CENTRO UNIVERSITÁRIO DE SANTA CATARINA

ENGENHARIA DE SOFTWARE

Alexandre Tessaro Vieira

Aplicação Web para Divulgação e Colaboração de Startups

Joinville

2025

Resumo

Neste documento é apresentada a especificação para o desenvolvimento de uma aplicação web voltada à conexão entre pessoas e ideias, com foco na criação e colaboração de projetos de startups. A aplicação permitirá a divulgação de iniciativas inovadoras e facilitará a formação de equipes multidisciplinares, atuando como um elo entre empreendedores e talentos. O sistema será construído com tecnologias modernas como React, Node.js e PostgreSQL, seguindo boas práticas de desenvolvimento ágil, integração e entrega contínuas (CI/CD), além dos princípios de Clean Code e SOLID. O objetivo central é estimular o ecossistema de inovação, promovendo colaboração eficiente desde as fases iniciais de uma ideia até sua concretização.

1. Introdução

Contexto

O ecossistema de inovação e startups é cada vez mais dinâmico e competitivo. Nesse cenário, a formação de boas parcerias é um fator crítico para o sucesso de novas empresas. Embora existam ferramentas colaborativas no mercado, são escassas aquelas que têm como foco principal conectar ideias a pessoas dispostas a desenvolvê-las desde os estágios iniciais de concepção.

Justificativa

Facilitar a conexão entre projetos e talentos é fundamental para impulsionar a inovação tecnológica. Criar uma aplicação que centralize essas conexões contribui para um ambiente empreendedor mais acessível, possibilitando que ideias com potencial se tornem viáveis ao encontrar os perfis certos para sua execução.

Objetivos

- Objetivo Principal:
  Desenvolver uma aplicação web que permita a divulgação de ideias e projetos, conectando seus idealizadores a pessoas interessadas em colaborar no desenvolvimento, formando equipes de maneira ágil e assertiva.
- Objetivos Secundários:
  Aplicar boas práticas de engenharia de software durante o desenvolvimento do sistema, incluindo:
  • Integração e entrega contínuas (CI/CD)
  • Desenvolvimento orientado a testes (TDD)
  • Princípios do Clean Code
  • Fundamentos da arquitetura SOLID

2. Descrição do Projeto

Tema do Projeto

O projeto propõe o desenvolvimento de uma aplicação web voltada à colaboração e divulgação de iniciativas inovadoras no ecossistema de startups. A plataforma terá como foco principal conectar ideias a pessoas com perfis e habilidades complementares, fomentando a criação de times e projetos com potencial de crescimento.

Principais funcionalidades:

- Exibição de perfis públicos de projetos, contendo descrição, objetivos, tecnologias desejadas e status atual.
- Sistema de matchmaking inteligente, conectando idealizadores a colaboradores com base em interesses e competências.
- Ferramenta de divulgação estruturada, facilitando a formação de equipes multidisciplinares e alinhadas às necessidades dos projetos.

Problemas a Resolver

- Dificuldade de idealizadores em encontrar pessoas interessadas e comprometidas em colaborar com suas ideias.
- Barreiras na formação de equipes técnicas e multidisciplinares, principalmente nas fases iniciais dos projetos.
- Ausência de uma plataforma centralizada para divulgar, acompanhar e interagir com projetos em estágio inicial.

Limitações

- Questões jurídicas, como elaboração de contratos, termos de parceria ou acordos legais, não serão abordadas nesta versão.
- A aplicação será desenvolvida exclusivamente para ambiente web responsivo; não está prevista a versão mobile nativa nesta fase.
- Integrações financeiras (ex: pagamentos, investimentos ou crowdfunding) não estarão disponíveis na primeira entrega.

3. Especificação Técnica

A especificação técnica é o alicerce que orienta a construção da aplicação, servindo como referência para desenvolvedores, designers, analistas e demais stakeholders ao longo do projeto. Esta seção descreve de forma detalhada os requisitos funcionais e não funcionais, as tecnologias adotadas, os fluxos esperados de interação, bem como as decisões arquiteturais que garantem a escalabilidade, segurança, usabilidade e performance da solução.

A proposta é desenvolver uma plataforma web colaborativa voltada ao ecossistema de startups, com foco na divulgação de projetos e conexão entre talentos e ideias. A seguir, são apresentados os requisitos do sistema, os padrões adotados para comunicação entre os componentes, os módulos principais e os critérios de qualidade que nortearão a implementação da solução.

