Guia de Desenvolvimento da Aplicação

1. Pré-requisitos

1.1 Ferramentas Necessárias

Para rodar e desenvolver a aplicação localmente, o desenvolvedor precisará das seguintes ferramentas instaladas:

- **.NET 6.0 ou superior**: Para rodar a API back-end.
- **Visual Studio 2022 ou superior**: Para desenvolver tanto o back-end quanto o front-end. É importante ter o workload de desenvolvimento ASP.NET e web instalado.
- **SQL Server 2016 ou superior**: Para persistência de dados. Pode ser a versão Express para ambiente de desenvolvimento.
- **SQL Server Management Studio (SSMS)**: Para gerenciar o banco de dados localmente.
- **Node.js e NPM**: Para o desenvolvimento do front-end com ASP.NET MVC e Razor.
- **Postman** ou **Insomnia**: Para testar as APIs.

2. Configuração Inicial do Projeto

2.1 Configuração do Front-End

O front-end é desenvolvido com ASP.NET MVC e Razor. A solução pode ser aberta com o arquivo:

`Front\ThomaGregFront\ThomaGregFront.sln`

Configuração Inicial:

1. Abra o arquivo `ThomaGregFront.sln` no Visual Studio.
2. Certifique-se de que o **Node.js** está instalado corretamente. O projeto pode ter pacotes NPM que precisarão ser restaurados.
3. Abra o arquivo `appsettings.json` e configure as informações necessárias:

```
{
    "Bearer": {
        "Usuario": "administrador",
        "Senha": "senha@123"
    },
    "UrlApiCrud": "http://localhost:5183/"
}
```

Isso configura o **usuário e senha** para autenticação no back-end e o endereço da API que o front-end consumirá.

2.2 Configuração do Back-End

O back-end é desenvolvido como uma API REST utilizando **.NET Core 6.0+**. A solução pode ser aberta com o arquivo:

`API\POC.ThomasGreg.CadastroCliente.sln`

Configuração Inicial:

1. Abra o arquivo `POC.ThomasGreg.CadastroCliente.sln` no Visual Studio.
2. Verifique o arquivo `appsettings.json` e configure a **connection string** para o banco de dados:

```
{
    "ConnectionStrings": {
        "DefaultConnection": "Server=DESKTOP-
8N14EVU\\SQLEXPRESS; Database=ThomasGreg;
Trusted_Connection=True; TrustServerCertificate=True;"
    },
    "JwtSettings": {
        "Secret":
"sua_chave_secreta_com_256_bits_de_tamanho_aqui_32_caracteres!"
,
        "Issuer": "ThomasGregAPI",
        "Audience": "SeusClientes",
        "ExpiresInMinutes": 60
    },
    "AllowedHosts": "*",
    "Serilog": {
        "Using": [ "Serilog.Sinks.Console" ],
```

```
            "MinimumLevel": {
                "Default": "Debug",
                "Override": {
                    "Microsoft": "Warning",
                    "System": "Warning"
                }
            },
            "WriteTo": [
                { "Name": "Console" },
                {
                    "Name": "File",
                    "Args": {
                        "path": "C:\\Desenvolvimento\\Log\\api-
    thomasgreg.txt",
                        "rollingInterval": "Day",
                        "outputTemplate": "{Timestamp:dd-MM-yyyy
    HH:mm:ss} | [{Level:u3}] | {ThreadId} |
    {Message:lj}{NewLine}{Exception}"
                    }
                }
            ]
        }
    }
```

