PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

CryptoBank - Api bancária para transações em cryptos

Diego Paredes Canellas

RELATÓRIO DE PROJETO FINAL DE GRADUAÇÃO

CENTRO TÉCNICO CIENTÍFICO - CTC
DEPARTAMENTO DE INFORMÁTICA
Curso de Graduação em Ciência da Computação

Rio de Janeiro, julho de 2024

Diego Paredes Canellas

CryptoBank - Api bancária para transações em cryptos

Relatório de Projeto Final, apresentado ao Curso de Ciência da Computação da PUC-Rio como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Edmundo Bastos Torreão

Departamento de Informática

Rio de Janeiro, julho de 2024

Resumo

Canellas Diego, Paredes. Torreão Edmundo, Bastos. CryptoBank : Api bancária para transações em cryptos. Rio de Janeiro, 2024. Relatório de Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este projeto visa desenvolver uma API para um banco que permita aos clientes realizar transações usando Ethereum, uma criptomoeda popular, além das transações padrões de um banco convencional, como empréstimos, saques, contas correntes e transferências. Oferecendo aos clientes uma maneira conveniente e segura de se envolverem com criptomoedas enquanto acessam os serviços bancários tradicionais. O objetivo é proporcionar ao banco uma vantagem competitiva, aproveitando o crescente interesse em moedas digitais e abrindo novas oportunidades de negócios.

Palavras-Chave

Ethereum, Criptomoedas, API Bancária, Transações Financeiras, Integração Bancária, Mercado Financeiro.

Abstract

Canellas Diego, Paredes. Torreão Edmundo, Bastos. CryptoBank : Api bancária para transações em cryptos. Rio de Janeiro, 2024. Relatório de Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This project aims to develop an API for a bank that allows customers to conduct transactions using Ethereum, a popular cryptocurrency, in addition to the standard transactions of a conventional bank, such as loans, withdrawals, checking accounts, and transfers. Offering customers a convenient and secure way to engage with cryptocurrencies while accessing traditional banking services. The goal is to provide the bank with a competitive advantage by leveraging the growing interest in digital currencies and opening up new business opportunities.

Keywords

Ethereum, Cryptocurrencies, Banking API, Financial Transactions, Banking Integration, Financial Market.

Sumário

1. Introdução **1**
2. SItuação Atual **3**
3. Proposta e Objetivos do Trabalho **4**
4. Análise de requisitos do sistema **5**
4.1. Uso do termo "sistema" 5
4.2. Escopo do sistema 5
4.3. Requisitos Funcionais 5
4.4. Cadastrar Cliente 6
4.5. Abrir conta corrente 7
4.6. Consulta de saldo em conta corrente 8
4.7. Transferência/Saque 9
4.8. Pagamento 10
4.9. Solicitação de Empréstimo 11
4.10. Solicitar cartão 12
4.11. Quitar parcela do empréstimo 13
4.12. Utilizar cartão de débito 14
4.13. Utilizar cartão de crédito 15
4.14. Bloquear cartão 16
4.15. Emitir extrato 17
4.16. Simulação de empréstimo 18
4.17. Realização de empréstimo 19
4.18. Avaliar estoque 20
4.19. Compra de Ethereum com Real Brasileiro 21
4.20. Depósito 22
4.21. Diagrama de Classes 23
5. Design e Estrutura da API **24**
5.1. Estrutura de uma API 25
5.3. Injeção de Dependência 28
6. Implementação **29**
6.1. Entity Framework 30
6.2. JWT 32
7. Conclusão **33**
8. Implementações futuras **34**
9. Referências **35**

1. Introdução

Nos últimos anos, o setor bancário tem testemunhado uma transformação significativa impulsionada pela blockchain (por definição entende-se blockchain, como um mecanismo de armazenamento digital que armazena todas as informações de transações que são feitas na rede). Alinhado a essas transformações, dois termos passaram a se tornar comuns dentro do mercado financeiro, são eles, contratos inteligentes e aplicativos descentralizados (dapps).

Os contratos inteligentes são “programas de computador que executam regras pré-estabelecidas de forma automatizada que são executadas dada uma condição específica é atendida. Um exemplo é um empréstimo. Uma pessoa poderia emprestar valores para outra sem a necessidade de recorrer a uma entidade para garantir a transação – todos os termos e condições poderiam ser programados.”[1]

Entende-se por aplicativos descentralizados, aplicativos muito semelhantes a aplicações de computador tradicionais porém utilizando a estrutura da blockchain, ou seja não são centralizados em apenas uma empresa.[2]

O surgimento e a ascensão das criptomoedas, com destaque para o Ethereum, atualmente apenas atrás do bitcoin como a criptomoeda de maior valor de mercado, têm marcado significativamente o cenário financeiro global. “O Ethereum, diferenciando-se de outras criptomoedas por sua capacidade de suportar contratos inteligentes e aplicativos descentralizados”[3], surgiu como uma plataforma muito popular com aplicações que vão além de ser apenas uma forma de moeda digital. Com isso em mente, alguns bancos, atentos à evolução do mercado financeiro, estão explorando maneiras de integrar essa tecnologia em seus serviços tradicionais.

A complexidade e o potencial do Ethereum têm despertado o interesse dos bancos, que agora se veem diante de uma oportunidade única de expandir e aprimorar suas ofertas de serviços. Ao considerar a integração de criptomoedas em seus sistemas, os bancos tentam modernizar suas operações, além de oferecer aos clientes uma variedade maior de serviços financeiros. Além disso, a adoção do Ethereum pode melhorar a eficiência operacional dos bancos, reduzindo custos e

simplificando processos.

No entanto, essa integração não é isenta de desafios. Os bancos enfrentam questões complexas relacionadas à regulamentação, segurança e conformidade ao explorar o uso do Ethereum em suas operações. “As regulamentações para as criptomoedas variam amplamente de país para país e podem mudar rapidamente. Isso pode criar incertezas e desafios para as instituições financeiras empreendedoras que desejam trabalhar com criptomoedas.”[4] A necessidade de garantir a conformidade com os padrões regulatórios estabelecidos e proteger os ativos dos clientes é uma consideração fundamental nesse processo.

Apesar dos desafios, a integração do Ethereum nos serviços bancários representa um passo significativo em direção à modernização e inovação no setor financeiro. O Banco Central trata as integrações com as criptomoedas como prioridade para 2024, além de tratar suas inovações no mercado financeiro uma “nova era da economia digital”.[5]

Ao adotar essa tecnologia emergente, os bancos podem posicionar-se de forma mais competitiva em um ambiente cada vez mais digital e oferecer para seus clientes uma experiência financeira mais moderna, sofisticada e abrangente.

2. SItuação Atual

Recentemente o mercado de criptomoedas viu um avanço significativo, fazendo crescer os bancos que operam em criptomoedas e assets exclusivamente digitais, alguns nomes mais notáveis e conhecidos no mercado são a Kraken Financial e o banco Avanti ambos são pioneiros quando o assunto é criptomoedas e transações bancárias.

