CODATA AI: uma experiência com modelos de GPT no setor público

Pedro Lucas da Silva dos Santos

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2025

Pedro Lucas da Silva dos Santos

CODATA AI: uma experiência com modelos de GPT no setor público

Relatório Técnico apresentado ao curso de Ciência da Computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Adriana Carla Damasceno

Maio de 2025

Catalogação na publicação
Seção de Catalogação e Classificação

S237c Santos, Pedro Lucas da Silva Dos.
CODATA AI: uma experiência com modelos de GPT no setor público / Pedro Lucas da Silva Dos Santos. - João Pessoa, 2025.
45 f. : il.

Orientação: Adriana Carla Damasceno.
TCC (Graduação) - UFPB/CCSA.

1. Inteligência Artificial. 2. CODATA. 3. GPT. 4. OpenAI. I. Damasceno, Adriana Carla. II. Título.

UFPB/CI CDU 004.8

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Ciência da Computação intitulado ***CODATA AI: uma experiência com modelos de GPT no setor público*** de autoria de Pedro Lucas da Silva dos Santos, aprovado pela banca examinadora constituída pelos seguintes professores:

Profa. Dra. Adriana Carla Dmasceno
Universidade Federal da Paraíba

Profa. Dra. Danielle Rousy Dias Ricarte
Universidade Federal da Paraíba

Profa. Dra. Cecir Barbosa de Almeida Farias
Universidade Federal de Campina Grande

João Pessoa, 18 de maio de 2025

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

No laser da madame, no radinho da empregada.

DEDICATÓRIA

Eu dedico este trabalho a duas pessoas muito especiais na minha vida: minha mãe, fonte de motivação para tudo o que fiz e faço para crescer e ser alguém neste mundo, e minha avó materna, provedora de um carinho inconsciente que talvez eu nunca compreenda e que partiu antes que eu pudesse lhe proporcionar tudo de bom que planejei. Vocês são as Marias da minha vida. Te amo, vó, espero te encontrar no céu.

AGRADECIMENTOS

Gostaria de agradecer a todos aqueles que foram a minha rede de apoio ao longo dessa jornada e que sempre acreditaram em mim, em especial:

À minha madrinha Orzila, aquela que cuidou, investiu e me deu a oportunidade de virar gente.

Aos meus irmãos Diego, Aglaé e Bruno, que sempre acreditaram em mim e me incentivaram a ir atrás do que eu acreditava.

Ao homem que tenho como pai, Zezinho, que deu a mim e à minha mãe um lugar para morar e nos presenteou com minha irmã, Lara.

Aos meus amigos do Centro de Informática, que me fizeram sorrir e ajudaram a tornar os dias mais lúdicos e menos mórbidos durante a graduação. Sem vocês, eu não teria sobrevivido ao processo.

Aos meus colegas de trabalho Daniel Sorrentino e Rafael Vasconcelos, que acreditaram em mim e no projeto.

À CODATA, por me proporcionar um ambiente no qual pude aprender, ensinar e crescer, tanto como profissional quanto como ser humano.

Ao meu amiguinho Tobias, que esteve comigo em todas as fases decisivas que me trouxeram até aqui — do ensino médio até o penúltimo período do curso. Ele sempre ia ao meu quarto enquanto eu estudava, passava por debaixo da cadeira e saía. Parecia saber exatamente quando eu estava cansado e tenso. Eu te amo, meu bichinho.

RESUMO

A adoção da inteligência artificial (IA) no setor público alinha-se ao objetivo de otimizar processos e aumentar a eficiência na prestação de serviços à sociedade. Nesse contexto, a Companhia de Processamento de Dados da Paraíba (CODATA) busca soluções inovadoras para aprimorar o trabalho de seus colaboradores, de modo a favorecer o aumento da produtividade em todos os departamentos da organização.

Apesar do avanço das ferramentas de IA, muitas soluções disponíveis não atendem plenamente às demandas específicas do setor público, como segurança, confiabilidade e personalização. Esse cenário evidencia uma oportunidade de pesquisa para o desenvolvimento de uma aplicação que supra essas lacunas, alinhando-se às necessidades institucionais da companhia.

