**Web Aplicação de Divulgação e Colaboração de Startups**

Alexandre Tessaro Vieira

Engenharia de Software

01/01/2025

**Resumo**  
Este documento descreve o desenvolvimento de uma aplicação no qual tem como objetivo conectar ideias a pessoas para a criação de startups. Primordialmente, a aplicação permitirá a divulgação de ideias de projeto e facilitará a formação de equipes, criando assim uma ponte entre ideias e pessoas. Pretende-se usar tecnologias novas como React, Node.js e PostgreSQL, obedecendo práticas de desenvolvimento ágil, integração e entrega contínuas (CI/CD) e princípios de Clean Code. O projeto visa fomentar um ecossistema de inovação, incentivando a colaboração e a estruturação de times.

1. **Introdução**

**Contexto**

O ambiente de inovação e startups é cada vez mais dinâmico e competitivo. No ambiente corporativo, a busca por bons parceiros é essencial para o desenvolvimento e a consolidação de novas empresas. No entanto, ainda não existe nenhuma plataforma dedicada exclusivamente em promover a conexão entre pessoas e ideias.

**Justificativa**

Facilitar a conexão entre projetos e pessoas é fundamental para fomentar a inovação tecnológica. Portanto criar uma plataforma que centralize essas conexões gera um ecossistema empreendedor para projetos que talvez jamais fossem viabilizados.

**Objetivos**

* Objetivo Principal: Desenvolver uma aplicação web que permita a divulgação de projetos, conectando quem a idealizou com pessoas interessadas em participar do desenvolvimento, promovendo assim a formação de equipes.
* Objetivos Secundários: Implementar boas práticas de engenharia de software ao longo do desenvolvimento, incluindo integração e entregas contínuas (CI/CD), Test-Driven Development (TDD), princípios do Clean Code e os fundamentos do SOLID.

1. **Descrição do Projeto**

**Tema do Projeto**

O projeto propõe o desenvolvimento de uma plataforma voltada à colaboração e divulgação de iniciativas no ecossistema de startups. A aplicação contará com funcionalidades como:

* Perfis públicos dos projetos, com informações sobre descrição, objetivo e status atual.
* Sistema de matchmaking para conectar pessoas com ideias a outras interessadas em participar do desenvolvimento.
* Ferramenta de divulgação para facilitar a formação de equipes com talentos alinhados às necessidades de cada projeto.

**Problemas a Resolver**

* Dificuldade enfrentada por criadores de ideias para encontrar pessoas interessadas e dispostas a contribuir com seus projetos.
* Obstáculos na formação de equipes qualificadas e multidisciplinares.
* Falta de uma plataforma centralizada para divulgar e acompanhar protótipos e informações sobre projetos em fase inicial.

**Limitações**

* Questões jurídicas, como elaboração de contratos ou acordos legais, não serão tratados nessa versão do projeto.
* A aplicação será focada exclusivamente em ambiente web responsivo; não está prevista a criação de um aplicativo mobile nativo neste estágio.
* Funcionalidades relacionadas a integrações financeiras, como sistemas de pagamento, não estarão disponíveis na primeira versão.

1. **Especificação Técnica**
   1. **Requisitos de Software**

**Lista de Requisitos de Software**

* **Requisitos Funcionais (RF)**
  + RF01: Permitir cadastro, login e autenticação de usuários e projetos.
  + RF02: Permitir a criação, edição e exclusão de projetos por seus criadores.
  + RF03: Dispobinilizar sistema de busca com filtros avançados para localizar projetos, usuários e oportunidades.
  + RF04: Implementar sistema de “match” entre criadores de ideias e usuários interessados com base em interesses, habilidades e objetivos.
  + RF05: Integrar perfis de usuários com APIs externas, como LinkedIn e GitHub.
  + RF06: Permitir que usuários solicitem participação em projetos e que os criadores possam aceitar ou recusar candidaturas.
  + RF07: Oferecer área de perfil público para usuários e projetos, com informações detalhadas, histórico e status.
  + RF08: Notificar os usuários sobre interações relevantes (convites, matches, atualizações de projeto, etc.).
  + RF09: Dispobinilizar painel de administração para moderar conteúdos e gerenciar usuários.
  + RF10: Suportar comentários ou seções de feedback nos projetos.
* **Requisitos Não Funcionais (RNF)**
  + RNF01: A aplicação deverá ser responsiva, adaptando-se a diferentes tamanhos de tela (desktop, tablet, mobile).
  + RNF02: Todas as ações críticas (criação, exclusão, alterações de dados) devem ser auditáveis e rastreáveis.
  + RNF03: O sistema deverá suportar ao menos 1000 usuários simultâneos na fase inicial.
  + RNF04: A autenticação deverá ser segura, utilizando padrões como Oauth 2.0 ou Firebase Authentication.
  + RNF05: O código deverá seguir princípios de Clean Code e SOLID, com cobertura de testes automatizados (TDD).
  + RNF06: O sistema deverá ter tempo de resposta inferior a 2 segundos para as principais interações do usuário.
  + RNF07: O backend deverá estar preparado para escabilidade horizontal (ex: uso de containers e orquestração com Docker/Kubernetes).
  + RNF08: A base de dados deverá ser segura e com backup periódico automatizado.
  + RNF09: A aplicação deverá ter integração contínua (CI) e entrega contínua (CD) configuradas para facilitar testes e deploys frequentes.

