DEVBANK: RELATO DE EXPERIÊNCIA SOBRE O DESENVOLVIMENTO DE UM SISTEMA DE SERVIÇOS FINANCEIROS PARA UM BANCO DIGITAL NO PROJETO INTEGRADOR

Amanda Alice de Oliveira Farias[1]
Ana Beatriz Madruga D'Emery [2]
Lucas Floriano da Silva[3]
Luiz Claudio da Silva Aguiar Filho[4]
Messias Rafael Batista[5]

RESUMO

Este artigo relata o desenvolvimento de um aplicativo bancário, voltado para o público de desenvolvedores, explorando o mundo financeiro e sua complexidade. O desenvolvimento se deu utilizando os conhecimentos da linguagem Java aplicada a orientação a objetos, linguagem muito utilizada na programação por sua facilidade de execução. Foram utilizadas ferramentas que facilitassem a modelagem de requisitos através da criação de diagramas de casos de uso, mostrando a relação entre o ambiente e o sistema, e o diagrama de classe, indicando as classes, atributos e objetos instanciados. Para realizar a criação dos diagramas, foi utilizada a ferramenta Draw IO, software muito utilizado no mundo da programação para fazer diagramas, fluxogramas e organogramas. O aplicativo busca ser atender a necessidade de seus usuários no universo financeiro, com propostas inovadoras.

Palavras-chaves: Diagramas; Aplicativo; Desenvolvedores; Mundo Financeiro; Complexidade; Linguagem Java, Orientação a Objetos; Modelagem de Requisitos; Diagramas de casos de uso; Diagrama de classe; Draw IO, Necessidade dos usuários; Propostas inovadoras.

ABSTRACT

This article reports the development of a banking application, aimed at developers, exploring the financial world and its complexity. The development took place using knowledge of the Java language applied to object orientation, a language widely used in programming due to its ease of execution. Tools were used to facilitate requirements modeling through the creation of use case diagrams, showing the relationship between the environment and the system, and the class diagram, indicating classes, attributes and instantiated objects. To create the diagrams, the Draw IO tool was used, a software widely used in the world of programming to make diagrams, flowcharts and organization charts. The application seeks to meet the needs of its users in the financial universe, with innovative proposals.

[1] Graduanda do Curso de Sistemas para Internet E-mail: amanda.alice.pb@gmail.com
[2] Graduanda do Curso de Sistemas para Internet E-mail: biademery@gmail.com
[3] Graduando do Curso de Sistema de Informação E-mail: lucasfloriano09@gmail.com
[4] Graduando do Curso de Sistema de Informação E-mail: lfilho2017@gmail.com
[5] Professor Orientador, Tecnólogo em Sistemas para Internet pelo Centro Universitário de João Pessoa (UNIPÊ), Mestrado em Engenharia da Computação voltada a Inteligência Computacional na Universidade de Pernambuco (UPE), MBA em Engenharia de dados pela UNIESP, Docente do Curso Superior em Sistema de informação/Sistemas para Internet da disciplina de Linguagem de Programação I. E-mail: prof2118@iesp.edu.br

1 INTRODUÇÃO

A programação orientada a objetos (POO) é um paradigma de programação que se baseia no conceito de objetos, que podem ser definidos como instâncias de uma classe. A POO permite criar programas mais modulares, escaláveis e reutilizáveis em relação a outros paradigmas de programação. No contexto dos bancos digitais, a POO pode ser utilizada para desenvolver sistemas mais eficientes, seguros e confiáveis. Um banco digital é uma instituição financeira que opera exclusivamente pela internet, sem a necessidade de agências físicas. As transações bancárias são realizadas de forma remota por meio de plataformas digitais, o que exige um sistema robusto e confiável.

Pretende-se apresentar os principais conceitos da POO aplicados ao desenvolvimento de sistemas bancários digitais. Para isso, será realizada uma revisão bibliográfica sobre o tema, buscando identificar as melhores práticas e tendências do mercado financeiro. Além disso, será apresentado um estudo de caso sobre um banco digital que utiliza a POO em sua arquitetura de software, a fim de ilustrar as vantagens e desafios da utilização dessa abordagem.

