Cin Chat

Grupo 1

FLAVIO HENRIQUE MIRANDA JUNIOR (fhmj)
IGOR DE AZEVEDO CORREA REGO (iacr)
MARIA CLARA GOMES ALVES(mcga)
MARCUS VINICIUS LEITE DA SILVA (mvls)
MATHEUS OLIVEIRA PESSOA (mop2)

Recife, 13 de agosto de 2025

Prof. Vinicius Cardoso Garcia (vcg@cin.ufpe.br)

Sumário

1. Introdução

- **Breve contextualização do projeto**

  O presente projeto visa desenvolver uma solução inovadora para otimizar a comunicação e o acesso à informação em instituições de ensino superior. Tomando como base o Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), identificou-se uma lacuna significativa na forma como docentes, discentes e o setor administrativo interagem com documentos e processos institucionais. A proposta é criar um sistema que utilize inteligência artificial generativa para facilitar a busca, compreensão e automatização de informações, reduzindo a burocracia e a sobrecarga administrativa.

- **Objetivos principais**

  Os objetivos principais deste projeto são:

- Facilitar a comunicação: Promover um fluxo de comunicação mais claro e eficiente entre a instituição e seus membros.
- Agilizar o acesso à informação: Centralizar e tornar acessíveis as informações institucionais, que atualmente se encontram dispersas.
- Automatizar processos burocráticos: Reduzir a intervenção humana em tarefas simples, como a assinatura de documentos de estágio.
- Melhorar a experiência do usuário: Oferecer respostas rápidas e precisas para dúvidas frequentes, permitindo autonomia aos usuários.
- Reduzir a sobrecarga administrativa: Diminuir o volume de demandas simples encaminhadas aos setores administrativos, especialmente em períodos críticos.

- **Justificativa da escolha do tema**

  A escolha do tema é justificada pela relevância do problema em instituições de grande porte como a UFPE, que conta com cerca de 40 mil alunos e 4 mil técnicos administrativos. A ineficiência no acesso à informação e a morosidade em processos burocráticos geram retrabalho, atrasos e sobrecarga dos setores, como evidenciado pelo volume de dúvidas durante o período de matrícula. Além disso, a crescente tendência do mercado em aplicar IA generativa à educação aponta para um potencial de ganhos reais de eficiência e melhoria na experiência de toda a comunidade acadêmica.

- **Abordagem metodológica utilizada (Scrum, Kanban, Híbrida, etc.)**

  A abordagem metodológica utilizada para o desenvolvimento do projeto foi Kanban. Essa escolha se deu pela necessidade de maior flexibilidade na gestão das tarefas, permitindo que a equipe conseguisse atender à agenda de todos os integrantes e, ainda assim, cumprir com a data de entrega estipulada. A natureza visual e a capacidade de adaptação do Kanban se mostraram ideais para o contexto do projeto.

2. Contexto e Problema

- **Descrição do problema abordado**

  O problema central que o produto busca resolver é a ineficiência na comunicação e no acesso a informações institucionais em universidades. Especificamente no CIN da UFPE, os principais problemas são:

- **Dificuldade de localização de documentos:** Documentos importantes estão espalhados em diferentes plataformas e pastas, dificultando sua localização.
- **Baixa clareza na redação dos documentos:** A linguagem utilizada nos documentos muitas vezes é complexa, dificultando a compreensão por parte dos alunos.

- **Relevância e impacto do problema (dados, fontes, validações)**

  A relevância do problema é alta devido ao seu impacto direto na operação da universidade e na experiência de alunos, professores e técnicos. Na UFPE, o volume de dúvidas recebidas pelo setor de graduação é tão grande em períodos críticos (como o de matrícula) que compromete a operação, podendo levar à perda de controle da fila de atendimento. A sobrecarga administrativa é um problema real, com e-mails evidenciando que muitas dúvidas simples acabam consumindo tempo de técnicos que poderiam focar em problemas mais complexos. A falta de um sistema acessível e inteligente para consulta de informações oficiais gera ineficiência e insatisfação.

- **Domínio de aplicação e motivação do projeto**

  O domínio de aplicação do projeto é o ambiente de instituições de ensino superior, com foco inicial no Centro de Informática da UFPE. A motivação principal é aprimorar a comunicação institucional e otimizar processos

administrativos, utilizando as capacidades da inteligência artificial generativa para oferecer um acesso mais claro, rápido e inteligente às informações. A intenção é reduzir a sobrecarga dos setores administrativos e melhorar a experiência de alunos e servidores, permitindo que se concentrem em suas atividades principais.

3. Público-Alvo e Personas

- **Descrição do(s) público(s)-alvo**

  Os principais públicos-alvo deste produto são os alunos e professores do Centro de Informática (CIN) da UFPE, bem como os técnicos administrativos dos setores de graduação. O sistema é projetado para atender às necessidades de todos que interagem com informações e processos institucionais na universidade.

- **Persona(s) detalhada(s): nome fictício, descrição, dores, motivações, comportamentos**

  Ana Clara, 20 anos, Estudante de Ciência da Computação

  - **Perfil:** Aluna do 4º período no CIN/UFPE, envolvida em projetos e buscando estágio.
  - **Dores:**
    - Dificuldade para encontrar e entender documentos institucionais.
    - Insegurança em processos burocráticos (como estágio e matrícula).
    - Demora nas respostas da secretaria, principalmente em períodos críticos.
  - **Motivações:**
    - Quer resolver dúvidas com autonomia e agilidade.
    - Busca clareza nas informações e menos retrabalho.
  - **Uso do produto:** Consulta sobre regras, prazos e processos acadêmicos; automatização de tarefas simples; acesso centralizado e claro a documentos.

4. Proposta de Valor e Solução

- **Proposta de valor clara**

  A proposta de valor do produto reside em oferecer uma solução integrada e inteligente para o acesso a informações institucionais, proporcionando praticidade e especificidade na busca por respostas. O sistema visa garantir que a maior parte das perguntas sejam respondidas de forma autônoma, sem intervenção humana do suporte, otimizando o tempo de alunos, professores e técnicos administrativos e reduzindo a sobrecarga dos setores

- **Diferenciais frente a soluções existentes**

  O produto proposto se diferencia das soluções existentes na Universidade Federal de Pernambuco por:

  - **Integração de IA Generativa:** Atualmente, não existe nenhuma solução semelhante na UFPE que integre inteligência artificial generativa à automatização de processos administrativos e à centralização de informações acadêmicas.
  - **Solução Integrada e Inteligente:** Embora algumas instituições usem chatbots para dúvidas frequentes, o sistema em desenvolvimento oferece uma solução mais abrangente e inteligente, capaz de unificar fluxos, interpretar documentos com IA e reduzir significativamente o retrabalho.
  - **Praticidade e Especificidade:** Diferentemente das soluções atuais que oferecem baixa especificidade nas respostas e alta necessidade de intervenção humana, nossa solução permitirá perguntas altamente específicas que só poderiam ser respondidas manualmente ou após extensa pesquisa em documentos.
  - **Redução da Sobrecarga e Otimização:** O sistema contribui diretamente para a redução do volume de consultas operacionais e demandas simples, otimizando o uso dos recursos humanos e evitando colapsos na operação dos setores administrativos em períodos de pico.