3.1 Requisitos de Software

Lista de Requisitos de Software

Requisitos Funcionais (RF)

Código	Prioridade	Requisito Funcional
RF01	Must	O sistema deve permitir que o usuário se cadastre com e-mail e senha, validando duplicidade e formato.
RF02	Must	O sistema deve permitir login com e-mail e senha, retornando token JWT em caso de sucesso.
RF03	Must	O sistema deve permitir logout seguro, invalidando o token de acesso.
RF04	Should	O sistema deve permitir recuperação de senha via e-mail com token de redefinição.
RF05	Must	O sistema deve permitir que o usuário edite seu perfil (nome, bio, habilidades, redes sociais, imagem).
RF06	Must	O sistema deve permitir que o usuário crie, edite e exclua projetos, informando título, descrição, tecnologias e status.
RF07	Must	O sistema deve disponibilizar busca com filtros (tecnologia, categoria, status, etc.) para localizar projetos e usuários.
RF08	Must	O sistema deve permitir que o usuário envie solicitações para participar de projetos.
RF09	Must	O sistema deve permitir que o criador do projeto aceite, recuse ou bloqueie solicitações de participação.
RF10	Should	O sistema deve sugerir conexões entre usuários e projetos com base em interesses e habilidades (sistema de match).
RF11	Could	O sistema pode permitir a integração com APIs externas (LinkedIn, GitHub) para enriquecer o perfil do usuário.
RF12	Should	O sistema deve disponibilizar perfis públicos de usuários e projetos, acessíveis sem login completo.
RF13	Should	O sistema deve permitir comentários e feedback nos projetos, com possibilidade de resposta pelo criador.
RF14	Must	O sistema deve enviar notificações em tempo real sobre interações relevantes (solicitações, convites, updates).
RF15	Should	O sistema deve oferecer um painel de administração para moderação de usuários e projetos.
RF16	Must	O sistema deve registrar logs de ações críticas (cadastro, login, exclusão de projeto, etc.) para auditoria.
RF17	Could	O sistema pode exibir métricas no perfil (número de participações, avaliações, projetos ativos).

Requisitos Não-Funcionais (RNF)

Código	Prioridade	Requisito Não Funcional
RNF01	Must	A aplicação deve ser responsiva, adaptando-se corretamente a diferentes tamanhos de tela (desktop, tablet, mobile).
RNF02	Must	Todas as ações críticas (como login, criação, exclusão e atualização de dados) devem ser auditáveis e rastreáveis.
RNF03	Must	O sistema deve suportar no mínimo 1000 usuários simultâneos durante a fase inicial do projeto.
RNF04	Must	A autenticação deve seguir padrões seguros como OAuth 2.0 ou Firebase Auth, com proteção contra ataques comuns (brute-force, session hijacking).
RNF05	Should	O código deve seguir os princípios de Clean Code e SOLID, com cobertura de testes unitários, de integração e E2E baseados em TDD.
RNF06	Must	O tempo de resposta para ações principais (login, visualização de projetos, matches) deve ser inferior a 2 segundos.
RNF07	Must	O backend deverá ser preparado para escalabilidade horizontal utilizando Docker e Kubernetes, com suporte a autoscaling.
RNF08	Must	A base de dados deve ser protegida por mecanismos de segurança e contar com backups automáticos periódicos.
RNF09	Should	A aplicação deve ter integração e entrega contínuas (CI/CD) com pipelines configuradas no GitHub Actions.
RNF10	Must	Deve ser implementado controle de rate limiting para limitar o número de requisições por IP/usuário e prevenir abusos (ex: DDoS).
RNF11	Should	Em caso de falhas nas APIs externas (GitHub, LinkedIn), o sistema deve exibir mensagens amigáveis e aplicar estratégias de retry e fallback, além de registrar o erro em logs.
RNF12	Should	A aplicação deve ser otimizada para dispositivos móveis, com uso de lazy loading, compressão de imagens, e tags como picture e srcset.
RNF13	Must	A aplicação deve estar em conformidade com a LGPD, incluindo consentimento explícito, anonimização de dados sensíveis e direito ao esquecimento.
RNF14	Must	O sistema deve manter logs de auditoria com segregação de armazenamento (ex: banco, arquivos e LogDNA/ELK), acessíveis apenas por administradores.
RNF15	Should	Devem ser realizados testes automatizados de segurança para detecção de vulnerabilidades como XSS, CSRF e SQL Injection. A aplicação deve estar apta a receber pentests externos.
RNF16	Could	A aplicação pode oferecer um modo offline limitado, utilizando IndexedDB para exibição de conteúdo previamente acessado (ex: perfil do usuário, projetos visualizados).
RNF17	Could	O sistema pode exibir mensagens de erro detalhadas para desenvolvedores (modo dev), com stack trace, enquanto oculta detalhes em produção por segurança.
RNF18	Should	A camada de cache (Redis) deve ser utilizada para acelerar consultas de alta frequência (projetos e perfis muito acessados).

Representação dos Requisitos (UML)

