ESCOLA SESI PLANALTO
CURSO TÉCNICO DO ENSINO MÉDIO

EVELYN OLIVEIRA CARLOS
JOÃO MIGUEL FREIRE DE OLIVEIRA MENDES
JOÃO VICTOR PINHEIRO REIS
NICOLY STEPHANY CORRÊA MARIANO

DOCUMENTO DE REQUISITOS DE PROJETO SCHOOL SECRETARY

Goiânia
2025

1. Introdução

1.1. Descrição Geral

O School Secretary é uma aplicação web desenvolvida para o gerenciamento de informações escolares. Suas principais características são: armazenamento remoto em servidor, interface simples e clara, estrutura otimizada e leve, além de autenticação para acesso seguro aos dados. O sistema é voltado para instituições de ensino que necessitam registrar e organizar informações acadêmicas e administrativas de forma centralizada e confiável.

1.2. Objetivos Principais

- Permitir o registro de informações escolares de forma centralizada.
- Oferecer uma interface clara e simples.
- Garantir armazenamento seguro e acesso autenticado aos dados.
- Assegurar que dispositivos de baixo custo com processamento limitado utilizem a aplicação de maneira eficiente.

1.3. Público-Alvo e Requisitos de Uso

O público-alvo principal é composto por instituições de ensino, professores, estudantes e equipe administrativa. As necessidades desse segmento se concentram em uma plataforma para o registro e organização de dados escolares, com foco nas seguintes características:

- Faixa etária: ampla, abrangendo desde estudantes em idade escolar até profissionais da educação.
- Proficiência tecnológica: nível básico a avançado.
- Contexto de uso: gerenciamento de informações acadêmicas e administrativas em ambiente escolar.

1.4. Escopo do Projeto

- O que será entregue:
    - Interface clara e de fácil navegação.
    - Fluxo de trabalho intuitivo.
    - Autenticação segura para acesso aos dados.
    - Funcionalidades básicas para registro e consulta de informações escolares.

    - Armazenamento de arquivos de documentos em PDF
- O que não será incluído:
    - Suporte para formatos complexos de dados ou registros avançados, como tabelas detalhadas, listas de verificação ou subcategorias.
    - Customização extensiva de interface ou elementos gráficos.

2. Requisitos do Projeto

2.1. Requisitos Funcionais

1. O usuário deve ser capaz de se registrar com e-mail e senha, incluindo recuperação de senha.
2. O usuário deve poder acessar e modificar informações escolares armazenadas remotamente no servidor.
3. O sistema deve permitir que o usuário adicione, edite e exclua registros de dados escolares.
4. O sistema deve oferecer funcionalidades de CRUD (Create, Read, Update, Delete) para os registros, tanto para administradores quanto para usuários autorizados.

2.2. Requisitos Não Funcionais

1. **Desempenho:** o tempo de carregamento da interface principal não deve exceder 3 segundos.
2. **Segurança:** o sistema deve criptografar senhas e proteger os dados escolares armazenados no banco de dados.
3. **Escalabilidade:** a arquitetura do backend deve conseguir suportar um aumento de até 50% no número de usuários simultâneos.

2.3. Restrições

- **Tecnologia:** o projeto deve utilizar Next.js com TypeScript no frontend, Axios para comunicação HTTP e Django no backend.
- **Prazo:** a primeira versão funcional (MVP) deve ser entregue em, no máximo, 2 meses.
- **Orçamento:** o custo com infraestrutura não deve ultrapassar 10 dólares por dia.

3. Arquitetura do Sistema

3.1. Frontend

- **Tecnologias:** TypeScript, Axios, React, NextJS, JSON, HTTP e CSS puro.
- **Estrutura da API:** Componentes React, funções em TypeScript, operações HTTP com Axios e comunicação em formato JSON, visando a interação com o backend para o gerenciamento de dados escolares.

