Universidade Federal Rural do Semi-Árido
Centro Multidisciplinar Pau dos Ferros
Departamento de Engenharias e Tecnologia
Projeto Detalhado de Software

Sistema Bancário

Pau dos Ferros – RN
Outubro de 2023

Arthur Kellyson Pinheiro De Negreiros - 2020022215
Pedro Makson Fontes Da Costa - 2021010616

Sistema Bancário

Relatório apresentado à Universidade Federal Rural do Semi-Árido, na disciplina de Projeto Detalhado de Software, como requisito para avaliação na 3ª unidade, solicitado pela professora Huliane Medeiros da Silva.

Pau dos Ferros – RN
Outubro de 2023

SUMÁRIO

1. Introdução

A presente proposta do projeto visa criar um sistema bancário robusto, seguro e eficiente, onde os clientes possam realizar uma variedade de operações financeiras com facilidade. Este sistema não só abrange o cadastro de usuários, mas também se destaca pela inovação ao incorporar a aprovação de contas por parte dos gerentes. Além disso, o sistema oferece funcionalidades de login seguras, permitindo aos clientes acessar suas contas com confiança. A capacidade de fazer depósitos, saques e transferências entre contas, juntamente com um sistema de solicitação e aprovação de empréstimos, torna o projeto altamente abrangente e relevante. Com ênfase na segurança e proteção dos dados dos clientes, este projeto oferece uma solução completa para as necessidades bancárias modernas.

2. Justificativa

O desenvolvimento deste sistema bancário é de grande importância, uma vez que permite aos clientes do "Imperial Bank" gerenciar suas contas bancárias e realizar operações financeiras de forma segura e conveniente. Além disso, a aplicação de conceitos de orientação a objetos e padrões de projeto é fundamental para criar um sistema bem estruturado, de fácil manutenção e escalabilidade. A criação de um sistema bancário é relevante no contexto atual, onde a segurança e a eficiência no setor financeiro são essenciais.

3. Objetivos

3.1 Objetivo geral

O objetivo principal deste projeto é desenvolver um sistema bancário que permita aos clientes do "Imperial Bank" realizar operações financeiras, gerenciar suas contas e solicitar empréstimos. O sistema deve ser seguro, eficiente e seguir os princípios da orientação a objetos.

3.2 Objetivos específicos

- Implementar um sistema de cadastro de usuários com a aprovação de um gerente.
- Desenvolver funcionalidades de login e autenticação de usuários.
- Permitir que os clientes realizem depósitos e saques em suas contas.
- Possibilitar a transferência de fundos entre contas de diferentes usuários.
- Implementar a solicitação e aprovação de empréstimos.
- Garantir a segurança das transações e informações dos clientes.

4. Metodologia

A metodologia de desenvolvimento deste projeto envolve a utilização da linguagem de programação Java e a aplicação de padrões de projeto, como o padrão Singleton, que desempenha um papel fundamental na gestão da conexão com o banco de dados (que foi realizada a partir das dependências disponibilizadas pelo Apache Maven). O Singleton garante que apenas uma instância da classe de conexão seja criada e compartilhada por todo o sistema. Isso otimiza o uso dos recursos do sistema, evita a sobrecarga de criar múltiplas conexões e contribui para um desempenho mais eficiente.

Além disso, outro padrão de projeto aplicado é o padrão Facade, que simplifica a interação do cliente com o sistema bancário. O Facade fornece uma interface unificada para várias partes do sistema, permitindo que os clientes acessem os serviços do banco de forma simplificada e intuitiva.

Nesse contexto, a orientação a objetos desempenha um papel essencial, aplicando conceitos como herança, encapsulamento e polimorfismo para criar um sistema bem estruturado e de fácil manutenção. A combinação desses elementos proporciona uma base sólida para o desenvolvimento de um sistema bancário robusto e eficaz.

5. Resultados

5.1 Requisitos do sistema

5.1.1 Requisitos Funcionais

[RF001] Cadastrar usuário.

O sistema deve permitir que o usuário realize um cadastro a partir das informações de suas seguintes credenciais, seja para usuário físico ou jurídico.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF002] Login.

O sistema deve permitir que o usuário realize login no sistema a partir das informações de suas credenciais CPF/CNPJ e senha.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF003] Menu.

Após efetuar o login, o sistema deve apresentar uma tela de menu com as seguintes opções disponíveis para o usuário: Ver Extrato, sacar, depositar, transferir, Empréstimo, atualizar dados, alterar senha, ver saldo, ver informações da conta, deletar conta, sair.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF004] Ver Extrato.