A justificativa para este trabalho é a crescente importância dos bancos digitais no cenário financeiro, o que torna relevante o estudo das tecnologias utilizadas em sua criação. A POO é uma abordagem que tem se mostrado eficiente na criação de sistemas complexos e flexíveis, tornando-se uma opção bastante atraente para os desenvolvedores de bancos digitais.

Assim, o objetivo geral deste trabalho é analisar a aplicação da POO no desenvolvimento de sistemas bancários digitais, buscando identificar suas vantagens e desafios. Os objetivos específicos incluem apresentar os conceitos básicos da POO, descrever a arquitetura de um banco digital, analisar casos de sucesso de bancos digitais que utilizam a POO em sua arquitetura de software e discutir as vantagens e desafios da utilização da POO em sistemas bancários digitais.

Espera-se que este trabalho possa contribuir para o avanço da pesquisa e do desenvolvimento tecnológico da programação aplicada ao setor financeiro, além de fornecer subsídios para a criação de sistemas mais eficientes, seguros e confiáveis para os clientes dos bancos digitais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentada a fundamentação teórica do presente trabalho. Serão abordadas as ferramentas utilizadas para do desenvolvimento do aplicativo e a linguagem de programação aplicada.

2.1 Java Orientado a Objetos

A linguagem de programação Java foi utilizada no desenvolvimento do aplicativo bancário devido à sua experiência e popularidade. De acordo com Décio Heinzelmann Luckow e Alexandre Altair de Melo (2010), uma das características distintivas da linguagem Java é a

sua execução em uma Java Virtual Machine (JVM). Essa peculiaridade permite que qualquer dispositivo eletrônico capaz de executar uma máquina virtual seja capaz de executar programas em Java, proporcionando assim a portabilidade e a possibilidade de execução em qualquer lugar.

Tratando-se de orientação a objetos, "o programador visualiza seu programa em execução como uma coleção de objetos cooperantes que se comunicam por meio de troca de mensagens. Cada um dos objetos é uma instância de uma classe. Todas as classes formam uma hierarquia de classes unidas via relacionamento de herança." (Kaminski, 1996, p.4). Desta forma, a linguagem Java orientada a objetos pode proporcionar mais facilidade para programar um aplicativo bancário que possui diversas classes que conversam umas com as outras para o funcionamento de um objeto.

2.2 Diagrama de Caso de Uso

O levantamento de requisitos é um ponto importante para a construção de um software. A análise desses requisitos pode ser feita junto ao cliente ou membros da equipe de desenvolvimento. "Um software costuma ter vários tipos de usuário. Um projeto tem interessados em várias unidades organizacionais da empresa. Logo, o desafio é construir uma visão da especificação de requisitos que seja compreendida por várias pessoas presentes no lado do cliente." (VAZQUEZ e SIMÕES, 2016). Como facilitador para o desenvolvimento de um software, para que o desenvolvedor compreenda a visão do cliente, a modelagem é uma peça importante na construção desses requisitos.

Assim, o diagrama de caso de uso tem o importante papel de mostrar integração ambiente e sistema. De acordo com Sommerville (2011), os diagramas de caso de uso têm a função de servirem como um especificador de sistemas, pois descreve o seu comportamento. Por exemplo, o desempenho ou a confiabilidade requerida do sistema.

2.3 Diagrama de Classe

"Os diagramas de classe em UML podem ser expressos em diferentes níveis de detalhamento. Quando você está desenvolvendo um modelo, o primeiro estágio geralmente é o de olhar para o mundo, identificar os objetos essenciais e representá-los como classes. A maneira mais simples de fazer isso é escrever o nome da classe em uma caixa. Você também pode simplesmente observar a existência de uma associação, traçando uma linha entre as classes." (SOMMERVILLE, 2011, p.90). O diagrama de classes tem por objetivo mapear de forma precisa as classes, atributos, métodos e o relacionamento entre os objetos.

