Desenvolvimento de um Aplicativo Bancário em Python: Estudo de Caso BliBank

1st Bernardo Euclides Faria Gomes
Engenharia de Controle e Automação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
bernardoeuclides1962@gmail.com

2nd Israel Jardim de Souza
Engenharia de Controle e Automação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
israel.sdjardim@gmail.com

3rd Lariane Gonçalves Mendes
Engenharia de Controle e Automação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
larianegoncalvesm@gmail.com

***Abstract*—Este artigo apresenta o desenvolvimento do sistema bancário digital denominado BliBank, utilizando conceitos de Programação Orientada a Objetos (POO). Implementado em Python, o sistema abrange funcionalidades essenciais tanto para clientes quanto para administradores, incluindo cadastro de usuários, operações financeiras, gestão do cartão de crédito e investimentos. O estudo detalha a estrutura do sistema, a metodologia aplicada e os resultados alcançados durante o desenvolvimento. Aborda-se a aplicação dos pilares da POO — encapsulamento, herança, polimorfismo, modularização e abstração — e inclui cenários de testes e validações. Através deste estudo, demonstra-se como a POO facilita a construção de um sistema modular, extensível e de fácil manutenção, essencial para aplicações bancárias que exigem alta confiabilidade e segurança.**

***Index Terms*—banking application, modular development, Python, digital banking, account management, financial transactions.**

I. Introdução

Nos últimos anos, o uso de serviços bancários pela internet tem crescido de forma significativa, refletindo a tendência global de digitalização dos serviços financeiros. De acordo com o relatório da Federação Brasileira de Bancos (FEBRABAN) de 2023, o número de transações bancárias realizadas via internet banking cresceu 30% em comparação com o ano anterior [1]. Este aumento destaca a preferência crescente dos consumidores por soluções digitais práticas e seguras para gerenciar suas finanças. Na Figura 2, apresentamos um gráfico ilustrando o crescimento do uso do internet banking nos últimos anos, conforme os dados da Instituição, evidenciando um aumento constante e significativo no uso desses serviços.

Além disso, o perfil dos usuários de sistemas de internet banking é diversificado, abrangendo desde jovens adeptos a tecnologia até pessoas idosas que buscam agilidade nas transações. A Pesquisa Nacional por Amostras de Domicílios (PNAD), realizada pelo Instituto Brasileiro de Geografia e Estatística (IBGE), revela que a faixa etária predominante entre os usuários de internet banking varia entre 25 e 45 anos, com uma utilização significativa entre pessoas acima de 60 anos que buscam praticidade e segurança [2]. O gráfico ilustrando essa distribuição etária é apresentado na Figura 1, destacando o crescimento constante do uso desta ferramenta nas faixas etárias.

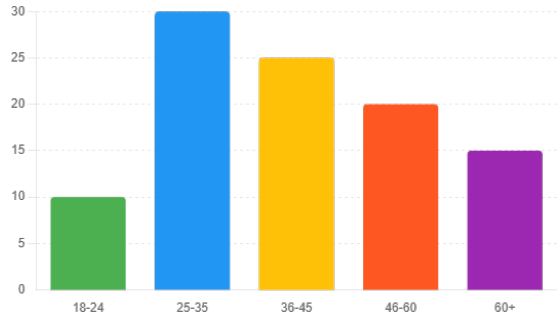

Fig. 1. Distribuição Etária dos Usuários de Internet Banking.

Neste contexto, identificamos a oportunidade de elaborar o BliBank, um sistema bancário desenvolvido em Python [3] e que utiliza amplamente os conceitos de Programação Orientada a Objetos (POO). As Funcionalidades do BliBank são comumente encontradas em bancos digitais conhecidos, como NuBank [4] e Banco Inter [5]. Esses aplicativos serviram de modelo para a criação do nosso Internet Banking.

Com a aplicação de Herança, Polimorfismo, Abstração, Encapsulamento, Classe e Objeto, nosso objetivo foi criar uma solução intuitiva e eficiente, em um sistema complexo com uma estrutura bem definida, facilitando a incorporação de novas funcionalidades e a manutenção do código existente. A escolha de Python, como linguagem de programação, se deu pela sua simplicidade e poder, que aliados aos conceitos de POO, permitem o desenvolvimento de um sistema mais eficiente, graças à sua infinidade de bibliotecas. O software desenvolvido visa proporcionar uma experiência clara e simples, eliminando as incertezas nas transações financeiras, trazendo uma interface amigável e funções acessíveis.