- **Hipóteses iniciais e validações realizadas**

  As hipóteses iniciais para este projeto são as seguintes:

  - **Hipótese 1: A dificuldade de localização e clareza dos documentos institucionais gera uma sobrecarga significativa nos setores administrativos.**
    - **Validação:** Esta hipótese é validada pela **observação direta e experiências pessoais** dos membros da equipe, que

enfrentaram o **caos no período de matrícula** na UFPE. Além disso, a **análise de exemplos de e-mails** recebidos pelo setor de graduação, demonstrando a simplicidade das dúvidas e o impacto negativo na operação, reforça essa premissa.

- **Hipótese 2: Um sistema que utilize IA generativa para interpretar e centralizar informações pode resolver grande parte das dúvidas simples de forma autônoma.**
  - **Validação:** A validação desta hipótese é baseada em **conversas com colegas** que expressam a necessidade de um acesso mais eficiente à informação. A **tendência de mercado** em aplicar IA generativa na educação também sustenta o potencial dessa solução em oferecer ganhos de eficiência e melhorar a experiência do usuário.
- **Hipótese 3: A automatização do acesso à informação liberará os técnicos administrativos para focarem em problemas mais complexos e urgentes.**
  - **Validação:** Esta hipótese é inferida diretamente da **experiência com a sobrecarga** em períodos de pico. A equipe acredita que, ao reduzir o volume de perguntas repetitivas, o tempo dos funcionários será otimizado, impactando positivamente a eficiência geral do setor.

5. MVP e Funcionalidades

1. Visão do MVP (Produto Mínimo Viável)

O MVP visa reduzir a dificuldade de alunos e docentes em encontrar informações institucionais e, consequentemente, a sobrecarga dos setores administrativos. O público-alvo principal é o estudante universitário (como a persona “Ana Clara”), que precisa de respostas rápidas e confiáveis sobre normas e processos acadêmicos. A solução mínima de maior valor é um assistente de IA em formato de chat que responde perguntas com base em documentos oficiais da instituição (via RAG), oferecendo uma interface simples para autenticação, criação de conversa e troca de mensagens.

2. Funcionalidades Centrais Implementadas

- Back-end (API):
  - Gestão de Usuários:
    - `GET /users/`: listar usuários.
    - `POST /users/`: criar usuário (nome, email, senha).

        - `POST /users/login`: login e emissão de token JWT.
    - Gestão de Conversas (Chats):
        - `POST /chats/`: criar nova conversa associada ao usuário.
        - `GET /chats/user/{userId}`: listar conversas do usuário.
        - Proteção por JWT aplicada às rotas de chats e mensagens.
    - Interação e Mensagens:
        - `POST /messages/`: enviar mensagem de usuário para uma conversa (registra no histórico e aciona resposta da IA).
        - `GET /messages/{chatId}`: recuperar histórico completo de mensagens da conversa (ordem cronológica).
    - Documentação da API:
        - `GET /api-docs`: documentação interativa (Swagger/OpenAPI) cobrindo endpoints de usuários, chats e mensagens.
- Front-end (Interface do Usuário):
    - Autenticação:
        - Tela de autenticação (`AuthPage`) com formulários de login (`LoginForm`), registro (`RegisterForm`) e redefinição de senha (`ResetPasswordForm`).
        - Persistência de sessão (token) e guarda de rotas via `Index.tsx` + `useAuth`.
    - Chat:
        - Tela principal de chat (`ChatPage`) com layout de trabalho.
        - Painel lateral (`ChatSidebar`) para listar conversas, selecionar conversa ativa, iniciar nova conversa e excluir conversa; menu do usuário (logout e deletar conta).
        - Interface de mensagens (`ChatInterface`): exibir histórico com distinção visual usuário/assistente, timestamps, envio de mensagens com Enter, auto-scroll.
        - Criação de conversa via modal (`NewChatDialog`) com pergunta inicial.
    - Experiência do Usuário:
        - Feedbacks por toast (sucesso/erro), responsividade (sidebar colapsável em mobile) e uso de componentes acessíveis (shadcn/ui + Radix).

- **Priorizações e decisões sobre escopo**

  Dada a restrição de tempo para o desenvolvimento do Produto Mínimo Viável (MVP), a equipe priorizou as funcionalidades principais de um sistema de chat. A decisão de focar em um núcleo funcional robusto para a interação por conversas foi estratégica para garantir uma entrega de valor

dentro do prazo estipulado. Isso incluiu a capacidade de os usuários enviarem perguntas e receberem respostas geradas pela IA, otimizando a comunicação e o acesso à informação de forma prática. Funcionalidades adicionais, embora importantes, foram adiadas para fases futuras do projeto, permitindo que o foco principal fosse na resolução do problema central de forma eficaz e ágil no curto período disponível.

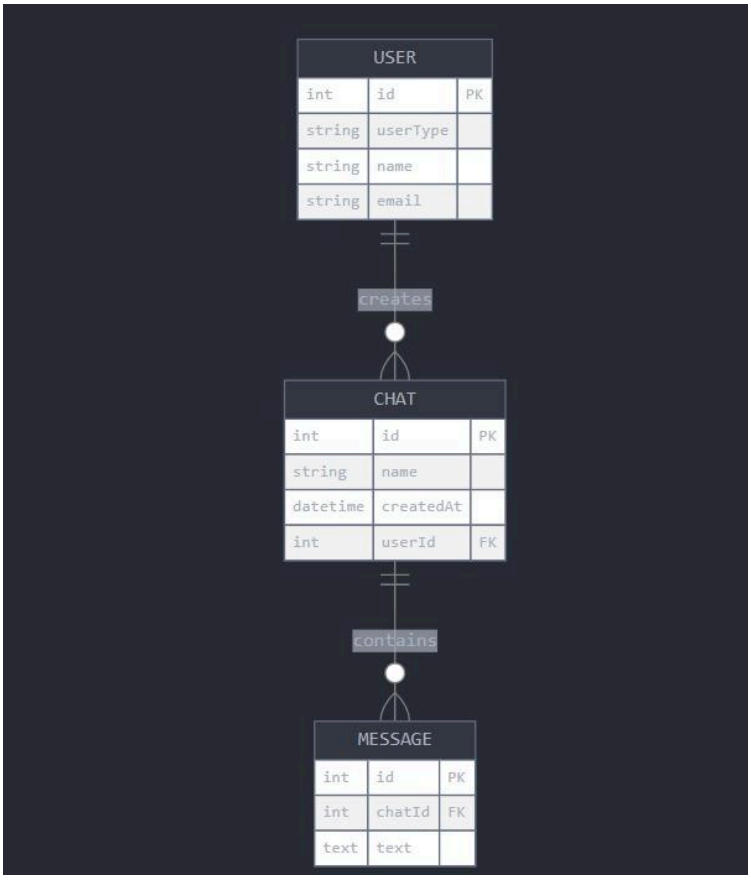
- **Protótipos (screenshots ou links para Figma, Esboços, Wireframes, etc.)**

  Link para o figma: https://www.figma.com/design/xr28oV3BN0pApY3gLqhN8m/Cin-Chat?node-id=0-1&t=7MwpOUlTLzRblQmy-1

