

**PROJETO INTERDISCIPLINAR**

# **Aplicativo de gestão financeira**

Rubik soluções financeiras

**Alunos:**

<b>RGM</b>	<b>Nome</b>
29682860	Matheus Ribeiro dos Santos
30074819	Samuel Marques de Oliveira
29672422	João Victor de Oliveira Silva
30154243	Vinícius Roberto Figueiredo

São Paulo

2023

UNIVERSIDADE CRUZEIRO DO SUL

PROJETO INTERDISCIPLINAR

# Aplicativo de gestão financeira

Rubik soluções financeiras

Trabalho apresentado como parte do requisito para aprovação na Disciplina de Projeto Interdisciplinar do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Cruzeiro do Sul.

**Orientadores:** Prof. Fabio Pereira da Silva e Prof. Agnaldo Silibert Mota

São Paulo

2023

## Sumário

1. Apresentação.....	4
1.1. Justificativa e Motivação .....	4
1.2. Dados do Sistema.....	4
1.3. Código .....	5
1.3.1. Tela de Login .....	5
1.3.2. Tela de cadastro.....	6
1.3.2. Saldo Bancário.....	6
1.3.3. Transferência .....	7
1.3.4. Histórico .....	8
1.3.5. Investimento.....	8
1.3.6. Atendimento.....	9
1.3.7. Banco de Dados .....	10
2. Requisitos de Técnica de Desenvolvimento de Algoritmos .....	11
2.1. Backlog .....	11
2.2. User Story.....	12
2.3. Design UI .....	13
3. Requisitos de Programação Orientada a Objetos.....	16
3.1. Requisitos Funcionais.....	16
3.2. Requisitos Não Funcionais .....	17
3.3. Regra de Negócio .....	19
3.4. Diagramas .....	20
4. Consideração Finais .....	21
5. Referências.....	22
APENSO 1 – Cronograma de entrega de atividades. ....	23

# **1. APRESENTAÇÃO**

## **1.1. Justificativa e Motivação**

O escopo principal deste projeto é criar um software de qualidade, para o qual a documentação é essencial. Será ofertada uma atenção especial para a criação de documentação abrangente e precisa, juntamente com a adoção de metodologias eficazes de gerenciamento de projetos.

Com a criação de uma documentação completa, será possível garantir uma visão geral clara e detalhada do projeto, incluindo as fases de desenvolvimento e análise de requisitos. Isso permitirá uma melhor colaboração entre a equipe envolvida, e ajudará a garantir que o produto final atenda às necessidades do cliente.

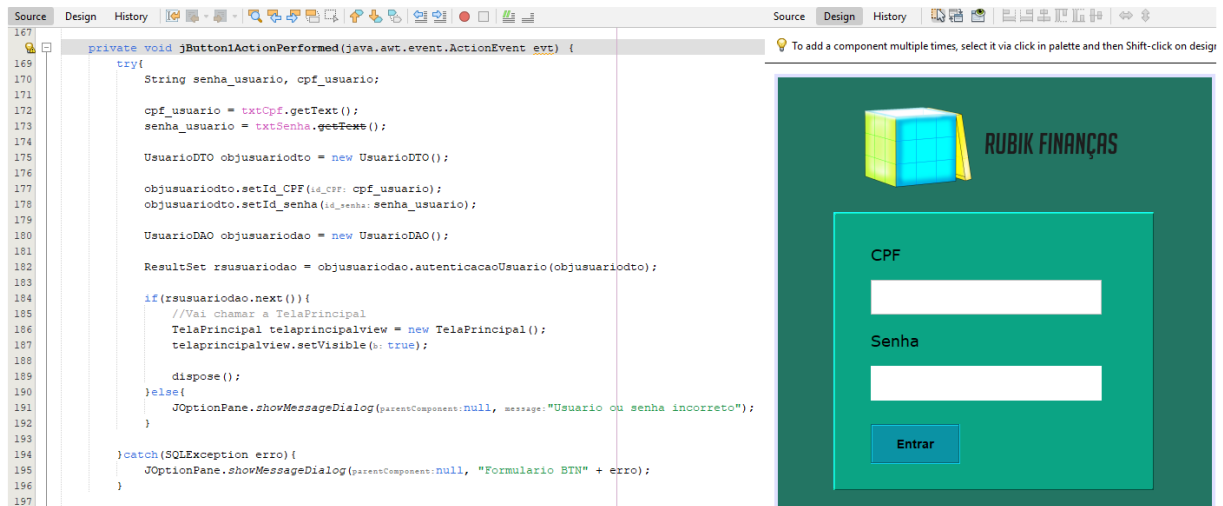
Em resumo, o objetivo principal deste projeto é criar um software de qualidade, com uma documentação completa e metodologias eficazes de gerenciamento de projetos. Isso irá nos proporcionar um entendimento imprescindível para um profissional atuante no ramo, garantindo que nós sejamos aptos a concluir com sucesso todas as fases que compõe a criação e estruturação de documentação de software, englobando às necessidades do cliente.

## **1.2. Dados do Sistema**

Um banco é essencial na atualidade, imprescindível para as atividades diárias da maior parte da população mundial, o mesmo provê uma gama de funções ágeis para manejo de renda e investimentos, tendo como pilar a segurança tanto física da moeda quanto digital. O aplicativo Rubik Finanças proporciona uma interação fácil e descomplicada de se usar, mantendo todos os métodos utilizados no mercado financeiro proporcionando investimentos seguros, preservação de dados, controle de gastos e transações financeiras.

## 1.3. Código

### 1.3.1. Tela de login



[Link](#)

A classe **JButton1ActionPerformed** é responsável por realizar quatro ações no código usando o método *try* e *catch*. Essas ações incluem a criação de duas variáveis do tipo *string*, como a **senha\_usuario**, para armazenar informações inseridas nos campos de entrada da **TelaLogin.java** (interface gráfica). A classe ([UsuarioDTO](#)) é utilizada para armazenar essas informações de forma segura, pois ela contém os *Getters* e *Setters* necessários para acessar a classe privada. A partir disso, um objeto **objusuariodto** é criado e recebe as informações das variáveis criadas no início da classe, como a **senha\_usuario**. O objeto **objusuariodto** armazena essas variáveis para comparar com o banco de dados.