Os códigos desenvolvidos estão disponiıveis em um repositório [6] do GitHub que pode ser acesso por meio deste link: https://github.com/LarianeMendes/Projeto-de-POO

A. As principais contribuições deste estudo são:

- Eficiência na Gestão Bancária: A abordagem modular facilita a manutenção e a escalabilidade do sistema, permitindo rápidas implementações de novas funcionalidades e melhorando a eficiência operacional dos bancos.

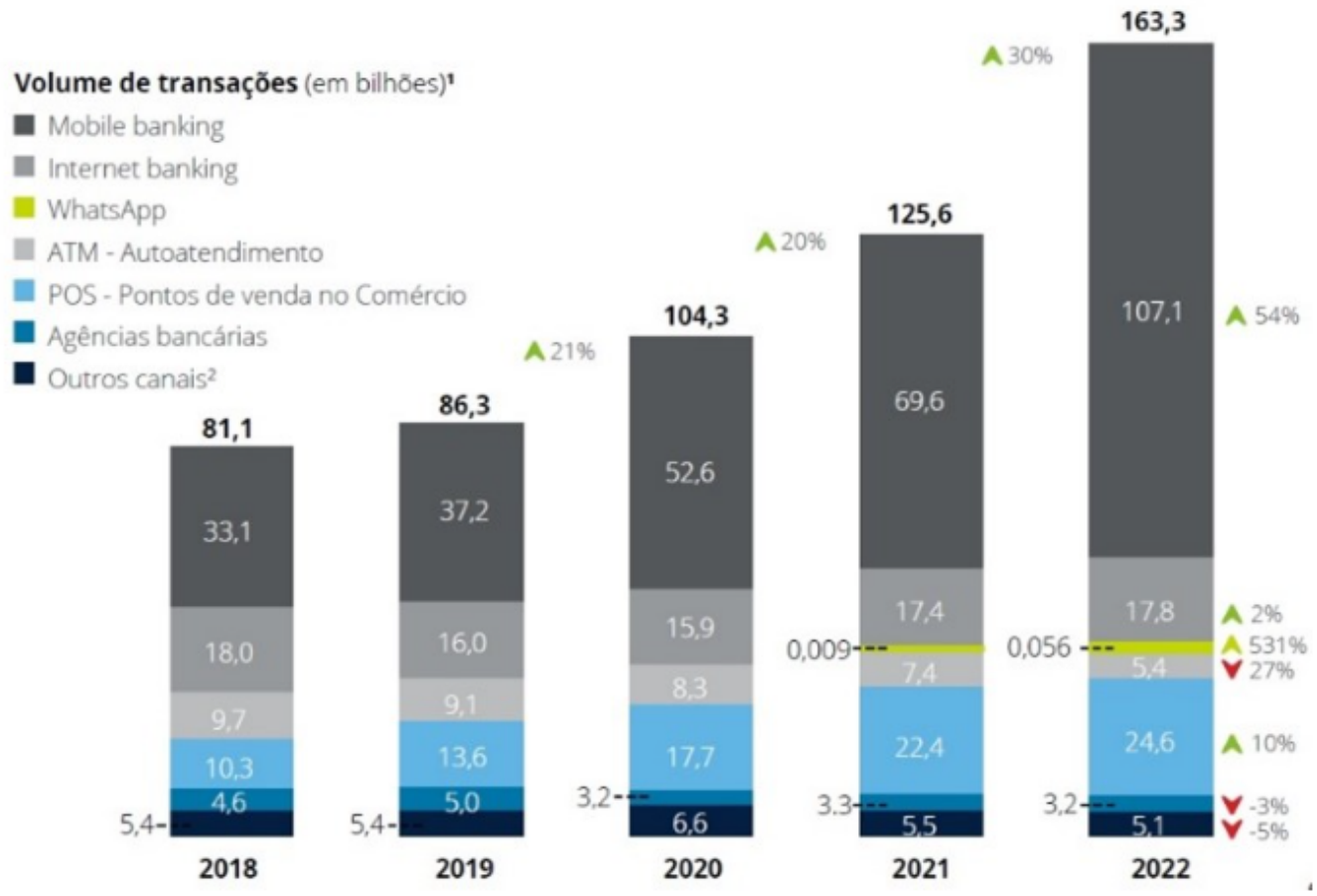

Fig. 2. Uso Internet Banking x Ano.