6. Modelagem e Design

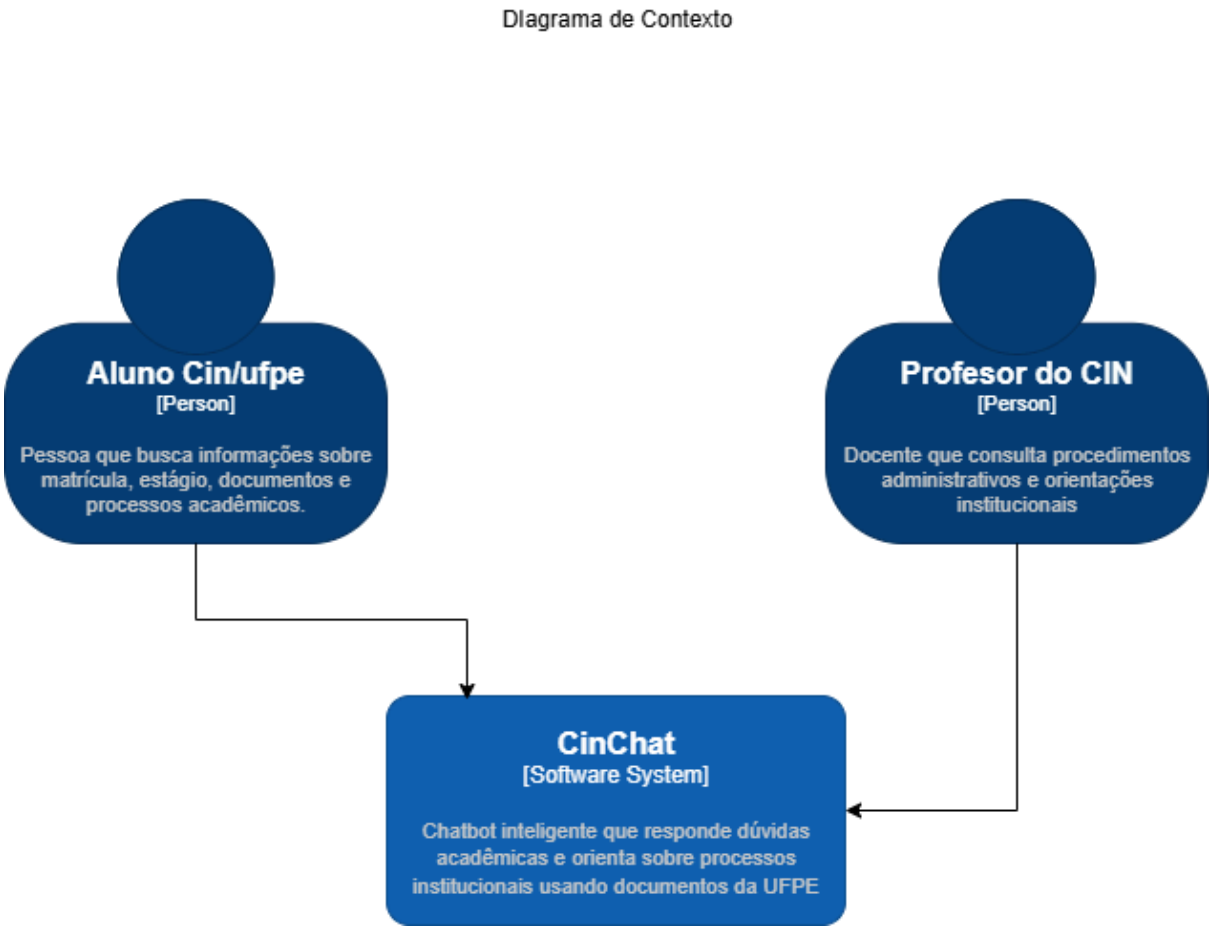
Esta seção detalha a arquitetura e o design do sistema proposto, utilizando a metodologia C4 para representar os diferentes níveis de abstração.

Diagrama ER (Entidade-Relacionamento)

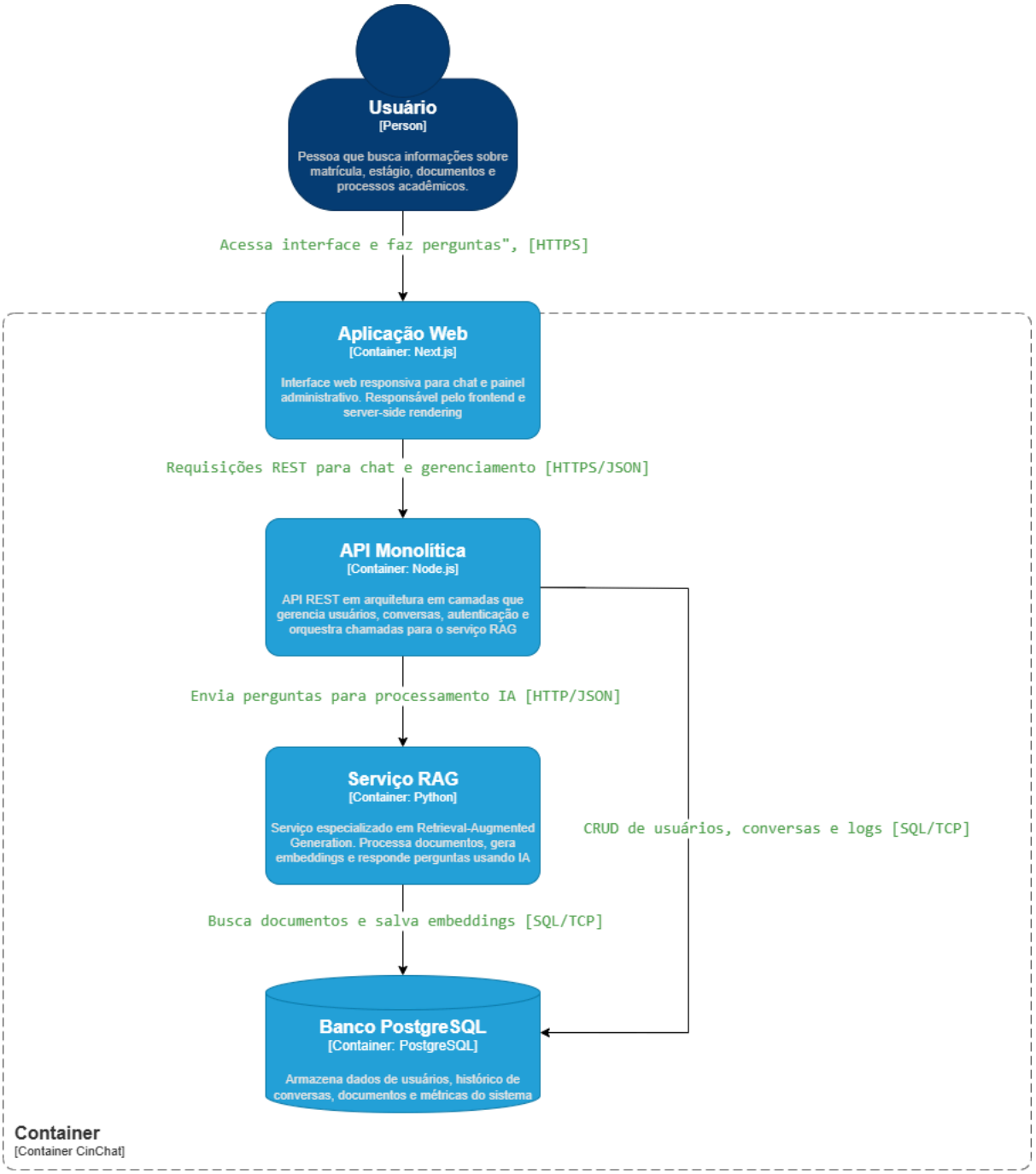


C4 Model:

