INSTITUTO INFNET

ENGENHARIA DE DADOS E IA

PROJETO INTEGRADOR DE FRONT-END COM FRAMEWORKS

Teste de Performance 3

Alunos: Marcus Vinícius dos Santos Liberato, Arthur Bezerra de Souza e Louriane Bueno.

Projeto: Plataforma Web de Compra e Venda de Criptomoedas

Equipe: Marcus (Front-end), Louriane (UI/UX), Arthur (Back-end)

18/02/2026

1. DEFINIÇÃO DO PROJETO

O projeto escolhido para desenvolvimento é uma **Plataforma Web de Compra e Venda de Criptomoedas**. Esta aplicação permitirá aos usuários realizarem transações de compra e venda de diferentes criptomoedas em tempo real, acompanhar cotações, visualizar histórico de transações e gerenciar seu portfólio digital.

A escolha deste projeto se justifica pela relevância atual do mercado de criptomoedas e pela oportunidade de trabalhar com dados em tempo real, APIs externas robustas e funcionalidades que exigem alta responsividade e segurança. Além disso, o projeto permite aplicar todos os conceitos de desenvolvimento front-end com frameworks modernos, incluindo React para web e React Native para mobile.

A plataforma será desenvolvida com foco na experiência do usuário, oferecendo uma interface intuitiva tanto para usuários iniciantes quanto para traders experientes, com recursos visuais como gráficos interativos e dashboards personalizáveis.

2. DEFINIÇÃO DE ESCOPO E OBJETIVOS

Objetivo Principal:

Desenvolver uma aplicação web responsiva e intuitiva que permita aos usuários realizarem operações de compra e venda de criptomoedas, com acesso a dados em tempo real de APIs externas e uma versão otimizada para dispositivos móveis.

Funcionalidades Específicas:

- **Autenticação e Segurança:** Sistema de login/cadastro seguro com JWT, recuperação de senha e autenticação de dois fatores.
- **Dashboard Principal:** Visualização em tempo real das principais criptomoedas com gráficos de variação, volume de negociação e tendências.
- **Sistema de Compra/Venda:** Interface para realizar transações com confirmação em tempo real, histórico de ordens e simulador de operações.
- **Carteira Digital:** Gerenciamento do portfólio pessoal com balanço total, distribuição de ativos e histórico de movimentações.
- **Consumo de API Externa:** Integração com APIs como CoinGecko e/ou Binance para dados atualizados de cotações.
- **Responsividade Total:** Adaptação completa para desktop, tablet e smartphones.
- **Funcionalidades Mobile:** Gestos touch como swipe para navegação, pull-to-refresh para atualizar cotações e gestos de pinça para zoom em gráficos.
- **Notificações:** Alertas de preço personalizáveis e notificações de transações concluídas.
- **Modo Escuro/Claro:** Personalização visual para melhor experiência do usuário.

3. GESTÃO ÁGIL DO PROJETO (SCRUM)

Metodologia Adotada

O projeto será desenvolvido seguindo o framework Scrum, com foco em entregas incrementais, controle contínuo do progresso e adaptação frequente com base em feedbacks. O trabalho será organizado em sprints de duas semanas, com planejamento, execução, revisão e retrospectiva claramente definidos.

Definition of Ready (DoR)

Um item do Product Backlog será considerado pronto para entrar em uma Sprint quando atender aos seguintes critérios objetivos:

- História de usuário escrita no formato "Como…, eu quero…, para…"
- Critérios de aceitação claros e testáveis
- Prioridade definida pelo Product Owner
- Estimativa de esforço atribuída (Story Points)
- Dependências identificadas
- Dados, APIs e recursos necessários disponíveis
- Interface ou fluxo previamente validado no Figma (quando aplicável)

Somente itens que atendam a todos esses critérios poderão ser selecionados durante a Sprint Planning.

Definition of Done (DoD)

Uma entrega será considerada concluída (Done) somente quando atender todos os critérios abaixo:

- Funcionalidade implementada conforme critérios de aceitação
- Código revisado (code review) e integrado à branch principal
- Testes unitários básicos implementados e aprovados
- Interface validada visualmente conforme o design aprovado
- Sem erros críticos no console ou falhas de execução
- Funcionalidade integrada ao restante do sistema
- Documentação técnica mínima atualizada
- Incremento disponível para demonstração na Sprint Review

Sem o atendimento integral desses critérios, o item não será considerado entregue.