- Interface Intuitiva: Na tela inicial do aplicativo, o cliente consegue ver o saldo e tem acesso rápido a diversas funções, como realizar operações financeiras e gerenciamento do cartão de crédito. Do mesmo modo, o administrador consegue ajustar limite de crédito, encerrar contas e aprovar solicitações de cartões.

B. Organização do trabalho:

Este artigo está organizado da seguinte forma: a Seção 2 apresenta uma revisão da literatura sobre tecnologias e metodologias utilizadas no desenvolvimento de aplicativos bancários. A Seção 3 detalha a metodologia adotada para o desenvolvimento do BliBank, incluindo a estrutura do sistema e suas principais funcionalidades. Na Seção 4, são descritas as implementações dos diferentes módulos do sistema com exemplos de código e os resultados obtidos. Finalmente, a Seção 5 conclui o trabalho, resumindo os resultados e propondo melhorias e direções para pesquisas futuras.

II. Referencial Teórico

A Programação Orientada a Objetos (POO) é um paradigma de programação que organiza o software como uma coleção de unidades de dados, ou objetos, que podem conter dados na forma de campos (atributos) e código na forma de procedimentos (métodos). A POO se baseia em quatro conceitos principais: encapsulamento, abstração, herança e polimorfismo. [7] [8] [9]

A. Conceitos Principais da POO

- Encapsulamento: Consiste em agrupar dados e métodos que manipulam esses dados dentro de uma única unidade chamada classe. Isso protege os dados de acesso externo não autorizado e permite que o estado interno de um objeto seja alterado apenas por meio de métodos definidos.
- Abstração: Refere-se à capacidade de um programa de ignorar alguns aspectos das informações que estão sendo manipuladas, concentrando-se apenas nos detalhes mais relevantes. Em POO, isso é realizado através de classes e objetos, onde a complexidade dos dados e operações é ocultada para fornecer uma interface simples ao usuário.
- Herança: Permite que uma nova classe herde os atributos e métodos de uma classe existente. Isso promove o reuso de código e a criação de uma hierarquia de classes. A classe que é herdada é chamada de superclasse ou classe base, e a classe que herda é chamada de subclasse ou classe derivada.

- Polimorfismo:Refere-se à capacidade de diferentes classes de serem tratadas como instâncias de uma mesma classe por meio de uma interface comum. Métodos podem ser implementados de diferentes maneiras em diferentes classes, mas podem ser chamados através da mesma interface, permitindo que um método tenha várias formas.

B. Vantagens da POO

- Modularidade: Permite a criação de sistemas modulares, onde cada objeto representa uma parte do sistema, facilitando manutenção e atualização.
- Reuso de Código: Com herança e polimorfismo, o código pode ser reutilizado e estendido com facilidade.
- Facilidade de Manutenção: O encapsulamento protege os dados internos de alterações não autorizadas, reduzindo erros e facilitando a depuração.
- Clareza e Organização: Classes e objetos permitem uma melhor organização do código, tornando-o mais legível e compreensível.

C. Desvantagens da POO

- Complexidade Inicial: Para iniciantes, entender e aplicar os conceitos de POO pode ser mais difícil em comparação a paradigmas de programação mais simples.
- Sobrecarga de Performance: Em alguns casos, o uso extensivo de objetos pode levar a uma sobrecarga de desempenho devido à abstração e à dinâmica de chamadas de métodos.

A POO é amplamente utilizada em linguagens de programação modernas como Java, C++, Python e C(sharp), e continua sendo uma abordagem fundamental para o desenvolvimento de software escalável.

III. Metodologia

A metodologia adotada para o desenvolvimento do BliBank seguiu uma abordagem estruturada e cuidadosa, abrangendo desde a concepção inicial até a implementação e testes finais.