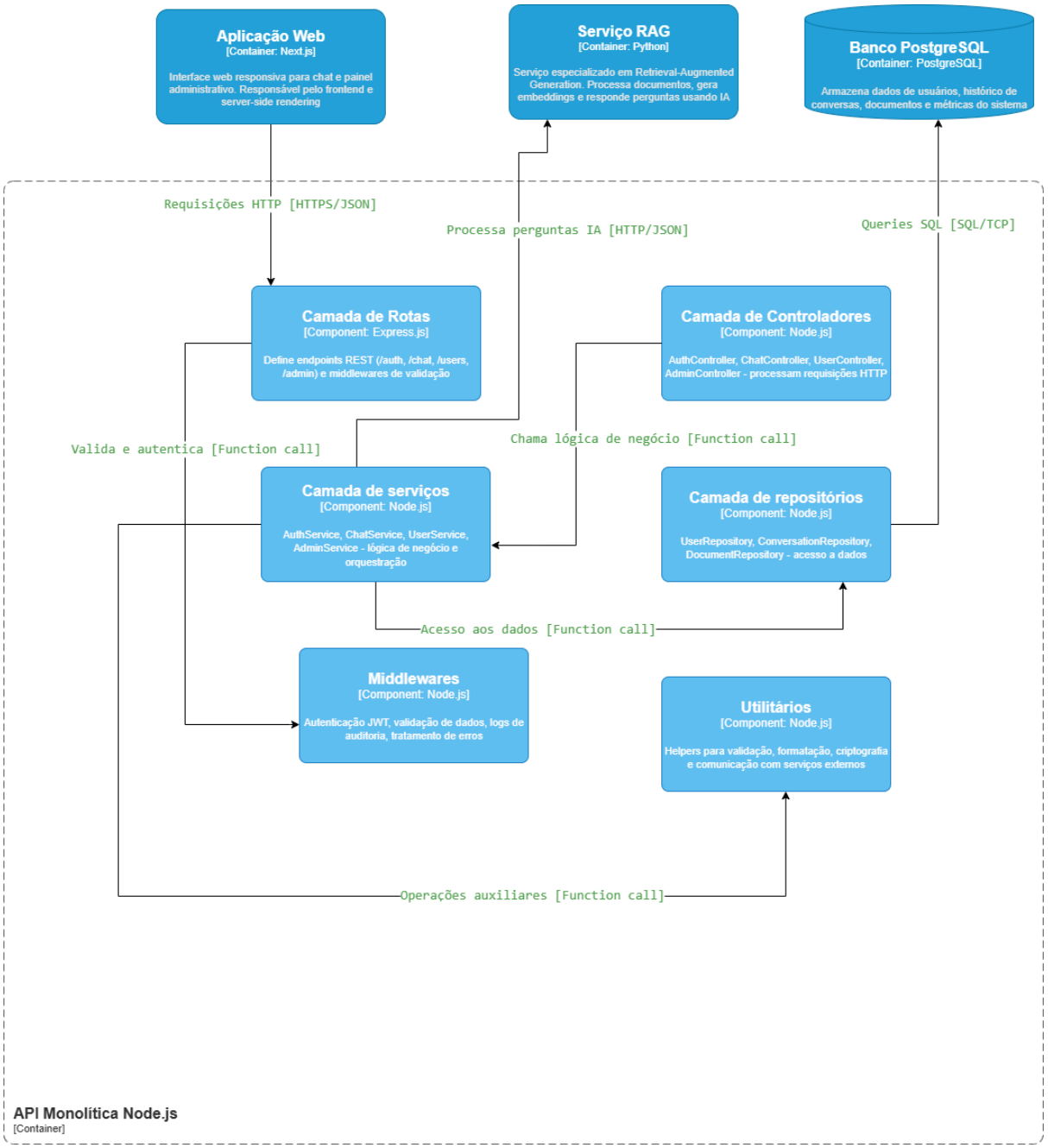
Nível de Contexto



Nível de Contêiner



Nível de Componentes



7. Processo de Desenvolvimento

Metodologia Adotada

Optamos por utilizar o Kanban como metodologia de desenvolvimento. Essa escolha foi motivada pela característica da equipe, que apresentava picos de sazonalidade no trabalho e dificuldade em alinhar horários entre todos os integrantes. Como a disponibilidade de cada membro variava de semana para semana, o Kanban se mostrou mais adequado do que o Scrum, pois não exigia o cumprimento de ciclos rígidos. Assim, o fluxo de trabalho foi gerenciado por meio de cards, que eram movimentados conforme cada integrante avançava nas tarefas de acordo com sua disponibilidade.

Organização da Equipe e Papéis

A equipe foi dividida em duas frentes principais:

- **Back-end:** Igor, Marcus e Flávio
- **Front-end:** Matheus e Clara

Ferramentas Utilizadas

- **GitHub Projects:** organização das tarefas no formato Kanban
- **Figma + Lovable:** prototipação das interfaces
- **Notion:** registro inicial de informações e construção de uma base de conhecimento compartilhada

Planejamento vs. Execução

No início do projeto, foi elaborado um planejamento detalhado sobre onde queríamos chegar. Embora nem todas as etapas tenham sido cumpridas nos prazos originalmente previstos, conseguimos, ao longo do desenvolvimento, recuperar parte do atraso. Ao final, o roadmap foi executado quase por completo, garantindo a entrega da maior parte das funcionalidades planejadas.

1. **Objetivo Principal do MVP**
- Reduzir dificuldade de encontrar informações e sobrecarga administrativa: **Realizado**
2. **Funcionalidades de Backend (API)**
- Gestão completa de usuários (criação, listagem, login com JWT): **Realizado**
- Gestão de conversas (criar, listar por usuário, proteção JWT): **Realizado**
- Interação de mensagens (enviar, histórico, acionamento da IA): **Realizado**
- Documentação interativa da API (Swagger/OpenAPI): **Realizado**
- Funções administrativas avançadas para gestão de usuários ou conteúdos da IA via API**: Não Realizado**
3. **Funcionalidades de Frontend (Interface do Usuário)**
- Tela de autenticação completa (login, registro, redefinição de senha): **Realizado**
- Persistência de sessão e guarda de rotas: **Realizado**
- Tela principal de chat com sidebar e interface de mensagens: **Realizado**

- Modal para criação de nova conversa com pergunta inicial: **Realizado**
- Experiência do usuário aprimorada (feedbacks, responsividade, componentes acessíveis): **Não Realizado**
- Funcionalidades de personalização da interface do usuário além da responsividade básica: **Não Realizado**

4. **Integração de IA e Base de Conhecimento**

- Utilizar IA em formato de chat com base em documentos oficiais (via RAG): Realizado
- Mecanismos de atualização contínua e automatizada da base de conhecimento: Não Realizado
- Aprimoramento da IA e personalização (modelos avançados, ajuste fino, personalização por perfil): Não Realizado

5. **Experiência de Usuário e Feedback**

- Oferecer praticidade e especificidade na busca por informações, autonomia ao usuário e redução do retrabalho: Realizado
- Módulo de feedback do usuário integrado para aprendizado contínuo da IA e acompanhamento da satisfação: Não Realizado

8. Garantia de Qualidade e Segurança

Estratégias e ferramentas de testes

Adotamos uma abordagem mista de testes, combinando **testes automatizados** e **testes manuais**.

- **Testes automatizados**: Utilizamos **Jest** (JavaScript/TypeScript) e **Pytest** (Python) para validar o funcionamento da aplicação. Esses testes são integrados às nossas pipelines de automação, permitindo que falhas sejam detectadas rapidamente, ainda no início do ciclo de desenvolvimento.

- **Testes manuais**: Mantemos uma documentação simples com casos de teste definidos, que servem de guia para validar funcionalidades específicas de forma manual.