3.2. Backend

- **Tecnologias:** Django, Django REST Framework, Python.
- **Estrutura de API:** Django REST Framework para criação de endpoints que suportem o registro, consulta e atualização de informações escolares de forma segura e eficiente.

3.3. Banco de Dados

- **Tipo:** PostgreSQL, escolhido por sua robustez, confiabilidade e capacidade de lidar com dados estruturados eficientemente, complementando a arquitetura do Django.
- **Esquema Preliminar:** O banco de dados terá tabelas principais para armazenar informações de usuários, registros acadêmicos e dados administrativos, mantendo separação clara e relacionamentos seguros entre os dados.

3.3.1. Formato Geral do Esquema do Banco

Categoria Principal	Campo Original (Modelo Django)	Tipo de Dado
Dados do Estudante	full_name	Texto
	registration_number	Texto/Número
	phone_number	Texto
	email	E-mail
	cpf	Texto
	birthday	Data
	address	Texto
	group (ForeignKey)	Texto
	created_at	Data/Hora

Dados do Responsável	full_name	Texto
	student (ForeignKey)	Texto
	phone_number	Texto
	email	E-mail
	cpf	Texto
	birthday	Data
	address	Texto
	created_at	Data/Hora
Dados do Contrato	guardian (ForeignKey)	Texto
	student (ForeignKey)	Texto
	created_at	Data/Hora
Dados de Notas	student (ForeignKey)	Texto
	subject (ForeignKey)	Texto
	year	Número
	bimester	Lista (1º Bimestre, 2º Bimestre, 3º Bimestre, 4º Bimestre)
	value	Número
	created_at	Data/Hora
Dados de Presença	student (ForeignKey)	Texto
	date	Data
	presence	Booleano (Sim/Não, ou Presente/Falta)
	created_at	Data/Hora

4. Fluxo de Desenvolvimento

4.1. Metodologia

Para o desenvolvimento do School Secretary, será adotada a metodologia Ágil, com foco no framework Scrum. Esta abordagem permite que a equipe se adapte rapidamente a mudanças e entregue valor de forma contínua, conforme a natureza iterativa do desenvolvimento de software.

4.1.1. Por que Scrum?

- **Entregas Iterativas e Incrementais:** O projeto será dividido em Sprints (períodos fixos, geralmente de 1 a 4 semanas), ao final dos quais uma versão funcional do sistema será entregue, permitindo feedback constante.
- **Flexibilidade e Adaptação:** O Scrum permite que a equipe responda a novas prioridades e requisitos ao longo do desenvolvimento, em vez de seguir um plano rígido.
- **Colaboração e Transparência:** Promove a comunicação diária e a colaboração entre os membros da equipe, mantendo os stakeholders informados sobre o progresso e os desafios.
- **Foco no Valor:** Prioriza funcionalidades que agregam mais valor ao sistema e aos usuários finais, garantindo que os esforços estejam direcionados ao que é mais relevante.

4.1.2. Papéis e Cerimônias (Scrum Essencial):

- **Product Backlog:** Lista priorizada de todas as funcionalidades e melhorias desejadas para o School Secretary.
- **Sprint Planning:** No início de cada Sprint, a equipe seleciona os itens do Product Backlog a serem desenvolvidos.
- **Daily Scrums:** Reuniões rápidas para sincronização da equipe, compartilhamento de progresso e identificação de impedimentos.
- **Sprint Review:** Ao final de cada Sprint, a equipe apresenta o que foi concluído aos stakeholders para coleta de feedback.
- **Sprint Retrospective:** Reunião para reflexão sobre o Sprint e identificação de oportunidades de melhoria no processo.

4.2. Sprints e Fases

O projeto será executado em quatro sprints, garantindo desenvolvimento iterativo e contínuo. Cada sprint possui focos definidos:

Sprint 1 (Semanas 1-2): Configuração Inicial e Prototipagem

- **Configuração do Ambiente de Desenvolvimento:** Instalação e configuração das ferramentas e dependências necessárias (IDE, frameworks, bibliotecas).
- **Prototipagem de UX/UI:** Criação de wireframes e mockups das principais interfaces, definindo o fluxo de usuário e experiência de navegação.
- **Criação das Telas de Autenticação:** Desenvolvimento das interfaces de login e registro, incluindo validação de formulários e arquitetura visual.
- **Testes de Conexão (Frontend):** Verificação inicial da comunicação das telas com APIs de teste ou mock.