A. Definição de Requisitos

A primeira etapa consistiu na definição dos requisitos básicos do sistema. Esta fase envolveu um processo de brainstorming, focado nas necessidades diárias e nas expectativas dos brasileiros em relação aos serviços bancários. Observou-se como o banco é utilizado no dia a dia, levando em consideração aspectos como a conveniência de poder realizar operações financeiras rapidamente pelo celular, a importância de uma gestão eficiente de crédito e a facilidade de gerenciar investimentos de forma prática e intuitiva. Com base nessas observações, foram identificadas as seguintes funcionalidades essenciais: cadastro de usuários, login e logout, operações financeiras (depósitos, saques, transferências e investimentos) e gerenciamento de cartões de crédito, incluindo solicitação, aprovação, registro de compras, pagamento de faturas e solicitação de aumento de limite. Esses requisitos foram definidos para garantir que todas as necessidades dos usuários fossem atendidas de forma eficiente e prática. Além disso, o nome ”BliBank” foi criado como uma homenagem aos autores do projeto: Bernardo, Lariane e Israel. A escolha do nome reflete a colaboração e o esforço conjunto da equipe no desenvolvimento do banco digital.

B. Fluxograma do Desenvolvimento

Após a definição dos requisitos, foi criado um fluxograma, Figura 3, para ilustrar o processo metodológico adotado no desenvolvimento do BliBank. Ele foi desenvolvido pelo grupo, que discutiu e mapeou aspectos como a sequência de passos, desde a definição dos requisitos até os testes e validações finais, as interações entre as diferentes funcionalidades e os usuários, destacando como cada operação se conecta e contribui para o funcionamento do sistema, e os pontos críticos onde poderiam ocorrer problemas, permitindo a antecipação de soluções para evitar falhas. O fluxograma ajudou a proporcionar uma visão clara do fluxo de informações e das operações dentro do sistema, sendo fundamental para identificar possíveis pontos de melhoria e otimizar a eficiência do desenvolvimento.

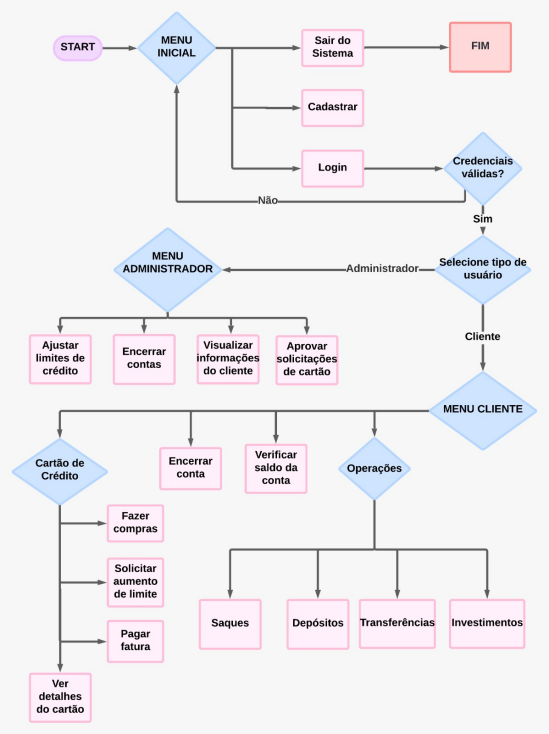

Fig. 3. Fluxograma do projeto

No fluxograma, o menu inicial oferece opções de cadastro, login e saída do sistema. Após o login, o usuário pode escolher entre os perfis de administrador ou cliente. Administradores têm a capacidade de ajustar limites de crédito, encerrar contas, visualizar informações dos clientes e aprovar solicitações de cartão de crédito. Já os clientes podem verificar o saldo, realizar saques, depósitos, transferências, investimentos e gerenciar seus cartões de crédito. As ações foram devidamente testadas, garantindo segurança e ausência de erros nas transações.

C. Modelagem do Sistema

Para garantir uma estrutura clara e bem definida, utilizamos diagramas UML (Unified Modeling Language) para modelar as classes e suas interações. As principais classes identificadas foram: Usuario, Cliente, Administrador, GestaoConta, CartaoCredito e Investimentos. Essa organização permitiu visualizar a estrutura do sistema e definir as relações entre as classes, garantindo que todos os componentes necessários fossem incluídos e corretamente integrados. Esta modelagem foi essencial para alinhar o grupo e assegurar que todos tivessem um entendimento comum sobre a arquitetura do sistema. Os diagramas UML utilizados estarão em anexo a este trabalho.

D. Aplicação dos Pilares da Programação Orientada a Objetos

Neste projeto, aplicamos os pilares da Programação Orientada a Objetos (POO) para criar um sistema modular, escalável e de fácil manutenção. A seguir, detalhamos como cada um desses pilares foi implementado no BliBank:

1) Encapsulamento: O encapsulamento foi aplicado para proteger os dados sensíveis dos usuários. Atributos como senha e CPF foram definidos como privados, acessíveis apenas através de métodos específicos (getters e setters) que garantem a integridade e a segurança dos dados. A figura 4 apresenta um exemplo do funcionamento do encapsulamento.