O sistema deve possibilitar que o usuário visualize suas informações bancárias.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF005] Sacar.

O sistema deve possibilitar que o usuário possa sacar um valor que seja menor ou igual ao saldo.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF006] Transferir.

O sistema deve possibilitar que o usuário possa realizar transferências entre contas do mesmo banco.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF007] Depositar.

O sistema deve possibilitar que o usuário deposite um valor.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF008] Empréstimo.

O sistema deve possibilitar que o usuário realize uma solicitação ao gerente um empréstimo de acordo com o valor de sua renda e/ou da receita da empresa e, que ele possa pagar as parcelas de seu empréstimo, visualizar empréstimo e voltar ao menu.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF009] Ver saldo.

O sistema deve possibilitar que o usuário visualize o seu saldo bancário.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF0010] Logout.

O sistema deve possibilitar que o usuário possa sair de sua conta a partir da opção de Logout .

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF011] Deletar conta.

O sistema deve possibilitar que o usuário possa apagar sua conta a partir da opção Deletar conta.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF012] Aceitação para conta.

O sistema deve possibilitar que o gerente realize se sim ou não que o usuário possa ter acesso ao sistema a partir de seu cadastro.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF013] Aprovação de empréstimo.

O sistema deve possibilitar que o gerente possa aprovar ou recusar uma solicitação de empréstimo do usuário.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF014] Atualizar dados.

O sistema deve possibilitar que o usuário possa atualizar a suas credenciais.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF015] Alterar senha.

O sistema deve possibilitar que o usuário possa alterar sua senha.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF016] Ver informações da conta.

O sistema deve possibilitar que o usuário possa visualizar as informações de sua conta.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

5.1.2 Requisitos Não Funcionais

[RNF001] Restrição de Cadastro.

O sistema só deverá permitir efetuar o cadastro se o cliente for maior que 18 anos.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RNF 002] Validação de cadastro.

O sistema só poderá permitir um cadastro no sistema, se todos os campos obrigatórios forem preenchidos corretamente a partir de suas validações, pois nome completo não pode ter número ou CPF/CNPJ não pode ter letras

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RF003] Limite de Saque.

O sistema deve possibilitar que o usuário possa apenas valores que sejam menores ou igual ao seu saldo atual.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RNF 004] Transferir valor.

O sistema só deverá permitir efetuar uma transferência bancária caso o valor desejado a ser transferido seja menor ou igual ao saldo atual.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

[RNF 005] Restrição de Login.

O sistema só deverá permitir que o acesso só seja efetuado se houver as informações das credenciais CPF/CNPJ e senha, desde que as mesmas estejam válidas no sistema.

Prioridade: (X) Essencial ( ) Importante ( ) Desejável

5.2 Diagramas de Classes

Na criação do diagrama de classe pela equipe, foi constituída a ferramenta Astah UML, que teve como objetivo trazer um esboço da estrutura do projeto, foi determinado que haveria seis entidades para o projeto que seriam: Conta, Emprestimo, Gerente, Usuario, UsuarioFisico e UsuarioJuridico.

Segue abaixo o modelo do diagrama de classe:

5.2.1 Classes entidades

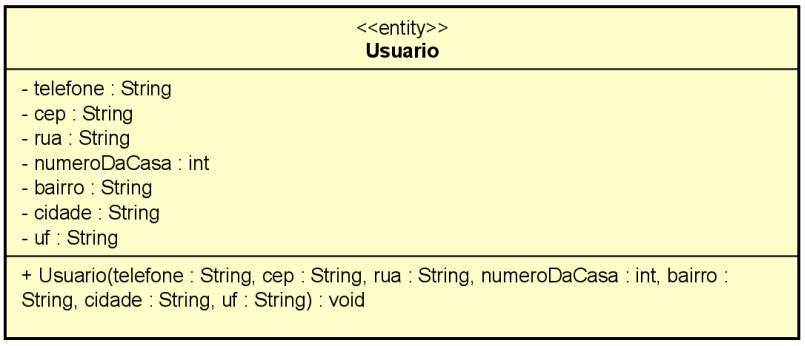

Figura 01: Diagrama de classe entidade *Usuario (classe mãe)*

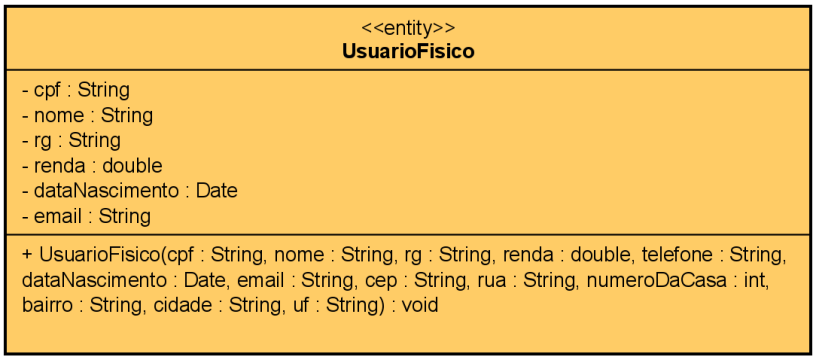

Figura 02: Diagrama de classe entidade *UsuarioFisico (classe filha)*

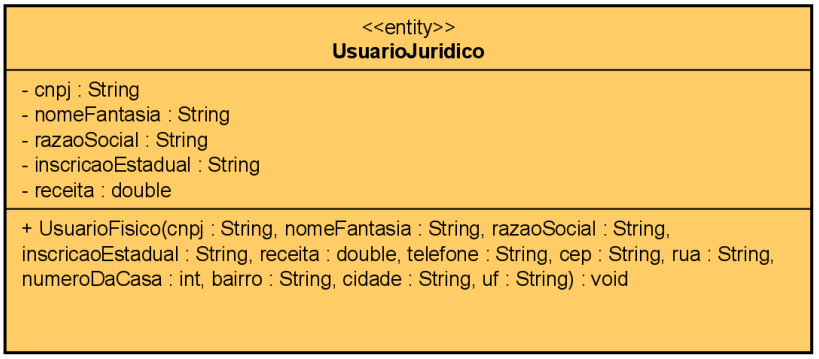

Figura 03: Diagrama de classe entidade *UsuarioJuridico (classe filha)*

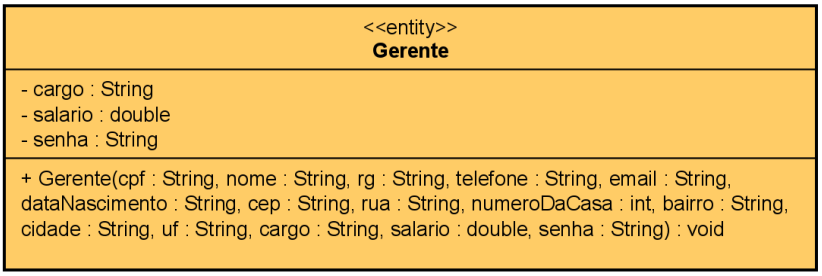

Figura 04: Diagrama de classe entidade *Gerente (classe filha)*

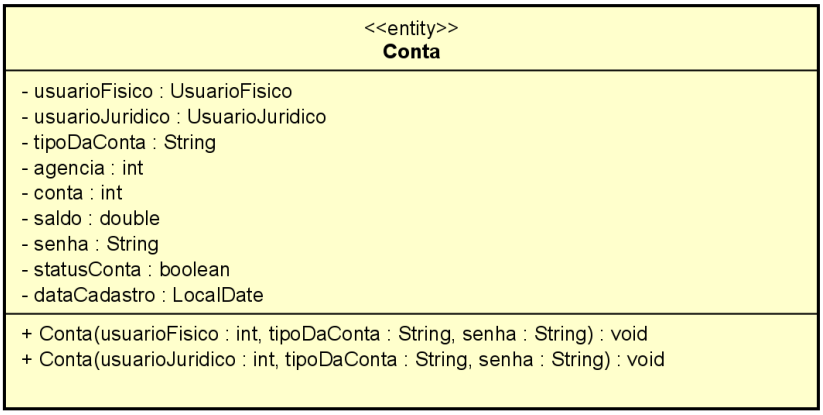

Figura 05: Diagrama de classe entidade *Conta*

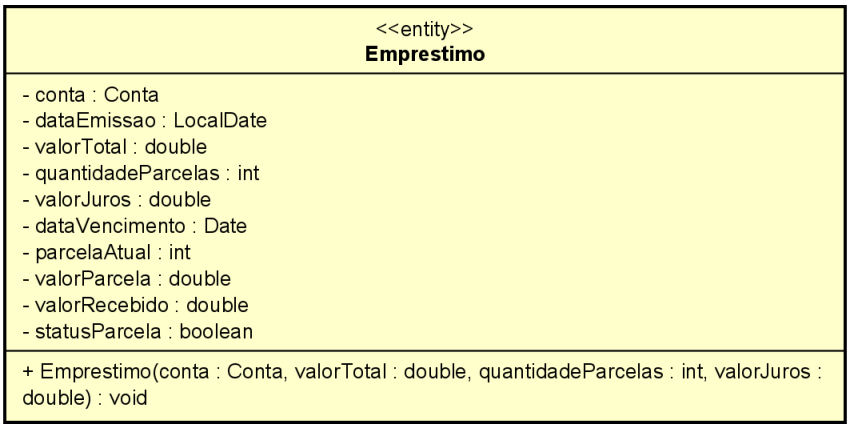

Figura 06: Diagrama de classe entidade *Emprestimo*

5.2.2 Classes auxiliares (Controllers)

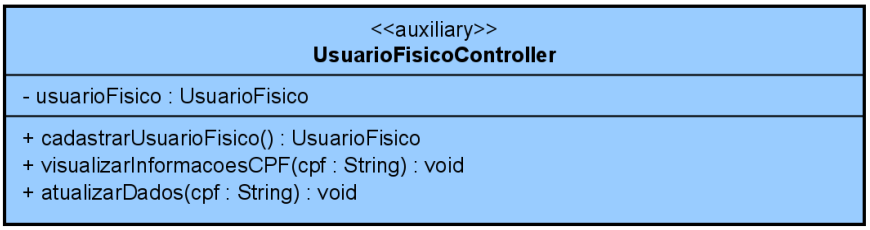

Figura 07: Diagrama de classe auxiliadora *UsuarioFisicoController*

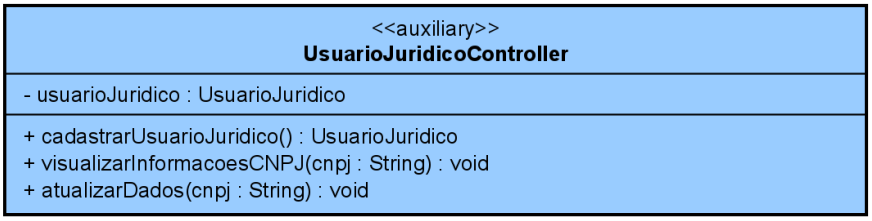

Figura 08: Diagrama de classe auxiliadora *UsuarioJuridicoController*

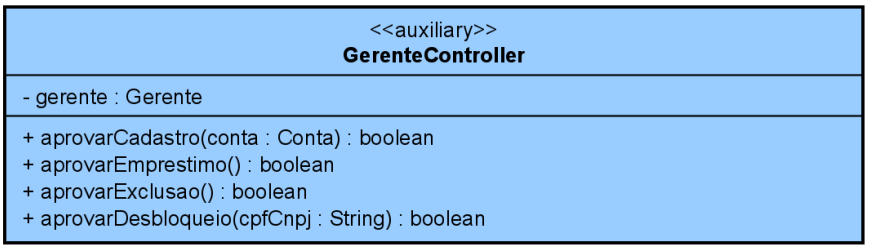

Figura 09: Diagrama de classe auxiliadora *GerenteController*

<<auxiliary>>
ContaController

- conta : Conta

+ cadastrarConta(tipoConta : int) : Conta
+ logar(cpfCnpj : String, senha : String) : boolean
+ sacar(cpfCnpj : String, valor : double) : boolean
+ depositar(cpfCnpj : String, valor : double) : void
+ transferir(contaOrigem : String, contaDestino : String, valor : double, senha : String) : boolean
+ imprimirExtrato(cpfCnpj : String) : void
+ alterarSenhaLogado(cpfCnpj : String, senha : String) : void
+ alterarSenhaDeslogado() : void
+ visualizarInformacoes(cpfCnpj : String) : void
+ imprimirSaldo(cpfCnpj : String) : double
+ excluirConta(cpfCnpj : String) : void
+ desbloquerConta() : void

Figura 10: Diagrama de classe auxiliadora *ContaController*

<<auxiliary>>
EmprestimoController

- emprestimo : Emprestimo

+ solicitarEmprestimo(cpfCnpj : String) : void
+ visualizarEmprestimo(cpfCnpj : String) : void
+ pagarParcelaEmprestimo(cpfCnpj : String) : void