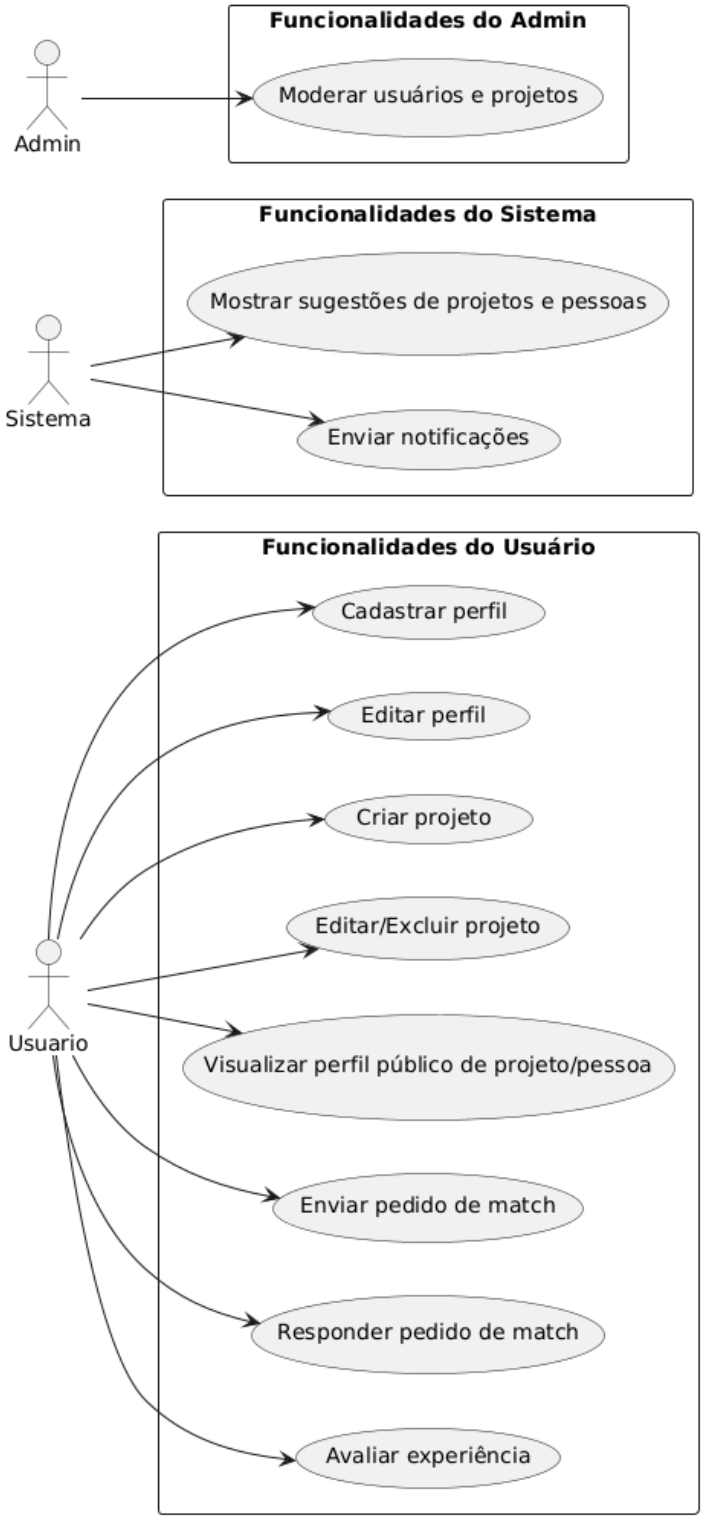

Figura 1 - Representação dos Requisitos (UML)

Principais Casos de Uso

Usuário

- Cadastrar perfil: Criar uma conta preenchendo informações pessoais, profissionais e interesses.
- Editar perfil: Atualizar nome, habilidades, redes sociais, foto e outras informações do perfil.
- Criar projeto: Registrar novo projeto com título, descrição, objetivos, status e tecnologias desejadas.
- Editar/Excluir projeto: Modificar ou remover projetos criados pelo usuário.
- Enviar pedido de match: Demonstrar interesse em participar de um projeto ou convidar outros usuários para colaborar.
- Responder pedido de match: Aceitar ou recusar convites recebidos para participar de projetos.
- Visualizar perfil público: Acessar informações detalhadas de projetos ou usuários sem necessidade de autenticação completa.
- Avaliar experiência (feedback): Fornecer feedback sobre colaborações concluídas ou projetos encerrados.

Sistema

- Mostrar sugestões de projetos e pessoas: Gerar recomendações baseadas em interesses e histórico de navegação do usuário.
- Enviar notificações: Notificar automaticamente sobre novos matches, convites, atualizações de projetos e mensagens.

Administrador

- Moderar usuários e projetos: Acessar painel administrativo para analisar, bloquear ou remover conteúdos ou perfis inadequados.

3.2 Considerações de Design

Discussão sobre as escolhas de design:

- Front-end: React
  Usarei React por ser uma biblioteca madura, consolidada, com um vasto ecossistema de pacotes. Ele proporciona uma UI moderna, reativa e facilmente escalável, aumentando a velocidade de desenvolvimento. Também permite o uso de service workers e IndexedDB para suportar navegação offline básica.
- Back-end: Node.js + Express
  Essa stack oferece simplicidade na criação de APIs, velocidade de desenvolvimento e uma integração eficiente com bancos de dados relacionais. É uma solução madura e escalável, com bom suporte para aplicações que exigem desempenho em tempo real, se necessário.
- API: REST
  Optei pelo REST por ser uma alternativa simples, robusta e amplamente adotada, especialmente para comunicação com serviços externos como GitHub e LinkedIn. O uso de GraphQL chegou a ser considerado pelas vantagens em consultas específicas, mas foi deixado de lado nesta primeira versão para favorecer a simplicidade da arquitetura inicial.
- Segurança:
  A autenticação será feita com JWT ou OAuth 2.0, utilizando access tokens e refresh tokens para garantir sessões seguras e com expiração adequada. As senhas serão armazenadas com hashing seguro (bcrypt), e as entradas do usuário serão sanitizadas para prevenir vulnerabilidades como XSS e SQL Injection (seguindo práticas OWASP).