O Kraken Financial, por exemplo, recebeu recentemente uma carta bancária[20] pelo estado de Wyoming, que os permite prover serviços bancários como pagamentos, depósitos e empréstimos.[21] Já em contrapartida o banco Avanti é mais como uma ponte entre bancos digitais e bancos tradicionais pois oferecem serviços em dólar porém com suporte a criptomoedas.

Esses bancos se beneficiam da tecnologia blockchain para realizar suas transações com segurança, transparência e eficiência, além da descentralização conhecida pela blockchain.

Apesar de tudo, essas instituições bancárias ainda enfrentam desafios comuns conhecidos anteriormente como a volatilidade das criptomoedas, as dificuldades regulamentares além da pouca escalabilidade, porém continuam no caminho inovando e colaborando com os sistemas bancários tradicionais e prometem ainda mais inovações para o futuro.

Em resumo, o aumento do número de instituições que trabalham em criptomoedas vem evoluindo e se transformando dado o grande avanço e a explosão do mercado de cripto nos últimos anos, com as instituições pioneiras guiando o caminho mais bancos tendem a começar a aderir aos *assets* digitais e aos benefícios das criptomoedas em seus serviços.

3. Proposta e Objetivos do Trabalho

Este projeto visa criar uma RESTful API para facilitar transações bancárias utilizando a tecnologia blockchain. Embora o desenvolvimento de contratos inteligentes e a integração total com a rede Ethereum estejam fora do escopo deste trabalho devido a restrições de tempo das disciplinas de Projeto Final, será elaborada uma API robusta e escalável para processar transações de forma segura e eficiente. A API permitirá que os usuários realizem operações, como transferências, depósitos e consultas de saldo, de maneira transparente.

A API será projetada para ser de fácil integração, oferecendo endpoints bem definidos com entradas de dados simples e bem documentadas e retornos de dados em formato JSON para garantir uma experiência de usuário fluida. Embora não implementados neste trabalho como mencionado anteriormente, será implementada uma base sólida para futuras implementações, de contratos inteligentes e aplicativos descentralizados explorando os conceitos fundamentais da Ethereum.

A API será projetada com endpoints bem definidos para lidar com diferentes operações bancárias, como transferências, depósitos e consultas de saldo. Os endpoints serão acessíveis via HTTP, seguindo os princípios RESTful para garantir uma integração suave e uma comunicação eficiente entre o cliente e o servidor, além disso a arquitetura da API será modular seguindo os padrões da clean architecture (arquitetura limpa). O retorno de dados será em formato JSON, proporcionando uma estrutura flexível e fácil de interpretar para os desenvolvedores. Além disso, serão implementadas medidas de segurança como JWT e validações de dados.

A introdução desta API para transações bancárias utilizando Etherium tem o potencial de revolucionar o setor financeiro, oferecendo uma alternativa descentralizada e segura aos sistemas tradicionais. Além disso, a transparência e a imutabilidade oferecidas pela blockchain podem aumentar a confiança dos usuários e facilitar transações em larga escala.

4. Análise de requisitos do sistema

4.1. Uso do termo "sistema"

Entende-se, no âmbito deste relatório, que o termo 'sistema' refere-se especificamente ao serviço web desenvolvido. Este serviço web, caracterizado por sua autonomia funcional, com lógica de negócios, modelo de dados e capacidade de deploy independentes, é denominado 'sistema' para ressaltar sua complexidade e completude como uma solução de software. A utilização desta terminologia tem por objetivo facilitar a compreensão da amplitude e funcionalidade do serviço web em questão. A consistência no emprego deste termo ao longo do documento é essencial para garantir clareza e evitar ambiguidades.

4.2. Escopo do sistema

Ao definir o escopo, delimitamos as funcionalidades essenciais do software, evitando o acúmulo de recursos desnecessários que podem comprometer o tempo e a qualidade final do produto. A definição do escopo é um diálogo entre todos os envolvidos, além disso fazer uma boa definição de escopo é uma parte crucial do planejamento de um projeto, pois evita mudanças e alterações durante o andamento do trabalho.[6]

Com um escopo bem definido, o desenvolvimento se torna mais eficiente e previsível. É possível estimar prazos com maior precisão, gerenciar os recursos de forma otimizada e evitar surpresas desagradáveis no meio do caminho.

4.3. Requisitos Funcionais

De acordo com Craig Larman em "Applying UML and Patterns", os casos de uso representam os requisitos funcionais do sistema, delineando as funcionalidades essenciais[7].

Os mesmos estão representados abaixo.

4.4. Cadastrar Cliente

Nome:	Cadastro de usuários
Objetivo:	Permitir que o cliente se cadastre e crie uma conta
Atores:	Aplicação client da API - (Cliente)
Pré-condição:	
Pós-condição (cenário de sucesso):	Cliente cadastrado
Pós-condição (cenário de insucesso):	Cliente não cadastrado
Trigger:	Aplicação client da API (Cliente) envia a mensagem de “Realizar Cadastro”
Fluxo Principal:	1. A aplicação Client da API (Cliente) envia mensagem com os campos ‘Nome’, ‘E-mail', 'Senha', ‘Telefone’ e ‘CPF/CNPJ’ e ‘Endereço’ que é optativo para o cadastro. 2. O sistema valida os campos [RN1][RN2][E1][E2] 3. O sistema gera um número da conta do cliente 4. O sistema retorna mensagem de sucesso.
Fluxos Alternativos:	
Fluxos de Exceção	[E1] Há campos inválidos/não preenchidos. 1. O sistema retorna a mensagem ‘Há informações inválidas’. [E2] Transação de Inclusão com CPF/CNPJ já existente 1. O sistema retorna mensagem de CPF/CNPJ já existente
Regras de Negócio	[RN1] Um novo cliente não pode ter um CPF/CNPJ já existente. [RN2] O campo de CPF/CNPJ é numérico e deve seguir o formato da Receita Federal

4.5. Abrir conta corrente

Nome:	Abrir Contas correntes para clientes
Objetivo:	Permitir que o usuário abra uma conta corrente
Atores:	Aplicação client da API - (banco) e Aplicação client da API - (Cliente)
Pré-condição:	Cliente cadastrado, Usuário logado
Pós-condição (cenário de sucesso):	Usuário logado.
Pós-condição (cenário de insucesso):	Erro ao abrir conta
Trigger:	Aplicação client da API envia a mensagem “abrir conta corrente”
Fluxo Principal:	1. A aplicação Client da API envia o token JWT. 2. O sistema valida o token.[E1][RN1] 3. A aplicação Client da API habilita a função de conta corrente para o usuário, 4. O sistema informa as taxa mensal da conta 5. O sistema retorna mensagem de sucesso e um JSON com o número da conta corrente.
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. [E1] Token Inválido. 1. O sistema retorna a mensagem ‘Token inválido’.
Regras de Negócio:	[RN1] o Cliente deve fornecer um token JWT válido.