Política de Refinamento do Backlog

O Product Backlog Refinement ocorrerá uma vez por semana, com duração aproximada de 1 hora, conduzido pelo Product Owner (Arthur) com participação de toda a equipe.

Durante o refinamento serão realizadas:

- Detalhamento de histórias futuras
- Quebra de histórias grandes em tarefas executáveis
- Revisão de prioridades

• Estimativas de esforço

• Identificação de riscos e dependências técnicas

4. CRONOGRAMA DO PROJETO

Calendário Geral do Projeto

Período do projeto: 20/01/2026 a 16/03/2026
Duração: 8 semanas
Sprints: 4 sprints de 2 semanas

Sprint 1 – Planejamento e Fundação
20/01 a 02/02/2026

Tarefa	Responsável	Esforço (h)	Dependência	Critério de Aceite
Criar repositório e CI/CD	Marcus	6h	—	Build automático funcionando
Estrutura inicial React/Next	Marcus	10h	Repo criado	App renderiza página base
Wireframes no Figma	Louriane	12h	—	Fluxos aprovados
Definir arquitetura back-end	Arthur	10h	—	Documento técnico validado

Marco M1: Protótipo navegável + ambiente configurado
Critério de Aceite: Aplicação inicial funcionando e fluxos validados no Figma

---

Sprint 2 – MVP Visual
03/02 a 16/02/2026

Tarefa	Responsável	Esforço (h)	Dependência	Critério de Aceite
Página Login	Marcus	8h	UI aprovada	Login renderizado
Dashboard com Recharts	Marcus	12h	API disponível	Gráfico exibindo dados
API de autenticação	Arthur	10h	Arquitetura definida	JWT funcionando
Ajustes responsivos	Louriane	8h	Páginas criadas	Mobile validado

Marco M2: MVP navegável
Critério de Aceite: Login → Dashboard funcional com dados simulados

Sprint 3 – Integração
17/02 a 02/03/2026

- Integração front-end com API real
- Implementação de compra e venda

- Testes de fluxo completo

Marco M3: Sistema integrado
Critério de Aceite: Usuário realiza login, visualiza dados reais e simula operação

---

Sprint 4 – Finalização e Deploy
03/03 a 16/03/2026

- Otimizações de performance
- Deploy em produção
- Documentação final

Marco M4: Aplicação publicada
Critério de Aceite: Sistema online, documentado e demonstrável

---

Estratégias para Cumprimento de Prazos:

Estratégia	Descrição
Daily Scrum (15 min/dia)	Reunião rápida para identificar bloqueios e revisar progresso
Kanban Board (Trello/Jira)	Visualizar tarefas: A Fazer / Em Progresso / Concluído
Sprint Review	Apresentar entregas e receber feedback do Product Owner
Sprint Retrospective	Discutir pontos positivos e melhorias
Versionamento (GitHub)	Controle de versões e integração contínua
Documentação Contínua	Cada integrante regista suas atividades diariamente
Testes Automatizados	Reduzir erros e retrabalho

5. CRIAÇÃO DE HISTÓRIAS DE USUÁRIO

ARTEFATOS DO SCRUM:

Product Backlog:

Lista priorizada de todas as funcionalidades, melhorias e correções necessárias para o produto. Será mantido e atualizado pelo Product Owner (Arthur, neste caso), contendo itens como:

• Sistema de autenticação com JWT
• Dashboard com gráficos em tempo real
• Integração com API CoinGecko
• Sistema de compra/venda de criptomoedas
• Versão mobile com gestos touch
• Notificações de preço

Sprint Backlog:

Subconjunto do Product Backlog selecionado para cada Sprint. A equipe seleciona os itens que podem ser completados dentro do período de 2 semanas, decompostos em tarefas menores e atribuídos a cada membro da equipe.

Incremento:

Produto potencialmente entregável ao final de cada Sprint. Por exemplo, ao final da Sprint 2, teremos uma interface funcional com páginas de Home, Login e Dashboard implementadas.

PAPÉIS DO SCRUM:

Product Owner (Arthur):

Responsável por maximizar o valor do produto. Define as prioridades do backlog, comunica a visão do produto e toma decisões sobre funcionalidades. Por ser desenvolvedor back-end e ter conhecimento técnico da integração com APIs, foi escalado para exercer a função de Product Owner.