Tipos de testes aplicados

- **Unitários**: Desenvolvemos apenas testes unitários, garantindo que funções e componentes individuais funcionem como esperado.

- *(Não aplicamos, até o momento, testes de integração ou de sistema.)*

Ferramentas de análise estática utilizadas

- **Trivy**: Integrado às automações para executar análises SAST (Static Application Security Testing), identificando vulnerabilidades de segurança no código e nas dependências.

- **npm-audit**: Ferramenta simples e rápida para detecção de vulnerabilidades em pacotes Node.js, também executada nas automações.

Estratégias de segurança adotadas

- As análises SAST são realizadas de forma automatizada com o **Trivy**, gerando relatórios de vulnerabilidades.

- Caso sejam detectadas vulnerabilidades de severidade **alta**, as automações sinalizam imediatamente para a equipe.

- Utilizamos o **npm-audit** como complemento para inspeção de dependências e rápida verificação de segurança.

9. Arquitetura Técnica e Implementação

Link repositório:

https://github.com/marcusvxf/Engenharia-de-Software-grupo-1

- **Linguagens, frameworks e bibliotecas utilizadas**
- Back-end:
  - Linguagem e Runtime: Node.js, TypeScript
  - Framework Web: Express
  - Autenticação/Segurança: jsonwebtoken (JWT), cors, dotenv
  - ORM/DB Client: Prisma, @prisma/client
  - Utilidades: axios, body-parser, ulid
  - Dev tooling: ts-node-dev, typescript
- Front-end:
  - Linguagem e Framework: TypeScript, React 18
  - Bundler/Dev Server: Vite, @vitejs/plugin-react-swc
  - UI/Design System: Tailwind CSS, shadcn/ui (construído sobre Radix UI)
  - Navegação e Estado: react-router-dom, (TanStack) @tanstack/react-query
  - Formulários e Validação: react-hook-form, zod, @hookform/resolvers

    - Datas e Ícones: date-fns, lucide-react
    - UX/Feedback: sonner
- Banco de Dados:
    - PostgreSQL (via Prisma ORM)
- Testes:
    - Back-end: Jest, ts-jest
    - Front-end: Jest, jest-environment-jsdom, @testing-library/react, @testing-library/jest-dom, @testing-library/user-event
- Documentação:
    - Swagger: swagger-ui-express, swagger-autogen
- Observação (serviço RAG separado – Python, fora do package.json):
    - FastAPI, LangChain (ChatOpenAI, OpenAIEmbeddings), Chroma (vector store), PyMuPDF/fitz para extração de PDF

- **Estrutura geral do projeto (explicação dos principais módulos)**
- Raiz do repositório (monorepo):
    - back-end/: API REST em Node/Express com TypeScript, Prisma, JWT e Swagger. Expõe recursos de usuários, chats e mensagens.
    - front-end/: SPA em React/TypeScript com Vite e Tailwind/shadcn, implementando autenticação (Context), interface de chat (Sidebar/Interface/Dialog) e consumo da API.
    - RAG_CIN/: Serviço Python (FastAPI) para RAG: upload e indexação de PDFs (embeddings), e endpoint de busca com geração de resposta e fontes.
    - nginx/: Artefatos de NGINX para servir/reverso proxy (prod).
    - Documentos auxiliares: README(s), BUILD.md, CONTRIBUTING.md e docker/docker-compose.
- back-end/src (módulos principais):
    - routes/: Define endpoints REST por recurso:
        - `user.routes.ts`: CRUD básico e login.
        - `chat.routes.ts`: criação e listagem de conversas (protegidas por JWT).
        - `message.route.ts`: envio de mensagens e histórico (protegidas por JWT).
    - controllers/: Orquestram a lógica de requisição/resposta, validam entradas e chamam serviços:
        - `user.controller.ts`, `chat.controller.ts`, `message.controller.ts`.
    - services/: Camada de negócio/acesso a dados via Prisma:

        - `user.service.ts`, `chat.service.ts`, `message.service.ts`, `external/rag.service.ts`.
    - middlewares/: Cross-cutting concerns:
        - `authenticate.middleware.ts`: valida JWT e injeta `req.user`.
    - prisma/: `schema.prisma` (modelos `User`, `Chat`, `Message`) e migrações.
    - swagger.json e swagger.ts: Documentação OpenAPI e geração via swagger-autogen.
    - index.ts/server.ts: bootstrap do app (Express), montagem de middlewares e rotas, servidor HTTP.
- front-end/src (módulos principais):
    - pages/: `Index.tsx` (guarda de autenticação), `AuthPage.tsx` (login/registro/reset), `ChatPage.tsx` (layout principal de chat).
    - components/:
        - auth/: Formulários de Login, Registro e Reset.
        - chat/: Sidebar (lista/ações), Interface (mensagens), NewChatDialog (criação).
        - ui/: Biblioteca shadcn/ui (botões, dialog, inputs etc.).
    - hooks/: `useAuth.tsx` (Context + persistência via localStorage), `useChat.tsx` (operações de chat), `use-toast.ts` etc.
    - types/: Definições TypeScript para auth e chat.
    - lib/, index.css, App.tsx, main.tsx e configuração Vite/TS/ESLint.
- RAG_CIN/ (serviço RAG):
    - main.py / api_routes.py: FastAPI com rotas `/upload/` (PDF) e `/search/` (RAG), pipeline de embeddings (OpenAIEmbeddings + Chroma) e LLM (ChatOpenAI). Extração de PDF com fitz, splitting de texto, retorno com fontes.

- **Estilos, Padrões arquiteturais e de design adotados**
    - Arquitetura em Camadas (Layered Architecture) no back-end:
        - routes → controllers → services → ORM (Prisma)
        - Benefícios: separação clara de responsabilidades, testabilidade e manutenção facilitada.
    - Monorepo:
        - Reúne back-end, front-end, serviço RAG e infra (nginx) no mesmo repositório, padronizando versionamento e facilitando orquestração/dev local.
    - ORM (Object-Relational Mapping) com Prisma:
        - Mapeia modelos `User`, `Chat`, `Message` para PostgreSQL; migrações versionadas; type-safety ponta a ponta.
    - API RESTful:

        - Rotas organizadas por recursos (`/users`, `/chats`, `/messages`), uso consistente de verbos HTTP (GET/POST/DELETE), respostas e códigos adequados (200/201/400/401/404/500).
    - Autenticação JWT:
        - Middleware `authenticate.middleware.ts` protegendo rotas de chats/mensagens; token em `Authorization: Bearer`.
    - Documentação OpenAPI/Swagger:
        - `swagger.json` servido em `/api-docs` com swagger-ui-express; geração automatizada com swagger-autogen.
    - Front-end component-driven:
        - Separação por páginas, componentes e hooks; Context API para sessão; UI consistente com shadcn/ui + Radix; responsividade e acessibilidade como guias.
    - Configuração 12-factor:
        - Uso de variáveis de ambiente (`DATABASE_URL`, `OPENAI_API_KEY`, `VITE_SERVER_PATH`, chaves JWT) para segregar config do código.
    - Serviço RAG dedicado:
        - Microserviço Python especializado em indexação/vetorização e QA com fontes, integrável ao chat.