Figura 11: Diagrama de classe auxiliadora *EmprestimoController*

5.2.3 Diagrama de classes (sem padrões)

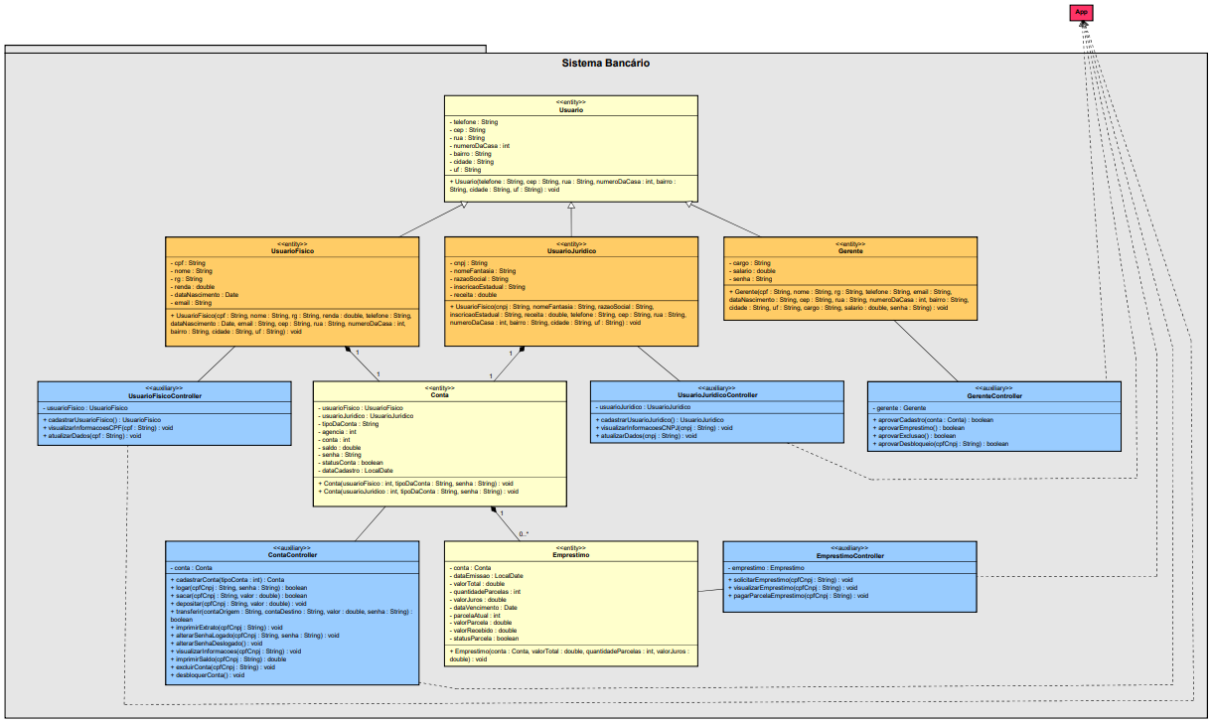

Figura 12: Diagrama de Classes (sem padrões)