Scrum Master (Louriane):

Facilita o processo Scrum e remove impedimentos. Louriane, como designer UI/UX e documentadora, tem uma visão holística do projeto, podendo mediar conflitos e garantir que a equipe siga as práticas ágeis corretamente.

Time de Desenvolvimento (Marcus, Louriane, Arthur):

Equipe auto-organizada e multifuncional responsável por entregar incrementos do produto. Cada membro contribui com suas especialidades para atingir os objetivos da Sprint.

EVENTOS DO SCRUM:

Sprint Planning (Início de cada Sprint - 2h):

Reunião para definir o objetivo da Sprint e selecionar itens do Product Backlog. A equipe discute o que pode ser entregue e como o trabalho será realizado. Exemplo: Na Sprint 1, definimos que o objetivo é ter o protótipo navegável e ambiente configurado.

Daily Scrum (Diariamente - 15min):

Reunião rápida onde cada membro responde: O que fiz ontem? O que farei hoje? Há impedimentos? Isso mantém todos sincronizados e identifica problemas rapidamente.

Sprint Review (Final de cada Sprint - 1h):

Demonstração do incremento produzido para stakeholders. A equipe apresenta o que foi concluído e coleta feedback para ajustes futuros.

Sprint Retrospective (Após Review - 1h):

Reflexão da equipe sobre o processo. Discutimos: O que funcionou bem? O que pode melhorar? Quais ações tomaremos? Isso promove melhoria contínua.

5.1 Tabela comparativa das histórias de usuário

Elemento	Histórias de usuário originais	Histórias de usuário atualizadas	Melhorias Aplicadas
Perfil do usuário	Usuário (genérico)	Usuário investidor	Definição mais precisa do público-alvo, alinhada ao contexto do sistema
Objetivo da história	Visualizar informações sobre criptomoedas	Visualizar cotações das principais criptomoedas em tempo real	Maior clareza funcional e foco em dados relevantes
Contexto de uso	Não especificado	Dashboard responsivo e multiplataforma	Inclusão explícita de uso em desktop e dispositivos móveis
Benefício esperado	Acompanhar investimentos	Tomar decisões de investimento informadas	Explicitação do valor entregue ao usuário
Critérios de aceitação	Implícitos / não detalhados	Critérios objetivos e mensuráveis (atualização automática, responsividade, performance)	Facilita validação, testes e aceite da funcionalidade
Requisitos técnicos	Não especificados	Integração com API externa, gráficos, responsividade	Maior alinhamento entre requisitos funcionais e implementação
Aderência ao Scrum	História inicial de alto nível	História refinada, pronta para Sprint Planning	Adequação ao Product Backlog Refinement

6. PLANEJAMENTO DO DESENVOLVIMENTO FRONT-END

PASSOS INICIAIS:

a) **Definição da Stack Tecnológica (Dia 1-2):**

- Front-end Web: React.js com Next.js para SSR/SSG
- Front-end Mobile: React Native com Expo
- Gerenciamento de Estado: Redux Toolkit ou Zustand

• Estilização: Tailwind CSS (web) e StyleSheet (mobile)

• Gráficos: Chart.js ou Recharts

• HTTP Client: Axios com interceptors

b) Configuração do Ambiente (Dia 3-5):

• Criar repositório no GitHub com estrutura monorepo

• Configurar ESLint, Prettier e Husky para padronização

• Configurar CI/CD com GitHub Actions

• Criar ambiente de desenvolvimento local com Docker

• Configurar variáveis de ambiente (.env)

c) Arquitetura do Projeto (Dia 6-7):

• Definir estrutura de pastas (components, pages, hooks, services, utils)

• Criar componentes base reutilizáveis (Button, Input, Card, Modal)

• Implementar sistema de temas (dark/light mode)

• Configurar rotas e navegação

MARCOS PRINCIPAIS (MILESTONES):

Marco Entregável Semana

MARCO	ENTREGÁVEL	SEMANA
M1 - Fundação	Protótipo Figma aprovado	Semana 2
M2 - MVP Visual	Interface funcional com navegação completa	Semana 4
M3 - Integração	Front-end conectado ao back-end com dados reais	Semana 6
M4 - Release	Aplicação publicada e documentada	Semana 8

Critérios de Qualidade:

• Cobertura de testes unitários ≥ 70%

• Score do Lighthouse ≥ 90 em Performance e Acessibilidade

• Tempo de First Contentful Paint < 1.5s

• Zero erros críticos de segurança no OWASP Top 10

7. INTERATIVIDADE NA APLICAÇÃO WEB/MOBILE

O ReactJS será utilizado na versão web com gráficos em Recharts. O React Native permitirá reutilização da lógica de negócio através de hooks e services compartilhados.