- **Justificativa das principais decisões técnicas**
    - TypeScript no back-end e front-end:
        - Segurança de tipos, melhor DX (autocompletar/IntelliSense), refatoração segura e redução de bugs em escala.
    - Arquitetura em camadas no back-end:
        - Controllers tratam I/O HTTP, Services encapsulam regras de negócio e acesso a dados; facilita testes unitários/mocks e desacopla camadas.
    - Prisma como ORM:
        - Esquema declarativo com migrações versionadas, type-safety do modelo ao controller, geração do cliente tipado e produtividade superior em CRUDs.
    - Express no Node.js:
        - Framework minimalista, estável e com vasto ecossistema; fácil composição com middlewares (JWT, CORS) e adoção generalizada em REST APIs.
    - JWT para autenticação:
        - Sessões stateless, escalável em ambientes distribuídos, simples de verificar em gateway/reverse proxy e clientes SPAs.

- Swagger/OpenAPI:
  - Documentação interativa e "viva", contrato claro entre clientes e servidor, facilita testes manuais e integração de terceiros.
- Vite + React no front-end:
  - Build e HMR extremamente rápidos, DX superior; React 18 com composição por componentes e ecossistema maduro; plugin SWC acelera transpilações.
- Tailwind + shadcn/ui + Radix:
  - Padronização visual rápida, acessibilidade por padrão (Radix), produtividade em construção de UI consistente.
- Serviço RAG separado (FastAPI + LangChain + Chroma):
  - Desacoplamento de preocupações: pipeline de NLP/embeddings independente; escala separada; uso de ferramentas especializadas do ecossistema Python.
- PostgreSQL:
  - Banco relacional robusto, transacional e amplamente suportado pelo Prisma; adequado para relacionamentos entre `User`, `Chat`, `Message`.
- Testes com Jest e Testing Library:
  - Padronização de testes unitários e de componentes, mocks/fixtures maduros, amplo suporte na comunidade.
- Integração e Deploy:
  - Presença de `docker-compose` e artefatos NGINX sinalizam estratégia de conteinerização, roteamento e futura composição de serviços (API, SPA, RAG, proxy).
- Configuração e Observabilidade:
  - Variáveis de ambiente centralizam segredos e endpoints, simplificando switches entre ambientes; erros padronizados com códigos HTTP e mensagens claras (compatível com monitoramento posterior).

10. Implantação e DevOps

Realizamos a implantação de forma totalmente automatizada por meio do **GitHub Actions**, utilizando a **DigitalOcean** como provedor de hospedagem e o **Docker Hub** para armazenamento das imagens utilizadas.

Dado o início do projeto, adotamos um arquivo **Docker Compose** para gerenciar os serviços, o que facilitou a configuração e execução inicial da aplicação.

Fluxo de automação

Nossas pipelines realizam, de forma sequencial:

1. **Análise estática do código** (SAST);
2. **Build da nova imagem** e envio para o Docker Hub;
3. **Deploy em produção** na DigitalOcean.

Essas etapas são executadas automaticamente sempre que um **pull request** é fechado na branch `main` ou um commit é feito diretamente nela.

Outros pipelines

Além do deploy principal, mantemos pipelines adicionais para:

- Executar testes a cada novo pull request aberto;
- Atualizar arquivos como `docker-compose.yml` e configurações do **NGINX**.

Facilidade de execução

Graças ao uso de containers, nossa aplicação pode ser iniciada rapidamente com o comando:

```
docker compose up -d
```

na raiz do projeto. Uma vez em execução, ela pode ser acessada diretamente pelo endereço IP da instância na DigitalOcean.

Link para as actions: Github actions

11. Colaboração e Versionamento

Adotamos o padrão **Conventional Commits** para manter nossos commits organizados, consistentes e de fácil leitura. Além disso, utilizamos **templates padronizados para Pull Requests** no GitHub, garantindo clareza e uniformidade

Também aplicamos o processo de **code review**, permitindo identificar possíveis problemas, sugerir melhorias e assegurar a qualidade do código antes da integração à branch principal.

Link para documentação do conventional commits: commits

Link GitHub Insights:

https://github.com/marcusvxf/Engenharia-de-Software-grupo-1/pulse

Participação individual (resumo por integrante)

Igor Corrêa:

Minha participação no projeto concentrou-se no desenvolvimento, estabilização e documentação da API do back-end. Atuei diretamente na depuração do ambiente de desenvolvimento, resolvendo conflitos de dependências críticas e configurando os scripts de execução para a nossa stack TypeScript. Fui responsável pela implementação completa do módulo de 'Chats', desenvolvendo os endpoints RESTful para criar e listar conversas (`POST /chats` e `GET /chats/user/:userId`), seguindo a arquitetura em camadas já estabelecida.

Para garantir a qualidade e facilitar a integração com o front-end, liderei a configuração da documentação interativa com Swagger, escrevendo as anotações para todos os endpoints da aplicação (`Users`, `Chats` e `Messages`). Adicionalmente, contribuí para as boas práticas de versionamento, ajustando o `.gitignore` para a segurança do repositório e seguindo o fluxo de trabalho com feature branches, commits padronizados e Pull Requests.

Matheus:

No início, participei auxiliando com decisões de escopo, arquiteturais, de metodologia e etc... ao passar dessa fase inicial, liderei o desenvolvimento do front-end, o qual foi feito inicialmente no Lovable e posteriormente, foram feitos ajustes diretamente pela IDE, onde além da aplicação, também foram realizados testes manuais e automatizados com jest.

Na reta final do projeto, eu me beneficiei da arquitetura BFF, que possibilitou que eu contribuísse com outras partes do projeto, então, realizei alguns bug fix no backend, fiz a integração entre o serviço de RAG e o servidor Nodejs e fiz alguns ajustes pontuais no RAG, após isso, subi um banco de dados remoto apenas para testes, fiz as migrations e pude usar o backend localmente para finalizar a integração com o frontend, que ainda tinha alguns bugs.

Marcus:

Inicialmente colaborei com a decisão do escopo do projeto com outros membros. Atuei também repassando e organizando atividades no github project principalmente no escopo do back-end.

Atuei no desenvolvimento em si realizando novos endpoints como de mensagens no back-end. Mas foquei no desenvolvimento de automações, realizei a criação dos workflows que temos hoje utilizando github actions desenvolvendo fluxo de ci/cd com elas, além de ser responsável pela implantação da nossa aplicação no ambiente da digital ocean e pela configuração de serviços como o docker hub e a configuração do nginx, para conseguirmos ter a aplicação de pé.

Clara: Minha participação no projeto concentrou-se principalmente nas fases de planejamento e documentação. Como aluna, trouxe a perspectiva direta das dores e necessidades do usuário final, o que foi crucial para a definição das personas e para a validação das hipóteses iniciais do projeto. Atuei ativamente na pesquisa e análise de documentos para a construção da base de conhecimento inicial e ajudei a refinar a proposta de valor e os diferenciais da solução, garantindo que o produto estivesse alinhado com o problema que buscávamos resolver

Flávio:

Minha participação no projeto concentrou-se no desenvolvimento e implementação do módulo RAG (Retrieval-Augmented Generation) e na criação da funcionalidade de interação entre o usuário e o chat. Atuei na configuração da pipeline de ingestão e indexação de documentos PDF, desenvolvendo o fluxo para leitura de arquivos com suporte tanto a texto digital quanto a OCR para documentos digitalizados.

Implementei a lógica de busca contextual e geração de respostas com LangChain e OpenAI, garantindo que o retorno fosse sempre em português e com base apenas nas informações disponíveis nos documentos indexados. Participei também da integração desse módulo com o sistema de conversas, permitindo que o usuário interagisse de forma contínua com o chat e obtivesse respostas relevantes e contextualizadas.