Sprint 2 (Semanas 3-4): Desenvolvimento do Backend e Autenticação

- **Criação do Schema do Banco de Dados:** Modelagem das tabelas, colunas, chaves primárias e estrangeiras, incluindo usuários, registros acadêmicos e dados administrativos.
- **Implementação do Backend:** Desenvolvimento da lógica de negócios e da estrutura do servidor.
- **Criação das APIs de Autenticação:** Desenvolvimento dos endpoints para login e registro, com foco em segurança (hashing de senhas, tokens).
- **Testes de Conexão (Backend):** Testes de unidade e integração para garantir comunicação correta com o banco de dados e funcionamento das APIs.

Sprint 3 (Semanas 5-6): Implementação do Banco de Dados e CRUD

- **Configuração do SQL:** Criação do banco de dados e implementação do schema detalhado, incluindo execução de scripts DDL.
- **Desenvolvimento de Operações CRUD:** Implementação de endpoints e lógica para criação, leitura, atualização e exclusão de registros escolares.
- **Testes de Integração:** Validação da comunicação entre frontend, backend e banco de dados.

Sprint 4 (Semanas 7-8): Desenvolvimento da Página Principal e Ajustes Finais

- **Configuração da Página Principal:** Desenvolvimento da interface principal, onde os usuários podem visualizar e gerenciar os registros escolares.
- **Implementação de Filtros e Organização:** Funcionalidade para agrupar e exibir registros por categoria, data ou palavra-chave, facilitando a organização das informações.
- **Testes de Aceitação do Usuário (UAT):** Realização de testes com usuários reais para coleta de feedback e ajustes de usabilidade.
- **Finalização e Deploy:** Preparação do ambiente de produção e implantação do sistema.

4.3. Ferramentas de Gerenciamento

- Git e GitHub para controle de versão.
- Uso de Trello para organizar tarefas e fluxos de trabalho.

5. Design e Prototipagem

5.1. Wireframes e Mockups

A fase de design será utilizada para visualizar e refinar a interface do School Secretary, garantindo clareza e fluxo de trabalho intuitivo para usuários escolares. Serão criados wireframes e mockups interativos utilizando Figma, permitindo validar a navegação, organização de registros e componentes antes da implementação em Next.js com TypeScript.

5.2. Fluxo de Navegação

Após o login, o usuário será direcionado à página principal, onde poderá acessar de forma clara e intuitiva todas as funcionalidades do sistema. Para cada aba acessada será dada uma visualização dos dados e ainda um link para acessar a página onde mais dados daqueles são acessados. Nesse caso, o professor enxerga a aba de notas escolares e, entrando nela, seleciona uma turma e poderá ver as notas dos alunos da turma, nessa mesma página poderá ir para a próxima, onde irá inserir dados. Isso se repete para tudo, de forma bastante semelhante.

5.3. Estilo Visual: Monocromático Elegante

O estilo visual do projeto será pautado na simplicidade e elegância do design monocromático, utilizando uma paleta de cores neutras que transmitem profissionalismo e

clareza. O objetivo é criar uma interface limpa, intuitiva e esteticamente agradável, onde o foco recaia no conteúdo e na funcionalidade.