No dashboard web, será utilizado um gráfico de Recharts para exibir as variações de preço. Os dados serão consumidos da API CoinGecko e atualizados automaticamente a cada 30 segundos, permitindo análise visual clara da tendência de mercado.

INTERAÇÕES BÁSICAS PLANEJADAS:

a) Autenticação e Onboarding:

- Formulário de login com validação em tempo real
- Feedback visual imediato para campos inválidos
- Animação de loading durante autenticação
- Tour guiado para novos usuários com highlights interativos

b) Dashboard e Visualização de Dados:

- Gráficos interativos com hover para detalhes (tooltip)
- Filtros dinâmicos por período (1h, 24h, 7d, 30d)
- Ordenação de tabelas por diferentes critérios (clique no cabeçalho)
- Pesquisa instantânea de criptomoedas com autocomplete
- Drag-and-drop para reorganizar widgets do dashboard

c) Sistema de Transações:

- Calculadora de conversão em tempo real ao digitar valor
- Slider para definir quantidade de compra/venda
- Modal de confirmação com resumo da operação
- Notificação toast após conclusão da transação
- Histórico com scroll infinito e paginação lazy

d) Interações Mobile Específicas:

- Pull-to-Refresh: Atualizar cotações puxando a tela
- Swipe Actions: Deslizar em itens para ações rápidas
- Bottom Sheet: Painel deslizante para detalhes
- Haptic Feedback: Vibração em ações importantes

• Shake to Refresh: Agitar dispositivo para atualizar

GARANTIA DE EXPERIÊNCIA FLUIDA:

a) Princípios de UX Aplicados:

• Feedback Imediato: Toda ação do usuário recebe resposta visual em < 100ms
• Estados de Loading: Skeletons e spinners para indicar carregamento
• Tratamento de Erros: Mensagens claras e ações de recuperação
• Consistência: Padrões de interação uniformes em toda aplicação
• Acessibilidade: Navegação por teclado, alto contraste, leitor de tela

b) Otimizações Técnicas:

• Debounce em inputs de pesquisa (300ms)
• Throttle em scroll evento (60fps)
• Virtualização de listas longas
• Prefetch de dados para navegação antecipada
• Cache local com invalidação inteligente

8. FRAMEWORKS: REACTJS E REACT NATIVE

Os frameworks ReactJS e React Native são fundamentais para atender aos requisitos do projeto, proporcionando uma base sólida para desenvolvimento web e mobile com código compartilhado e alta performance.

REACTJS PARA APLICAÇÃO WEB:

a) Responsividade:

React permite criar componentes modulares que se adaptam a diferentes tamanhos de tela. Utilizando hooks como useMediaQuery e bibliotecas como Tailwind CSS, podemos implementar layouts responsivos de forma eficiente. O Virtual DOM do React garante que mudanças de layout sejam renderizadas rapidamente.

b) Consumo de API Externa:

Com hooks como useState e useEffect, React facilita o gerenciamento de dados assíncronos. Bibliotecas como React Query ou SWR oferecem cache, revalidação automática e estados de loading/erro para integração com APIs como CoinGecko. O padrão de componentes permite encapsular lógica de fetching em custom hooks reutilizáveis.

c) Interatividade:

O sistema de eventos do React e o estado reativo permitem criar interfaces altamente interativas. Gráficos com Chart.js ou Recharts respondem a interações do usuário em tempo real. O Context API e Redux gerenciam estado global de forma previsível, essencial para uma plataforma de trading.

d) Performance:

React.memo, useMemo e useCallback previnem re-renderizações desnecessárias. Code splitting com React.lazy e Suspense carregam componentes sob demanda. Next.js adiciona SSR/SSG para melhorar SEO e tempo de carregamento inicial.

