BancoDigitalAPI - Documentação

1 - Objetivo

O projeto **BancoDigitalAPI** simula funcionalidades básicas de um banco digital, permitindo: - Sacar valores de uma conta corrente - Depositar valores em uma conta - Consultar saldo de uma conta

A API foi desenvolvida utilizando **.NET 8**, **GraphQL** e **MySQL**, e possui também endpoints REST para testes com Swagger.

2 - Estrutura do Projeto

```
BancoDigitalAPI/
│
├── Data/
│   └── AppDbContext.cs          # Contexto do EF Core para acesso ao
banco MySQL
│
├── Models/
│   └── ContaCorrente.cs         # Modelo da conta corrente
│
├── Services/
│   └── ContaService.cs          # Lógica de negócio (sacar, depositar,
saldo)
│
├── GraphQL/
│   ├── Mutations/ContaMutation.cs   # Mutations GraphQL (sacar e
depositar)
│   └── Queries/ContaQuery.cs        # Query GraphQL (saldo)
│
├── Program.cs                   # Configuração do projeto, endpoints
REST, GraphQL e Swagger
└── appsettings.json             # Configurações do banco de dados
```

3 - Banco de Dados

O projeto utiliza **MySQL**. A tabela `Contas` é definida pelo **DbContext**:

```
public DbSet<ContaCorrente> Contas => Set<ContaCorrente>();
```

- **Conta** (int): número da conta
- **Saldo** (decimal): saldo atual da conta

Connection string no `appsettings.json`:

```
{
  "ConnectionStrings": {
    "DefaultConnection":
```

```
"Server=localhost;Database=FuncionalDB;Uid=root;Pwd=admin;"
  }
}
```

4 - Modelo: ContaCorrente

```
public class ContaCorrente
{
    public int Conta { get; set; }
    public decimal Saldo { get; set; }
}
```

- Representa a conta do cliente.

5 - Serviço: ContaService

Responsável pela **lógica de negócio**:

```
public async Task<ContaCorrente> SacarAsync(int conta, decimal valor)
public async Task<ContaCorrente> DepositarAsync(int conta, decimal valor)
public async Task<decimal> ObterSaldoAsync(int conta)
```

- **SacarAsync:** subtrai o valor do saldo se houver saldo suficiente, caso contrário lança `GraphQLException`.
- **DepositarAsync:** adiciona o valor ao saldo da conta.
- **ObterSaldoAsync:** retorna o saldo da conta.

6 GraphQL

- **Mutation: ContaMutation.cs**

```
public class ContaMutation
{
    public async Task<ContaCorrente> Sacar([Service] ContaService
service, int conta, decimal valor)
    public async Task<ContaCorrente> Depositar([Service] ContaService
service, int conta, decimal valor)
}
```

- **Query: ContaQuery.cs**

```
public class ContaQuery
{
    public async Task<decimal> Saldo([Service] ContaService service, int
conta)
}
```

Exemplos de GraphQL:

```
mutation { sacar(conta: 54321, valor: 140) { conta saldo } }
mutation { depositar(conta: 54321, valor: 200) { conta saldo } }
query { saldo(conta: 54321) }
```

7 - Endpoints REST

Método	URL	Parâmetros	Descrição
POST	/api/depositar	conta, valor	Deposita valor na conta
POST	/api/sacar	conta, valor	Saca valor da conta
GET	/api/saldo/{conta}	conta	Consulta saldo da conta

Exemplo:

```
POST https://localhost:56187/api/depositar?conta=54321&valor=200
```

8 - Program.cs

- Configura **DbContext** com MySQL
- Registra **ContaService**
- Configura **GraphQL**
- Adiciona **Swagger** para documentação REST
- Mapeia endpoints REST (`MapPost` e `MapGet`)
- Habilita GraphQL Playground (`/graphql`)

9 - Swagger

- URL: `https://localhost:56187/swagger`
- Permite testar os endpoints REST
- GraphQL é testado no Playground (`/graphql`)

10 - Conclusão

O projeto está estruturado para: - Separar **camadas de serviço**, **modelo** e **GraphQL** - Permitir testes via **GraphQL Playground** e **Swagger** - Funcionar com **MySQL** usando migrations do EF Core - Facilitar manutenção e escalabilidade