Documentação do Projeto **Sevencc Bank**

**1. OBJETIVO**

**Visão Geral e Objetivos do Projeto**

O **Sevencc Bank** é um banco inovador que visa proporcionar uma experiência bancária segura, acessível e moderna. Nosso objetivo é criar um sistema bancário que permita aos clientes realizarem as principais operações financeiras de maneira prática e segura. Para isso, estamos desenvolvendo um módulo essencial do sistema bancário, que consiste em uma API capaz de armazenar e gerenciar transações como saques, depósitos e transferências. Além disso, o sistema contará com um aplicativo móvel que consome essa API, oferecendo conveniência e acessibilidade aos usuários.

**Escopo do Projeto**

O escopo inicial do projeto contempla:

* Desenvolvimento de uma API para gerenciar transferências bancárias.
* Implementação de um módulo de cadastro que armazena dados de clientes, agências e contas correntes.
* Criação de um aplicativo móvel que consome a API, proporcionando uma interface intuitiva para os clientes realizarem suas transações.
* Implementação de funcionalidades para consultar extratos e recalcular saldos.

Fora do escopo:

* Módulos avançados de investimento, caixinhas, empréstimos, seguros ou programas de bonificações como cashback ou pontos (podem ser considerados em futuras expansões).

**2. ARQUITETURA**

O **Sevencc Bank** adota uma arquitetura em camadas, que separa as responsabilidades em diferentes componentes, seguindo o padrão MVC:

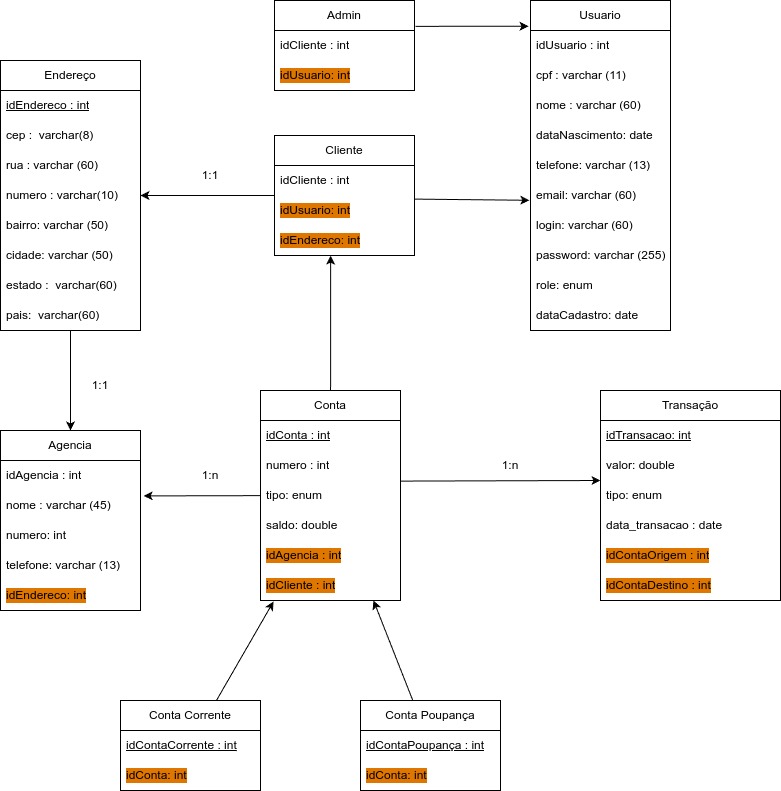
* **Camada de Apresentação:** Responsável pela interação com o usuário, sendo representada pelo aplicativo móvel.
* **Camada de Serviço:** Contém a lógica de negócios, onde são implementadas as regras e operações financeiras.
* **Camada de Acesso a Dados:** Gerencia a persistência dos dados, utilizando JPA/Hibernate para interagir com o banco de dados relacional.
* **Camada de Segurança:** Implementa mecanismos de autenticação e autorização, garantindo a segurança das transações.

**3. STACK TECNOLÓGICO**

* **Backend:** Spring Boot 3.3.2, utilizando JPA/Hibernate para mapeamento objeto-relacional.
* **Banco de Dados:** MySQL para persistência em produção e H2 em memória para testes.
* **Segurança:** Spring Security, com autenticação baseada em JWT.
* **Frontend e Mobile:** O aplicativo móvel é desenvolvido utilizando Kotlin, consumindo a API RESTful.
* **Testes:** Utilizamos a biblioteca JUnit com o framework Mockito, e para verificar a cobertura de testes utiliza o JaCoCo.
* **Gerenciamento de Dependências:** Maven
* **Documentação da API:** Swagger

**4. MODELAGEM**

**Diagrama ER Atualizado**



O modelo de dados foi redesenhado para garantir integridade e eficiência. O diagrama ER inclui as seguintes entidades:

 **Endereço**: Armazena informações de localização, como CEP, rua, número, bairro, cidade, estado e país. Relaciona-se com cliente e agência para vincular o endereço.