No código do BliBank, por exemplo, na classe Usuario, métodos como get_senha e get_cpf foram implementados para acessar os dados de forma controlada. Isso evita o acesso direto aos dados, protegendo-os de modificações indevidas.

```python
class Usuario:
    def __init__(self, nome: str):
        self.__nome = nome  # Atributo privado

    def set_nome(self, novo_nome: str):
        self.__nome = novo_nome

    def get_nome(self) -> str:
        return self.__nome

usuario = Usuario("Bernardo")

# Altera o nome do usuário usando o método set_nome
usuario.set_nome("Benado")
# Imprime o nome do usuário usando o método get_nome
print(usuario.get_nome())  # Saída: Benado

# Tentativa de acesso direto ao atributo privado (causará erro)
print(usuario.__nome)
# ERRO: AttributeError: 'Usuario' object has no attribute '__nome'
```

Fig. 4. Pseudo código para exemplificar o uso do encapsulamento.

2) Herança: A herança permitiu a criação de classes específicas (Cliente e Administrador) a partir da classe base Usuario, reutilizando código e garantindo consistência. A classe Usuario define atributos e métodos comuns a todos os tipos de usuários, enquanto Cliente e Administrador expandem essa funcionalidade para incluir métodos específicos de cada tipo de usuário. Isso promove a reutilização de código e facilita a manutenção e expansão do sistema. Essas relações podem ser observadas na Figura 5, que é um trecho do diagrama UML geral do BliBank.

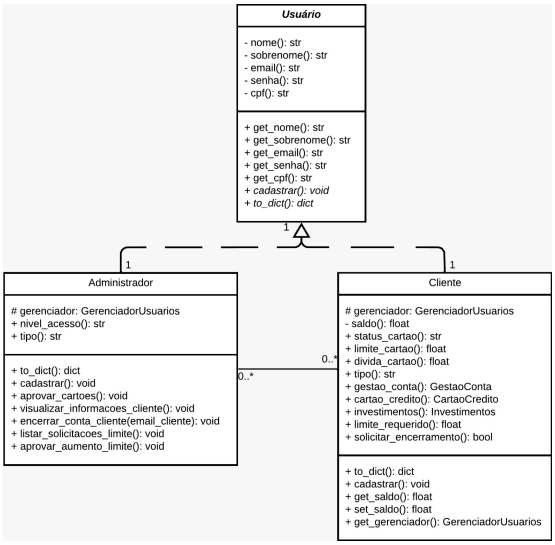

Fig. 5. Trecho do UML que demonstra a Herança do código

3) Polimorfismo: O polimorfismo foi implementado através de métodos abstratos e sobrecarga de métodos, permitindo que diferentes classes respondessem de maneira específica a chamadas de métodos definidos na classe base. Por exemplo, os métodos cadastrar e to_dict foram definidos na classe Usuario como abstratos e implementados de maneira específica nas classes Cliente e Administrador. Isso permite que o sistema trate objetos de diferentes classes de maneira uniforme, promovendo a flexibilidade e a extensibilidade do código.

4) Abstração: A abstração foi utilizada para definir interfaces comuns, permitindo que diferentes tipos de usuários compartilhassem a mesma estrutura básica, mas com implementações específicas para suas necessidades. A classe Usuario é uma classe abstrata que define métodos que devem ser implementados pelas classes derivadas Cliente e Administrador. Isso facilita a extensão do sistema para incluir novos tipos de usuários no futuro, sem a necessidade de grandes mudanças no código existente.

5) Modularização: A arquitetura do sistema foi planejada para maximizar a modularidade e a reutilização de código. O projeto foi organizado em diferentes módulos, cada um responsável por uma parte específica da funcionalidade do sistema. A estrutura de diretórios do projeto foi dividida conforme organização apresentada na Figura 6.