Versão Inicial da Arquitetura:

- Front-end (React):
  Aplicação SPA que se comunica com a API por meio de requisições REST, com suporte offline básico utilizando IndexedDB para cache local de conteúdos visualizados.
- API (Node.js + Express):
  Responsável pelo CRUD de usuários e projetos, autenticação, sistema de matchmaking, envio de mensagens, notificações e controle de permissões.
- Banco de Dados (Postgres):
  O banco de dados escolhido é o PostgreSQL, por ser robusto, seguro, open-source e altamente confiável para aplicações que demandam estrutura relacional. É ideal para garantir integridade referencial, realizar consultas complexas com joins e suportar transações ACID. Sua compatibilidade com ferramentas modernas (ORMs como Prisma, Sequelize e suporte a extensões como PostGIS) o torna uma escolha sólida para o cenário da aplicação.

Ele será responsável por armazenar entidades como usuários, projetos, relações de interesse, mensagens, comentários e logs de auditoria.

- Caching (Redis):
  Utilizado como cache de alto desempenho para consultas frequentes, como listagem de projetos populares, sugestões de match e perfis acessados com frequência. Adotado no padrão *cache-aside*.
- Infraestrutura com Docker + Kubernetes: Todos os serviços serão containerizados com Docker. O deploy será realizado em um cluster Kubernetes, permitindo escalabilidade horizontal através do Horizontal Pod Autoscaler com métricas de CPU/memória. O Nginx Ingress Controller gerenciará o tráfego externo com balanceamento de carga automático.
- CI/CD (GitHub Actions):
  Pipeline configurado para realizar testes automatizados, análise estática, builds e deploy contínuo. A esteira permitirá entregas frequentes e controle de qualidade desde o início do projeto.

Padrões e Práticas Arquiteturais:

- Clean Architecture (Layers: Presentation, Application, Domain, Infrastructure):
- Separação de responsabilidades clara, permitindo escalabilidade, testes e manutenções facilitadas.
- Princípios SOLID:
  Aplicados para promover coesão, baixo acoplamento, reutilização e testabilidade de código.
- Repository Pattern
  Centraliza e abstrai o acesso ao banco de dados, facilitando a troca futura por outra base e a manutenção do código.
- Controller – Service – Repository
  Organizar o serviço nas camadas de controller, service e repository, aumentando a separação de responsabilidades.
- DTOS (Data Transfer Objects)
  Usar DTOs para transferir apenas os dados necessários para o Front-end, aumentando a eficácia, a seguridade e a simplicidade nas comunicações.

Estratégia de Testes Automatizados

Para assegurar a qualidade do software, prevenir regressões e promover confiança no processo de desenvolvimento contínuo, será adotada uma estratégia de testes automatizados desde os estágios iniciais, alinhada à prática de Test Driven Development (TDD). Essa abordagem permitirá validar requisitos funcionais e não funcionais com rapidez e precisão.

As ferramentas escolhidas contemplam diferentes níveis de testes:

- Jest
  Utilizado para testes unitários, focando na lógica de negócio isolada (ex: validação de entrada, regras de associação entre projetos e usuários, cálculo de match, etc.). É amplamente adotado no ecossistema Node.js, com boa performance e suporte a mocks, spies e coverage.
- Supertest
  Usado para testes de integração da API, validando endpoints REST do backend Express. Os testes verificarão comportamento completo das rotas, incluindo autenticação, persistência no banco de dados e resposta adequada a erros.
- Cypress
  Aplicado em testes end-to-end (E2E) para simular a jornada real do usuário na interface. Os fluxos testados incluirão login, cadastro, criação de projetos, envio de match, entre outros. A ferramenta permite validar tanto o comportamento funcional quanto o visual da aplicação.

Todos os testes serão integrados à esteira de CI/CD (GitHub Actions), executando automaticamente a cada commit ou pull request. A pipeline será responsável por:

- Executar testes unitários, de integração e E2E.
- Verificar cobertura de testes.
- Validar padrões de código (lint).
- Garantir que o build esteja íntegro antes do deploy.

Essa estratégia contribui diretamente para um processo de desenvolvimento ágil, seguro e sustentável, promovendo qualidade contínua em todas as entregas.

Segurança, LGPD e Logs de Auditoria

A aplicação será desenvolvida com foco em segurança e proteção de dados, seguindo diretrizes da LGPD (Lei Geral de Proteção de Dados) e boas práticas de segurança da OWASP.

Conformidade com a LGPD:

- Coleta de dados será feita mediante consentimento explícito do usuário (checkbox no momento do cadastro).
- Dados sensíveis (como e-mail e perfil profissional) serão armazenados com criptografia e exibidos com controle de visibilidade.
- O sistema implementará o direito ao esquecimento, permitindo que o usuário solicite a exclusão total dos seus dados.

- Os dados utilizados para estatísticas ou análises internas poderão ser anonimizados de forma irreversível.

