhorar a clareza e separação de responsabilidades:
   1. **ApiClient**: Responsável pelas interações com a API.
   2. **Models**: Representa as estruturas de dados, como Usuario, Profile e Address.
4. Adicione tratamentos de erros para:
   1. Falha na conexão com a API.
   2. Respostas inválidas da API.
   3. Token de autenticação ausente ou expirado.

### 

### 

### 

### 

### 

### **Instruções**

1. **Configuração Inicial**
   1. Acesse a API fornecida pelo instrutor (executando localmente em http://127.0.0.1:8000).
   2. Certifique-se de que a API está funcionando corretamente antes de iniciar o desenvolvimento.
2. **Estrutura do Projeto**
   1. Crie um projeto Console App no Visual Studio ou qualquer editor de sua preferência.

Adicione o pacote **Newtonsoft.Json** ao projeto:  
dotnet add package Newtonsoft.Json

1. **Código-Base** Utilize o código-base abaixo como ponto de partida. Complete as funcionalidades de acordo com as instruções acima.

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

namespace ApiComAutenticacao

{

public class Usuario { /\* Definição do modelo \*/ }

public class Profile { /\* Definição do modelo \*/ }

public class Address { /\* Definição do modelo \*/ }

public class ApiClient

{

private const string BaseUrl = "http://127.0.0.1:8000";

public async Task<string> SignUp(string username, string password, string email)

{

// Implementação da requisição POST para /signup

}

public async Task<string> Authenticate(string username, string password)

{

// Implementação da requisição POST para /token

}

public async Task<List<Usuario>> FetchProtectedData(string token)

{

// Implementação da requisição GET para /protected

}

}

class Program

{

static async Task Main(string[] args)

{

ApiClient apiClient = new ApiClient();

// Implementação da lógica do programa

}

}

}

### **Critérios de Avaliação**

1. **Funcionalidade Completa**:
   1. O programa deve ser capaz de realizar o fluxo completo: cadastro → autenticação → acesso a dados protegidos.
2. **Clareza do Código**:
   1. O código deve ser organizado, comentado, e com boa separação de responsabilidades.
3. **Tratamento de Erros**:
   1. O programa deve tratar possíveis erros, como credenciais inválidas e falta de conexão com a API.
4. **Output Legível**:
   1. Os dados retornados pelo endpoint /protected devem ser exibidos no console de forma clara e organizada.

### **Dicas**

1. Teste cada funcionalidade de forma isolada antes de integrar no fluxo completo.
2. Use o **Postman** ou similar para testar os endpoints da API manualmente antes de implementar no C#.
3. Se tiver dúvidas, consulte a documentação oficial de **HttpClient** e **Newtonsoft.Json**.

Boa sorte! 🎯