4.6. Consulta de saldo em conta corrente

Nome:	Consulta de saldo em conta corrente em cripto
Objetivo:	Consultar saldo em conta do usuário
Atores:	Aplicação Client da API (banco)
Pré-condição:	Usuário possui conta corrente; Usuário cadastrado; Usuário logado
Pós-condição (cenário de sucesso):	Sistema retorna saldo do usuário
Pós-condição (cenário de insucesso):	Erro ao verificar saldo
Trigger:	Aplicação client da API envia a mensagem “Consultar saldo”
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente e o token JWT. 2. O sistema valida o token.[E1][RN1] 3. O sistema retorna um JSON com o valor do saldo do usuário.
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Token Inválido. 1. O sistema retorna a mensagem ‘Token inválido’.
Regras de Negócio:	[RN1] o Cliente deve fornecer um token JWT válido.

4.7. Transferência/Saque

Nome:	Transferência em Cripto
Objetivo:	Transferência de fundos
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário possui conta corrente; Usuário logado
Pós-condição (cenário de sucesso):	Sistema transfere o valor
Pós-condição (cenário de insucesso):	Erro ao transferir
Trigger:	Aplicação client da API envia a mensagem "Transferir"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, valor, informações da conta que vai receber a transferência (conta destino) e token JWT. 2. O sistema valida as informações e o token.[E1][RN1][E3][RN3] 3. O sistema transfere o dinheiro para a conta destino. [E2][RN2]
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Conta destino inválida 1. O sistema retorna a mensagem ‘Conta inválida’. [E2] Saldo insuficiente para a transferência 1. O sistema retorna a mensagem ‘Saldo insuficiente’. [E3] Token inválido 1. O sistema retorna a mensagem ‘Token inválido’.
Regras de Negócio:	[RN1] O usuário deve possuir o valor informado [RN2] Conta destino deve ser válida [RN3] O Cliente deve fornecer um token JWT válido.

4.8. Pagamento

Nome:	Pagamento em cripto
Objetivo:	Pagamento de conta
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário possui conta corrente; Usuário logado
Pós-condição (cenário de sucesso):	Sistema paga o valor para a conta devida
Pós-condição (cenário de insucesso):	Erro ao pagar
Trigger:	Aplicação client da API envia a mensagem "Pagar"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, valor, informações do código numérico da conta e token JWT.[E1][RN1] 2. O sistema valida as informações e o token JWT.[E2][RN2][E3][RN3] 3. O sistema efetua o pagamento.
Fluxos Alternativos:	1. O sistema retira o dinheiro que o usuário possui. 2. O sistema gera uma dívida para o usuário com o valor faltante
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] O usuário não possui o valor necessário em conta. 1. O sistema retorna a mensagem ‘Saldo Insuficiente’. [E2] Código numérico inválido 1. O sistema retorna a mensagem ‘Código numérico inválido’. [E3] Token inválido 1. O sistema retorna a mensagem ‘Token inválido’.
Regras de Negócio:	[RN1] O cliente deve ter a quantia necessária em conta corrente. [RN2] O cliente deve fornecer um código numérico válido. [RN3] O Cliente deve fornecer um token JWT válido.

4.9. Solicitação de Empréstimo

Nome:	Solicitação de Empréstimo em cripto
Objetivo:	Empréstimo bancário
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário possui conta corrente; Usuário logado
Pós-condição (cenário de sucesso):	Sistema gera um empréstimo para o cliente
Pós-condição (cenário de insucesso):	Erro no empréstimo
Trigger:	Aplicação client da API envia a mensagem "Gerar empréstimo"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, valor do empréstimo, número de parcelas, informações do usuário e token JWT. 2. O sistema valida as informações e o token.[E1][E2][RN1][RN2] 3. O sistema calcula os juros 4. O sistema calcula o valor das parcelas que devem ser pagas com juros de acordo com a data de cada parcela. 5. O sistema informa valor dos juros 6. O sistema transfere o dinheiro para a conta.
Fluxos Alternativos:	
Fluxos de Exceção:	[E1] Valor excessivo para o total disponível para o cliente. 1. O sistema retorna a mensagem ‘Valor excede o permitido para este cliente’. [E2] Token inválido 1. O sistema retorna a mensagem ‘Token inválido’.
Regras de Negócio:	[RN1] Histórico da conta corrente deve ser compatível com valor de empréstimo solicitado [RN2] O Cliente deve fornecer um token JWT válido.

4.10. Solicitar cartão

Nome:	Solicitar cartão
Objetivo:	Solicitar cartão virtual
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário possui conta corrente; Usuário logado
Pós-condição (cenário de sucesso):	Sistema envia dados do cartão virtual para o cliente
Pós-condição (cenário de insucesso):	Erro no envio
Trigger:	Aplicação client da API envia a mensagem "Solicitar cartão"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, dados do usuário, tipo do cartão e limite disponível 2. O sistema valida as informações e o token.[E1][E2][E3][E4][RN1][RN2][RN3][RN4] 3. O sistema gera um cartão do tipo informado 4. O sistema envia um JSON com o número do cartão, data de validade e código de segurança.
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Usuário não pode solicitar cartão. 1. O sistema retorna a mensagem ‘Usuário não pode solicitar cartão. [E2] Tipo cartão inválido 1. O sistema retorna a mensagem ‘Tipo cartão inválido’. [E3] Token inválido 1. O sistema retorna a mensagem ‘Token inválido’. [E4] Limite insuficiente 1. O sistema retorna a mensagem ‘Limite Insuficiente’.
Regras de Negócio:	[RN1] Cartões de crédito devem ser concedidos para contas corrente respeitando os critérios vigentes do banco. [RN2] Os tipos devem ser crédito, débito ou ambos. [RN3] O Cliente deve fornecer um token JWT válido. [RN4] Cartões de débitos devem respeitar os limites do usuário.

4.11. Quitar parcela do empréstimo

Nome:	Quitar parcela empréstimo
Objetivo:	Quitar parcela de um empréstimo bancário
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário deve possuir empréstimo; Usuário logado; Usuário possui conta corrente
Pós-condição (cenário de sucesso):	Parcela de empréstimo do cliente quitado
Pós-condição (cenário de insucesso):	Parcela de empréstimo não quitado
Trigger:	Aplicação client da API envia a mensagem "Quitar parcela"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, dados do empréstimo, valor pago, valor restante, número da parcela e token JWT. 2. O sistema valida as informações e o token.[E3][RN3] 3. O sistema valida se todo o valor foi pago.[E1][RN1][E2][RN2] 4. O sistema envia mensagem de sucesso com a parcela paga.
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Valor excedente 1. O sistema retorna a mensagem ‘Valor excede o total do empréstimo’. [E2] Empréstimo quitado 1. O sistema retorna a mensagem ‘Empréstimo já foi quitado’. [E3]Token inválido 1. O sistema retorna a mensagem ‘Token inválido’.
Regras de Negócio:	[RN1] O valor pago não pode exceder o total da parcela/empréstimo [RN2] O usuário não pode quitar um empréstimo que já foi quitado. [RN3] O Cliente deve fornecer um token JWT válido.