A utilização do diagrama de classes é colabora para o desenvolvimento de softwares, aplicativos, sistemas, etc. Pois, a visão de um todo pode ser vista a partir de sua construção para compreensão do objeto que será construído. Esta modelagem é fundamental para o processo criativo dos desenvolvedores.

3 METODOLOGIA

Este documento descreve as metodologias utilizadas do desenvolvimento do sistema utilizando a linguagem de programação Java. Durante o processo de criação, foram aplicados conceitos aprendidos em sala de aula, tais como o uso de métodos “get” e “set”, criação de classes e implementação de herança entre classes. Durante a criação do código, foi realizado uma análise detalhada dos requisitos do sistema, com o objetivo de compreender as funcionalidades necessárias e os dados que seriam manipulados. A partir desta análise, foram identificados os principais objetos e suas respectivas responsabilidades. Com base nos requisitos identificados, foram projetadas as classes necessárias para a implementação do sistema. Utilizou-se a técnica de modelagem orientada a objetos para identificar as características e comportamentos de cada classe. Foram Aplicados os conceitos de encapsulamento, abstração e modularidade. Sobre a linguagem, Java foi escolhida como base para implementação do sistema.

Através do uso de IDEs (Integrated Development Environments), como o IntelliJ, IDE escolhida para utilização, foram desenvolvidas as classes e os métodos necessários. Foram utilizados os métodos "get" e "set" para o encapsulamento dos atributos das classes, permitindo o acesso controlado aos dados. Além disso, foram aplicados conceitos de herança, onde foram criadas classes pai e filhas, aproveitando as características comuns e estendendo as funcionalidades específicas.

3.1 Projeto de leitura para remição do apenado

4 RESULTADO E DISCUSSÃO

Pix.java

```
private static String gerarPixCopiaColaAleatorio() {
        String caracteresPermitidos = "0123456789";
        StringBuilder pix = new StringBuilder();

        // 20 caracteres antes
        for (int i = 0; i < 20; i++) {
            int randomIndex = new Random().nextInt(caracteresPermitidos.length());
            char randomChar = caracteresPermitidos.charAt(randomIndex);
            pix.append(randomChar);
        }

        pix.append("br.gov.bcb.pix1001qrcodes-pix.devbank.com.br/v4/b");

        // 58 caracteres depois
        for (int i = 0; i < 58; i++) {
            int randomIndex = new Random().nextInt(caracteresPermitidos.length());
            char randomChar = caracteresPermitidos.charAt(randomIndex);
            pix.append(randomChar);
        }

        pix.append("BR71822RECEBEDOR0000CIDADE");

        // 8 caracteres depois
        for (int i = 0; i < 8; i++) {
            int randomIndex = new Random().nextInt(caracteresPermitidos.length());
            char randomChar = caracteresPermitidos.charAt(randomIndex);
            pix.append(randomChar);
        }

        // 7 caracteres depois "*"
        pix.append("*");

        // 7 caracteres depois
        for (int i = 0; i < 7; i++) {
            int randomIndex = new Random().nextInt(caracteresPermitidos.length());
            char randomChar = caracteresPermitidos.charAt(randomIndex);
            pix.append(randomChar);
        }

        // Consultando e usando o valor atualizado do StringBuilder
        return pix.toString();
    }
```