- Paleta de Cores:
    - **Primária:** #FFFFFF (Branco Puro) - Utilizado para fundos principais, áreas de conteúdo e elementos de destaque que precisam de leveza.
    - **Secundária:** #000000 (Preto Puro) - Empregado para textos principais, ícones, bordas e elementos de contraste.
    - **Tons de Cinza:**
        - Aproximadamente #F2F2F2 (Cinza Claro Quase Branco) - Para fundos secundários, divisores sutis ou estados de elementos (ex: hover).
        - Aproximadamente #CCCCCC (Cinza Médio Claro) - Para textos secundários, placeholders ou elementos que precisam de menor destaque.
        - Aproximadamente #666666 (Cinza Médio Escuro) - Para textos de apoio, descrições ou elementos de interface que não são o foco principal.
        - Aproximadamente #333333 (Cinza Escuro Quase Preto) - Para títulos, subtítulos ou elementos que exigem forte contraste sem ser preto puro.
    - **Cor de Destaque:** Um tom sutil de vermelho para alertas, como deletar dados.
- Tipografia:
    - Fontes: Optar por fontes sem serifa (sans-serif) para a maioria dos textos, garantindo legibilidade e modernidade.
        - Fonte Principal (Títulos e Destaques): Inter em tamanho maior.
        - Fonte Secundária (Corpo de Texto): Inter em tamanho médio e menor.
    - Hierarquia: Variações de peso (light, regular, semi-bold, bold) e tamanho para criar hierarquia visual clara, sem a necessidade de cores adicionais.
- Ícones:
    - Estilo: Ícones minimalistas, de linha fina (outline) ou com bordas (border).
    - Cores: Em cinza-claro ou preto em fundo claro e em cinza-escuro ou branco em fundo escuro.
- Elementos de Identidade Visual:
    - Espaçamento: Priorizar o uso generoso de espaços em branco para criar uma sensação de leveza, organização e sofisticação.

- Linhas e Bordas: Utilizar linhas finas e bordas sutis em tons de cinza para separar seções ou destacar elementos.
- Botões e Campos:
  - Design Minimalista: Botões e campos de formulário devem seguir a estética limpa e minimalista. Evite gradientes, texturas e bordas espessas.
  - Bordas: Utilize bordas finas e arredondadas para dar um toque moderno e suave, mantendo a consistência visual.
  - Estados: Os estados do botão (normal, hover, ativo) devem ser diferenciados por mudanças sutis, como uma alteração leve na cor do fundo ou na espessura da borda. Isso garante a usabilidade sem poluir o design.
  - Tipografia: O texto em botões e campos de entrada deve ser claro e legível, usando a mesma tipografia do restante do design.

6. Plano de Testes

6.1. Testes Unitários

- **Ferramentas:** Vitest para frontend, Pytest e Django Tests para backend.
- **Cobertura:** Todo o conjunto principal de unidades do projeto, incluindo funções, classes ou componentes que desempenham as principais tarefas no sistema.

6.2. Testes de Integração

- **Ferramentas:** Vitest para frontend, Pytest e Django Tests para backend.
- **Cobertura:** Fluxo de registro de usuário, comunicação entre frontend e backend, criação, edição e exclusão de dados, além do registro de tais modificações no banco de dados.

6.3. Testes de Ponta a Ponta (End to End, ou E2E)

- **Ferramentas:** Cypress para frontend
- **Cobertura:** Principais elementos interativos da tela, como: botões, links, abas, etc.

6.4. Testes de Usabilidade

- **Metodologia:** Testes com usuários reais para validar a experiência de navegação.

6.5. Ferramentas de Teste

Para garantir a qualidade e a confiabilidade do projeto, utilizaremos as seguintes ferramentas, com exemplos práticos de cenários de teste para cada tipo.

Vitest (Testes Unitários): O Vitest será usado para validar funcionalidades isoladas do código, garantindo que cada "peça" funcione corretamente por si só.

- Cenário de Teste: Validar a formatação de data
    - Descrição: Criar um teste que verifica se uma função de formatação de data (formatDate('2025-08-11') retorna o valor esperado (11/08/2025).
- Cenário de Teste: Validar o estado de um componente
    - Descrição: Testar se um componente de botão, ao receber a propriedade “disabled”, desabilita realmente o botão e não permite cliques.