5.2.4 Diagrama de classes (com padrões)

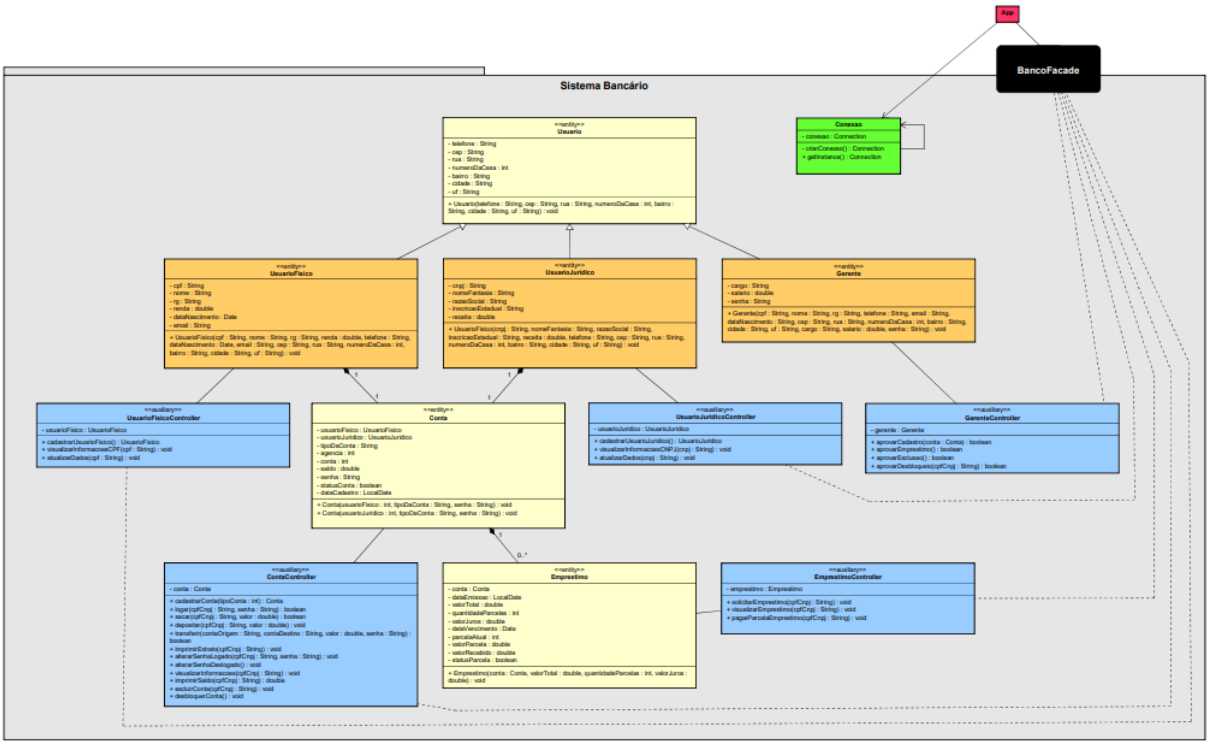

Figura 13: Diagrama de Classes (com padrões)

Respositório Git Hub

5.3 Explicação sobre as classes do sistema

Classe: Usuario.java
Descrição: A classe Usuario.java representa um modelo de entidade para armazenar informações sobre um usuário em um sistema. Ela encapsula uma série de atributos que descrevem um usuário, incluindo detalhes pessoais e de endereço

Classe: UsuarioFisico.java
Descrição: A classe UsuarioFisico.java é uma subclasse da classe Usuario.java e representa um modelo de entidade para armazenar informações sobre um usuário pessoa física em um sistema. Ela herda todos os atributos e métodos da classe Usuario.java, adicionando detalhes específicos de um usuário pessoa física..

Classe: UsuarioJuridico.java
Descrição: A classe UsuarioJuridico.java também é uma subclasse da classe Usuario.java e representa um modelo de entidade para armazenar informações

sobre um usuário pessoa jurídica em um sistema. Assim como a classe UsuarioFisico.java, ela herda todos os atributos e métodos da classe Usuario.java, mas inclui informações específicas de uma empresa ou organização.

Classe: Gerente.java
Descrição: A classe Gerente.java é uma subclasse da classe Usuario.java e representa um modelo de entidade específico para gerentes em um sistema. Ela herda características e atributos da classe Usuario.java e adiciona informações adicionais específicas para gerentes

Classe: Emprestimo.java
Descrição: A classe Emprestimo.java representa um modelo de entidade que é usado para gerenciar informações relacionadas a empréstimos em um sistema. Ela é útil para representar e gerenciar informações sobre empréstimos, permitindo que um sistema acompanhe o estado de cada empréstimo, a data de vencimento, o valor das parcelas e outros detalhes relacionados aos empréstimos feitos por clientes ou usuários.

Classe: Conta.java
Descrição: A classe Conta.java representa um modelo de entidade para representar informações relacionadas a contas bancárias no sistema. Ela armazena detalhes sobre as contas, incluindo informações sobre o titular da conta, tipo de conta, número da agência, número da conta, saldo, senha, status da conta e data de cadastro

Classe: Conexao.java
Descrição: A classe Conexao.java é responsável por criar e gerenciar uma conexão com um banco de dados PostgreSQL.

Classes: UsuarioFisicoDAO.java, UsuarioJuridicoDAO.java, GerenteDAO.java, EmprestimoDAO.java e ContaDAO.java
Descrição: Tem como objetivo encapsular a lógica de acesso e manipulação de dados em um banco de dados, fornecendo uma interface consistente e abstrata para interagir com os dados de uma aplicação. As classes citadas anteriormente

desempenham funções específicas relacionadas a usuários, gerentes, empréstimos e contas, respectivamente, permitindo que a aplicação realize operações como inserção, busca, atualização e remoção de informações nos respectivos domínios. Essas classes facilitam a separação de preocupações em um sistema, garantindo que as operações de acesso a dados sejam tratadas de forma coesa e organizada, o que contribui para a manutenção e escalabilidade do sistema.