4.12. Utilizar cartão de débito

Nome:	Utilizar cartão
Objetivo:	Utilizar cartão de débito
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário deve possuir cartão com a função débito; Usuário logado
Pós-condição (cenário de sucesso):	Cartão utilizado
Pós-condição (cenário de insucesso):	Erro na utilização do cartão
Trigger:	Aplicação client da API envia a mensagem "Utilizar cartão"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, dados do cartão, valor a pagar, banco destino e token JWT 2. O sistema verifica se o usuário tem saldo suficiente em sua conta corrente. [E1][RN1] 3. O sistema tira o valor da conta do usuário e faz o pagamento para a conta no banco destino. 4. O sistema envia uma mensagem de sucesso para o usuário.
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Saldo insuficiente 1. O sistema retorna a mensagem ‘Saldo insuficiente’.
Regras de Negócio:	[RN1] O usuário deve possuir saldo suficiente para a transação

4.13. Utilizar cartão de crédito

Nome:	Utilizar cartão crédito
Objetivo:	Utilizar cartão de crédito
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário deve possuir cartão com a função crédito; Usuário logado
Pós-condição (cenário de sucesso):	Cartão utilizado
Pós-condição (cenário de insucesso):	Erro na utilização do cartão
Trigger:	Aplicação client da API envia a mensagem "Utilizar cartão"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, dados do cartão e valor a pagar 2. O sistema verifica se o usuário tem limite suficiente em seu cartão. [E1][RN1] 3. O sistema gera valor a pagar na conta do cartão de crédito 4. O sistema envia uma mensagem de sucesso para o usuário.
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Limite insuficiente 1. O sistema retorna a mensagem ‘Limite insuficiente’.
Regras de Negócio:	[RN1] O usuário deve ter limite suficiente para realizar a transação.

4.14. Bloquear cartão

Nome:	Bloquear cartão
Objetivo:	Bloquear cartão
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário deve possuir cartão; Usuário logado
Pós-condição (cenário de sucesso):	Cartão bloqueado
Pós-condição (cenário de insucesso):	Erro ao bloquear cartão
Trigger:	Aplicação client da API envia a mensagem "Bloquear cartão"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente e dados do cartão 2. O sistema verifica se o cartão já está bloqueado. [E1][RN1] 3. O sistema bloqueia o cartão.
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Cartão bloqueado 1. O sistema retorna a mensagem ‘O cartão já foi bloqueado’.
Regras de Negócio:	[RN1] O cartão não pode estar bloqueado

4.15. Emitir extrato

Nome:	Emitir extrato
Objetivo:	Emitir um extrato bancário
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário deve possuir conta corrente; Usuário logado
Pós-condição (cenário de sucesso):	Extrato emitido
Pós-condição (cenário de insucesso):	Erro ao emitir extrato
Trigger:	Aplicação client da API envia a mensagem "Emitir extrato"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente e dados do cartão, periodo do extrato bancário 2. O sistema emite um extrato com as informações da conta corrente
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos.
Regras de Negócio:	

4.16. Simulação de empréstimo

Nome:	Simular empréstimo
Objetivo:	Simular empréstimo e juros estimados
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário deve possuir conta corrente; Usuário logado
Pós-condição (cenário de sucesso):	Empréstimo simulado
Pós-condição (cenário de insucesso):	Erro na simulação
Trigger:	Aplicação client (banco) da API envia a mensagem "Simulação de pagamento de parcela de Empréstimo"
Fluxo Principal:	1. O banco envia mensagem com as informações da conta corrente, valor do empréstimo e valor da parcela do empréstimo [A1] 2. O sistema calcula a quantidade de parcelas mensais em função da taxa de juros vigente 3. O sistema envia a quantidade de parcelas mensais para o usuário.
Fluxos Alternativos:	[A1] Quantidade de parcelas pré-determinada 1. O banco envia mensagem com as informações da conta corrente , valor do empréstimo e a quantidade de parcelas mensais 2. O sistema calcula o valor da parcela mensal em função da taxa de juros vigente 3. O sistema envia o valor da parcela de parcelas mensais para o usuário.
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos.
Regras de Negócio:	

4.17. Realização de empréstimo

Nome:	Realização do empréstimo
Objetivo:	Permitir que o cliente do banco contrate um empréstimo
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário deve possuir conta corrente; Usuário logado
Pós-condição (cenário de sucesso):	Empréstimo realizado
Pós-condição (cenário de insucesso):	Erro na realização do empréstimo
Trigger:	Aplicação client (banco) da API envia a mensagem "Realização de empréstimo"
Fluxo Principal:	1. O usuário envia mensagem com as informações da conta corrente, valor do empréstimo desejado e a quantidade de parcelas mensais [E1][RN1] 2. O sistema transfere o valor do empréstimo para conta do usuário e registra dados do empréstimo
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Usuário não pode solicitar o empréstimo. 1. O sistema retorna a mensagem ‘critérios para empréstimo não respeitados’
Regras de Negócio:	[RN1] Empréstimos devem ser concedidos para contas corrente respeitando os critérios vigentes do banco

4.18. Avaliar estoque

Nome:	Avaliar saldo em cripto ou em real
Objetivo:	Avaliar estoque de cripto ou valor em real necessários na tesouraria do banco
Atores:	Aplicação Client da API (banco)
Pré-condição:	Usuário administrador do sistema
Pós-condição (cenário de sucesso):	Valor de saldo avaliado
Pós-condição (cenário de insucesso):	Erro na avaliação do saldo
Trigger:	Aplicação client (banco) da API envia a mensagem "Avaliar saldo"
Fluxo Principal:	1. O banco envia mensagem com o tipo da moeda 2. O sistema valida o tipo da moeda [E1][RN1] 3. O sistema retorna o valor total necessário (saldo do banco) na moeda informada baseado nos saldos das contas correntes do clientes do banco
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Tipo de moeda inválido 1. O sistema retorna a mensagem ‘Tipo de moeda inválido’
Regras de Negócio:	[RN1] Os tipos de moedas aceitáveis são ‘Cripto’ e ‘Real’