Cypress (Testes End-to-End): O Cypress será utilizado para simular a jornada completa do usuário na interface, garantindo que o fluxo de navegação e as interações funcionem de ponta a ponta.

- Cenário de Teste: Login de usuário
    - Descrição: O teste deve simular a digitação de credenciais válidas nos campos de login, clicar no botão de "Entrar" e verificar se o usuário é redirecionado para a página inicial, exibindo o seu nome ou um painel de controle.
- Cenário de Teste: Criação e exclusão de dados
    - Descrição: O teste deve navegar até uma página de registro (pessoa, atividade, horário, etc.), preencher o formulário, salvar, verificar se o banco recorda os dados e, em seguida, simular também a exclusão desse dado.

Pytest e Django Testas (Testes de Integração): Utilizaremos essas ferramentas para garantir que os diferentes componentes do backend (views, serializers, modelos) se comuniquem corretamente e que as regras de negócio sejam respeitadas.

- Cenário de Teste: Criar nota com usuário autenticado
    - Descrição: Simular uma requisição POST com um token de autenticação válido para o endpoint de criação de notas. O teste deve verificar se a resposta é um

status code 201 Created, e se a nova nota foi persistida no banco de dados com os dados enviados.

- Cenário de Teste: Acesso não autorizado
    - Descrição: Tentar acessar o endpoint de listagem de notas (ex. GET /student/grade) sem um token de autenticação. O teste deve validar que a resposta é um status code 401 Unauthorized, garantindo que a segurança da API está funcionando.
- Cenário de Teste: Atualização de dados
    - Descrição: Simular uma requisição PUT para atualizar um registro existente. O teste deve verificar se a resposta é um 200 OK e se o valor no banco de dados foi realmente alterado como esperado.

7. Implantação

7.1. Ambiente de Produção

- **Serviços:** Docker Compose.
- **Configuração:** O ambiente será configurado em um servidor na nuvem. Utilizaremos o Docker Compose para orquestrar os containers do backend (Django), frontend (NextJS) e do banco de dados (Postgres).

7.2. Processo de Deploy

O processo de deploy será totalmente baseado em containers. Os componentes do projeto serão construídos como imagens Docker e publicados no Docker Hub. A implantação em qualquer servidor remoto ocorrerá por meio de um arquivo compose.yaml. Este arquivo instruirá o Docker Compose a baixar as imagens necessárias, configurar as redes e iniciar todos os containers, garantindo que eles se comuniquem e formem a aplicação completa de forma coesa e isolada.

7.3. Manutenção e Monitoramento

A manutenção e o monitoramento são etapas cruciais para garantir que o School Secretary permaneça disponível, performático e livre de erros após a implantação. Portanto, haverá verificação de bugs frequente, e esses serão desfeitos assim que detectados.

7.4. Estratégias e Ferramentas Propostas

- **Monitoramento de Logs:** Centralizaremos os logs de todas as partes da aplicação (servidor, backend, frontend) para facilitar a busca, análise e depuração de problemas.
- **Alertas e Notificações:** Serão configurados alertas para eventos críticos, como picos de erro ou degradação de performance. As notificações serão enviadas para canais de comunicação da equipe, garantindo que os problemas sejam endereçados prontamente.

8. Cronograma

Data de Início	Data de Término	Descrição da Atividade
04/08/2025	20/08/2025	Fase 1: Setup e Modelagem Completa (Concluída): Definição de requisitos e escopo do projeto. Configuração do ambiente Docker Compose. Criação e migração de todas as estruturas de dados no backend, incluindo gestão de pessoas (alunos, professores, funcionários), dados acadêmicos (notas, presença, desempenho), contratos, agenda, eventos, usuários e permissões.
25/08/2025	27/08/2025	Fase 2: Backend - APIs de Registro de Pessoas (Alunos, Professores, Funcionários): Implementação completa das APIs REST (CRUD) para gerenciar o cadastro de alunos, professores e funcionários.
01/09/2025	03/09/2025	Fase 2: Frontend - Páginas de Cadastro de Pessoas: Criação das interfaces de usuário (UI) e integração com as APIs para o cadastro e listagem de alunos, professores e funcionários.
08/09/2025	10/09/2025	Fase 3: Backend - Autenticação (JWT) e Autorização/Permissões: Configuração completa do sistema de autenticação via JWT. Desenvolvimento da lógica de grupos de usuários e permissões, garantindo acesso diferenciado.
15/09/2025	17/09/2025	Fase 3: Frontend - Páginas de Login e Gerenciamento de Usuários: Desenvolvimento das páginas de login e registro, e implementação de gerenciamento básico de usuários/perfis, tudo integrado com as APIs de autenticação/autorização.