**Representação dos Requisitos (UML)**

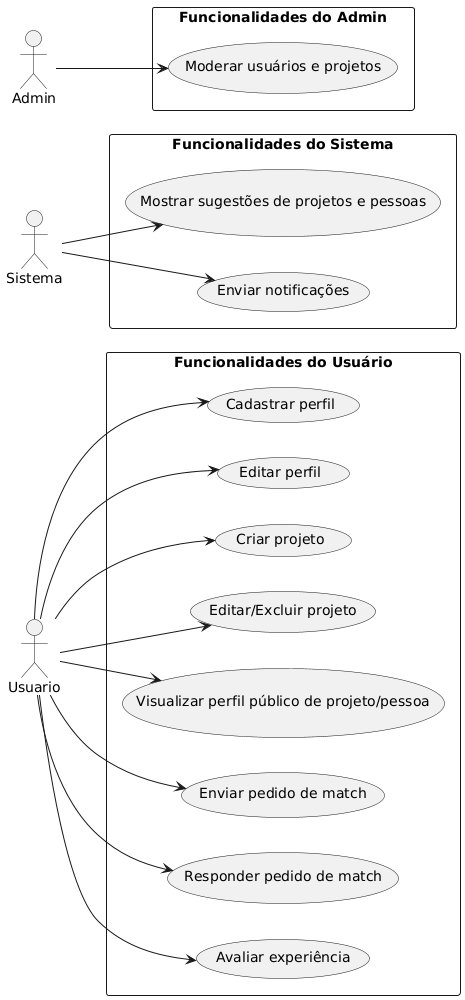


Figura 1 - Representação dos Requisitos (UML)

**Principais Casos de Uso**

* [Usuário] – Cadastrar perfil

Permite que o usuário crie uma conta e preencha suas informações pessoais, profissionais e interesses.

* [Usuário] – Editar perfil

Permite atualizar informações como nome, habilidades, redes sociais e foto.

* [Usuário] – Criar projeto

Permite que o usuário crie um novo projeto com título, descrição, objetivos, status atual e tecnologias desejadas.

* [Usuário] – Editar/Excluir projeto

Permite modificar ou remover projetos que o usuário criou.

* [Usuário] – Enviar pedido de match

Permite demonstrar interesse em participar de um projeto ou convidar alguém para colaborar.

* [Usuário] – Responder pedido de match

Aceitar ou recusar convites recebidos.

* [Sistema] – Mostrar sugestões de projetos e pessoas

Com base em interesses e histórico de navegação do usuário;

* [Sistema] – Enviar notificações

Notificações automáticas sobre matches, atualizações e mensagens.

* [Admin] – Moderar usuários e projetos

Acesso a painel administrativo para análise, bloqueio ou remoção de conteúdos ou perfis inadequados.

* [Usuário] – Visualizar perfil público de projeto/pessoa

Permite ver informações detalhadas de projetos ou usuários, sem necessidade de login completo.

* [Usuário] – Avaliar experiência (feedback)

Permite dar feeback sobre uma colaboração concluída ou projeto encerrado.

* 1. **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.

* Back-end: Node.js + Express

Essa stack oferece simplicidade na criação de APIs, velocidade de desenvolvimento e uma integração tranquila com bancos de dados relacionais. É uma saída madura, que se destaca principalmente na construção de aplicações em tempo real, se necessário.

* API: REST

Optei pelo REST por ser uma alternativa simples, robusta e facilmente integrável, principalmente para a comunicação com outros serviços, como o GitHub, que é principalmente REST.

Ainda assim, o GraphQL chegou a ser considerado pelas vantagens nas buscas, já que ele proporciona maior eficácia na seleção de determinados conteúdos.

* Segurança:

Implementação de autenticação com JWT ou OAuth 2.0.

Uso de refresh de tokens junto de acess tokens para maior segurança.

Armazenamento seguro de senhas (bcrypt) e tratamento das entradas para prevenir vulnerabilidades (OWASP).

**Versão Inicial da Arquitetura:**

Front-end (React): Uma SPA que se comunica com o back-end via API.