A classe ([UsuarioDAO](#)) é responsável por se comunicar com o banco de dados e recebe as informações passadas pelo *Setter* do objeto **objusuariodto** para compará-las com a base de dados correspondente. Por fim, a função de condição *if* e *else* é utilizada para chamar a ([TelaPrincipal.java](#)) e fechar a **TelaLogin.java**, caso a classe **UsuarioDAO** retorne informações correspondentes. Caso contrário, a condição se encaixa no *else*, exibindo uma mensagem de erro informando que "usuário ou senha incorreto".

### 1.3.2. Tela de cadastro

O método **CadastrarUsuario** inclui uma string SQL que será usada no PreparedStatement para inserir os dados fornecidos pelo usuário na interface gráfica. Antes de solicitar os dados, é instanciada a classe **GeraNum** para gerar números de conta e agência aleatórios, que serão atribuídos aos dados mencionados anteriormente.

Na classe da interface gráfica, uma condição if/else é utilizada para impedir que menores de idade criem contas. Após o clique no botão "cadastrar", o processo é concluído e a tela é redirecionada de volta para a página de login.



[Link](#)

### 1.3.3. Saldo Bancário

O saldo bancário é obtido através da pesquisa usando métodos concentrados na classe **UsuarioDAO**. O método em questão chama-se **listarSaldo**, que registra a entrada do usuário na classe **UsuarioDAO/ArrayList PerfilSaldo**. Em seguida, o saldo é retornado para ser exibido na interface gráfica quando o usuário clica no botão "Saldo bancário".



[Link](#)

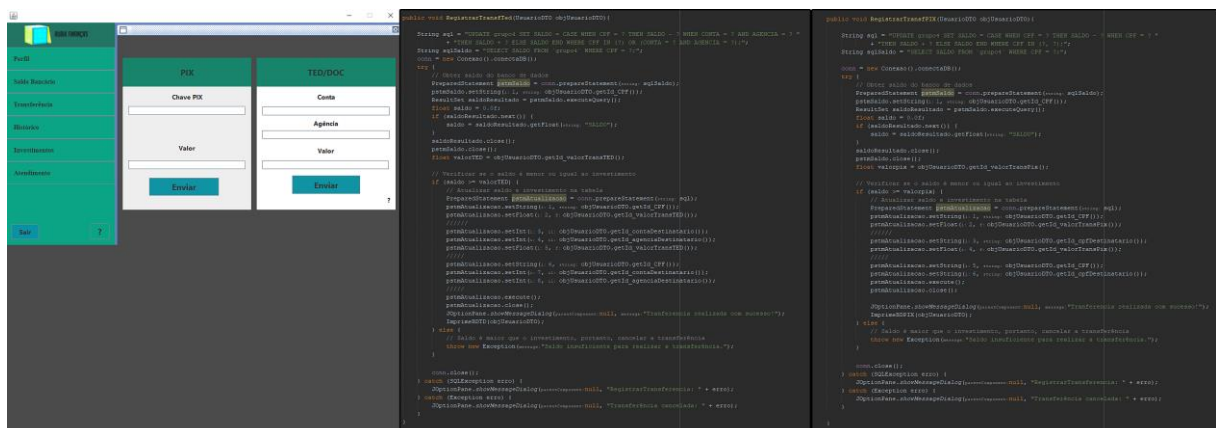
### 1.3.4. Transferência

Os métodos desenvolvidos para as funções de transferência são como irmãos gêmeos bivitelinos, possuindo características distintas. A diferença entre eles reside nas chaves de transferência utilizadas. Enquanto as transações por TED e DOC exigem as informações de conta e agência do destinatário, o Pix requer apenas o CPF.

Dentro dessa perspectiva, vou explicar o método mais curto, que é o Pix. Nesse método, foram criadas duas strings com sintaxe SQL. A primeira string, denominada sqlSaldo, tem a finalidade de buscar no banco de dados o saldo atual da conta. Já a segunda string, chamada sql, é responsável por atualizar os dados no banco de dados.

Dentro do bloco try, é criado o primeiro PreparedStatement, que utiliza a string sqlSaldo para obter e armazenar o valor do saldo atual em uma variável do tipo Float denominada "saldo". Em seguida, o valor inserido pelo usuário na interface gráfica, no campo "Valor" do PIX, é armazenado em uma variável Float chamada "valorPix".

Posteriormente, é iniciada uma estrutura condicional If/Else, que verifica se o valor armazenado na variável "saldo" é maior ou igual ao valorPix. Caso essa condição seja atendida, o programa continua para o próximo PreparedStatement, que irá processar a string sql e exibir a mensagem "Transferência realizada com sucesso!" na tela. No caso de a condição não ser atendida, o programa retorna um erro, exibindo a mensagem "Saldo insuficiente para realizar a transferência".



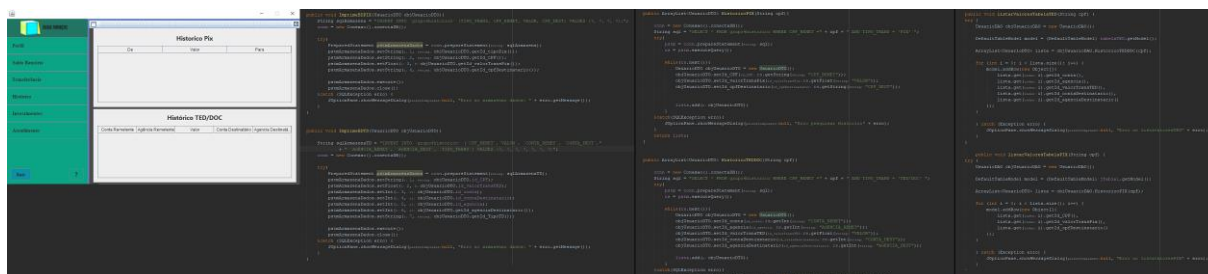
[Link](#)

### 1.3.5. Histórico

O método desenvolvido para o histórico aproveita-se de outros métodos utilizados durante os processos de transferência. Esses métodos são chamados **ImprimeBDPIX** e **ImprimeBDTD**, os quais armazenam as informações das transferências em um novo esquema (schema) no banco de dados chamado "**grupo4historico**". Portanto, o método que invoca essas informações é utilizado para criar um `ArrayList` no histórico, que será responsável por armazenar as informações a serem exibidas em suas respectivas tabelas.