REACT NATIVE PARA VERSÃO MOBILE:

a) Desenvolvimento Cross-Platform:

Com React Native, desenvolvemos para iOS e Android com uma única base de código JavaScript/TypeScript. Aproximadamente 80-90% do código pode ser compartilhado entre plataformas, reduzindo tempo de desenvolvimento e custos de manutenção.

b) Gestos Touch Nativos:

React Native Gesture Handler e Reanimated 2 permitem implementar gestos complexos com performance nativa. Pull-to-refresh, swipe, pinch-to-zoom e long press são executados na thread nativa, garantindo 60fps mesmo em dispositivos mais antigos.

c) Acesso a APIs Nativas:

React Native oferece acesso a funcionalidades nativas como câmera (para QR codes), notificações push, biometria (Face ID/Touch ID para segurança) e armazenamento seguro (Keychain/Keystore para credenciais).

d) Experiência do Usuário:

Componentes nativos como FlatList oferecem virtualização automática para listas longas. Animações nativas com Animated API ou Reanimated criam transições suaves. A navegação com React Navigation proporciona padrões de UX familiares para cada plataforma.

BENEFÍCIOS COMBINADOS:

Benefício	ReactJS (Web)	React Native (Mobile)
Código Compartilhado	Hooks e lógica de negócio	Mesma base JavaScript/TypeScript
Ecossistema	npm com milhares de pacotes	Expo e bibliotecas nativas
Comunidade	Documentação extensa	Suporte ativo e atualizações
Performance	Virtual DOM otimizado	Bridge nativa otimizada
Manutenção	Componentes modulares	Hot reload e OTA updates

A escolha de ReactJS e React Native permite que nossa equipe trabalhe com tecnologias complementares, maximizando reuso de código e conhecimento. O padrão de componentização do React se alinha perfeitamente com a metodologia Scrum, permitindo desenvolvimento incremental e entregas frequentes de valor.

Marco	Descrição	Período	Entregáveis	Critérios Verificáveis de Aceite
M1 Fundação do Projeto	Estrutura inicial e planejamento concluídos	Semana 2	Repositório configurado, CI/CD ativo, protótipo Figma aprovado	Aplicação React inicial renderizando; pipeline executando com sucesso; protótipo navegável validado
M2 MVP Visual	Interface principal funcional	Semana 4	Login, Dashboard, gráficos de cotações	Usuário consegue realizar login e visualizar dashboard com dados simulados sem erros
M3 Integração Completa	Integração entre front-end e back-end	Semana 6	Consumo de API real, fluxo de compra e venda	Dados reais exibidos; simulação de compra/venda executada corretamente; sem falhas críticas
M4 Release Final	Aplicação pronta para entrega	Semana 8	Deploy em produção, documentação final	Sistema publicado, acessível online, documentação entregue e demonstração funcional realizada

9. INSTRUÇÕES PARA EXECUÇÃO DO PROJETO

Passo a passo para abrir e executar o projeto no VS Code:

1) Extrair o arquivo .zip

a. Faça download do arquivo **Plataforma_Criptmoeda;**
b. Clique com o botão direito → **Extrair tudo…** (ou use o WinRAR/7-Zip → **Extrair para…**).
c. Escolha uma pasta (ex.: Documentos) e extraia. Após extrair, você deve ter uma pasta com esta estrutura:

- public/
- src/
- index.html
- package.json
- pnpm-lock.yaml
- vite.config.js
- README.md

2) Abrir a pasta correta no VS Code

a. Abra o **VS Code**.
b. Vá em **File > Open Folder…** (Arquivo > Abrir Pasta…).
c. Selecione a pasta **que foi extraída** (a pasta "raiz" do projeto).
Importante: **não abra só a pasta src**.
Você precisa abrir a pasta que contém o package.json.