4.19. Compra de Ethereum com Real Brasileiro

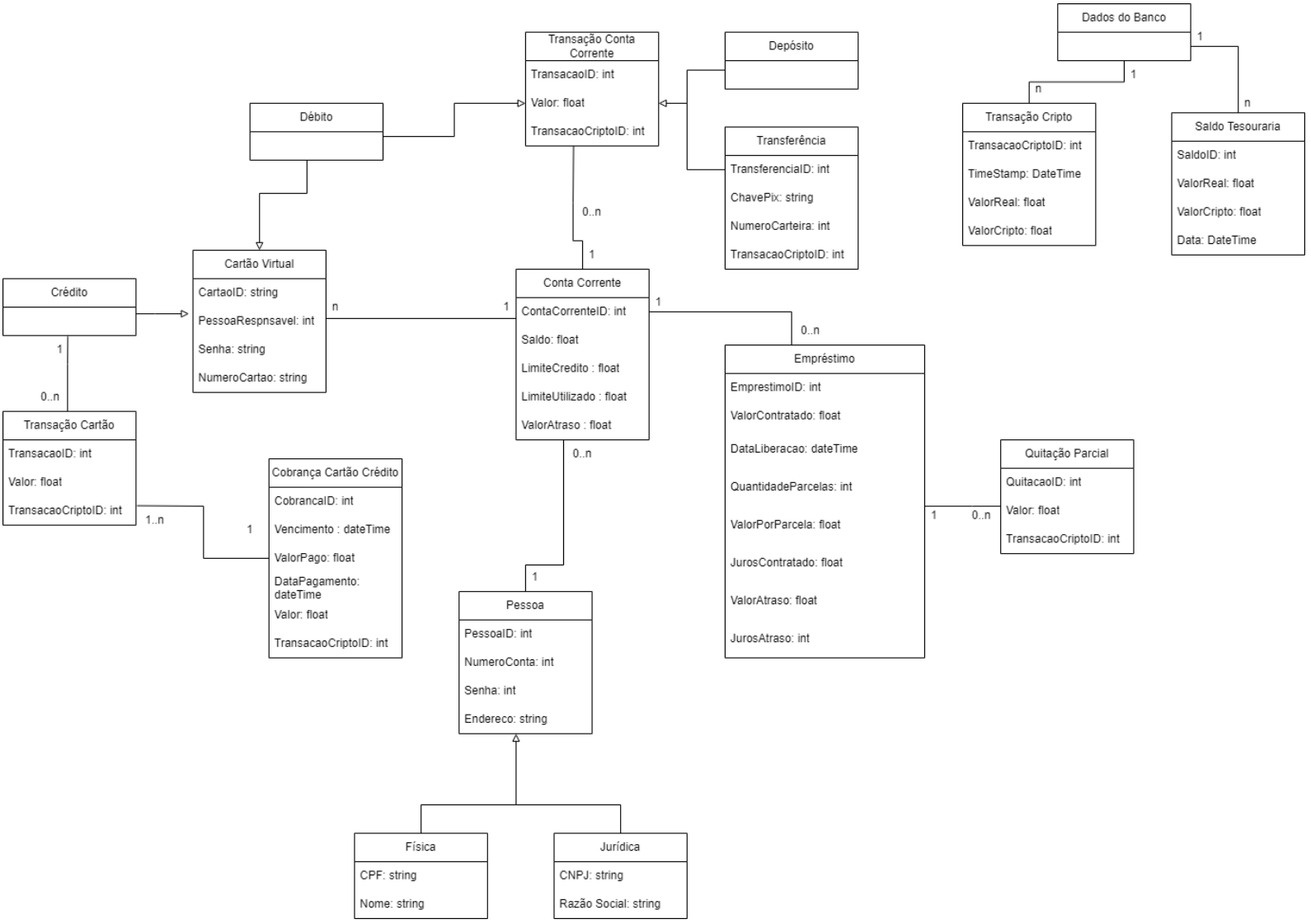
Nome:	Comprar Ethereum
Objetivo:	Compra de Ethereum com Real Brasileiro
Atores:	Aplicação Client da API (banco)
Pré-condição:	Usuário administrador do sistema
Pós-condição (cenário de sucesso):	Aumento de capital em Etherium.
Pós-condição (cenário de insucesso):	Erro na compra.
Trigger:	Aplicação client (banco) da API envia a mensagem "Comprar etherium"
Fluxo Principal:	1. O banco envia mensagem com a quantidade em real e o token JWT. 2. O sistema valida as informações [E1][RN1] 3. O sistema solicita a compra da cripto através de uma exchange. 4. O sistema atualiza o saldo em caixa de Etherium
Fluxos Alternativos:	
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Token inválido 1. O sistema retorna a mensagem ‘Token inválido’
Regras de Negócio:	[RN1] O banco deve fornecer um token JWT válido.

4.20. Depósito

Nome:	Depositar
Objetivo:	Depositar fundos em conta corrente.
Atores:	Aplicação Client da API (banco, cliente)
Pré-condição:	Usuário administrador do sistema
Pós-condição (cenário de sucesso):	Aumento de capital em conta do cliente.
Pós-condição (cenário de insucesso):	Erro no depósito.
Trigger:	Aplicação client (Cliente) da API envia a mensagem "Depositar"
Fluxo Principal:	1. O Cliente envia mensagem com o valor do depósito, tipo de moeda, conta destino e o token JWT. 2. O sistema valida as informações [E1][E2][RN1][RN2][A1] 3. O sistema atualiza o saldo em caixa de Etherium do cliente.
Fluxos Alternativos:	[A1] Tipo de moeda Real 1. O sistema recebe a transferência em sua carteira de cripto. 2. O sistema solicita a compra da cripto com o valor depositado em real para a exchange de criptomoedas. 3. O sistema atualiza o saldo em Etherium do cliente.
Fluxos de Exceção:	Códigos de erro: 401: Acesso negado. 406: Parâmetros inválidos. [E1] Tipo de moeda inválido 1. O sistema retorna a mensagem ‘Tipo de moeda inválido’ [E1] Token inválido 1. O sistema retorna a mensagem ‘Token inválido’
Regras de Negócio:	[RN1] O cliente deve fornecer um tipo de moeda válido (real ou etherium) [RN2] O cliente deve fornecer um token JWT válido.

4.21. Diagrama de Classes

Além disso, desenvolvemos um diagrama de classes para ajudar a entender melhor a estrutura do sistema representado neste trabalho.



Obs: Para efeito de simplificar o diagrama todas as transações que recebem a FK TransaçãoCriptoID deveriam ter ligação com Transação Cripto porém não foram explicitadas neste diagrama, visando melhor legibilidade.

5. Design e Estrutura da API

Para entendermos um pouco mais sobre a implementação desta API (Application Programming Interfaces), primeiro precisamos entender o que são APIs, como funcionam e como são estruturadas.