Além disso, dentro da interface, foi criado um método para cada tabela, no qual um laço de repetição é utilizado para imprimir as informações e pular automaticamente para uma nova linha a cada linha preenchida.

Essas melhorias visam garantir que o histórico seja atualizado e exibido corretamente, proporcionando uma organização adequada das informações no banco de dados e uma representação clara na interface do usuário.



[Link](#)

### 1.3.6. Investimento

O método utilizado na aba de investimentos pode ser comparado aos métodos de transferência, porém, os dados são armazenados no mesmo esquema de informações, como CPF e saldo, facilitando seu acesso. Portanto, esse método utiliza sintaxes simples de SQL para realizar atualizações e consultas no banco de dados.

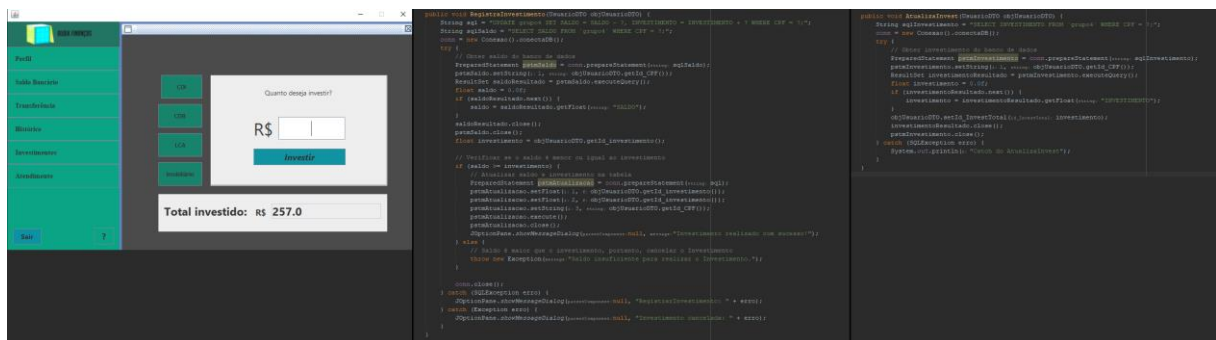
Da mesma forma que o método **RegistraTransfPix**, o método de investimento registra o saldo atual no banco de dados em uma variável do tipo `float` chamada



"saldo". Em seguida, prepara uma variável float chamado "investimento", que recebe a entrada do usuário na interface gráfica.

Em seguida, o método entra em uma estrutura condicional If/Else que verifica se a variável "saldo" é maior ou igual à variável "investimento". Se essa condição for atendida, o programa executa o **PreparedStatement**, que utiliza a string sql para atualizar os dados no banco de dados e exibir a mensagem "Investimento realizado com sucesso!" na tela. Caso a condição não seja atendida, o programa retorna a mensagem "Saldo insuficiente para realizar o investimento".

Além disso, o método **AtualizaInvest** é utilizado para recuperar o total investido no sistema e exibi-lo na interface gráfica. Esse método garante que as informações do investimento sejam atualizadas e apresentadas de forma correta, permitindo uma visualização adequada do total investido no sistema.

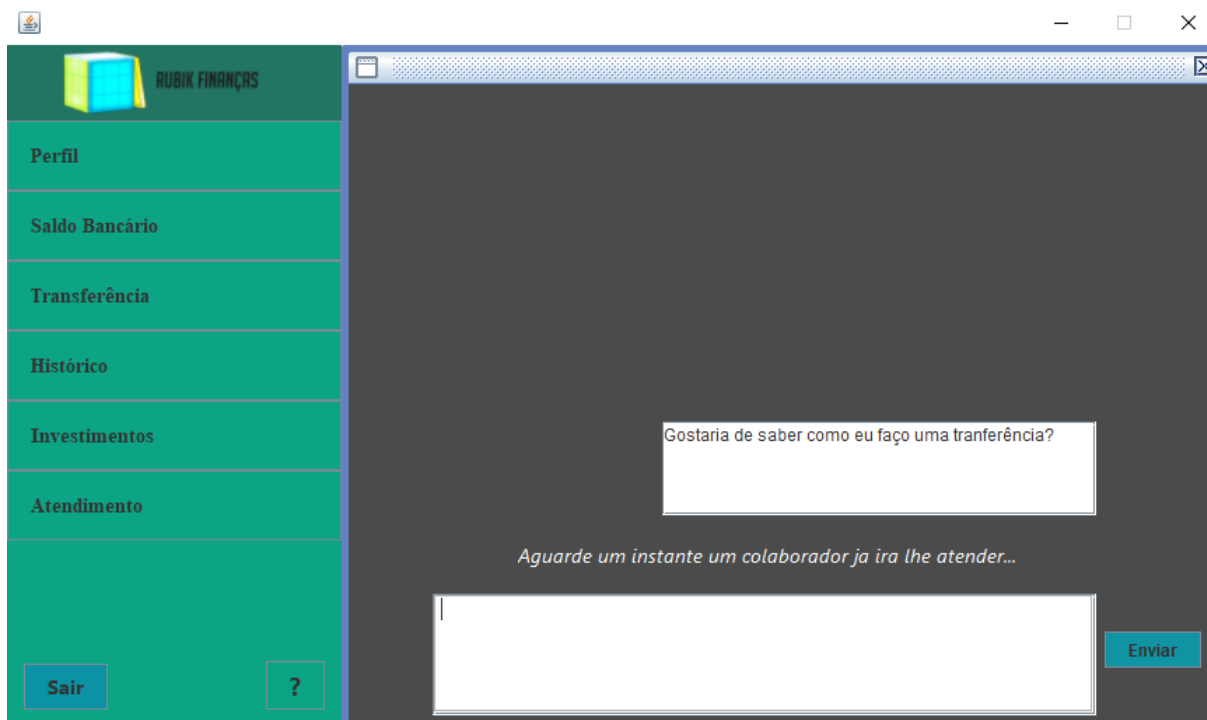


[Link](#)

### 1.3.7. Atendimento

O atendimento funciona como um espaço reservado (placeholder) para uma possível implementação de atendimento online. Essa funcionalidade foi incluída no sistema visando a futura expansão do software para oferecer suporte e interação em tempo real com os usuários.