Logs de Auditoria:

- Todas as ações críticas realizadas por usuários e administradores (login, alterações, exclusões, matches, denúncias) serão registradas.
- Os logs conterão: ID do usuário, ação realizada, timestamp, IP e status.
- Os dados de auditoria serão armazenados em:
    - Banco de dados relacional (PostgreSQL) para rastreamento transacional;
    - Arquivos de log (rotacionados por tempo);
    - Eventualmente integrados a ferramentas como LogDNA ou ELK Stack, dependendo da infraestrutura disponível.

Política de Refresh Token:

- Será adotada a autenticação via JWT (access token + refresh token):
    - Access token: expiração curta (15 minutos).
    - Refresh token: expiração de 7 dias, armazenado em cookie HTTPOnly + Secure, evitando acesso via JavaScript.
- O backend poderá invalidar tokens de forma explícita em casos de logout ou suspeita de comprometimento.

Isso permitirá que o modelo seja facilmente compartilhado, aumentando o grau de compreensão da arquitetura de desenvolvimento.

3.3 Arquitetura Visual – Diagramas C4

Para representar graficamente a arquitetura do sistema, foram utilizados os modelos C4 (Contexto, Contêiner, Componente e Código). Esses diagramas ajudam a visualizar, em diferentes níveis de abstração, como os elementos do sistema se relacionam entre si e com agentes externos

3.3.1 Diagrama de Contexto (Nível 1)

Mostra como o sistema interage com usuários e sistemas externos.

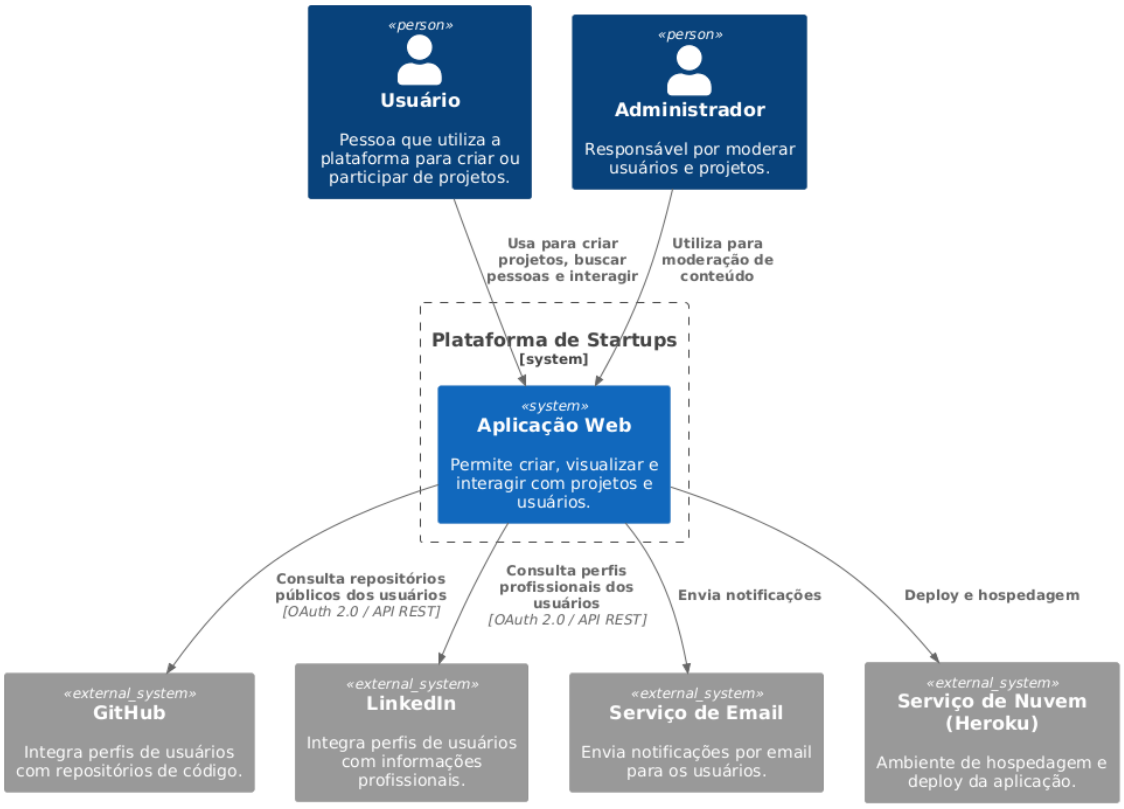

Figura 2 - Diagrama de Contexto (C1)

3.3.2 Diagrama de Contêineres (Nível 2)

Apresenta os principais contêineres da arquitetura: frontend, backend (API) e banco de dados.