Investimentos.java

```
public static void consultarRendimentosInvest() {
        if (extratoInvest == null) {
            extratoInvest = new ArrayList<>();
        }

        double rendimento = saldoInvestimentos * 0.10;
        saldoInvestimentos += rendimento;

        LocalDateTime dataHora = LocalDateTime.now();
        String registro = String.format("%02d:%02d:%02d - %02d/%02d/%04d | Rendimentos de Investimentos | Valor = R$ %.2f",
                dataHora.getHour(), dataHora.getMinute(), dataHora.getSecond(),
                dataHora.getDayOfMonth(), dataHora.getMonthValue(), dataHora.getYear(),
                rendimento);
        extratoInvest.add(registro);
        System.out.println(registro);
    }

    public static void consultarExtratoInvest() {
        if (extratoInvest == null) {
            extratoInvest = new ArrayList<>();
        }

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss - dd/MM/yyyy");

        if (extratoInvest.isEmpty()) {
            System.out.println("Não há movimentação disponível.");
            return;
        }

        for (String operacao : extratoInvest) {
            String[] partes = operacao.split(" \\| ");
            if (partes.length < 3) {
                continue; // Ignora operações inválidas no extrato
            }
            String dataHora = partes[0];
            String descricao = partes[1];
            String Valor = partes[2];

            LocalDateTime dataHoraOperacao;
            try {
                dataHoraOperacao = LocalDateTime.parse(dataHora, formatter);
            } catch (DateTimeParseException e) {
                System.out.println("Erro ao converter a data e hora do registro: " + dataHora);
                continue; // Ignora operações com formato de data e hora inválido
            }
            System.out.printf("%s  |  %s | %s\n", dataHoraOperacao.format(formatter), descricao, Valor);
        }
    }
```

ContaPoupanca.java

```
private static void calcularRendimento() {
        double rendimento = saldopoupanca * 0.02;
        saldopoupanca += rendimento;

        LocalDateTime dataHora = LocalDateTime.now();
        String registro = String.format("%02d:%02d:%02d - %02d/%02d/%04d | Rendimento da Conta Poupança | Valor = R$ %.2f",
                dataHora.getHour(), dataHora.getMinute(), dataHora.getSecond(),
                dataHora.getDayOfMonth(), dataHora.getMonthValue(), dataHora.getYear(),
                rendimento);
        extratopoupanca.add(registro);
        System.out.println(registro);
    }
```

4.1 Diagrama de Caso de Uso

Um diagrama de caso de uso para um banco digital é uma representação visual que descreve as interações entre os atores (usuários) e o sistema do banco digital. Ele destaca as funcionalidades oferecidas pelo sistema e como os usuários interagem com essas funcionalidades.

No diagrama, os atores são representados pelos usuários do banco digital, como clientes, gerentes e administradores. Os casos de uso são as funcionalidades específicas do sistema, como "Transferir TED", "consultar saldo" e "Solicitar 2° Via".

As relações entre os atores e os casos de uso são mostradas por meio de linhas, representando a interação entre eles. Por exemplo, um cliente pode interagir com os casos de uso "Transferir TED" e "consultar saldo".

Cada caso de uso possui uma descrição detalhada, que define as ações realizadas pelo sistema em resposta às solicitações dos usuários. Além disso, o diagrama pode mostrar extensões, inclusões ou generalizações de casos de uso, indicando cenários alternativos ou relacionamentos hierárquicos entre as funcionalidades.

O diagrama de caso de uso é uma ferramenta importante para capturar os requisitos funcionais do sistema e fornecer uma visão geral das interações entre os usuários e o banco digital. Ele ajuda a compreender as funcionalidades necessárias e serve como base para o desenvolvimento e o design do sistema, garantindo que as necessidades dos usuários sejam atendidas de maneira adequada.