Este trabalho tem como objetivo narrar o processo de desenvolvimento da aplicação CODATA AI, que faz uso dos modelos da OpenAI para apoiar e agilizar o processo de escrita de documentos por parte dos analistas de requisitos da empresa. A proposta é oferecer uma solução segura e adaptável, diferenciando-se das opções existentes no mercado.

A metodologia adotada, baseando-se na *Design Science Research* (DSR), envolveu o levantamento de requisitos junto à equipe da CODATA, a narrativa do processo de desenvolvimento da aplicação e a validação do sistema. Além disso, foram discutidos os principais desafios técnicos, lições aprendidas, limitações e perspectivas futuras da ferramenta.

Os resultados demonstraram que a aplicação foi bem-sucedida em auxiliar o trabalho dos analistas de requisitos, automatizando rotinas e, por conseguinte, reduzindo o tempo de escrita de documentos. A primeira versão do sistema em produção provou ser estável e eficiente para esse tipo de tarefa; no entanto, foram identificadas oportunidades de melhorias e possíveis cenários de expansão de funcionalidades.

Palavras-chave: Inteligência Artificial, Setor Público, GPT, OpenAI, Paraíba. CODATA. Design Science Research. Requisitos.

ABSTRACT

The adoption of artificial intelligence (AI) in the public sector is aligned with the goal of optimizing processes and increasing efficiency in the provision of services to society. In this context, the Data Processing Company of Paraíba (CODATA) seeks innovative solutions to improve the work of its employees, in order to favor increased productivity in all departments of the organization.

Despite the advancement of AI tools, many available solutions do not fully meet the specific demands of the public sector, such as security, reliability, and customization. This scenario highlights a research opportunity for the development of an application that fills these gaps, aligning with the institutional needs of the company.

This work aims to register the development process of the CODATA AI application, which uses OpenAI models to support and streamline the document writing process by the company's requirements analysts. The proposal is to offer a secure and adaptable solution, differentiating itself from the options available on the market.

The adopted methodology, based on *Design Science Research* (DSR), involved gathering requirements from the CODATA team, narrating the application development process, and validating the system. In addition, the main technical challenges, lessons learned, limitations, and future prospects of the tool were discussed.

The results demonstrated that the application was successful in assisting the work of requirements analysts, automating routines and, consequently, reducing the time spent writing documents. The first version of the system in production proved to be stable and efficient for this type of task; however, opportunities for improvement and possible scenarios for expanding functionality were identified.

Keywords: Artificial Intelligence, Public Sector, GPT, OpenAI, Paraíba. CODATA. Design Science Research. Requirements.

LISTA DE FIGURAS

1 Diagrama de Arquitetura do sistema. 29

2 Diagrama Entidade-Relacionamento do sistema. 30

3 Primeira tela desenvolvida do projeto. 31

4 Organização das pastas do projeto no Gitlab. 32

5 Tela de Login do sistema. 32

6 Exemplo de endpoint da API. 33

7 Primeira versão da tela principal do sistema em Vue.js. 34

8 Página de administração do sistema. 34

9 Versão definitiva da tela principal em ambiente de homologação. 36

10 Imagem da página inicial do SIGAA fornecida ao modelo. 41

11 Prompt de entrada dado ao modelo. 42

12 Histórias de usuário geradas na resposta. 42

LISTA DE TABELAS

1 Caso de teste [T01]. 37
2 Caso de teste [T02]. 37
3 Caso de teste [T03]. 38
4 Caso de teste [T04]. 38
5 Caso de teste [T05]. 39
6 Caso de teste [T06]. 39
7 Caso de teste [T07]. 40
8 Caso de teste [T08]. 40

LISTA DE ABREVIATURAS

API - Application Programming Interface

BDD - Based Driven Development

CD - Continuous Delivery

CI - Continuous Integration

CODATA - Companhia de Processamento de Dados da Paraíba

CRUD - Create, Read, Update and Delete

DBA - Database Administrator

DevOps - Development Operations

DRF - Django REST Framework

DSR - Design Science Research

GEDES - Gerência de Desenvolvimento de Sistemas

GDPR - General Data Protection Regulation

GPT - Generative Pre-trained Transformer

IA - Inteligência Artificial

JSON - JavaScript Object Notation

JWT - JSON Web Token

LGPD - Lei Geral de Proteção de Dados

MVP - Minimum Viable Product

OIDC - OpenID Connect

PO - Product Owner

QA - Quality Analyst (Analista de Qualidade)