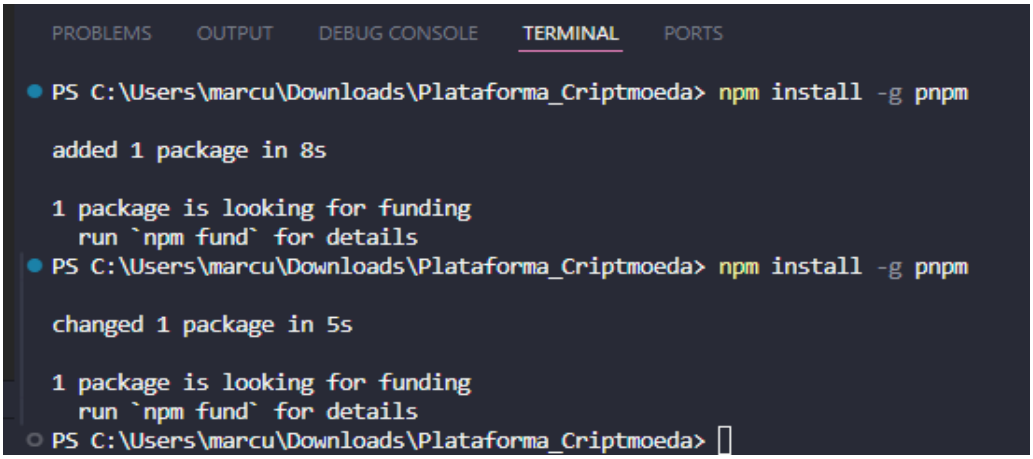
3) Abrir o terminal no VS Code

a. No VS Code, clique em **Terminal > New Terminal** (Terminal > Novo Terminal).
b. Confira se o terminal está na pasta certa (deve aparecer o caminho da pasta do projeto).

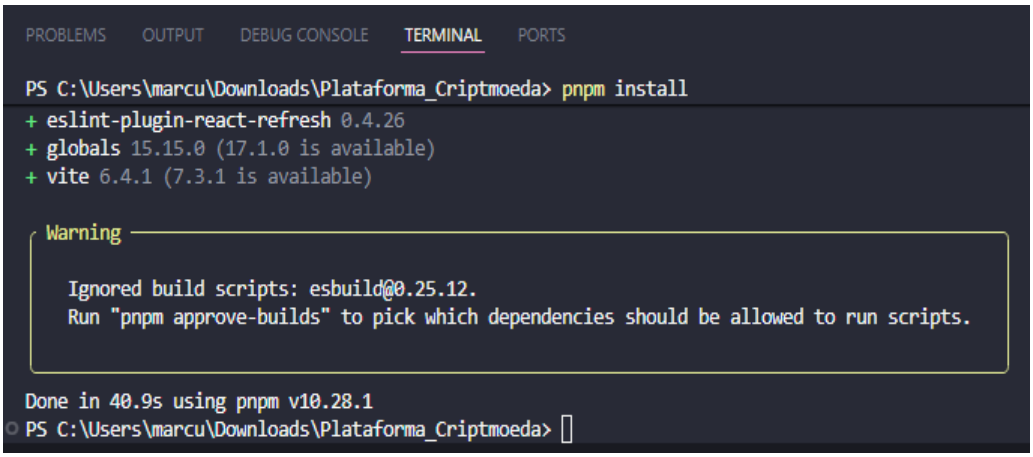
4) Instalar as dependências

No terminal do VS Code execute os seguintes comandos **NA PASTA DO PROJETO** para instalar o pnpm:

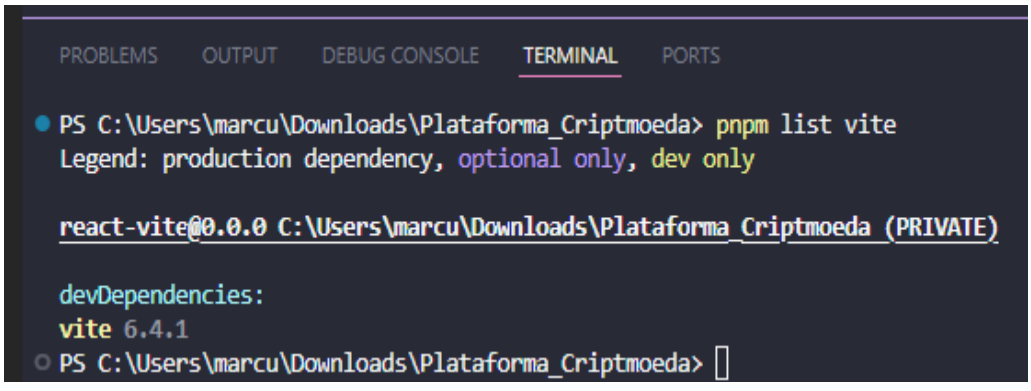
a. **npm install -g pnpm**
b. **pnpm install**



c. Ainda DENTRO DA PASTA DO PROJETO, rode: **pnpm install**



d. Verifique se o Vite está no projeto com o seguinte comando: **pnpm list vite**