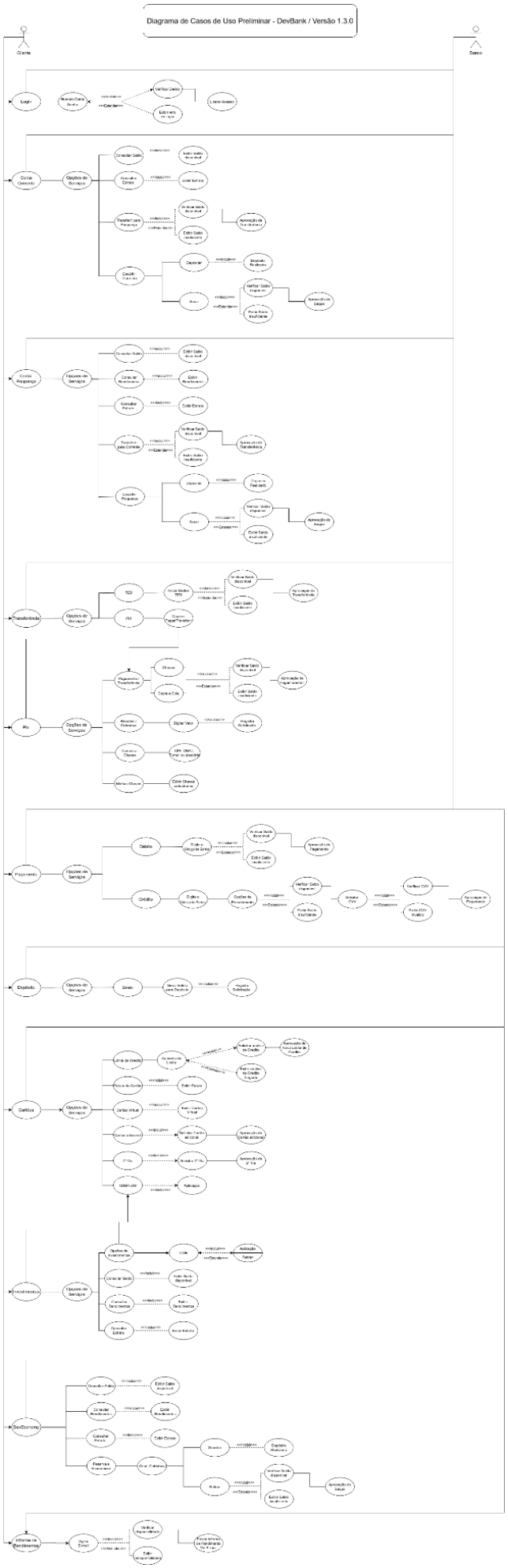


Diagrama de Casos de Uso Preliminar - DevBank / Versão 1.3.0
Cliente
Banco
Login
Número Conta Bancária
Verificar Saldo
Editar saldo
Liberar Acesso
Conta Corrente
Opções de Serviços
Consultar Saldo
Consultar Extrato
Transferir para Poupança
Depositar
Sacar
Conta Poupança
Opções de Serviços
Consultar Saldo
Consultar Rendimento
Consultar Extrato
Transferir para Corrente
Investir Poupança
Depositar
Sacar
Transferência
Opções de Serviços
TED
PIX
Pix
Opções de Serviços
Pagamento
Opções de Serviços
Débito
Crédito
Depósito
Opções de Serviços
Depósito
Cartões
Opções de Serviços
Investimentos
Opções de Serviços
Conta Investimento
Opções de Serviços
Informe de Rendimentos
Digitar Email

4.2 Diagrama de Classe

Um diagrama de classe para um banco digital é uma representação visual que mostra a estrutura do sistema, destacando as classes, atributos e relacionamentos entre elas. Ele é usado para modelar a estrutura do software e definir as entidades que compõem o banco digital.