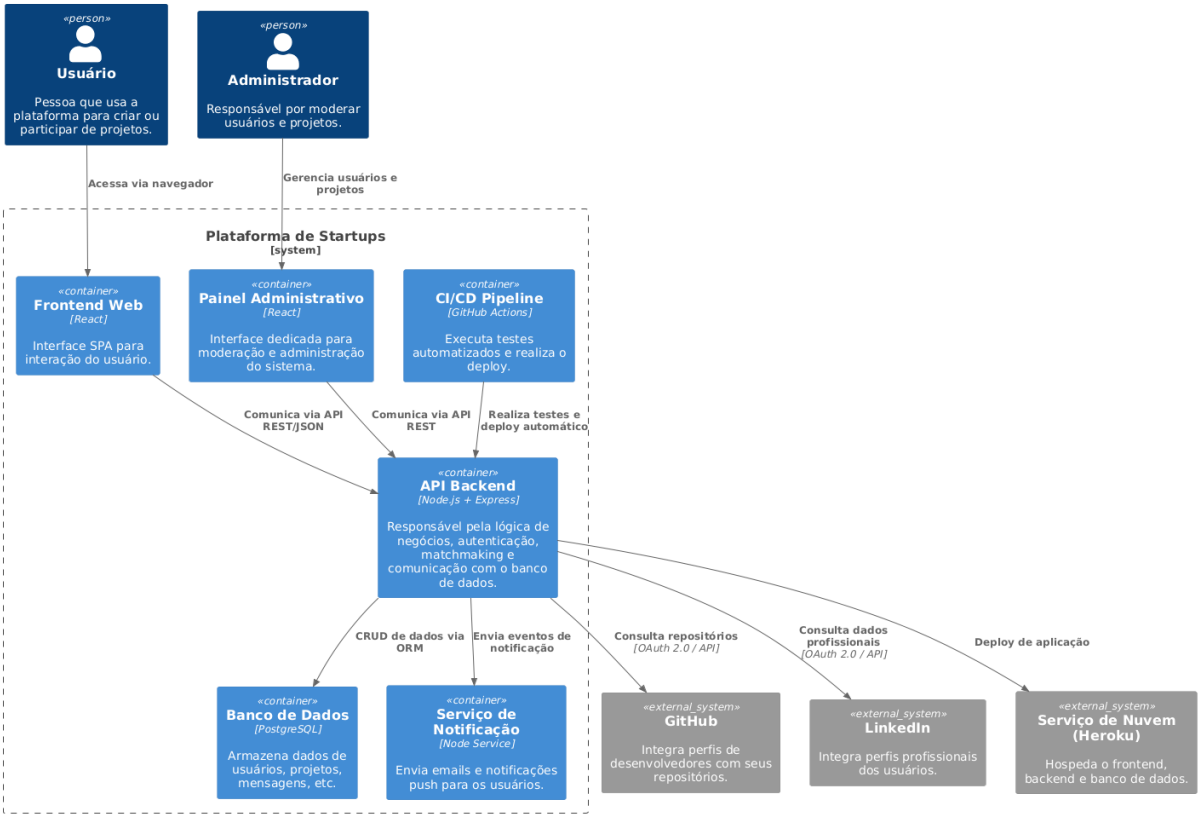

Figura 3 - Diagrama de Contêineres (C2)

3.3.3 Diagrama de Componentes (Nível 3)

Detalha os componentes da API, como controllers, services, repositories, e como se comunicam entre si.

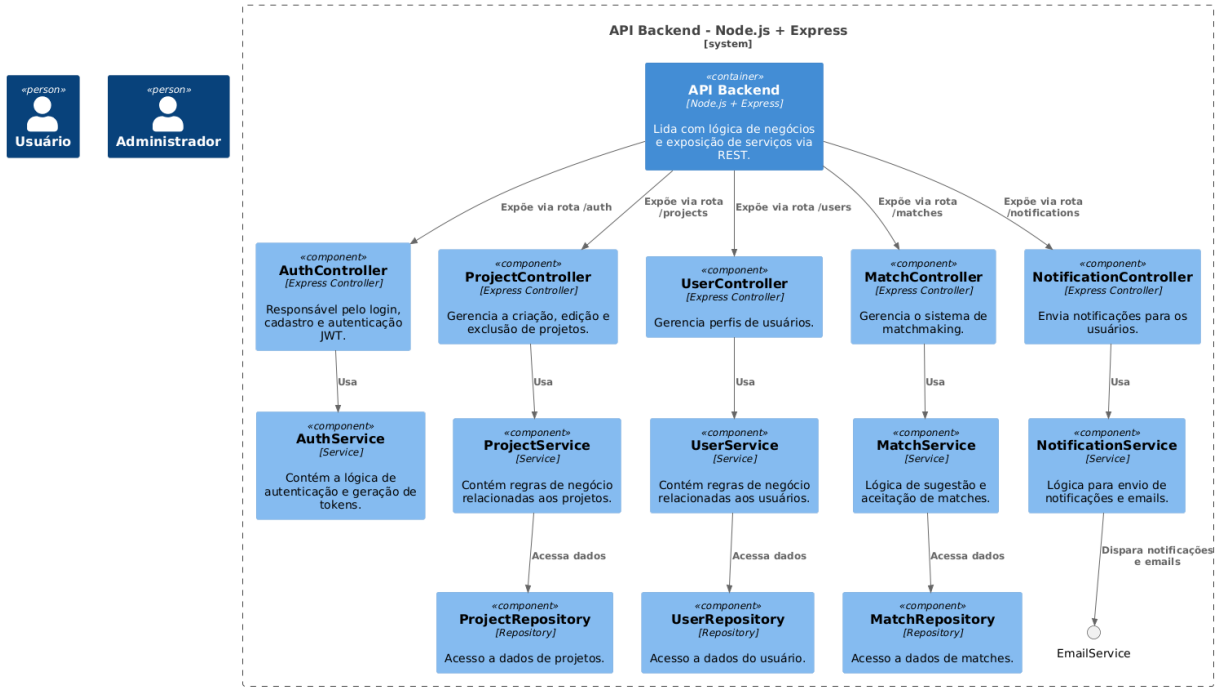

Figura 4 - Diagrama de Componentes (C3)

3.3.4 Diagrama de Código (Nível 4)

Explora detalhes internos de um componente específico, como as classes e métodos principais do módulo de match, autenticação ou outro núcleo.