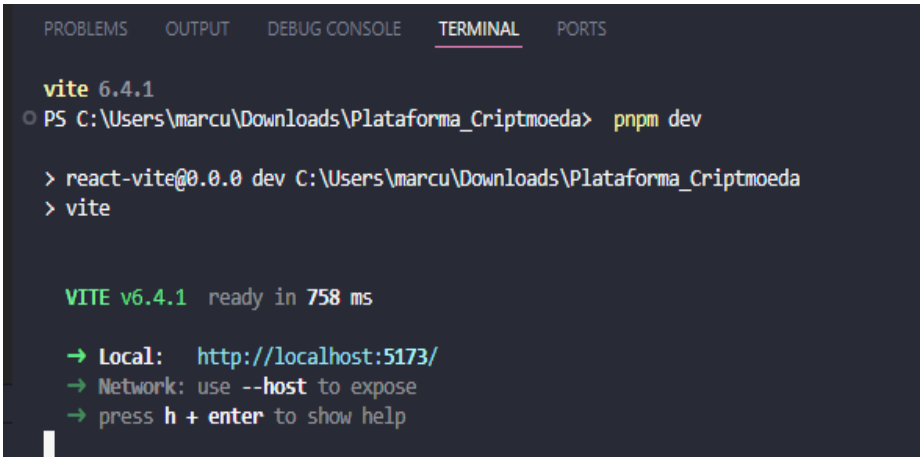
Se aparecer algo como “**vite@x.x.x”**, como na imagem acima, está tudo ok.

5) Rodar o projeto

No terminal do VS Code execute o seguinte comando para rodar o projeto:

a. **pnpm dev**

b.



6) Abrir no navegador

No terminal vai aparecer algo parecido com:

- Local: http://localhost:5173/

Abra esse endereço no navegador.

10. ESTRATÉGIA DE REUTILIZAÇÃO DE COMPONENTES

Neste projeto, a reutilização de componentes foi adotada como estratégia central para garantir organização, padronização visual e facilidade de manutenção da aplicação. A interface foi planejada com base no conceito de componentes reaproveitáveis, onde cada bloco visual segue uma estrutura previsível e pode ser renderizado dinamicamente a partir de dados.

A aplicação utiliza o React com foco em componentes funcionais e renderização baseada em arrays. Em vez de criar manualmente cada card ou seção repetida no HTML, foram definidos conjuntos de dados estruturados em arrays de objetos (como marketData, portfolioData, transactions e highlights). Esses dados são percorridos com o método map(), permitindo gerar automaticamente elementos visuais de forma consistente e escalável. Dessa forma, a criação de novos itens (por exemplo, adicionar uma nova criptomoeda no mercado) exige apenas a inclusão de um novo objeto no array correspondente, sem alterações na estrutura principal do layout.

Outro ponto relevante é o reaproveitamento de padrões visuais por meio de classes CSS compartilhadas. Elementos como botões (primary-button, outline-button, ghost-button), cards e grids utilizam estilos comuns que garantem identidade visual em toda a aplicação. Essa padronização reduz duplicação de código e facilita ajustes futuros, já que uma alteração em uma classe impacta automaticamente todos os componentes que a utilizam.

Além disso, a lógica de controle do aplicativo foi centralizada no App.jsx, com gerenciamento de estado por meio de useState, permitindo simular funcionalidades típicas de uma plataforma real (como autenticação, tema e navegação interna) sem depender de bibliotecas externas. Essa abordagem mantém o projeto mais didático, modular e fácil de evoluir, pois separa claramente responsabilidade de controle e apresentação.

Portanto, a estratégia de reutilização aplicada combina renderização dinâmica, organização por dados e reaproveitamento de estilos, tornando a aplicação mais flexível, consistente e preparada para futuras expansões.

11. Artefato: Backlog (Próximas Entregas – Plataforma de Criptomoeda)

Abaixo está o Backlog com as próximas atividades previstas, contemplando novas funcionalidades, testes, refactoring, segurança e qualidade. Ele pode ser usado como base para organizar as próximas sprints/entregas.