22/09/2025	24/09/2025	Fase 4: Backend - APIs de Notas e Presença: Criação das APIs (CRUD) para o registro e consulta de notas e presenças. Início da lógica para cálculo de desempenho acadêmico.
29/09/2025	01/10/2025	Fase 4: Frontend - Páginas de Notas, Presença e Desempenho: Desenvolvimento das interfaces para lançamento e visualização de notas, registro/consulta de presenças e exibição do desempenho acadêmico inicial.
06/10/2025	08/10/2025	Fase 5: Backend - APIs de Contratos e Geração de PDF: Desenvolvimento robusto da API para gerenciamento de contratos, incluindo a lógica final de backend para geração de PDFs (contratos e outros documentos).
13/10/2025	15/10/2025	Fase 5: Frontend - Páginas de Contratos e Download de PDF: Criação das interfaces para visualização, gerenciamento e download de PDFs de contratos e documentos.
20/10/2025	22/10/2025	Fase 6: Backend - APIs de Agenda de Atividades e Notificações de Eventos: Desenvolvimento das APIs (CRUD) para agenda de atividades e para notificações de eventos (com envio ou registro).
27/10/2025	29/10/2025	Fase 6: Frontend - Páginas de Agenda e Notificações: Implementação das interfaces para a agenda de atividades (calendário, listagem) e para a visualização/gerenciamento de notificações de eventos.

9. Orçamento e Recursos

9.1. Custos

Os custos para o desenvolvimento e operação do School Secretary serão cuidadosamente otimizados para garantir a sustentabilidade do projeto. A estratégia prioriza soluções eficientes e de bom custo-benefício, utilizando tecnologias e serviços que oferecem planos adequados para a fase inicial do sistema.

Servidores e Hospedagem

- **Estimativa de custo mensal:** R$ 50 - R$ 150.

- **Detalhes:** Para as fases iniciais, um Servidor Virtual Privado (VPS) ou um plano de hospedagem em nuvem de entrada, como AWS EC2, será suficiente. Esses serviços oferecem instâncias com 1-2 GB de RAM e 1 vCPU, adequadas para hospedar os containers do projeto. À medida que o número de usuários crescer, será possível escalonar recursos ou adicionar soluções complementares, como cache e CDNs.

Licenças de Software

Não haverá custo, pois tudo que será usado, no que se refere a artigos de software, será livre e/ou gratuito.

Serviços de Terceiros (APIs Externas)

- **Estimativa de custo mensal:** R$ 0.
- **Detalhes:** O projeto é simples e utilizará no máximo o que vem incluído no AWS EC2, ou seja, AWS EBS para armazenar persistentemente os dados.

Custo Total Estimado

- **Custo Mensal Total Estimado: R$ 50 - R$ 150**
- **Considerações:** Este valor reflete o custo mínimo para manter o projeto no ar, com os recursos necessários para o desenvolvimento e operação inicial. O total pode aumentar gradualmente à medida que o número de usuários cresce, exigindo mais recursos de servidor ou a adoção de planos mais robustos para serviços de terceiros.

9.2. Recursos Humanos

- **Equipe:** 4 Desenvolvedores Full Stack.
- **Papéis e Responsabilidades:** Os devs terão a responsabilidade de construir, configurar, testar e realizar a manutenção de todo o projeto.