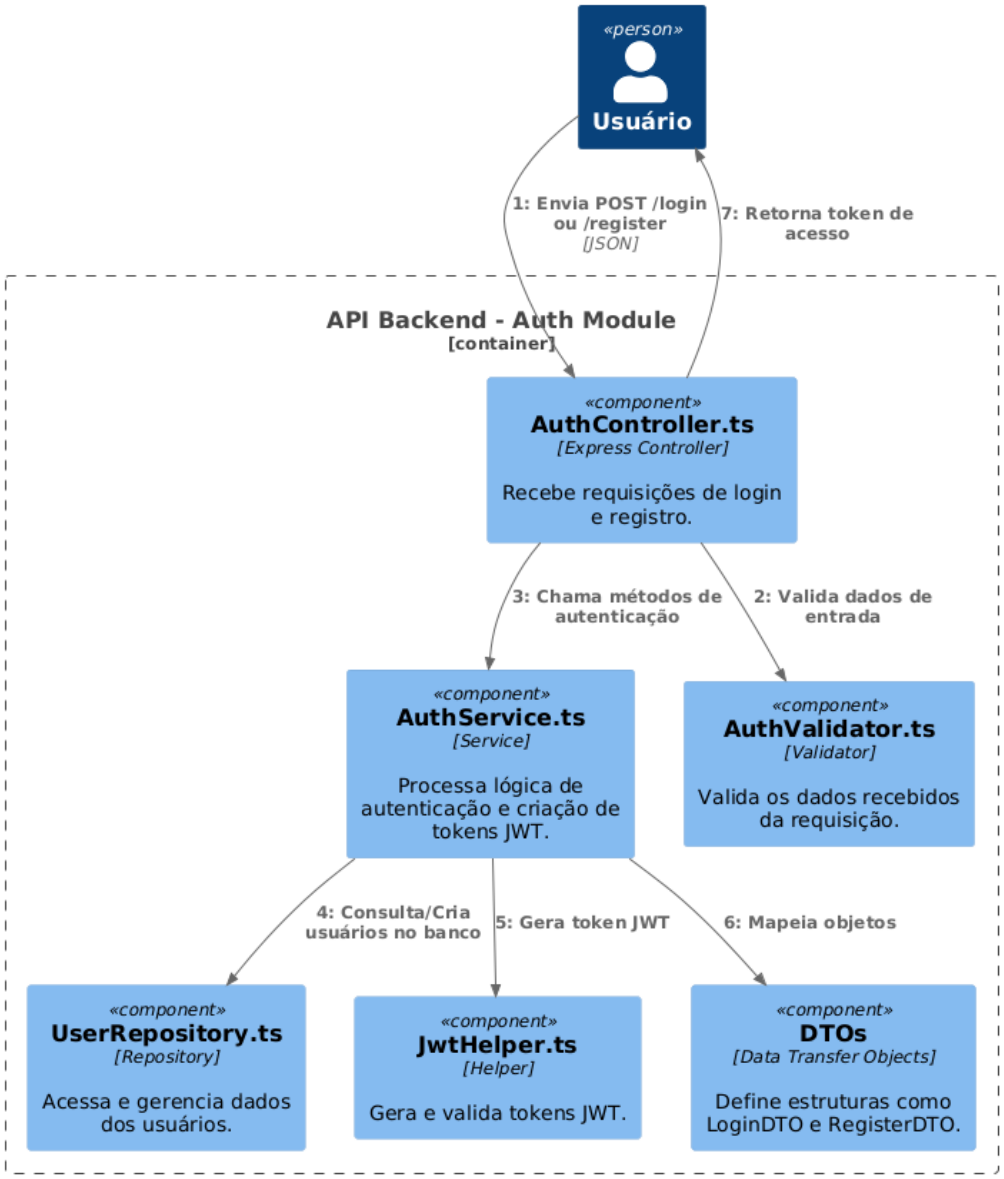

Figura 5 - Diagrama de Código (C4)

4. Próximos Passos:

Fase 1: Preparação e CI/CD

Semana	Atividades
1	Criar repositório GitHub com estrutura mínima de pastas (frontend, backend, infra, docs) Configurar Docker + Docker Compose para Node, React e PostgreSQL
2	Criar CI/CD com GitHub Actions: testes, lint, build Testar deploy simulado em ambiente de staging (ex: Render ou Railway)
3	Criar README inicial com setup do projeto Criar modelo de PR + arquivos .env.example Iniciar board no Trello ou Jira com épicos e tarefas priorizadas