Além disso, contribuí para a organização do código seguindo padrões de commits e versionamento, mantendo boas práticas no uso de branches e Pull Requests, e garantindo a compatibilidade e atualização das dependências do projeto

12. Lições Aprendidas

- **Desafios enfrentados**

  Durante o desenvolvimento e planejamento do projeto, alguns desafios foram identificados ou antecipados:

  - **Gerenciamento da disponibilidade da equipe:** Dada a flexibilidade da metodologia Kanban, foi um desafio coordenar as agendas individuais para garantir o progresso contínuo e a conclusão das tarefas dentro do prazo.
  - **Coleta e tratamento de dados:** A dispersão e, possivelmente, a baixa clareza na redação dos documentos institucionais representam um desafio significativo para a ingestão e o pré-processamento dos dados que alimentarão a IA.
  - **Garantia da especificidade e precisão da IA:** Assegurar que as respostas geradas pela inteligência artificial sejam de fato específicas, precisas e contextualmente relevantes para as dúvidas dos usuários, sem gerar informações incorretas (alucinações).

- **Ajustes feitos ao longo do caminho**

  Com base nos desafios e no aprendizado contínuo, os seguintes ajustes podem ser considerados ou foram feitos:

  - **Otimização do uso do Kanban:** Reforçar a disciplina na atualização do quadro Kanban e na comunicação assíncrona para maximizar a colaboração, apesar das agendas flexíveis.
  - **Foco na fase de coleta de dados:** Dedicar um esforço maior e mais estruturado na coleta, padronização e indexação dos documentos para garantir a qualidade da base de conhecimento da IA.
  - **Iterações constantes para feedback da IA:** Implementar ciclos curtos de feedback para refinar os modelos de IA, garantindo que as respostas atendam às expectativas de especificidade e precisão dos usuários.

- **Aspectos que funcionaram bem**

  Diversos aspectos contribuíram positivamente para o andamento do projeto:

- **Metodologia Kanban:** A flexibilidade do Kanban permitiu que a equipe se adaptasse às agendas individuais, mantendo o foco nas entregas e na data final.
- **Clareza do problema:** A identificação de um problema real e relevante, com validações baseadas em experiências diretas e dados da instituição, facilitou a definição dos objetivos e a motivação da equipe.
- **Potencial da IA Generativa:** A forte tendência de mercado e o potencial transformador da IA na educação mantiveram a equipe engajada e otimista com a solução.

- **Aspectos que poderiam ser melhorados**

  Para futuros desenvolvimentos, os seguintes aspectos podem ser aprimorados:

  - **Comunicação pró-ativa:** Embora o Kanban ajude, uma comunicação mais proativa sobre bloqueios ou dependências pode acelerar o fluxo de trabalho.
  - **Testes de usabilidade antecipados:** Realizar testes de usabilidade com usuários reais (alunos, professores, técnicos) em etapas mais iniciais do desenvolvimento para coletar feedback e fazer ajustes finos na interface e na experiência do usuário.
  - **Documentação interna:** Manter uma documentação interna mais detalhada sobre decisões de arquitetura e implementações técnicas para facilitar a manutenção e a entrada de novos membros na equipe.

- **Relato individual (cada membro da equipe)**

  Marcus Silva:

  - **Desafios:** Pra mim acredito ter sido a gestão de tempo, por ser um projeto que demanda um esforço grande, não organizar o tempo acabou se provando um problema.
  - **Aprendizados:** A organização de uma equipe e conceitos mais técnicos que tive o contato inicial como o Rag que foi feito.
  - **Contribuições:** Acredito que contribui desde da idealização até a implantação do produto, focando ali dentro da parte de devops onde construi nossos pipelines utilizando github actions, além de atuar

em algumas partes do back. Também realizei a configuração dos serviços e o deploy da aplicação na digital ocean.

Maria Clara:

- **Desafios:** Como aluna, senti de perto a dor da dificuldade de acesso a informações claras e centralizadas. Meu principal desafio foi traduzir essa experiência em requisitos funcionais e não funcionais que a solução pudesse de fato resolver, garantindo que a perspectiva do usuário final estivesse sempre no centro das discussões. Devido ao meu conhecimento técnico mais limitado, precisei de um esforço extra para compreender as nuances da implementação, mas isso me impulsionou a aprender.
- **Aprendizados:** Este projeto me ensinou muito sobre a colaboração em equipe, especialmente com agendas variadas, e como o Kanban é prático para manter todos focados. Além disso, a modelagem C4 foi um grande aprendizado, pois me permitiu entender a arquitetura do sistema de forma bem clara, mesmo sem um background técnico profundo, o que foi super útil na prática.
- **Contribuições:** Minha principal contribuição foi na fase de planejamento do projeto, incluindo a definição de personas e a validação das hipóteses iniciais, sempre buscando trazer a perspectiva do usuário final. Também fui ativa na documentação do projeto, organizando informações e ajudando a refinar a proposta de valor e os diferenciais da solução para garantir que atendesse às necessidades.

Matheus Pessoa:

- **Desafios:** Meu maior desafio foi a gestão de tempo, pois foi um período bastante corrido e com disciplinas bastante trabalhosas, além das diversas atividades profissionais extracurriculares que participo.
- **Aprendizados:** Acredito que meu maior aprendizado tenha sido em relação a “aprender” a contribuir com diversas partes do projeto, mesmo estando focado em uma. Com a arquitetura BFF, isso ficou muito claro e acabei aprendendo bastante com isso. Após o término do projeto, falei sobre com alguns amigos e um deles, que é desenvolvedor mobile no Itaú, me disse que eles também usam BFF em alguns projetos e me deu mais algumas dicas sobre isso.
- **Contribuições:** Minha principal contribuição foi o Desenvolvimento front-end. Além disso, pontuo: planejamento, bug-fix no backend e integração entre o servidor nodejs e o agente de IA com Rag.

Igor Corrêa:

- **Desafios:** Meu principal desafio pessoal foi a falta de experiência técnica aprofundada ao iniciar o projeto. Conceitos como o fluxo de trabalho colaborativo com Git, a depuração de ambientes de desenvolvimento complexos e a configuração de ferramentas como Swagger e TypeScript eram, em grande parte, novos para mim. Enfrentei uma curva de aprendizado íngreme, o que exigiu paciência e uma nova forma de pensar para solucionar problemas de maneira metódica.
- **Aprendizados:** O aprendizado mais significativo foi a importância da metodologia e da resiliência na resolução de problemas. Aprendi que um erro não é um bloqueio, mas uma pista. A longa depuração para configurar o Swagger, por exemplo, me ensinou a investigar problemas passo a passo: verificar caminhos de arquivos, analisar dependências entre pacotes, entender a configuração do ambiente e, acima de tudo, não desistir. Além do conhecimento técnico em Git, Node.js e TypeScript, o maior ganho foi ter desenvolvido uma abordagem mais estruturada e calma para enfrentar desafios técnicos, transformando a frustração inicial em um processo lógico de investigação.
- **Contribuições:** Minhas principais contribuições se concentraram na estruturação e na garantia de qualidade da API do back-end. Fui responsável pela **implementação da funcionalidade de 'Conversas' (Chats)**, o que incluiu a criação dos endpoints para criar novas conversas (`POST /chats`) e listar as conversas de um usuário (`GET /chats/user/:userId`), seguindo a arquitetura em camadas (Routes, Controllers, Services) do projeto. Adicionalmente, liderei a **configuração e implementação da documentação interativa da API com Swagger**. Isso envolveu a depuração do ambiente, a geração do arquivo de documentação e a criação da página `/api-docs`. Documentei detalhadamente todos os endpoints existentes (`Users`, `Chats` e `Messages`), criando um "contrato" claro e funcional para facilitar a integração com a equipe de front-end. Por fim, contribuí na melhoria dos processos de versionamento, corrigindo o `.gitignore` e garantindo a criação de commits e Pull Requests seguindo as boas práticas do projeto.