10. Riscos e Mitigações

10.1. Riscos Potenciais

- Risco 1: Atraso no cronograma devido a imprevistos técnicos, como desafios inesperados no desenvolvimento, problemas de integração entre frontend e backend ou necessidade de refatoração de código.
- Risco 2: Falhas de segurança após a implantação, incluindo vulnerabilidades a ataques como injeção de SQL, acesso não autorizado a dados de usuários ou exploração de brechas na autenticação.
- Risco 3: Perda de dados críticos, causada por erros humanos, falhas de hardware ou ataques maliciosos, impactando a confiabilidade do sistema.
- Risco 4: Falhas de autenticação, permitindo acesso de usuários não autorizados ou exploração de contas, comprometendo a segurança das informações.

10.2. Plano de Mitigação

- Mitigação 1: Alocação de buffer de tempo em cada fase do projeto, permitindo flexibilidade para resolver problemas técnicos sem comprometer o cronograma.
- Mitigação 2: Testes de segurança robustos, incluindo verificação de injeção de SQL, acesso a páginas restritas e testes de penetração, além de isolamento adequado dos dados.
- Mitigação 3: Backup periódico e automatizado de dados, armazenado seguramente em local separado do servidor principal, garantindo restauração rápida em caso de falhas.
- Mitigação 4: Políticas de autenticação rigorosas, com senhas fortes e implementação de autenticação de dois fatores (2FA), dificultando o acesso não autorizado mesmo em caso de comprometimento de credenciais.

11. Apêndices

11.1. Glossário de Termos

Axios: Biblioteca JavaScript para realizar requisições HTTP entre frontend e backend.

Backend: Parte do sistema responsável pela lógica de negócios, processamento de dados e comunicação com o banco de dados.

CRUD (Create, Read, Update, Delete): Conjunto de operações básicas para manipulação de dados.

Django: Framework web em Python utilizado para desenvolvimento do backend.

Django REST Framework (DRF): Extensão do Django para criação de APIs RESTful.

Frontend: Parte do sistema com a qual o usuário interage diretamente, geralmente composta por interfaces visuais.

Next.js: Framework React para desenvolvimento de aplicações web com suporte a SSR (Server-Side Rendering) e TypeScript.

TypeScript: Linguagem baseada em JavaScript que adiciona tipagem estática, aumentando a robustez do código.

PostgreSQL: Sistema de gerenciamento de banco de dados relacional, utilizado para armazenar dados estruturados.

VPS (Servidor Virtual Privado): Servidor virtualizado que oferece recursos dedicados para hospedagem de aplicações.

Wireframe: Representação esquemática da interface de uma aplicação, focada na estrutura e fluxo de navegação.

Mockup: Protótipo visual detalhado de uma interface, mostrando design e elementos gráficos.

CDN (Content Delivery Network): Rede de distribuição de conteúdo que melhora a entrega de arquivos e performance da aplicação.

2FA (Two-Factor Authentication): Método de autenticação que requer dois fatores diferentes para validar o acesso do usuário.

AWS EC2: Instância de servidor virtual da Amazon Web Services, adequada para aplicações de baixa demanda inicial.

11.2. Referências

1. Django (Backend): https://docs.djangoproject.com/en/5.2/
2. Django REST Framework (Backend): https://www.django-rest-framework.org/
3. Next.js (Frontend): https://nextjs.org/docs
4. TypeScript (Frontend): https://www.typescriptlang.org/docs/
5. Vitest (Frontend testing): https://vitest.dev/guide/
6. Figma (Interface design): https://www.figma.com/

7. React (Frontend): https://react.dev/learn
8. Axios (Frontend): https://axios-http.com/docs/intro
9. PostgreSQL (Banco de Dados): https://www.postgresql.org/docs/current/index.html
10. Docker (Containerização): https://docs.docker.com/engine/
11. Docker Compose (Instalação e Orquestração): https://docs.docker.com/compose/