3. Como Rodar a Aplicação Localmente

3.1 Banco de Dados

1. **Criar o Banco de Dados**: Antes de rodar a aplicação, é necessário criar o banco de dados. Para isso, siga os passos:
    a. Abra o **SQL Server Management Studio (SSMS)** e conecte-se ao seu servidor SQL.
    b. Crie um novo banco de dados com o nome `ThomasGreg` (ou o nome definido na `ConnectionStrings`).
2. **Adicionar e Aplicar Migrations**:
    a. O código que adiciona e aplica automaticamente as migrations já está implementado na aplicação através do método `AdicionarMigration`:

```
public static IServiceProvider AdicionarMigration(this
IServiceProvider services)
```

```
{
    using (var scope = services.CreateScope())
    {
        var dbContext =
scope.ServiceProvider.GetRequiredService<CadastroDbContext>();
        dbContext.Database.Migrate(); // Cria o banco de dados e
aplica as migrations
    }
    return services;
}
```

b. Ou seja, não é necessário adicionar manualmente o código para aplicar as migrations. O que o desenvolvedor deve fazer é **gerar as migrations** sempre que houver alterações no modelo de dados, utilizando o comando do .NET:

```
dotnet ef migrations add NomeDaMigration
```

3. **Configuração da Connection String**: A connection string do banco de dados é configurada no `appsettings.json`. Verifique se o caminho e as credenciais estão corretos para o ambiente local.

3.2 Rodar a Aplicação

- **Rodar o Front-End**: No Visual Studio, pressione **F5** para rodar o projeto do front-end (`ThomaGregFront.sln`). Isso vai iniciar o servidor local para o front-end.

- **Rodar o Back-End**: Da mesma forma, pressione **F5** no projeto `POC.ThomasGreg.CadastroCliente.sln` para rodar a API.

Agora, o front-end estará disponível na URL configurada e a API estará rodando no local definido.

4. Desenvolvendo uma Nova Feature

4.1 Desenvolvendo no Back-End

Passos para Desenvolver uma Nova Feature no Back-End:

1. **Criação de Command e Query**:
    a. Crie uma nova classe de comando ou consulta em `Application/Commands` ou `Application/Queries`, dependendo da operação.
    b. Para **commands** (escrita), defina a classe que implementa `IRequest`.
    c. Para **queries** (leitura), defina a classe que implementa `IRequest<TResult>`.
2. **Criação de Handler**:
    a. Crie um handler para o comando ou consulta em `Application/Handlers`. O handler vai ser responsável por processar o comando ou consulta e interagir com a camada de domínio.
3. **Implementação de Lógica de Negócio**:
    a. No `Domain`, adicione a lógica de negócio nas entidades ou Value Objects conforme necessário.
4. **Migrations**:
    a. Se houver alterações no banco de dados, **gere uma nova migration** com o comando:

```
dotnet ef migrations add NomeDaMigration
```

    b. **Note**: O código que aplica as migrations já está implementado na aplicação, portanto, não é necessário adicionar manualmente a lógica para aplicar migrations no banco.
5. **Testes**:
    a. Como não há testes automatizados na aplicação, a validação pode ser feita diretamente nos endpoints utilizando o **Postman** ou **Insomnia**.
6. **Autenticação**:
    a. Lembre-se de que a API usa autenticação **JWT**. Para testar, envie o **Bearer Token** (obtido ao fazer login com o usuário e senha configurados no front-end) com as requisições.

4.2 Desenvolvendo no Front-End

Passos para Desenvolver uma Nova Feature no Front-End:

1. **Criar ou Modificar Controllers**:
    a. Se for necessário adicionar um novo endpoint na API, crie ou modifique o controller existente dentro da pasta `Controllers`.
2. **Desenvolver Views**:
    a. As views são desenvolvidas com Razor. Se for criar uma nova página, adicione uma nova view em `Views` com o Razor syntax.
3. **Consumir API**:
    a. O front-end usa a URL da API configurada em `appsettings.json`. Utilize o **HttpClient** para consumir os endpoints da API.
4. **Testes no Front-End**:
    a. Utilize o navegador ou **Postman** para testar a comunicação entre o front-end e o back-end.

5. Considerações Finais

- **Desacoplamento**: A arquitetura segue o padrão **Clean Architecture** e utiliza **MediatR** para separar as responsabilidades.
- **Segurança**: A autenticação JWT garante que os dados sensíveis estejam protegidos.
- **Escalabilidade**: A utilização de **CQRS** e **AutoMapper** permite maior escalabilidade e facilita a manutenção do código.