Uma API (Interface de Programação de Aplicações) é um conjunto de regras e protocolos que permite que diferentes softwares se comuniquem entre si. Funciona como uma ponte que permite que uma aplicação acesse funcionalidades ou dados de outra aplicação de forma padronizada e segura, sem precisar conhecer os detalhes internos da implementação.[8]

Para exemplo deste trabalho a API tem como funcionalidade entregar informações de transações bancárias diretamente para o cliente sem que o cliente tenha conhecimento da implementação da mesma. Para simplificar podemos imaginar uma estrutura de um restaurante, o cliente chega a um restaurante, decide no cardápio o que quer comer e então realiza o pedido através do garçom, que prontamente vai a cozinha informa o cozinheiro e já com a comida pronta traz de volta para o cliente, note que o cliente não sabe como a comida foi feita e nem precisa saber, neste exemplo a API seria o garçom que está indo ao servidor, buscando as informações e trazendo para o cliente, que apenas vai consumir os dados, neste caso o prato de comida.

A imagem abaixo exemplifica na prática como é feita uma chamada em uma API, temos o Cliente sentado à mesa, ele envia uma requisição (o pedido) e a API trás uma resposta (o prato).

Essa resposta, por sua vez, é a linguagem que a API utiliza para comunicar os resultados da sua solicitação.

As APIs podem retornar respostas em diversos formatos, cada um com suas características e aplicabilidades em caso de sucesso:

- “JSON (JavaScript Object Notation) é um formato de dados usado para transmitir e armazenar dados de forma organizada e legível por máquinas. Ele é baseado na sintaxe de objetos JavaScript e é amplamente utilizado em APIs para enviar e receber dados entre diferentes sistemas.”[9]
- XML (Extensible Markup Language): Um formato padronizado e rico em metadados, utilizado em diversas aplicações e plataformas. Sua estrutura em árvore e suporte a namespaces permitem organizar e descrever dados de forma complexa.[10]

- Texto Simples: Normalmente utilizados em sistemas mais antigos e mais simples, APIs também podem retornar respostas em formato de texto simples, como CSV (Comma Separated Values) ou HTML.

Além dos dados solicitados, a resposta de uma API também inclui informações sobre o status da sua solicitação. O status HTTP da resposta indica se a solicitação foi bem-sucedida (por exemplo, código 200 OK) ou se ocorreu algum erro (por exemplo, código 404 Not Found).

Códigos de erro específicos da API podem fornecer detalhes adicionais sobre a falha da solicitação, auxiliando na identificação e resolução de problemas.

5.1. Estrutura de uma API

Este trabalho foi implementado seguindo a estrutura clássica de APIs modernas, assim facilitando a manutenção e leitura do código por se tratar de um padrão conhecido e difundido.

- Endpoints: Para iniciar temos a porta de entrada da API, os endpoints são os pontos de acessos que podem ser acessados por outras aplicações, eles representam a camada mais externa da aplicação e por isso é sempre preferível evitar acessos ao banco de dados e regras de negócios diretamente nessa camada. Cada endpoint possui uma URL de acesso que é a rota em que o cliente usará para acessar a funcionalidade específica.
- Métodos HTTPs: são as definições do que cada endpoint deve representar e são divididos entre para este trabalho foram implementados os seguintes.
    - GET: Utilizado para recuperar dados de um recurso.
    - POST: Utilizado para criar um novo recurso.
    - PUT: Utilizado para atualizar um recurso existente.
    - DELETE: Utilizado para excluir um recurso.
- Formato de dados de resposta: Como mencionado anteriormente uma API pode retornar diversos formatos diferentes de dados, para este trabalho ficou definido o uso do JSON por se tratar do formato mais famoso e mais difundido no meio.

5.2. Clean Architecture

Como mencionado anteriormente, a arquitetura deste projeto usou como base os conceitos de arquitetura limpa, essa abordagem nos permite criar códigos de fácil manutenibilidade e muito escaláveis, além disso evitando depreciação de técnica já que por sua vez divide o código entre funcionalidades, para conceito deste trabalho a estrutura seguida foi o padrão de repositórios.

Um repositório nada mais é do que uma classe responsável por abstrair a camada Data do resto do código já que é através dela que faremos os acessos ao banco de dados.[11]

Para este projeto implementamos a arquitetura detalhada abaixo.

- Camada da API: Esta camada contém apenas os endpoints da API e as chamadas correspondentes. Aqui, não há regras de negócio, garantindo que sua responsabilidade seja exclusivamente a exposição dos serviços para consumo externo. A camada da API só pode interagir com a camada Application.
- Camada Application: Nesta camada residem as regras de negócio da aplicação, os serviços (ou use cases) e os DTOs (Data Transfer Objects). Os serviços definem as operações que a aplicação pode realizar, implementando as regras de negócio. Eles operam sobre os DTOs para receber e retornar dados de forma estruturada. A camada Application tem visibilidade sobre todo o projeto, coordenando a lógica de negócio e orquestrando as operações necessárias para atender aos requisitos funcionais.
- Camada Data: Responsável pelo acesso ao banco de dados, esta camada abstrai toda a lógica de persistência. Ela interage exclusivamente com a camada Domain, manipulando entidades e realizando operações de leitura e escrita no banco de dados conforme necessário.
- Camada Domain: Aqui residem as entidades ou modelos de domínio da aplicação, que representam conceitos e regras de negócio da aplicação. Também inclui mapeamentos do banco de dados, garantindo a persistência das entidades de domínio sem expor detalhes de implementação para as camadas superiores.

Nessa arquitetura temos uma maior separação do que é regra de negócio e o que é banco de dados, além disso por se tratar de transações bancárias temos uma maior segurança impedindo que a camada da API acesse direto a camada Data.

Para exemplificar melhor a importância dessa separação entre camadas e entender como isso evita acoplamentos de código temos o exemplo a seguir: para esse projeto foi utilizado o entity framework para fazer o acesso, criação e gerenciamento do banco de dados, sendo assim todos as consultas, mapeamentos e lógicas de criação de tabelas ficam dentro da camada Data, cada tabela por sua vez teria um repositório (classe responsável pelas consultas nesta tabela), já as interfaces destes repositórios ficam na camada Application, imaginamos que queremos mudar o framework de banco de dados no meio do projeto assim podemos remover toda a camada do Data sem impactar o restante da API já que tudo que a camada Application precisa para funcionar são as interfaces reduzindo assim o acoplamento entre as camadas.

5.3. Injeção de Dependência

Para um melhor aproveitamento da arquitetura limpa precisamos trabalhar a comunicação entre camadas, para isso a abordagem adotada foi a injeção de dependência.

A Injeção de Dependências é um princípio da programação orientada a objetos que se concentra em passar as dependências necessárias para um objeto, em vez de o próprio objeto criar essas dependências. Isso pode ser feito de várias maneiras, como por construtor, método ou propriedade, e é usado para minimizar o acoplamento entre classes. [13]

Usamos a injeção de dependência em arquiteturas limpas principalmente para garantir que lógicas e regras de negócios estejam separadas e encapsuladas, além de manter ciclos de vida de objetos.