Classes: UsuarioFisicoController.java, UsuarioJuridicoController.java, GerenteControlle.java, EmprestimoController.java e ContaController.java
Descrição: Essas classes são componentes essenciais do projeto que seguem a arquitetura Modelo-Visão-Controlador (MVC). Elas desempenham um papel fundamental na lógica de controle da aplicação, processando solicitações do usuário, interagindo com classes de modelo (por exemplo, UsuarioFisico, UsuarioJuridico, Gerente, Emprestimo e Conta).

5.4 Exibição dos resultados

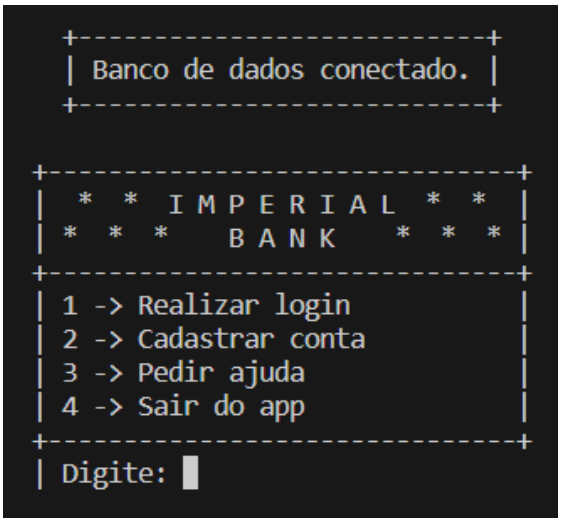

Figura 14: Tela inicial

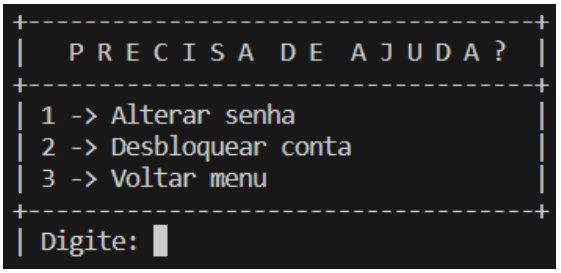

Figura 15: Tela de ajuda

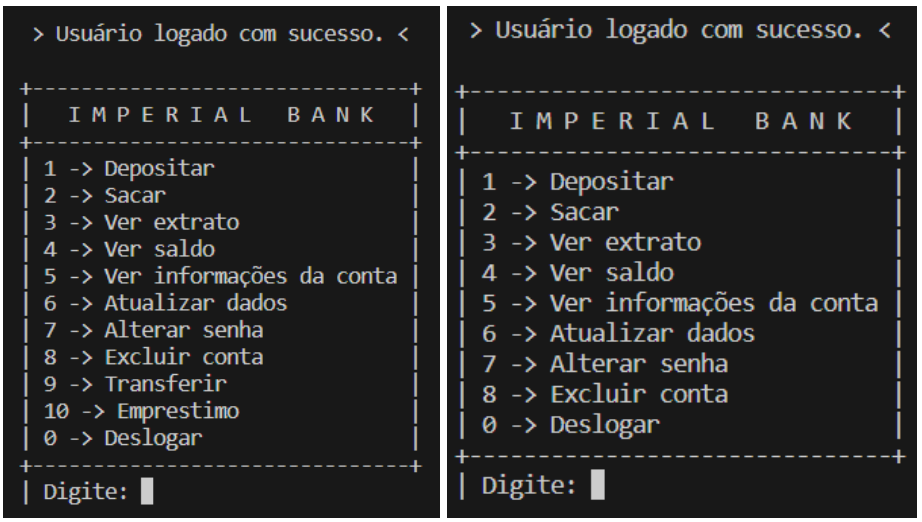

Figura 15: Tela de menu (conta corrente / conta poupança)