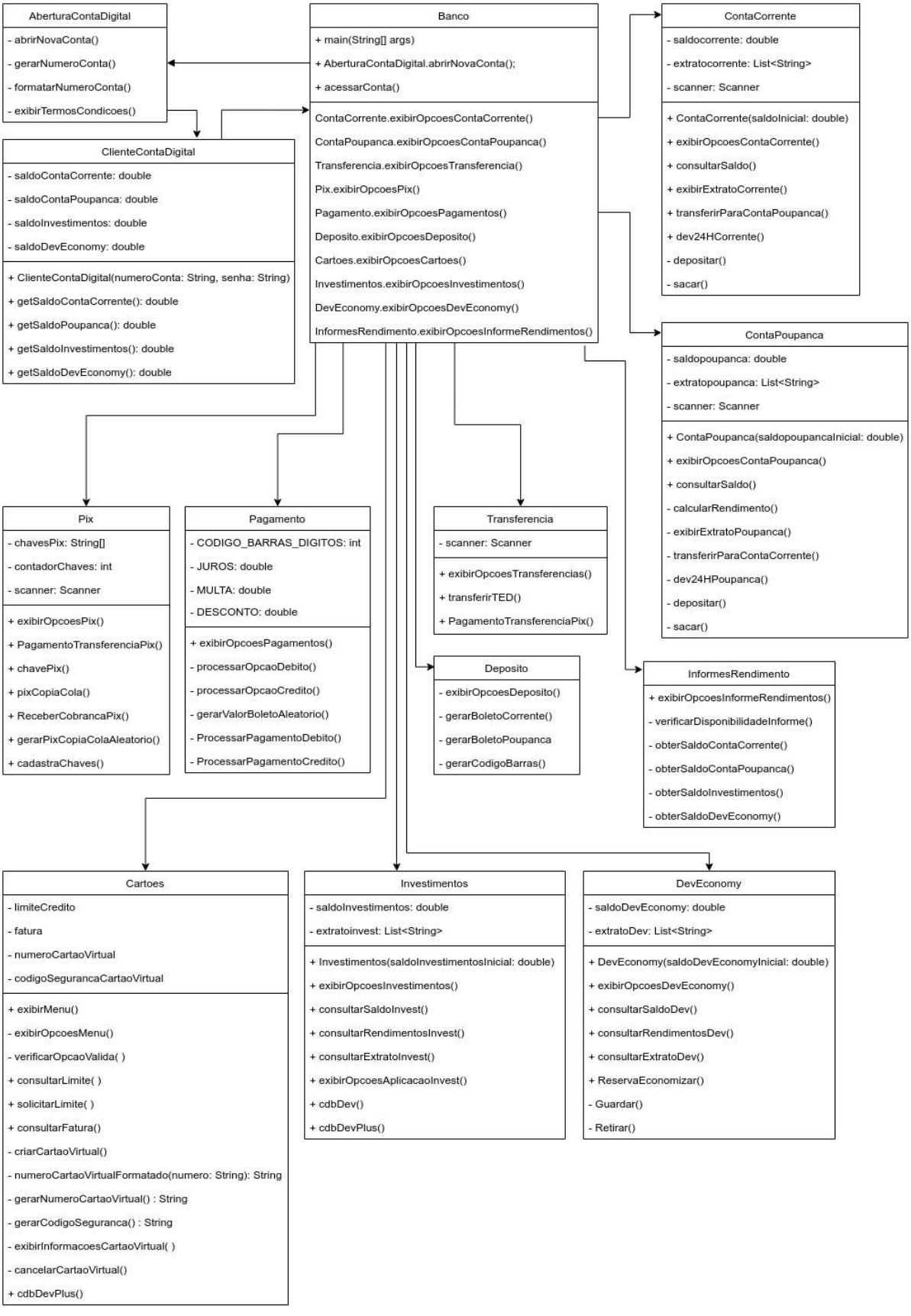
No diagrama, as classes representam os principais elementos do sistema, como "ClienteContaDigital" e "Banco". Cada classe possui atributos, que são as características ou propriedades daquela classe, por exemplo, "Número da Conta", "Senha" e "Saldo".

As classes são conectadas por relacionamentos, que podem ser associações, heranças ou dependências. Por exemplo, a classe "ClienteContaDigital" pode estar associada à classe "Conta Corrente", indicando que um cliente possui uma conta bancária. Além disso, a classe "Conta Corrente" pode herdar atributos e métodos da classe "ContaPoupança", que representa uma conta genérica.

O diagrama de classe também pode mostrar os métodos ou operações que podem ser realizados em cada classe, como "CalcularRendimento()" ou "TransferirTed()".

Em resumo, o diagrama de classe para um banco digital é uma representação visual da estrutura do sistema, mostrando as classes, atributos, relacionamentos e métodos. Ele ajuda a entender a organização do banco digital e serve como base para a implementação do software, definindo as entidades e as interações entre elas.