No contexto deste projeto temos:

- Controllers: Estão na camada da API. Eles dependem apenas dos serviços da aplicação e não têm conhecimento da implementação concreta desses serviços.
- Serviços: Localizados na camada Application, implementam as regras de negócio da aplicação. Eles podem depender de outros serviços ou de repositórios para realizar operações específicas.
- Repositórios: na camada do Data, são responsáveis pelo acesso aos dados. Um repositório pode depender de outros repositórios para obter dados relacionados ou realizar operações de persistência.

Como visto anteriormente a estrutura implementada impede que tenhamos comunicação direta entre camadas que não estão adjacente, por exemplo o controller nunca pode se comunicar direto com o repositório, o que violaria o princípio da arquitetura limpa, toda requisição deve passar pelo serviço antes de chegar ao banco de dados, isso garante que todos os dados recebidos sempre serão tratados pelas regras de negócios correspondentes.

6. Implementação

Este trabalho foi desenvolvido utilizando a linguagem de programação C# e para facilitar a integração foi utilizado o framework .NET. O .NET é um framework versátil, desenvolvido pela Microsoft que tem o objetivo de dar suporte e facilitar a criação de aplicações e sistemas em C#, oferecendo várias bibliotecas que facilitam o desenvolvimento na linguagem.[14]

Como vimos anteriormente esse projeto foi desenvolvido usando as melhores práticas e conceitos da programação orientada a objeto, o que é essencial para aplicações grandes e complexas que exigem uma grande e segura manipulação de dados, facilitando a manutenção e a modularização do sistema. Tratando-se de uma API de transações bancárias um ponto extremamente importante é a segurança, nisso o .NET se sobressai, pois oferece uma vasta gama de funcionalidades que facilitam o processo de segurança, como segurança por cargo, segurança por código de acesso e diversas bibliotecas de criptografia, além de uma fácil integração com tokens de segurança, como JWT, que foi utilizado neste trabalho.

Outro aspecto importante para a escolha do .NET foi a performance, dado a quantidade de transações e requisições médias de um banco se vê necessário o máximo de medidas para garantir agilidade e funcionalidade da API, o .NET permite transações em tempo de compilação (Just-In-Time), “O compilador Just-In-Time (JIT) é uma parte do Common Language Runtime (CLR) no .NET, responsável por gerenciar a execução de programas .NET, independentemente de qualquer linguagem de programação .NET. Um compilador específico de linguagem converte o código-fonte para a linguagem intermediária. Esta linguagem intermediária é então convertida em código de máquina pelo compilador Just-In-Time (JIT). Este código de máquina é específico para o ambiente do computador onde o compilador JIT está sendo executado.”[15]

6.1. Entity Framework

Este trabalho foi desenvolvido usando o Entity Framework, é uma ferramenta que facilita a manipulação e acesso ao banco de dados. Além disso o Entity Framework é um ORM (Object-Relational Mapper).[16]

ORM (Object Relational Mapper) é uma técnica de mapeamento objeto relacional que permite fazer uma relação dos objetos com os dados que os mesmos representam.[17]

Este framework funciona criando uma ponte entre a aplicação e o banco de dados e foi dividido seguindo a seguinte estrutura:

- Model: A primeira etapa é definir as classes em C# que representam os objetos das tabelas do nosso banco, cada tabela deve ter sua model representada na aplicação, no sentido único cada entidade do banco representa um objeto.
- Contexto: Depois disso criamos um contexto, que herda o contexto base do entity framework o ‘DbContext’, que por sua vez possuem os ‘DbSet’[18] de cada entidade por exemplo o DbSet<ContaCorrente> representa todas as contas correntes do sistema.
- Operações: As operações (Adicionar, Ler, Atualizar e Deletar) ficam nos repositórios, porém são operados como objetos, para efeito de exemplo imagine que esteja inserindo uma conta corrente nova, para isso eu crio um objeto ContaCorrente, preencho as informações e adiciono ao meu DbSet<ContaCorrente>. Vale ressaltar que para garantir a integridade do banco de dados todas as operações são feitas dentro de uma transação que seguem o seguinte padrão:
    - Iniciamos a transação.
    - Realizamos todas as operações necessárias dentro de um bloco try-finally.
    - Passando a execução para o finally, caso o bloco try tenha sido executado até o final, executamos o CommitTransaction, que é responsável por concluir a transação e inserir o valor novo no banco de dados, caso contrário executamos um RollbackTransaction que desfaz as alterações.
- Migrações: Por último temos as migrações, como vimos o Entity framework

te dá suporte para migrações que facilitam alterações e ajustes na estrutura do banco de dados de acordo com a necessidade, se quiséssemos adicionar uma coluna nova a tabela de conta corrente com o banco já estruturado por exemplo poderíamos simplesmente adicionar a propriedade a classe ContaCorrente adiciona-la no mapeamento da classe e rodar a migração novamente.

Tendo tudo isso em mente, vemos que utilizar o entity framework para desenvolver essa API simplifica a manipulação de dados, acelera o desenvolvimento e garante um método eficiente e efetivo para manter o banco de dados.

Para testar a minha API foi utilizado o Swagger. Swagger é uma ferramenta de código aberto que permite que desenvolvedores, testem, documentem e interajam com APIs. Sua interface simplificada me permitiu executar todas as chamadas desta API diretamente do meu navegador.

6.2. JWT

Para garantir a comunicação segura entre clientes e servidores na minha API de transações bancárias, eu optei por usar JSON Web Tokens (JWT), que são uma solução popular para autenticação e autorização, devido à sua simplicidade e segurança.

Os JWTs são divididos em três partes: cabeçalho, payload (dados) e assinatura. O cabeçalho indica o tipo de token (JWT) e o algoritmo de assinatura (como HMAC SHA256). O payload contém informações do usuário, enquanto a assinatura verifica a autenticidade do token e garante que ele não foi alterado.[19]

Na minha implementação, o JWT é enviado no cabeçalho (header) das requisições HTTP. Quando um usuário faz login, o servidor gera um JWT com base no cpf ou cnpj do usuário e o assina com uma chave secreta, vale ressaltar que o token tem duração de 5 minutos, sendo assim o token funciona semelhante a uma sessão, o usuário tem esse tempo para realizar as suas transações. Esse token é então enviado de volta ao cliente.

Em requisições futuras, o cliente inclui o JWT no cabeçalho Authorization da requisição HTTP, no formato Authorization: Bearer <token>. O servidor então verifica o token, usando a chave secreta para garantir que ele é válido. Se o token for válido, o servidor processa a requisição e permite o acesso aos recursos protegidos. Caso contrário ou caso o token esteja expirado, retorna um erro 401 Acesso negado.