Embora ainda não esteja totalmente implementado, o módulo de atendimento foi projetado para possibilitar o suporte ao cliente por meio de um chat online ou de recursos similares. Isso permitirá que os usuários obtenham assistência personalizada, tirem dúvidas e recebam orientações em tempo real.



### 1.3.8. Banco de dados

O uso do banco de dados desempenhou um papel fundamental na criação deste projeto, sendo sua presença crucial para a maior parte do nosso software. Para garantir a eficiência e organização das informações, foram criados dois esquemas (schemas) distintos.

O primeiro esquema, denominado "**grupo4**", contém informações essenciais do cliente, como CPF, senha, e-mail, entre outros dados relevantes para a operação do software. Esse esquema é responsável por armazenar e gerenciar os detalhes pessoais dos clientes de forma segura e confiável.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
<input type="checkbox"/>	1 CPF	varchar(11)	utf8mb4_general_ci		No	None		
<input type="checkbox"/>	2 SENHA	varchar(255)	utf8mb4_general_ci		No	None		
<input type="checkbox"/>	3 NOME	varchar(255)	utf8mb4_general_ci		No	None		
<input type="checkbox"/>	4 AGENCIA	int(11)			No	None		
<input type="checkbox"/>	5 CONTA	int(11)			No	None		
<input type="checkbox"/>	6 EMAIL	varchar(255)	utf8mb4_general_ci		No	None		
<input type="checkbox"/>	7 SALDO	float			No	None		
<input type="checkbox"/>	8 INVESTIMENTO	float			No	None		







Além disso, foi desenvolvido um segundo esquema especificamente para armazenar as transações bancárias, excluindo o investimento. Essa separação permitiu uma melhor administração dos dados, simplificando a consulta e o rastreamento das transações realizadas pelos clientes. Dessa forma, garantimos uma organização eficiente das informações financeiras e a integridade dos registros bancários.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
<input type="checkbox"/> 1	CPF_REMET	varchar(11)	utf8mb4_general_ci		Yes	NULL		
<input type="checkbox"/> 2	VALOR	float			Yes	NULL		
<input type="checkbox"/> 3	CPF_DEST	varchar(11)	utf8mb4_general_ci		Yes	NULL		
<input type="checkbox"/> 4	CONTA_REMET	int(11)			Yes	NULL		
<input type="checkbox"/> 5	CONTA_DEST	int(11)			Yes	NULL		
<input type="checkbox"/> 6	AGENCIA_REMET	int(11)			Yes	NULL		
<input type="checkbox"/> 7	AGENCIA_DEST	int(11)			Yes	NULL		
<input type="checkbox"/> 8	TIPO_TRANS	varchar(11)	utf8mb4_general_ci		Yes	NULL		

## 2 REQUISITOS DE ENGENHARIA DE SOFTWARE

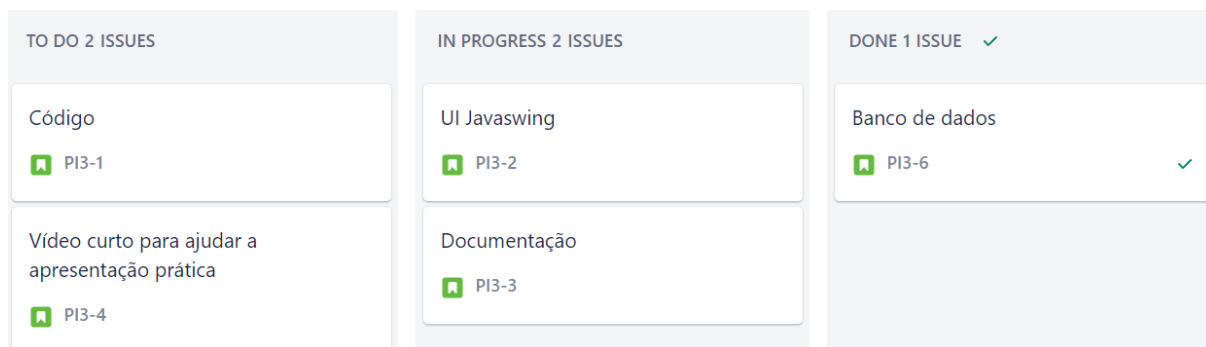
### 2.1. Backlog

Para aprimorar a organização das funções em nosso grupo, adotamos a metodologia ágil de backlog. Reconhecendo que cada membro possui habilidades e desafios distintos, implementamos um sistema de familiarização com o assunto para identificar as funções mais adequadas para cada um no projeto.

 PI3-1 Código  <a href="#">+ Epic</a>
 PI3-2 UI Javaswing
 PI3-3 Documentação
 PI3-4 Vídeo curto para ajudar a apresentação prática
 PI3-6 Banco de dados

Adotamos uma abordagem mais flexível, permitindo que o grupo mudasse o foco conforme necessário para oferecer assistência mútua ao membro que está com dúvidas ou necessitando de assistência.

Com essa abordagem ágil, pudemos melhorar a eficiência e eficácia do trabalho em equipe, garantindo que cada membro do grupo contribuísse de forma mais efetiva para o projeto. Além disso, a flexibilidade do sistema de backlog nos permitiu adaptar o andamento do projeto às mudanças nas necessidades e circunstâncias.



## 2.2. User Story

Como um aplicativo de finanças, é fundamental que possua a capacidade de gerenciar o saldo bancário dos usuários de forma eficiente, permitindo-lhes visualizar suas transações, consultar extratos e realizar operações bancárias, como transferências e pagamentos.

Além disso, um recurso essencial para atender às necessidades dos usuários é oferecer a opção de investir em ativos financeiros, tais como ações, títulos e fundos mútuos. Isso pode ser feito através da integração com corretoras de valores, permitindo que os usuários gerenciem seus investimentos diretamente do aplicativo.

Em resumo, um aplicativo de finanças bem projetado deve oferecer uma ampla gama de recursos que permitam aos usuários gerenciar suas finanças de forma eficiente, incluindo o gerenciamento do saldo bancário, a realização de operações bancárias e a possibilidade de investir em ativos financeiros.