ID	Tipo	Atividades	Objetivo	Prioridade	Critérios de aceite	Dependências
BL-01	Funcionalidade	**Autenticação (Login/Logout) + persistência de sessão**	Permitir acesso seguro e manter usuário logado	Alta	Login válido para autenticar; logout encerra sessão; rota protegida bloqueia acesso sem login	BL-02
BL-02	Segurança	**Estrutura de segurança básica (tokens, storage seguro, proteção de rotas)**	Evitar acesso indevido e vazamento de dados	Alta	Rotas privadas exigem token; token expira; tratamento de sessão inválida	—
BL-03	Funcionalidade	**Tela de Carteira (Portfolio) com visão detalhada por ativo**	Exibir posições, custo médio, variação e total	Alta	Lista de ativos + total; cálculo consistente; layout responsivo	BL-06
BL-04	Funcionalidade	**Histórico de transações (compra/venda) + filtros**	Permitir auditoria do usuário	Alta	Filtros por data/tipo/ativo; paginação ou carregamento incremental	BL-06
BL-05	Funcionalidade	**Fluxo de Compra e Venda (ordem simples)**	Permitir simular/realizar operações	Alta	Form valida; confirma operação; atualiza carteira e histórico	BL-03, BL-04
BL-06	Integração	**Camada de API (services) + tratamento de erros padronizado**	Separar UI de dados e padronizar integração	Alta	Services isolados; estados de loading/erro; retries (se aplicável)	—

ID	Tipo	Atividades	Objetivo	Prioridade	Critérios de aceite	Dependências
BL-07	Refactoring	**Normalizar componentes e pastas (UI, pages, services, hooks)**	Melhorar manutenibilidade	Alta	Estrutura definida; imports consistentes; componentes reutilizáveis	—
BL-08	Refactoring	**Custom Hooks (ex.: useMarketData, usePortfolio)**	Reuso e redução de duplicação	Média	Hooks com loading/error/data; testes básicos	BL-06
BL-09	Qualidade	**Validações de formulários (compra/venda, login) + mensagens amigáveis**	Reduzir erros de uso	Média	Mensagens claras; impede envio inválido; máscara/formatadores	BL-01, BL-05
BL-10	UX/UI	**Estados de loading, empty state e skeletons**	Melhorar percepção de desempenho	Média	Sem “tela branca”; skeleton nos cards/listas; empty state com CTA	BL-06
BL-11	UI Responsiva	**Ajustes para desktop largo + mobile-first refinado**	Melhor experiência em diferentes telas	Média	Breakpoints; grids alinhados; sem overflow; nav ok no mobile	—
BL-12	Observabilidade	**Logs e monitoramento front (console controlado + tracking de erro)**	Facilitar debug e estabilidade	Média	Logs padronizados; captura de erros em requests	BL-06
BL-13	Testes	**Testes unitários (helpers, hooks, formatadores)**	Evitar regressões	Alta	Cobertura mínima definida; testes passam no CI local	BL-08
BL-14	Testes	**Testes de componentes (React Testing Library)**	Garantir UI estável	Alta	Renderiza cards; interações principais; snapshots (se usados)	BL-07
BL-15	Testes	**Testes E2E (fluxos: login → carteira → compra/venda)**	Validar jornada completa	Média	Cenários críticos automatizados; execução local documentada	BL-01, BL-05
BL-16	Performance	**Otimizações (memo, lazy loading, split de rotas)**	Melhorar tempo de carregamento	Baixa/Média	Lighthouse melhora; bundle reduz; sem travamentos na home	BL-07
BL-17	Segurança	**Hardening: sanitização básica + proteção contra XSS em campos**	Reduzir risco em inputs	Média	Inputs tratados; nada é injetado na UI sem escape	BL-09
BL-18	Documentação	**README técnico + guia de execução + padrões do projeto**	Facilitar manutenção e entrega acadêmica	Média	Passo a passo rodar; scripts; arquitetura; decisões	—

ID	Tipo	Atividades	Objetivo	Prioridade	Critérios de aceite	Dependências
BL-19	Funcionalidade	**Página de Ativo (detalhe do Bitcoin/Ethereum etc.)**	Exibir gráfico/indicadores e info do ativo	Baixa/Média	Abre via clique; mostra dados; tratamento quando falhar	BL-06
BL-20	Qualidade	**Padronização de estilo (ESLint/Prettier) + regras do projeto**	Consistência de código	Média	Lint sem erros; formatação automática; commit limpo	BL-07