E. Interface Gráfica

A interface gráfica do BliBank foi desenvolvida com foco na usabilidade e na experiência do usuário. Utilizando a biblioteca Flet [10], a interface proporciona uma navegação intuitiva e acessível. As telas são organizadas de forma clara, permitindo que tanto clientes quanto administradores acessem facilmente as funcionalidades disponíveis. Por exemplo, na tela de login, usuários podem rapidamente inserir suas credenciais e escolher seu perfil. Uma vez logados, os clientes têm acesso direto às operações financeiras, visualização de saldo e gerenciamento

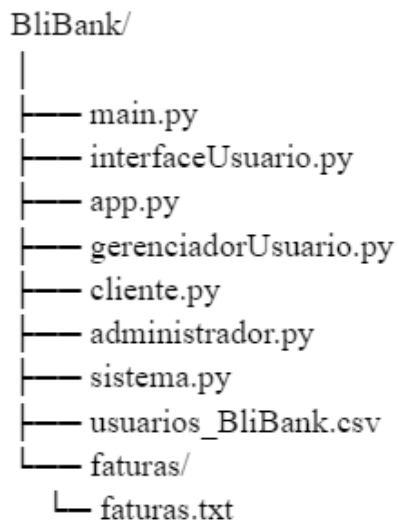

Fig. 6. Organização modular do BliBank.

de cartões. Administradores, por sua vez, encontram opções para gerenciar limites de crédito, aprovar cartões e encerrar contas. A combinação de uma interface gráfica bem projetada com as utilidades do sistema garante uma plataforma eficaz e confiável para todos os usuários do BliBank.

F. Testes e Validações

Após a implementação das classes, foram realizados testes unitários para validar o funcionamento das funcionalidades. O sistema foi testado em diferentes cenários para garantir a integridade das operações. Foram realizados vários testes unitários, abrangendo as operações do software. Cada funcionalidade foi testada isoladamente e em conjunto para assegurar que o sistema como um todo funcionasse conforme esperado. Além disso, cenários de erro foram simulados para garantir que o sistema lide adequadamente com situações inesperadas.

G. Manipulação e Tratamento de Dados

Durante o desenvolvimento do trabalho, utilizamos a biblioteca 'pandas' para gerenciar dados dos usuários armazenados em arquivos CSV. Com ele lemos e atualizamos dados facilmente, lidando com valores ausentes e convertendo tipos de dados, como CPF. Após a manipulação, salvamos os dados atualizados de volta nos arquivos CSV. Além disso, usamos arquivos TXT para armazenar faturas de cartões de crédito, realizando operações de leitura e escrita diretamente em Python. Essas técnicas garantem a manipulação e o tratamento de dados.

IV. Resultados

Neste tópico, iremos detalhar as principais aplicações do sistema e as validações de erro realizadas durante o desenvolvimento. Os resultados do projeto serão apresentados em duas subseções distintas. A primeira subseção abordará as aplicações práticas do sistema, enquanto a segunda subseção discutirá as validações de erro implementadas para garantir a plena funcionalidade do sistema.

A. Aplicações

As funcionalidades implementadas para os usuários do sistema incluem uma gama de operações financeiras e de gerenciamento. Tanto clientes quanto administradores podem realizar ações de login e logout. Os clientes têm acesso a funcionalidades específicas como depósitos, saques, transferências entre outras contas BliBank e investimentos. Eles também podem solicitar um cartão de crédito, registrar compras, pagar faturas e solicitar aumento de limite de crédito. No entanto, a aprovação de cartões de crédito, aumentos de limite e encerramento de contas é uma responsabilidade exclusiva dos administradores, garantindo um controle mais rigoroso sobre essas operações.

Os administradores possuem permissões adicionais para aprovar solicitações de cartões de crédito, ajustar limites de crédito e encerrar contas de clientes. Eles também podem visualizar informações detalhadas dos clientes para assegurar que todas as operações sejam realizadas de maneira segura e conforme as políticas do banco. Essa divisão clara de responsabilidades entre clientes e administradores garante a integridade e a segurança das operações dentro do BliBank.

Além das funcionalidades descritas, o BliBank oferece uma interface gráfica intuitiva. Proporcionando uma navegação fácil e acessível, permitindo que tanto clientes quanto administradores acessem rapidamente as funcionalidades disponíveis. A usabilidade aprimorada visa garantir uma experiência de usuário eficiente e agradável.

B. Cenários de Erro

Durante os testes, foram simulados diversos cenários de erro para garantir a robustez do sistema. A seguir, são apresentados alguns exemplos de erros detectados e suas mensagens correspondentes:

1) Tentativa de Login com Credenciais Inválidas: Ao tentar fazer login com credenciais inválidas, o sistema retorna a mensagem de erro presente na Figura 7.