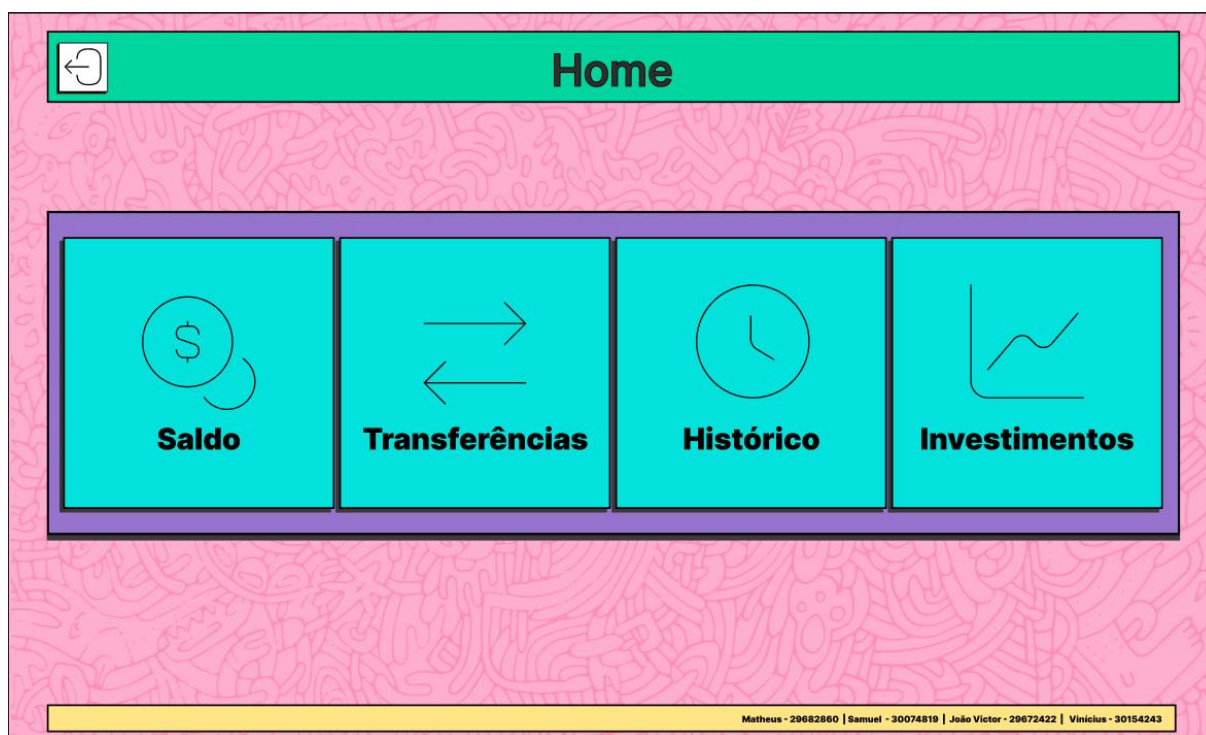
O atendimento ao cliente é uma parte crucial do sucesso de qualquer empresa e, portanto, deve ser fácil e eficiente para o usuário. É essencial que o aplicativo de finanças ofereça uma interface de comunicação clara e objetiva, para que os usuários

possam facilmente entrar em contato com a equipe de suporte em caso de dúvidas ou problemas.

Para garantir uma resolução rápida e eficaz, o aplicativo deve disponibilizar um sistema de chat, que permita aos usuários conversar diretamente com um colaborador. Isso deve ser feito de forma ágil e com respostas precisas, para que o usuário sinta que está sendo bem atendido e suas necessidades estão sendo atendidas.

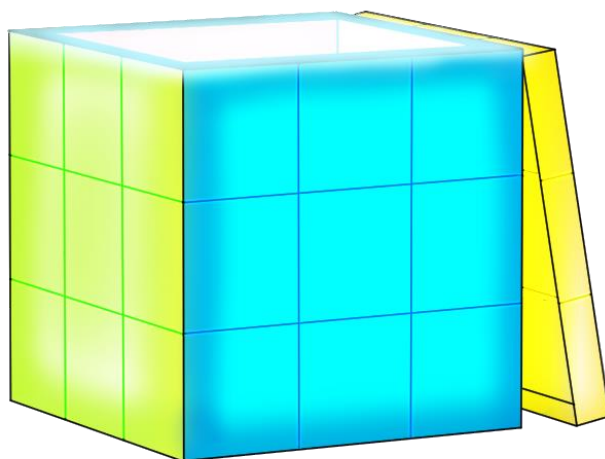
### 2.3. Design UI

O processo de idealização da interface gráfica passou por duas iterações, nas quais a interface foi descartada e refeita. O primeiro conceito apresentava um aspecto informal e desorganizado, e o design em si excedia as capacidades do próprio JavaSwing, o que levou à sua rápida eliminação.