7. Conclusão

O desenvolvimento de uma API para transações bancárias usando criptomoedas representa um grande avanço no mundo financeiro digital. Este trabalho explorou como criar, implementar e usar uma plataforma segura e eficiente para realizar transações com criptomoedas, destacando a importância de manter a segurança e seguir as regras para garantir a confiança no sistema.

Durante o desenvolvimento, usamos tecnologias modernas, como o framework .NET, para criar uma estrutura escalável e eficiente, além de implementar práticas para proteger contra fraudes e ataques cibernéticos. A API foi projetada para ser fácil de usar, com interfaces claras, facilitando a integração com diferentes aplicações financeiras e simplificando a gestão dos ativos digitais.

Os resultados mostram que a API atende às necessidades do mercado de criptomoedas, melhorando a eficiência das transações e ajudando na inclusão financeira e no acesso aos serviços bancários digitais. Com a evolução contínua das tecnologias de blockchain e a aceitação crescente das criptomoedas, a criação de APIs bancárias especializadas é essencial para atender às demandas do mercado e oferecer serviços inovadores e seguros.

Em resumo, este trabalho demonstrou que é possível criar uma infraestrutura bancária moderna e confiável, capaz de acompanhar o crescimento das transações com criptomoedas. A API desenvolvida é um passo importante na modernização dos serviços financeiros, promovendo inovação, segurança e acessibilidade no mundo das criptomoedas.

8. Implementações futuras

Como planos futuros para melhorar e aprimorar as funcionalidades desta API, pretendo implementar novas funcionalidades para alguns dos pontos em que julguei ser mais importante.

Primeiramente pretendo fazer a integração com a blockchain, ao longo deste trabalho pude estudar bastante sobre o tema e fiquei fascinado pela possibilidade de inovação no sistema bancário, além disso ao realizar a integração poderia automatizar diversos processos usando contratos inteligentes e garantir que todas as transações fossem seguras e transparentes.

Um ponto que não pode faltar na implementação de uma API bancária é a segurança, no momento de implementação deste trabalho estou usando o JWT como token de entrada nos endpoints, porém estou enviando e recebendo o mesmo sem criptografá-lo, o que é uma falha de segurança, já que algum usuário malicioso poderia interceptar a transação HTTP e obter o token para usar livremente, para isso pretendo implementar a criptografia de ponta a ponta, o que vai garantir que sempre que o token estiver em uma requisição ele vai estar criptografado.

Por último, pretendo validar o token JWT em todos os endpoints, garantindo que o CPF do usuário corresponda ao CPF associado ao token. Atualmente, estamos apenas verificando o tempo de expiração do token.

9. Referências

[1] Ethereum: como surgiu a segunda criptomoeda mais valiosa do mundo?

<https://www.infomoney.com.br/guias/o-que-e-ethereum> acesso em 22/04/2024

[2] O que são DApps? Saiba como funcionam e como utilizar

<https://www.blocknews.com.br/educacao/o-que-sao-dapps-saiba-como-funcionam-e-como-utilizar/> acesso em 25/04/2024

[3] Ethereum: o que é, preço, e como comprar. Disponível em:

<https://www.bity.com.br/blog/ethereum-o-que-e-preco-e-como-comprar/#:~:text=O%20Ethereum%20é%20uma%20das,possibilidades%20para%20o%20universo%20cripto.> acesso em 22/04/2024

[4] Vantagens e riscos do uso de moedas digitais na empresa. Disponível em:

<https://sebrae.com.br/sites/PortalSebrae/artigos/vantagens-e-riscos-do-uso-de-moedas-digitais-na-empresa,7604ac941b896810VgnVCM1000001b00320aRCRD#:~:text=Risco%20de%20hacking%3A%20as%20carteiras,país%20e%20podem%20mudar%20rapidamente.>acesso em 22/04/2024

[5] Banco Central quer integração do Drex com Ethereum e outros blockchains 'tradicionais'. Disponível em:

<https://exame.com/future-of-money/banco-central-quer-integracao-do-drex-com-ethereum-e-outros-blockchains-tradicionais/>acesso em 22/04/2024

[6] Definição de Escopo de Projeto: entenda o que é escopo e como montar o seu em apenas 4 etapas

<https://www.euax.com.br/2018/12/definicao-de-escopo-de-projeto/>acesso em 23/04/2024

[7] Applying UML and patterns - Craig Larman - 2a edição. Disponível em

<https://personal.utdallas.edu/~chung/SP/applying-uml-and-patterns.pdf>acess

o em: 22/04/2024.

[8] API: conceito, exemplos de uso e importância da integração para desenvolvedores. Disponível em:

<https://logsmartbrasil.com.br/2024/03/26/o-que-e-api/#:~:text=Em%20termos%20simples%2C%20uma%20API,compartilhando%20dados%20de%20forma%20estruturada.> acesso em 23/04/2024

[9] Saiba o que é JSON e como utilizar. Disponível em:

<https://www.alura.com.br/artigos/o-que-e-json> acesso em 22/04/2024

[10]Introduction to XML

<https://www.w3schools.com/xml/xml_whatis.asp> acesso em 22/04/2024

[11].NET 6.0 - Clean Architecture using Repository Pattern and Dapper with Logging and Unit Testing

<https://dev.to/techiesdiary/net-60-clean-architecture-using-repository-pattern-and-dapper-with-logging-and-unit-testing-1nd9> acesso em 27/05/2024

[13] Injeção de Dependências: Fundamentos e Importância na Arquitetura Clean Code

<https://www.dio.me/articles/injecao-de-dependencias-fundamentos-e-importancia-na-arquitetura-clean-code> acesso em 19/06/2024

[14] O que é o .NET?

<https://dotnet.microsoft.com/pt-br/learn/dotnet/what-is-dotnet> acesso em 22/06/2024

[15]What is Just-In-Time(JIT) Compiler in .NET

<https://www.geeksforgeeks.org/what-is-just-in-time-jit-compiler-in-dot-net> acesso em 23/06/2024

[16]Entity Framework Tools in Visual Studio

<https://learn.microsoft.com/en-us/visualstudio/data-tools/entity-data-model-tools-in-visual-studio?view=vs-2022> acesso em 02/07/2024

[17]ORM : Object Relational Mapper

<https://www.devmedia.com.br/orm-object-relational-mapper/19056> acesso em 02/07/2024

[18]Get started with Entity Framework 6

<https://learn.microsoft.com/en-us/ef/ef6/get-started> acesso em 02/07/2024

[19]Introduction to JSON Web Tokens

<https://jwt.io/introduction> acesso em 29/06/2024

[20]Kraken Becomes First Crypto Exchange to Charter a US Bank

<https://www.coindesk.com/business/2020/09/16/kraken-becomes-first-crypto-exchange-to-charter-a-us-bank> acesso em 01/07/2024

[21]An Analysis of Bank Charters and Selected Policy Issues

<https://crsreports.congress.gov/product/pdf/R/R47014> página 2. acesso em 02/07/2024