Flávio Henrique:

- **Desafios:** Implementar o módulo RAG com suporte a PDFs de texto e digitalizados via OCR, garantindo integração fluida com o chat e respostas sempre em português.
- **Aprendizados:** Domínio de fluxos híbridos de processamento de PDFs, indexação vetorial e uso de LangChain/OpenAI, além de

reforço nas boas práticas de versionamento e commits padronizados.
- **Contribuições:** Desenvolvi o módulo RAG completo, desde a leitura e tratamento dos PDFs até a busca e geração de respostas contextuais no chat, configurando o prompt para manter a consistência em português e contribuindo para a atualização das dependências e organização do código.

13. Conclusão

- **Avaliação da Solução Entregue Frente ao Problema Inicial**

  A solução proposta, focada em um sistema de gestão de informações acadêmicas com IA generativa, demonstrou grande potencial para **abordar diretamente os problemas iniciais** de dificuldade de localização de documentos, baixa clareza e sobrecarga administrativa.

  - **A centralização das informações** em uma base de conhecimento acessível pela IA resolve diretamente o problema de documentos dispersos, permitindo que usuários encontrem o que precisam rapidamente.
  - A capacidade da IA de **interpretar e resumir informações complexas** em respostas claras e específicas mitiga a dificuldade de compreensão dos documentos oficiais, tornando o conhecimento mais acessível a todos.
  - Ao **automatizar as respostas** a dúvidas simples e recorrentes, a solução libera o tempo dos técnicos administrativos, que podem se dedicar a questões mais complexas e que realmente exigem intervenção humana. Isso valida a hipótese de que o sistema pode reduzir significativamente o "caos no período de matrícula" e melhorar a eficiência do setor.

  Em resumo, a solução se alinha perfeitamente com a proposta de valor de oferecer praticidade e especificidade na busca por informações, transformando a experiência de comunicação institucional na UFPE.

- **Expectativas de Evolução do Projeto (Roadmap Futuro)**

  O projeto, em sua fase inicial, estabeleceu uma base sólida para a resolução dos problemas centrais. Para o futuro, vislumbram-se as seguintes etapas e expansões no roadmap:

- **Expansão da Base de Conhecimento:** Integrar mais fontes de dados e documentos institucionais de outros departamentos e setores da UFPE para abranger uma gama mais ampla de informações, além de desenvolver mecanismos para atualização contínua e automatizada.
- **Aprimoramento da IA e Personalização:** Investigar modelos de linguagem mais avançados e técnicas de ajuste fino (fine-tuning) para melhorar a precisão e a naturalidade das respostas da IA, explorando também a personalização com base no perfil do usuário.
- **Novas Funcionalidades:** Implementar funcionalidades para auxiliar em processos burocráticos mais complexos, como a geração de formulários pré-preenchidos ou o acompanhamento de status de solicitações, e adicionar um módulo de feedback do usuário para aprendizado contínuo.
- **Monitoramento e Análise:** Desenvolver painéis de monitoramento para acompanhar o desempenho da IA, o volume de consultas, os tipos de dúvidas mais frequentes e a satisfação do usuário, analisando os dados para identificar novas oportunidades.
- **Disponibilização para Outras Instituições:** Em um futuro mais distante, avaliar a possibilidade de adaptar e disponibilizar a solução para outras instituições de ensino superior que enfrentam desafios semelhantes.

- **Reflexão Final Sobre a Experiência de Desenvolvimento em Equipe**

  A experiência de desenvolvimento em equipe neste projeto foi marcada pela **colaboração e adaptação**. A escolha da metodologia Kanban se mostrou crucial para gerenciar a disponibilidade individual de cada membro e, ao mesmo tempo, manter o ritmo das entregas e o cumprimento dos prazos.

  A clareza sobre o problema a ser resolvido e a relevância do impacto da solução (validada pelas experiências pessoais e do entorno) foram fatores motivadores importantes. O potencial da IA generativa no contexto educacional manteve a equipe engajada e focada na inovação.

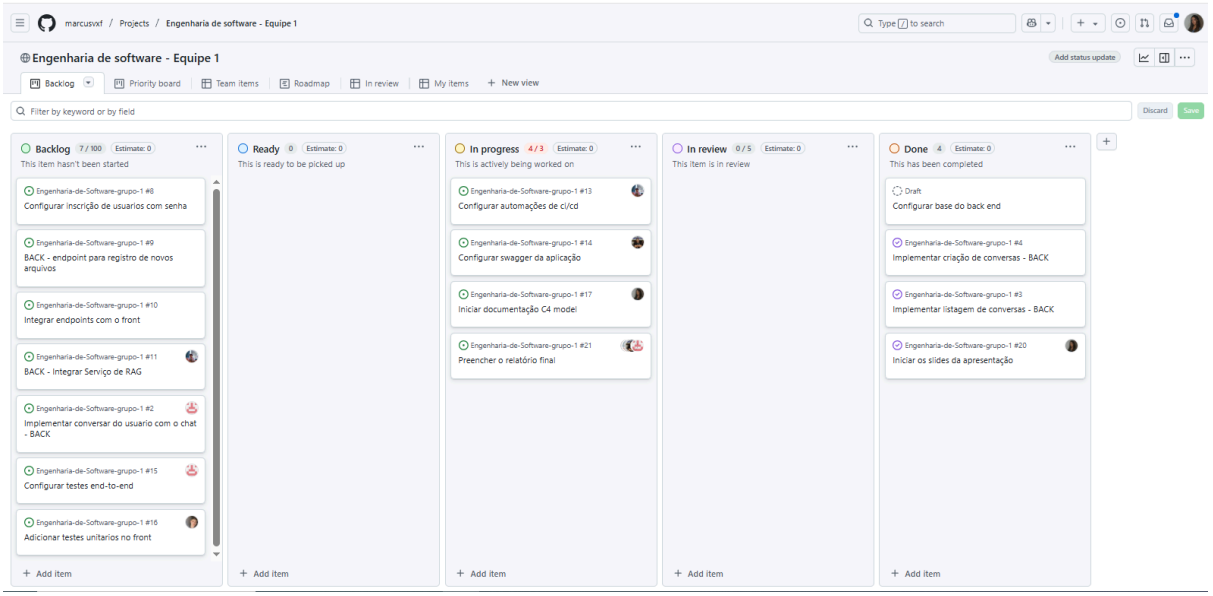
  Os desafios, como a coleta e tratamento de dados e a garantia da precisão da IA, impulsionaram a busca por soluções criativas e o aprendizado contínuo. A comunicação, mesmo que assíncrona, foi fundamental para o alinhamento e a superação de obstáculos.

  No geral, a experiência reforçou a importância de uma metodologia flexível, do trabalho em equipe coeso e da paixão por resolver problemas reais para

o sucesso de um projeto. Acreditamos que o aprendizado obtido será valioso para futuros empreendimentos.

14. Anexos

Anexo 1: Print do Quadro Kanban



Anexo 2: Link da aplicação (deploy)

[Aplicação]

Anexo 3: Relato pessoal que comprova a dificuldade dos alunos em processos acadêmicos