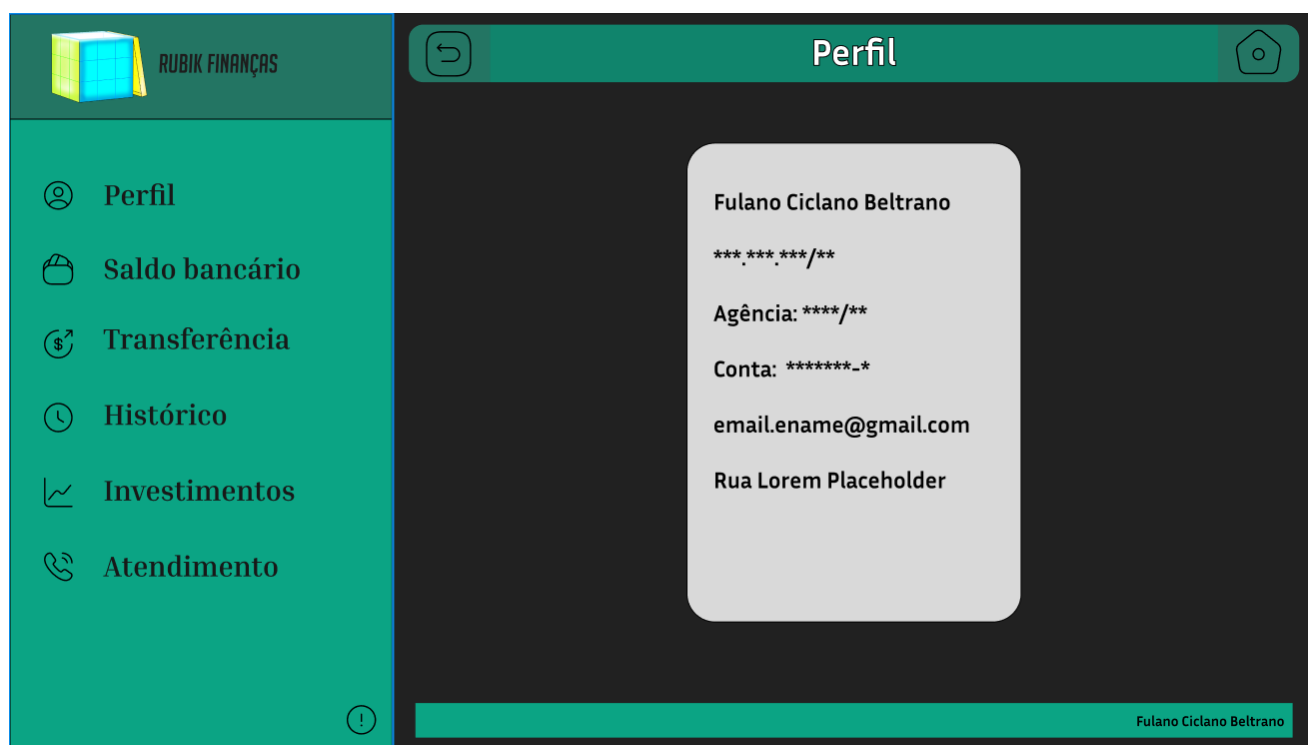
Dessa forma, decidimos por uma espécie de rebranding iniciando pela concepção do logotipo da marca. Além de suas competências básicas, como segurança e operações bancárias, um aplicativo bancário deve oferecer facilidades para o dia a dia dos clientes.

Para isso, optamos por utilizar o icônico cubo mágico (Rubik's cube) como símbolo do nosso banco. Entretanto, ao invés de apresentá-lo em sua forma tradicional, optamos por representá-lo com suas faces finalizadas e seu topo aberto, com uma luz se espalhando, como uma alegoria às soluções proporcionadas pelo nosso aplicativo.

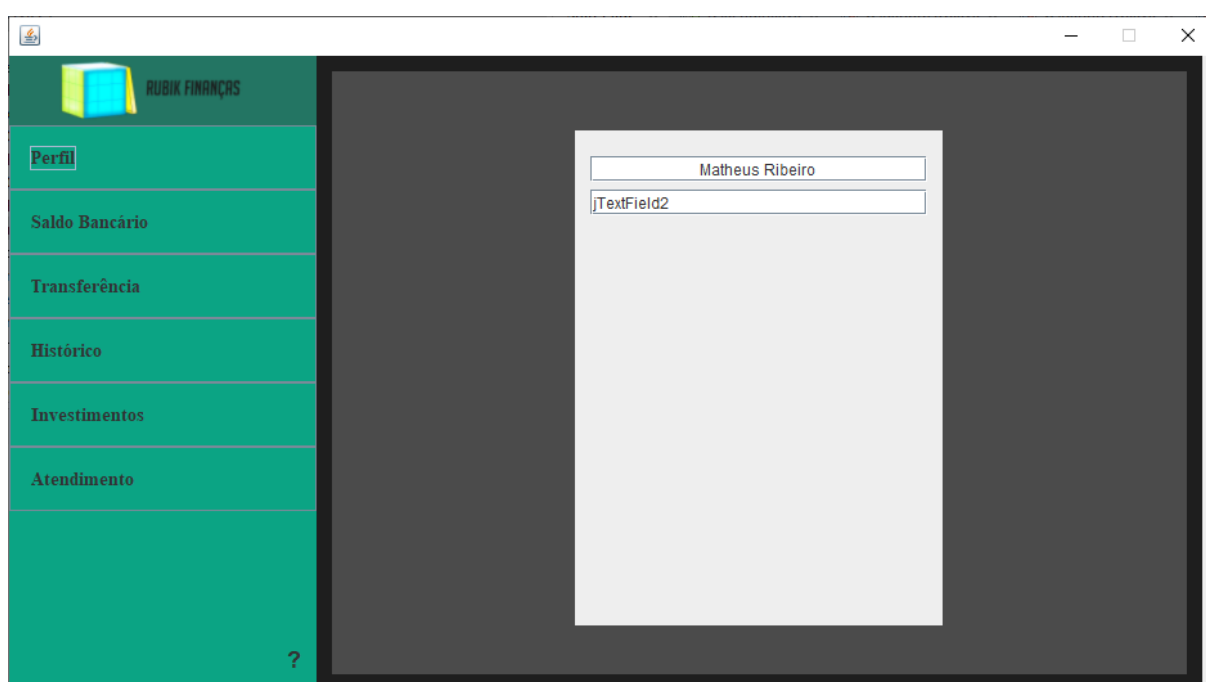


Após a conclusão do processo de rebranding, iniciamos o trabalho na nova interface gráfica. Dois aspectos foram considerados durante o processo de design: facilidade de navegação e sensibilidade à luz nos olhos, uma vez que a apresentação seria realizada em sala de aula. Para atender a esses requisitos, optamos por utilizar cores geralmente bem aceitas, com paletas condizentes.

A escolha de tons de cinza e verde predominantes contribuiu para uma interface de fácil visualização, com destaque para a usabilidade e eficiência do nosso aplicativo bancário. Além disso, foram incorporados outros elementos, tipografia clara e adequada, bem como a disposição adequada dos elementos gráficos para garantir a melhor experiência do usuário possível.