 **Usuário**: Armazena dados de autenticação e informações pessoais, como CPF, nome, data de nascimento, telefone, email, login, senha, papel do usuário (como cliente ou administrador) e data de cadastro.

 **Cliente**: Armazena dados pessoais e está associado a um usuário e a um endereço. Representa a entidade que pode realizar transações bancárias.

 **Admin**: Armazena a relação entre um cliente e seu status de administrador no sistema. Está vinculado ao cliente correspondente.

 **Agência**: Registra informações sobre a agência bancária, como nome, número da agência, telefone e o endereço associado.

 **Conta**: Gerencia os dados da conta bancária, incluindo número da conta, tipo (corrente ou poupança), saldo e as associações com cliente e agência.

 **ContaCorrente**: Gerencia os dados da conta corrente, associada a um cliente e a uma agência, através da entidade Conta.

 **ContaPoupança**: Gerencia os dados da conta poupança, associada a um cliente e a uma agência, através da entidade Conta.

 **Transação**: Registra todas as movimentações financeiras realizadas nas contas, como depósitos, saques e transferências, armazenando informações como valor, tipo e data da transação.

**5. COBERTURA DE TESTES**

* Testes Unitários - Camada Service:

Os testes unitários foram realizados para verificar a lógica de negócios implementada nos serviços da aplicação. As principais validações e funcionalidades testadas incluem:

**Operações Financeiras:** Garantir que as operações de depósito, saque e transferência sejam realizadas corretamente, atualizando os saldos das contas conforme esperado.

**Regras de Negócio:** Verificar a aplicação de regras específicas, como limites de transação, bloqueio de conta em caso de saldo insuficiente, e restrições de transações.

**Interação com o Repositório:** Testar a integração com o repositório de dados, garantindo que as operações de leitura e escrita sejam executadas corretamente, utilizando mocks para simular o comportamento do banco de dados.

**Tratamento de Exceções:** Verificar se exceções específicas são lançadas em casos de erros, como tentativa de saque em conta inativa ou com saldo insuficiente.

* Testes Unitários - Camada Model:

Os testes unitários na camada de Model foram realizados para garantir a consistência dos dados e validações nas entidades do sistema.

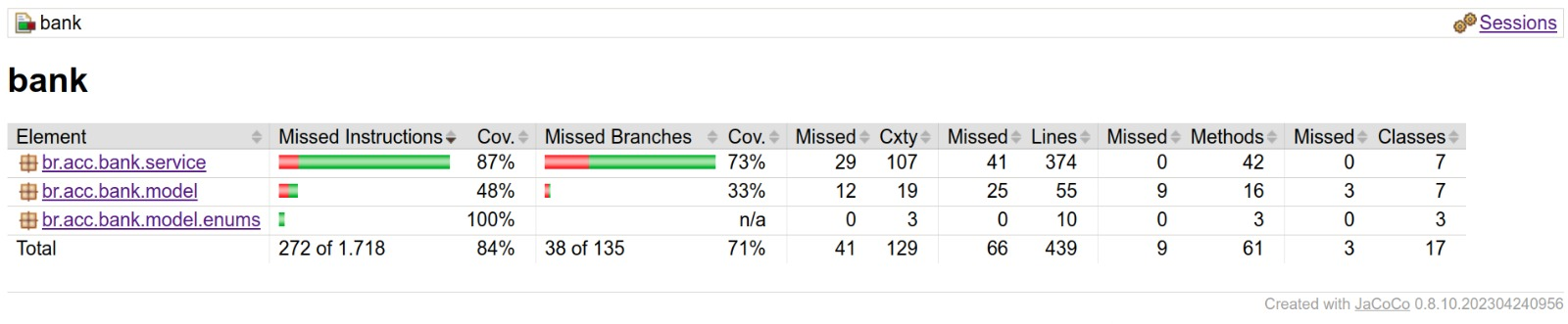
**Validação de Campos:** Testar a validação de campos obrigatórios, como nome, CPF, número da conta, e outros atributos essenciais das entidades.

**Relacionamentos Entre Entidades:** Garantir que os relacionamentos entre as entidades, como contas e clientes, estejam corretamente configurados e funcionando conforme esperado.

* Testes Unitários - Enum:

Os testes unitários em Enums foram realizados para garantir que os valores e comportamentos definidos nos enums estejam corretos.

**Verificação de Valores:** Testar se os enums contêm os valores corretos para representar estados ou tipos específicos, como status de transação (PENDENTE, CONCLUÍDA, CANCELADA) ou tipos de conta (CORRENTE, POUPANÇA).

**Mapeamento de Valores:** Garantir que os enums sejam mapeados corretamente em suas representações textuais ou numéricas, evitando erros de conversão.

De acordo com o relatório gerado através da ferramenta JaCoCo, indica que 84% das instruções do código do Sevencc Bank foram cobertas pelos testes, com 71% das ramificações (branches) testadas.

**6. CÓDIGO FONTE**

O código-fonte do Sevencc Bank se encontra no [Github](https://github.com/mateus-ssouza/sevencc-bank).