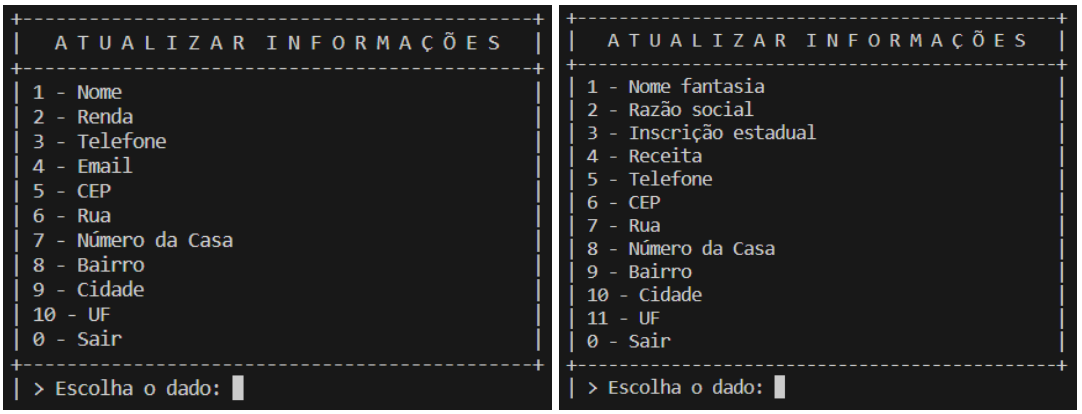

Figura 16: Tela de atualizar informações (pessoa física / pessoa jurídica)

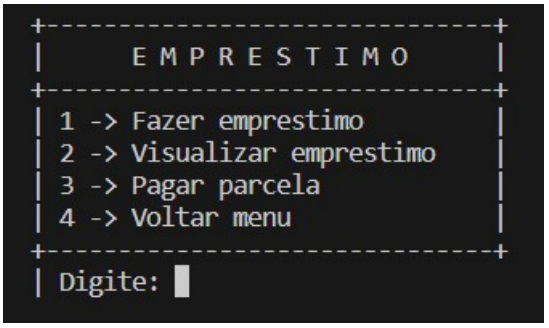

Figura 17: Tela de Empréstimos

5.5 Funcionalidades

5.5.1 Funcionalidades concluídas

Diante ao que se foi planejado na etapa de requisitos e do diagrama de classe podemos dizer que todas as funcionalidades foram implementadas.

5.5.2 Funcionalidades não concluídas

Não houve nenhuma funcionalidade que deixou de ser concluída, além do mais, acabou surgindo algumas outras a mais que precisassem ser implementadas no código.

5.5.3 Funcionalidades futuras

É deixado em aberto para prováveis aperfeiçoamentos do sistema futuramente, como uma interface gráfica e outras funcionalidades (como por exemplo, o usuário escolher pagar o empréstimo com o saldo da conta, habilitar o débito automático para pagar parcelas futuras, fazer mais funcionalidades para o Gerente) no Imperial Bank.

6. Conclusão

Ao longo do desenvolvimento deste projeto, enfrentamos uma série de desafios e dificuldades. Inicialmente, a etapa de levantamento de requisitos foi relativamente tranquila, graças à experiência prévia da equipe nesse aspecto. Isso nos permitiu acelerar a criação do diagrama de classes.

Durante a implementação do código, enfrentamos desafios decorrentes da necessidade de ajustar e refinar nossas ideias iniciais. Houve oscilações na abordagem à medida que novas soluções eram desenvolvidas para lidar com as mudanças necessárias e a reformulação de conceitos do projeto. O processo de conexão com o banco de dados foi mais tranquilo devido à familiaridade da equipe com o assunto, pois alguns membros já haviam cursado a disciplina equivalente.

Na etapa final, conduzimos testes rigorosos para garantir que as funcionalidades do programa atendessem ao que havia sido planejado, bem como às modificações necessárias que surgiram durante a parte prática do projeto.

Contudo, as maiores dificuldades encontradas estavam relacionadas à disponibilidade da equipe. Alguns membros estavam trabalhando e outros estavam envolvidos em projetos de pesquisa, o que consumia parte de nosso tempo disponível. Além disso, a discrepância nos horários de aulas dos membros dificultava a organização de reuniões, levando-nos a descartar, por exemplo, a implementação de uma interface gráfica que estava inicialmente planejada.

No entanto, superamos esses obstáculos graças ao apoio das aulas de orientação ao projeto juntamente com as aulas relacionadas a padrões de projetos. Essas fontes de suporte foram fundamentais para esclarecer dúvidas e orientar nosso progresso.

Em resumo, o desenvolvimento deste projeto nos permitiu adquirir valiosa experiência prática em programação orientada a objetos, aplicações de padrões de projetos e gerenciamento de sistemas complexos. Apesar dos desafios, estamos satisfeitos com o resultado final e com a oportunidade de aplicar nossos conhecimentos em um contexto real.

7. Referências

Refactoring Guru. (s.d.). **Singleton Design Pattern**. https://refactoring.guru/design-patterns/singleton. Acesso em 21 de out. de 2023.

Refactoring Guru. (s.d.). **Facade Design Pattern**. https://refactoring.guru/design-patterns/facade. Acesso em 21 de out. de 2023.