Diagrama de Classes Preliminar - DevBank / Versão 2.0.0



5 CONSIDERAÇÕES FINAIS

Durante o desenvolvimento do projeto, foram obtidos resultados significativos na busca por uma resposta ao problema apresentado inicialmente. O projeto se propunha a resolver o desafio de desmistificar a criação de um banco digital menos burocrático e intuitivo, e através de diversas etapas e abordagens, foram alcançados avanços notáveis.

No início do projeto, foi realizado um estudo detalhado problema, identificando suas principais causas e impactos. Isso permitiu uma compreensão mais aprofundada do contexto em que o problema estava inserido, facilitando a definição de metas e estratégias. Em seguida, foram realizadas pesquisas e análises extensas, buscando soluções existentes e melhores práticas relacionadas ao problema em questão. Foram identificadas diversas abordagens e tecnologias que poderiam ser aplicadas para solucionar o problema, e uma seleção criteriosa foi feita para determinar a mais adequada ao contexto do projeto.

Com base nessa seleção, foram desenvolvidos protótipos e realizados testes para validar a viabilidade e eficácia da solução proposta. Essa fase de experimentação permitiu a identificação de desafios e refinamento da abordagem, resultando em melhorias graduais e no aprimoramento do projeto como um todo. Durante o processo de implementação da solução, foram enfrentados obstáculos técnicos e desafios práticos.

No entanto, a equipe do projeto se manteve dedicada e colaborativa, buscando alternativas e soluções criativas para superar essas dificuldades. Essa abordagem resiliente e proativa permitiu avanços significativos na implementação da solução. Ao longo do projeto, também foi dada importância à coleta e análise de dados, a fim de monitorar o desempenho da solução e realizar ajustes quando necessário.

A partir dessas informações, foi possível verificar o impacto positivo da solução implementada, demonstrando sua eficácia na resolução do problema proposto.

No final do projeto, os resultados obtidos foram altamente satisfatórios. A solução desenvolvida foi capaz de resolver o problema inicial, proporcionando benefícios tangíveis, como a redução de custos, aumento da eficiência ou melhoria da qualidade. Além disso, o projeto contribuiu para a geração de conhecimento e aprendizado, destacando-se como um caso de sucesso na abordagem de desafios semelhantes.

Em resumo, o desenvolvimento do projeto resultou em avanços significativos na busca por uma resposta/solução ao problema inicialmente apresentado. Através de uma abordagem estruturada, pesquisa detalhada, prototipagem, testes e ajustes, foi possível alcançar um resultado satisfatório, solucionando o problema e trazendo benefícios para o contexto em questão.

5.1 AMEAÇAS AO PROJETO

Durante o desenvolvimento do projeto, é importante estar ciente das dificuldades que podem surgir e potencialmente invalidar o projeto. Essas dificuldades podem surgir em diferentes áreas e comprometer a viabilidade, eficácia ou relevância da solução proposta. A seguir, apresento um resumo das principais dificuldades que podem ser encontradas:

Viabilidade econômica: A falta de recursos financeiros suficientes ou a inviabilidade econômica da solução proposta podem invalidar o projeto. Os custos de implementação, manutenção e sustentabilidade da solução devem ser considerados cuidadosamente para garantir que o projeto seja viável do ponto de vista econômico.

Mudança no escopo ou requisitos: Durante o desenvolvimento do projeto, podem surgir mudanças no escopo ou nos requisitos. Essas mudanças podem afetar o planejamento, os prazos e a capacidade de entrega da solução. Se as mudanças forem significativas e não puderem ser acomodadas adequadamente, o projeto pode ser invalidado.

Falta de suporte ou expertise: A falta de suporte técnico especializado ou a ausência de conhecimento necessário para lidar com desafios específicos podem dificultar o progresso do projeto. A falta de expertise adequada na equipe ou a impossibilidade de obter suporte externo quando necessário pode inviabilizar a solução proposta.

Resistência à mudança: A resistência dos stakeholders ou das partes envolvidas afetadas pela solução proposta pode dificultar sua implementação. Resistência política, falta de apoio dos usuários finais ou conflitos de interesses podem comprometer a aceitação e o sucesso da solução.