API (Node/Express): Realiza o CRUD, a autenticação, o match, o envio de mensagens e o tratamento das solicitações.

Banco de Dados (Postgres): Armazena usuários, projetos, relações de interesse, mensagens, comentários e outras informações.

Docker: Containeriza tanto o back-end quanto o banco de dados, aumentando a portabilidade e facilitando o deploy.

Continuous Integration/Continuos Deployment (Github Actions): Realiza automaticamente os testes e faz o deploy nas instâncias de produção.

**Padrão de Arquitetura:**

* Clean Architectural Layers (Presentation, Application, Domain, Infrastructure)

Segregar responsabilidades em camadas para facilitar a manutenção, a escalabilidade e a organização do código.

* SOLID

Seguir os princípios SOLID para garantir um design de software robusto, coeso, facilmente testável e de baixo acoplagem.

* Repository Pattern

Utilizar o Repository Pattern para concentrar o acesso ao banco de dados, aumentando a coesão e facilitando a manutenção.

* 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.

**Modelos C4:**

* C4 Nível 1 (Contexto)

Mostrará o usuário, a plataforma, o banco de dados, o GitHub para integração e outros agentes envolvidos.

* C4 Nível 2 (Container)

Destacará os contêineres que compõem o sistema, como o Front-end (React), a API (Node), o banco de dados (Postgres) e o Docker, mostrando suas responsabilidades.

* C4 Nível 3 (Componente)

Explicará a organização interna de cada contêiner, mostrando as responsabilidades das classes controller, servisse, repository, etc.

* C4 Nível 4 (Código)

Foca no detalhamento de uma parte crítica da aplicação, como o módulo de autenticação ou sistema de matchmaking, ilustrando suas classes, métodos e fluxos internos.

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

* 1. **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

* + 1. **Diagrama de Contexto (Nível 1)**

Mostra como o sistema interage com usuários e sistemas externos.

A diagram of a software development process

AI-generated content may be incorrect.

Figura 2 - Diagrama de Contexto (C1)

* + 1. **Diagrama de Contêineres (Nível 2)**

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

A diagram of a software development process

AI-generated content may be incorrect.

Figura 3 - Diagrama de Contêineres (C2)

* + 1. **Diagrama de Componentes (Nível 3)**

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

A diagram of a program

AI-generated content may be incorrect.

Figura 4 - Diagrama de Componentes (C3)

* + 1. **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.

**A diagram of a project

AI-generated content may be incorrect.**

Figura 5 - Diagrama de Código (C4)

1. **Próximos Passos:**

* Criar o figma com protótipos das telas (low-fidelity e high-fidelity)
* Organizar o desenvolvimento no Trello ou Jira (criação de boards, cards, atividades)
* Preparar o ambiente de desenvolvimento (Docker Compose, configurações de banco de dados, variáveis de ambiente, etc.)
* Implementar o modelo de autenticação (JWT)
* Implementar o CRUD de usuários e projetos
* Implementar o match e o envio de mensagens
* Implementar o painel de moderação (com gerenciamento de conteúdos e usuários)
* Implementar os testes automatizados (TDD)
* Realizar o deploy para um serviço na nuvem (Docker + Heroku)
* Realizar uma revisão final, corrigindo eventuais problemas, otimizando desempenho e aumentando a estabilidade.

1. **Referências:**

* Clean Code – Martin, Robert C.
* Princípios SOLID – Fundamentos da engenharia de software orientados a objetos.
* Docker Compose Documentation – Define and run multi-container applications with Docker.
* JWT (JSON Web Tokens) – Introduction to JWT and secure authentication.
* React Documentation – A JavaScript library for building user interfaces.
* Node.js + Express Documentation – Fast, unopinionated, minimalist web framework for Node.js.
* PostgreSQL Documentation – The world’s most advanced open source relational database.

1. **Apêndices:**

* Modelo Entidade-Relacionamento (ER):

Diagrama representando a estrutura e os relacionamentos das entidades no banco de dados.

* Casos de Uso Detalhados:

Descrição estruturada das interações entre os usuários e o sistema, contemplando fluxos principais e alternativos.

* Wireframes das telas:

Protótipos visuais das interfaces da aplicação, desenvolvidos com ferramentas como Figma, para ilustrar a navegação e a disposição dos elementos.

* Mapeamento de Requisitos com Funcionalidades:

Tabela relacionando os requisitos funcionais com os componentes do sistema que os implementam.

* Checklist dos Testes:

Listagem de testes realizados (manuais e automatizados), contemplando os critérios de aceitação de funcionalidades.

* Plano de Deploy:

Descrição do processo de publicação da aplicação, com detalhes sobre ferramentas utilizadas (Docker, Heroku, GitHub Actions, etc).

1. **Avaliações de Professores:**