2) Tentativa de Depósito com Valor Inválido: Ainda que a interface possua vários mecanismos para evitar que você insira um número incorreto, caso isso ocorra, será apresentada a mensagem de erro presente na Figura 10.

3) Tentativa de Saque com Saldo Insuficiente: Ao tentar fazer um saque com saldo insuficiente, o sistema retorna a mensagem de erro presente na Figura 11.

4) Tentativa de Transferência para Email Não Cadastrado: Ao tentar fazer uma transferência para um email não cadastrado, o sistema retorna a mensagem de erro presente na Figura 8.

5) Tentativa de aprovar cartão, encerramento de conta e aumento de limite: Quando o administrador tentar fazer uma aprovação para algum cliente e a solicitação não existir, o sistema retorna a mensagem de erro presente na Figura 9

Essas telas demonstram a facilidade de uso e a organização das funcionalidades do sistema, proporcionando uma experiência de usuário intuitiva e eficiente. Através dos testes

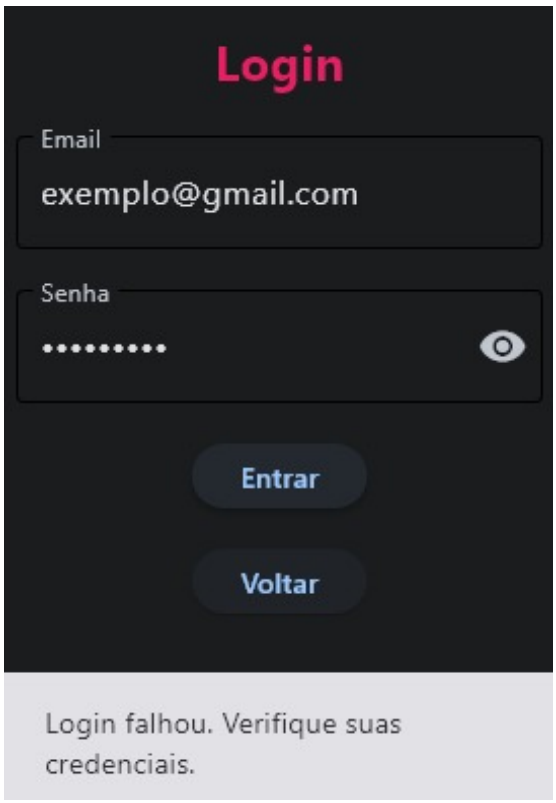

Fig. 7. Erro de login.

realizados, foi possível verificar que o sistema atende aos requisitos definidos e oferece uma plataforma segura e robusta para a realização de operações bancárias.

V. Conclusões

O desenvolvimento do BliBank demonstrou a eficácia da aplicação de conceitos de Programação Orientada a Objetos (POO) em um software de sistema bancário digital. As funcionalidades implementadas atenderam aos requisitos definidos, proporcionando uma experiência de usuário completa e segura. A utilização de herança, polimorfismo, interfaces, abstração e encapsulamento foi fundamental para a modularidade, extensibilidade do sistema e integridade de dados.

A modularidade do sistema facilita a manutenção e a expansão, permitindo que novas funcionalidades sejam incorporadas sem a necessidade de revisões significativas no código existente. A implementação de testes unitários foi crucial para identificar e corrigir erros, garantindo que o sistema funcione de maneira confiável em diversos cenários.

Estudos futuros podem explorar a integração de novos serviços e a otimização das operações financeiras. Além disso, a análise comparativa com outros sistemas bancários, como Nubank e Banco Inter, e o feedback dos usuários podem fornecer insights valiosos para melhorias contínuas.

Em conclusão, a construção do BliBank foi fundamental para fixar os conceitos estudados durante a matéria, além de ter proporcionado uma experiência prática de desenvolvimento de software para os estudantes. O conhecimento adquirido durante o desenvolvimento do BliBank pode servir como base para futuros projetos, contribuindo significativamente para nosso aprendizado.

