Documentação Técnica - BancoDigitalAPI

1. Objetivo

Esta API foi desenvolvida como parte de um desafio técnico, com o objetivo de simular funcionalidades básicas de um banco digital. A proposta envolve a criação de uma aplicação em .NET 8 com C#, utilizando GraphQL e testes unitários, sem a necessidade de autenticação, e com persistência em banco MySQL.

2. Tecnologias Utilizadas

- C# com .NET 8 - ASP.NET Core - MySQL 8 - GraphQL (HotChocolate) - Entity Framework Core - Swagger (Swashbuckle) - xUnit (testes unitários) - Banco InMemory e MySQL para testes

3. Estrutura do Projeto

- `Models/ContaCorrente.cs`: representa a entidade da conta com número e saldo. - `Data/AppDbContext.cs`: contexto do EF Core para acesso ao banco de dados. - `Services/ContaService.cs`: lógica de negócio da API. - `GraphQL/Queries/ContaQuery.cs`: consulta de saldo via GraphQL. - `GraphQL/Mutations/ContaMutation.cs`: mutations de saque, depósito e criação de conta. - `Program.cs`: configuração de serviços, endpoints, GraphQL e Swagger. - `Tests/ContaServiceTests.cs`: testes unitários com xUnit.

4. Endpoints REST

- `POST /api/conta`: cria nova conta - `GET /api/saldo/{conta}`: consulta saldo da conta

5. Mutations GraphQL

- `mutation { sacar(conta: 54321, valor: 100) { conta saldo } }` - `mutation { depositar(conta: 54321, valor: 200) { conta saldo } }` - `mutation { criarConta(conta: 12345, saldoInicial: 500) { conta saldo } }`

6. Query GraphQL

- `query { saldo(conta: 54321) }`

7. Testes Unitários

Os testes unitários cobrem: - Consulta de saldo - Saque com e sem saldo - Depósito - Criação de conta nova e conta existente Banco de dados usado nos testes: InMemory e MySQL (BancoDigitalTestDB). Cobertura de testes: acima de 85%.

8. Observações Finais

A API está pronta para ser executada em ambiente local com Swagger e GraphQL Playground. Para rodar os testes, use o comando `dotnet test`. As migrations devem ser aplicadas previamente

ao banco MySQL de teste.