No entanto, um obstáculo na criação da interface persistia devido às limitações do JavaSwing, que tornavam o concept superior à interface. Diante dessa situação, optamos por disponibilizar todos os elementos da interface, porém com menos incrementos visuais, a fim de garantir a melhor experiência possível ao usuário dentro das capacidades do JavaSwing. Buscamos, assim, encontrar um equilíbrio entre a estética e a funcionalidade, sem comprometer a eficiência do aplicativo bancário. Dessa forma, foram feitas adaptações e ajustes necessários para garantir a adequação do design aos recursos disponíveis, sem prejudicar a qualidade da experiência do usuário.



### 3. Requisitos de Análise e Projeto de Sistemas

#### 3.1. Requisitos Funcionais

- O programa deve realizar operações de bancos.
- O programa deve realizar transferências.
- O programa deve realizar DOC/TEDS.



- O programa deve realizar PIX.
- Deve ter um repositório de usuários.
- Deve oferecer um histórico de todas transações e depósitos em conta.
- Registro de usuários: permite que os usuários criem uma conta na plataforma e acessem os serviços oferecidos pela startup.
- A aplicação financeira deve conter integração com sistemas bancários para operações.
- Deve ter suporte via chat.
- Suporte ao cliente: oferece suporte ao cliente em tempo real por meio de chat ou outras formas de comunicação.

### **3.2. Requisitos Não Funcionais**

- Os clientes confiam em startups para proteger suas informações pessoais, incluindo dados financeiros e outros dados sensíveis. A startup deve garantir que os dados do cliente sejam armazenados de forma segura, protegidos contra violações de segurança e acessíveis apenas para pessoal autorizado.
- A arquitetura da startup deve ser escalável para lidar com um aumento na demanda de clientes, sem afetar a qualidade do serviço ou a velocidade de resposta do sistema. O objetivo é garantir que o software possa acomodar um grande número de usuários simultâneos sem afetar negativamente a experiência do usuário.
- Os clientes esperam uma experiência de usuário intuitiva e fácil de usar. A startup deve garantir que seus serviços sejam fáceis de usar e navegar, com interfaces claras e intuitivas.

- Startups de confiança precisam ser flexíveis e capazes de se adaptar rapidamente a mudanças no mercado, demandas dos clientes e novas tecnologias. É importante que a startup possa lançar rapidamente novos recursos e serviços, bem como se adaptar a mudanças regulatórias e tendências de mercado, para atender às necessidades dos clientes e se manter competitiva.
- Um banco digital deve cumprir as regulamentações financeiras e as leis locais e internacionais. Isso pode incluir conformidade com a Lei Geral de Proteção de Dados (LGPD), padrões de segurança cibernética e outras normas regulatórias.
- A startup deve fornecer um serviço confiável e preciso para os usuários, com transações financeiras precisas e registros de dados precisos.
- Deve ter um banco de dados integrado.
- Deve conter uma tela de login com CPF e senha.
- Deve ter uma UI principal (Home) com todas as opções.
- O software deve prover de uma interface com janela fixa, para melhor se adaptar aos monitores de diversas resoluções, usando por padrão **1366 X 768** (WXGA).
- O software necessariamente precisa ter especificações de uso baixas tendo em mente a média de processadores no mercado de computadores empresariais e pessoais (i5-2400).
- O programa deve ser rápido ao apresentar os conteúdos com média de 3 segundos.

### 3.3. Regra de Negócio

Regras de negócio de uma startup são as diretrizes que definem o funcionamento do negócio, orientando as decisões e ações da empresa em relação aos seus objetivos e estratégias. Essas regras podem incluir aspectos como o modelo de negócio, a segmentação de mercado, as políticas de preços, as estratégias de marketing, as parcerias com outras empresas, as políticas de privacidade e segurança de dados, entre outros.

As regras de negócio são fundamentais para o sucesso de uma startup, pois ajudam a orientar as decisões da empresa e a garantir que ela esteja alinhada com seus objetivos e valores.

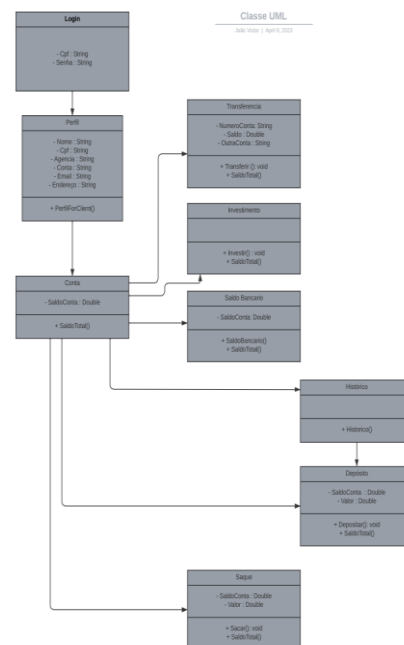
- TED e DOC, deveram seguir as regras das organizações bancárias.
- TED se for efetuado até 17 horas o saldo cairá na conta do destinatário mesmo dia.
- DOC o depósito é efetuado no dia posterior a realização da operação, podendo ter um atraso caso a mesma seja feita após as 22 horas do dia anterior.
- Ambas as operações TED e DOC seguem a mesma requisição de dados, sendo elas Nome completo, CPF ou CNPJ do destinatário, Tipo de conta (corrente ou poupança) e dados bancários.
- Caso tenha alguma incongruência nas informações ou mesmo algum problema relacionado servidor bancário a operação será cancelada e retornará o saldo debitado da conta no dia posterior à realização da mesma.
- As taxas de operações de TED / DOC variam entre instituições, a Rubik não adiciona taxas em transações de nenhuma natureza.
- A área de investimentos disponibilizada deve seguir as **Regras e parâmetros para intermediação – Resolução CVM 35/2021**.

- O atendimento deve ser realizado por um atendente remoto via chat ou por ligação.

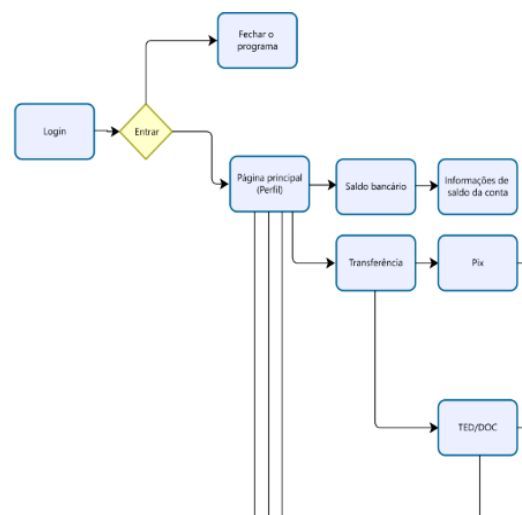
### 3.4. Diagramas

O **diagrama de classes** é uma ferramenta fundamental na modelagem de sistemas de software orientados a objetos, já optamos por um software na linguagem Java. Eles ajudam a representar visualmente as classes de objetos envolvidas no sistema e as relações entre elas.