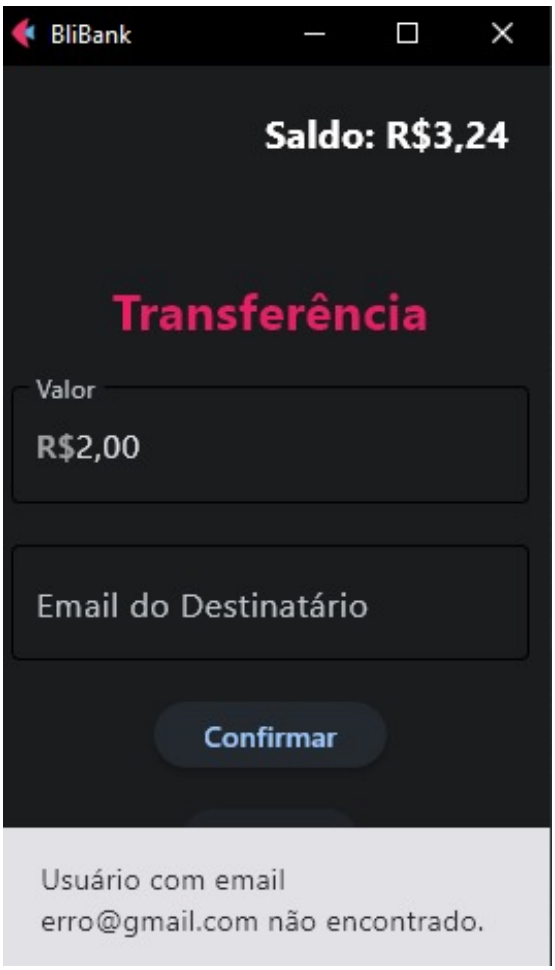

Fig. 8. Erro para transferência.

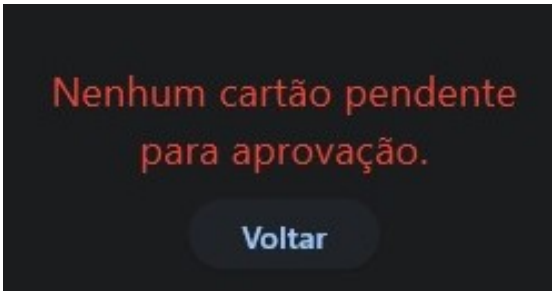

Fig. 9. Erro para aprovação de cartão.

Fig. 10. Erro de depósito.

Fig. 11. Erro de saque.

REFERENCES

[1] F. B. de Bancos FEBRABAN, "Brasileiro aumenta em 30% suas transações bancárias em 2022, e 8 em cada 10 operações são digitais," https://portal.febraban.org.br/noticia/3950/pt-br/, 2023, acessado: 18 de junho de 2024.

[2] I. B. de Geografia e Estatística IBGE, "Pesquisa nacional por amostra de domicÍlios contÍnua - pnad contÍnua," https://ftp.ibge.gov.br/Trabalho_e_Rendimento/Pesquisa_Nacional_por_Amostra_de_Domicilios_continua/Principais_destaques_PNAD_continua/2012_2021/PNAD_continua_retrospectiva_2012_2021.pdf, 2022, acessado: 18 de junho de 2024.

[3] P. S. Foundation, "Python 3.12.4 documentation," https://docs.python.org/3/, 2024, acessado: 18 de junho de 2024.

[4] I. de Pagamento Nu Pagamentos S.A, "Nubank," https://nubank.com.br/, 2024, acessado: 15 de maio de 2024.

[5] B. I. S.A., "Banco inter," https://www.bancointer.com.br, 2024, acessado: 15 de maio de 2024.

[6] L. G. Mendes, "Projeto de poo," https://github.com/LarianeMendes/Projeto-de-POO, 2024, acessado: 21 de junho de 2024.

[7] G. N. Lopes, "Poo," https://drive.google.com/drive/folders/1Av6FBH7SlNZeljknDFaI1v1jAwsQkR8s?usp=drive_link, 2024, acessado: 21 de junho de 2024.

[8] A. S. de Informática S.A Alura, "Python e orientaÇÃo a objetos," https://www.alura.com.br/apostila-python-orientacao-a-objetos, 2024, acessado: 21 de junho de 2024.

[9] C. S. F. Neves, "Minutos de desenvolvimento de sistemas: Abstração e encapsulamento na programação orientada a objetos," https://www.alura.com.br/apostila-python-orientacao-a-objetos, 2023, acessado: 21 de junho de 2024.

[10] A. S. Inc., "Flet," https://flet.dev/docs/, 2024, acessado: 15 de junho de 2024.