É essencial antecipar e avaliar cuidadosamente essas dificuldades ao longo do desenvolvimento do projeto. Identificar os riscos associados e elaborar estratégias de mitigação adequadas pode aumentar as chances de sucesso e minimizar o impacto dessas dificuldades, garantindo a validade e eficácia dele.

5.2 TRABALHOS FUTUROS

Aprimoramento de funções: Uma implementação pode ser direcionada ao aprimoramento das funções existentes no projeto, visando melhorar a eficiência, usabilidade ou desempenho. Isso pode envolver a otimização de algoritmos, a introdução de novas técnicas ou a melhoria da interface do usuário.

Implantação de novas funções: Além de aprimorar as funções existentes, o projeto pode se beneficiar da implantação de novas funcionalidades. Isso pode envolver a adição de recursos que ampliam a utilidade da solução proposta, atendendo a novas demandas e necessidades dos usuários.

Criação de uma interface web e mobile: Uma implementação importante pode ser a criação de uma interface web e mobile para a solução. Isso permitiria que os usuários acessassem e utilizassem o projeto de maneira mais conveniente, independentemente de seu dispositivo. A interface web e mobile também poderia oferecer recursos adicionais, como notificações em tempo real ou integração com outras plataformas.

Financiamento para o projeto: A obtenção de financiamento é fundamental para levar o projeto adiante. Isso pode envolver a busca por investidores, solicitação de financiamento governamental, crowdfunding ou parcerias estratégicas. O financiamento adequado permitiria alocar recursos necessários para o desenvolvimento, testes e implementação do projeto.

Criação de departamentos específicos para o gerenciamento da empresa: À medida que o projeto evolui e cresce, pode ser necessário criar departamentos específicos para gerenciar as diferentes áreas da empresa. Isso pode incluir departamentos de desenvolvimento, marketing, vendas, suporte ao cliente, recursos humanos, entre outros. Esses departamentos garantiriam a eficiente operação e evolução contínua do projeto.

Modulação de um banco de dados para registro de novos clientes: Para manter o registro e gerenciamento eficiente de novos clientes, uma implementação pode envolver a modulação de um banco de dados adequado. Isso permitiria o armazenamento seguro das informações dos clientes, facilitando a busca, atualização e análise dos dados relevantes.

Parcerias para possíveis programas de fidelidade dentro da empresa: Uma implementação estratégica seria a busca por parcerias para a criação de programas de fidelidade dentro da empresa. Esses programas podem incentivar a fidelização dos clientes, oferecendo benefícios exclusivos, descontos ou recompensas especiais. As parcerias com outras empresas podem enriquecer ainda mais os benefícios e atrações para os clientes.

Essas implementações podem contribuir para o desenvolvimento e sucesso contínuo do projeto, agregando valor aos usuários, expandindo as funcionalidades e atraindo mais clientes. Cada uma delas deve ser cuidadosamente planejada e executada, levando em consideração os recursos disponíveis, as necessidades do mercado e as metas do projeto.

Utilização de API REST: A implementação de uma API REST (Representational State Transfer) pode permitir a integração do projeto com outras aplicações e sistemas. Isso possibilita a troca de dados e funcionalidades entre diferentes plataformas, ampliando a interoperabilidade e a capacidade do projeto de se integrar a ecossistemas existentes.

REFERÊNCIAS

LUCKOW, Décio Heinzelmann; DE MELO, Alexandre Altair. **Programação Java para a WEB.** Novatec Editora, 2010.

KAMIENSKI, Carlos Alberto. **Introdução ao Paradigma de Orientação a Objetos**. Artigo, UFPB, 1996.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira. **Engenharia de Requisitos: software orientado ao negócio**. Brasport, 2016.

TEIXEIRA, Cássio Adriano Nunes; CUKIERMAN, H. **Algumas observações sobre os vínculos entre a Engenharia de Software e o pensamento moderno**. In: Workshop um Olhar Sociotécnico sobre a Engenharia de Software. 2006.