Nesse sentido, optamos por utilizar diagramas de classes como forma de auxiliar no processo de criação do software em questão. Para isso, elaboramos um diagrama completo que pode ser acessado de forma mais detalhada a imagem ao lado por meio deste [link](#) em nosso Google Drive.



O **diagrama de regras de negócio** é uma ferramenta que ajuda a garantir que os requisitos do cliente sejam devidamente compreendidos e atendidos no desenvolvimento de um aplicativo. Ele funciona como uma representação visual das regras de negócio, indicando como o sistema deve funcionar e como seus diversos componentes devem se interligar.



Por meio de rotas traçadas de forma gráfica, o diagrama de regras de negócio serve como um guia claro e acessível para o uso do software. Ele ajuda a identificar potenciais problemas ou conflitos nas regras de negócio definidas, bem como a validar e verificar se todas as funcionalidades requisitadas pelo cliente foram implementadas de acordo com o planejado.

Assim, o diagrama de regras de negócio é uma ferramenta essencial para garantir que o desenvolvimento do software ocorra de maneira assertiva, garantindo a satisfação do cliente e a qualidade do produto final. Segue o [link](#) para melhor visualização do arquivo de imagem disposto ao decorrer da explicação.

#### **4. CONSIDERAÇÃO FINAIS**

De maneira geral, nos deparamos com diversas dificuldades ao longo do projeto. Inicialmente, um dos maiores desafios foi a limitação do JavaSwing no desenvolvimento da interface gráfica. Posteriormente, o maior obstáculo foi a criação e integração do banco de dados. Tivemos dificuldade em determinar qual banco de dados utilizar, mas, após discussões em grupo, decidimos utilizar o MySQL por meio do XAMPP.

No que se refere à codificação, em geral, não tivemos dificuldades no início. No entanto, encontramos dificuldades ao finalizar a geração da transferência e do histórico. No caso da transferência, o problema ocorreu durante o cálculo, pois era feita uma subtração no saldo do remetente e um aumento no saldo do destinatário. O banco de dados não estava sendo atualizado corretamente, pois a operação de "Update" não estava sendo executada.

Uma nova dificuldade surgiu após a transferência, relacionada ao histórico. Após realizar a transferência, era necessário armazenar o valor no nosso banco de dados, juntamente com os dados do remetente e do destinatário. Para isso, foi necessário utilizar um ArrayList para armazenar o valor da transferência no banco de dados, permitindo assim registrar as transferências no histórico.

Embora os métodos tenham apresentado um problema simples, este perdurou por mais tempo que o previsto, cerca de uma semana e meia. Tal contratempo deveu-se a uma variável que não armazenava fixamente as informações, em razão da chamada dos Jframes. Ao clicar em um botão dentro da interface, esta invocava um novo objeto, o qual acabava por desvincular a classe UsuarioDTO, fazendo com que a mesma perdesse seus atributos.

## 5. BIBLIOGRAFIA

**Autenticação de Usuário - Login JAVA com MYSQL PARTE 1/2.** Youtube.

Disponível em: <https://www.youtube.com/watch?v=XxWV-RoJKS0&feature=youtu.be>. Acesso em: 29 abr. 2023.

**Autenticação de Usuário - Login JAVA com MYSQL PARTE 2/2.** Youtube.

Disponível em: <https://www.youtube.com/watch?v=N85H9rT63Vc>. Acesso em: 29 abr. 2023.

**Aula 6 - Diagrama de classes.** BlackBoard. Disponível em:

*Princípios de Análise e Projeto de Sistemas com UML*. Acesso em: 17 abr. 2023.

**[JAVA + NETBEANS] - (Aula 1) Interface Gráfica - JFrame, JLabels, JButtons e JTextFields.** Youtube. Disponível em:

<https://www.youtube.com/watch?v=9t9qTiSpE40>. Acesso em: 20 abr. 2023.

**CRUD - JAVA com MYSQL - Pesquisar (listar).** Youtube. Disponível em:

[https://www.youtube.com/watch?v=b5RZkC\\_qVVI&list=PLA177te8KCzejCXMA\\_Jd1sJU9pw-utKJ\\_&index=5](https://www.youtube.com/watch?v=b5RZkC_qVVI&list=PLA177te8KCzejCXMA_Jd1sJU9pw-utKJ_&index=5). Acesso em: 30 março 2023.

**Como Carregar Valores de uma TABELA (componente JTABLE). Setar Campos.** Youtube. Disponível em: [https://www.youtube.com/watch?v=Qcee-xNuWwC&list=PLA177te8KCzejCXMA\\_Jd1sJU9pw-utKJ\\_&index=6](https://www.youtube.com/watch?v=Qcee-xNuWwC&list=PLA177te8KCzejCXMA_Jd1sJU9pw-utKJ_&index=6). Acesso em: 30 março 2023.

**Link do arquivo no Github -** <https://github.com/mtrzx/Ultimo>

## APENSO 1 – CRONOGRAMA DE ENTREGA DE ATIVIDADES.

#	Descrição	Data		Prazo do cronograma em semanas																	
		Início	Término		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Definição do tema e planejamento inicial	23/03	27/03	P	X																
				R	X																
2	Backlog	08/04	22/04	P				X													
				R			X														
3	Diagramas	08/04	22/04	P				X													
				R			X														
4	UI/Banco de dados	30/03	26/04	P					X												
				R					X												
5	Codificação	30/03	06/05	P					X												
				R						X											
6	Documentação do projeto	07/04	10/05	P						X											
				R						X											
7	Entrega do projeto final e apresentação	15/05	19/05	P								X									
				R								X									

### OBS:

1) **P** = previsto; **R** = realizado