Fase 2: Protótipos, autenticação e CRUD básico

Semana	Atividades
4	Criar wireframes no Figma (low-fidelity e high-fidelity) Validar com usuários ou colegas se possível
5	Criar módulo de autenticação (JWT + Refresh Tokens) Configurar login, registro, recuperação de senha
6	Implementar CRUD de usuários (com testes unitários usando Jest) Definir DTOs e organizar controllers/services/repositories
7	Implementar CRUD de projetos Incluir testes unitários e integração com banco de dados (PostgreSQL)

Fase 3: Matchmaking, mensagens e moderação

Semana	Atividades
8	Implementar sistema de match (enviar/aceitar convites) Criar regras de negócios básicas para sugestão de usuários
9	Criar módulo de mensagens privadas Incluir notificações básicas (via email ou toast in-app)
10	Criar painel de moderação para admin (usuários e projetos) Lógica para denunciar/bloquear perfis
11	Validar testes E2E com Cypress Incluir testes de fluxo completo com autenticação, CRUD e match

Fase 4: Segurança, LGPD e otimizações

Semana	Atividades
12	Adicionar conformidade com a LGPD (termos, política de privacidade, consentimento) Criar sistema de auditoria (logs de ações críticas)
13	Implementar Redis para caching (ex: usuários mais acessados, projetos populares) Definir limites de taxa (rate limit com express-rate-limit)
14	Criar rotina de backup automático no banco (PostgreSQL + cron) Otimizar frontend para dispositivos móveis (lazy loading, compressão de imagens)
15	Rodar testes básicos de segurança: validação de JWT, proteção contra XSS/CSRF, validação de entrada (OWASP)

Fase 5: Finalização, deploy e apresentação

Semana	Atividades
16	Rodar todos os testes automatizados Refatorar código seguindo padrões SOLID e Clean Code
17	Realizar deploy da aplicação com Docker em produção (Heroku, Railway ou VPS) Documentar o processo de deploy (arquivo deploy.md)
18	Finalizar documentação técnica: requisitos, arquitetura, testes, decisões de design
19	Preparar pitch/apresentação do projeto Gravar demo funcional ou apresentação interativa (opcional) Entregar versão final validada

5. Referências:

Clean Code – Robert C. Martin

Princípios SOLID – Fundamentos da engenharia de software orientados a objetos

Docker Compose Documentation – Define e executa aplicações multi-contêiner com Docker.

JWT (JSON Web Tokens) – Introdução ao JWT e autenticação segura baseada em tokens.

React Documentation – Biblioteca JavaScript para construir interfaces de usuário.

Node.js + Express Documentation – Framework web minimalista e rápido para Node.js.

PostgreSQL Documentation – O banco de dados relacional open source mais avançado do mundo.

Redis Documentation – Banco de dados em memória, usado para cache e mensageria.

Jest Documentation – Framework para testes unitários em JavaScript.

Supertest Documentation – Biblioteca para testes de integração em APIs Node.js.

Cypress Documentation – Framework para testes end-to-end (E2E) em aplicações web.

OAuth 2.0 – Protocolo para autorização segura em APIs.

LGPD – Lei Geral de Proteção de Dados – Regulamentação brasileira sobre proteção de dados pessoais.

OWASP Cheat Sheet Series – Boas práticas para segurança de aplicações web.

GitHub Actions Documentation – Automação de integração e entrega contínua (CI/CD).

Kubernetes Documentation – Orquestração de contêineres para escalabilidade e gerenciamento de aplicações.

6. Apêndices:

Os apêndices complementam a documentação principal, oferecendo materiais de apoio visual, técnico e operacional que reforçam a compreensão do projeto e facilitam sua manutenção futura.

- Modelo Entidade-Relacionamento (ER):
  Diagrama representando a estrutura lógica do banco de dados, incluindo entidades, atributos e relacionamentos, utilizado como base para modelagem e implementação no PostgreSQL.
- Casos de Uso Detalhados:
  Documentação estruturada dos principais fluxos de interação entre os usuários (atores) e o sistema, incluindo descrições de cenários principais e alternativos, pré-condições, pós-condições e exceções.
- Wireframes das Telas:
  Protótipos de baixa e média fidelidade desenvolvidos em ferramentas como Figma, ilustrando a navegação e o layout da interface da aplicação (SPA - Single Page Application).
- Mapeamento de Requisitos e Funcionalidades:
  Tabela traçando a relação entre os requisitos funcionais e os módulos/componentes do sistema que os implementam, facilitando a rastreabilidade e priorização no desenvolvimento.
- Checklist de Testes Automatizados e Manuais:
  Relação dos testes realizados, organizados por tipo (unitário, integração, E2E), ferramentas utilizadas (Jest, Supertest, Cypress) e critérios de aceitação validados por funcionalidade.
- Plano de Deploy e Operação:
  Descrição do processo de publicação da aplicação, incluindo o uso de contêineres Docker, pipelines de CI/CD com GitHub Actions, estratégias de versionamento, ambientes (homologação/produção), e possíveis plataformas de hospedagem (como Heroku, Render, Railway ou AWS).

7. Avaliações de Professores:

Assinatura: _____

Assinatura: _____

Assinatura: _____