REST - Representational State Transfer

RF - Requisito Funcional

RNF - Requisito Não-Funcional

SGBD - Sistema de Gerenciamento de Banco de Dados

SSO - Single Sign-On

XSS - Cross-Site Scripting

Sumário

1 INTRODUÇÃO **17**

1.1 Contexto e Motivação 17

1.2 Definição do Problema 18

1.3 Objetivos 18

1.4 Estrutura do Relatório 19

2 DESCRIÇÃO DO PROCESSO **20**

2.1 Atores do Processo 20

2.2 Metodologia 21

3 TECNOLOGIAS UTILIZADAS **23**

3.1 Inteligência Artificial Generativa 23

3.2 Back-end 23

3.3 Front-end 24

3.4 Virtualização e Controle de Versão 25

4 DESENVOLVIMENTO DA SOLUÇÃO **26**

4.1 Elicitação de Requisitos 26

4.2 Requisitos Funcionais 26

4.3 Requisitos Não Funcionais 27

4.4 Regras de Negócio 28

4.5 Arquitetura do Sistema 29

4.6 Modelagem do Banco de Dados 29

4.7 Narrativa do Processo 31

4.8 Validação 36

4.9 Demonstração 41

4.10 Análise e Discussão dos Resultados 42

5 CONCLUSÕES E TRABALHOS FUTUROS **45**

REFERÊNCIAS **45**

1 INTRODUÇÃO

O desenvolvimento de sistemas vai além da mera codificação e testagem de componentes, atividades mecânicas e sequenciais no dia a dia laboral. Trabalhar nessa área é, antes de tudo, identificar problemas e viabilizar soluções. Talvez o maior anseio de um programador seja criar algo que verdadeiramente impacte a sociedade e melhore a vida das pessoas. Nesse sentido, este tópico se propõe a esclarecer os motivos e a importância do projeto CODATA AI para os interesses da Companhia de Processamento de Dados da Paraíba (CODATA), apresentando seu contexto, o problema que busca solucionar e as expectativas em torno de sua implementação.

1.1 Contexto e Motivação

A popularização de ferramentas de consulta e geração de *prompts* com uso de IA generativa, como o ChatGPT, da OpenAI, e o Gemini, do Google, favoreceu o ganho de produtividade em diversas áreas de conhecimento e em diferentes contextos. A adesão dessas tecnologias não só trouxe vantagens para a sociedade, mas também gerou dúvidas e preocupações [1]. No contexto do serviço público, uma das preocupações das empresas, órgãos e secretarias diz respeito ao fornecimento indevido de informações e dados governamentais sensíveis por parte de seus funcionários para esses sistemas. Não existem garantias de que esses dados estarão guardados apenas na sessão da conversa e que posteriormente serão destruídos, e nem de que tipo de informação os funcionários estão utilizando como entrada.

Além disso, a disponibilidade desse tipo de serviço pode ser afetada pelo não cumprimento de regulações em determinadas localidades. Em 2023, o ChatGPT foi bloqueado na Itália [2] após a identificação de inconsistências que violavam as diretrizes do Regulamento Geral sobre a Proteção de Dados (GDPR) europeu. No Brasil, o tratamento de dados pessoais em meios digitais é regido pela Lei Geral de Proteção de Dados (LGPD) [3]. As organizações, muitas vezes vistas como entes controladores de dados, para os efeitos desta lei, são as que detêm a responsabilidade pelas decisões relativas a custódia e manipulação desses dados. Nesse sentido, surge a necessidade de criar um ambiente controlado e seguro para o trânsito de dados, no qual a organização pode acompanhar e auditar qualquer uso indevido de informação, sem abrir mão da vanguarda tecnológica.

A engenharia de requisitos, segundo Marco Túlio Valente [4], é o nome que se dá ao conjunto de atividades relacionadas com a descoberta, análise, especificação e manutenção dos requisitos de um sistema. Constitui uma etapa fundamental dentro do ciclo de vida do software, pois é nessa fase que o time de requisitos fará o levantamento da viabilidade e dos recursos necessários para a execução do projeto, de acordo com as necessidades dos clientes e *stakeholders*. O uso de qualquer tecnologia que se proponha a automatizar e

acelerar o andamento de atividades durante esse processo é vista como uma vantagem competitiva que pode diminuir custos e garantir entregas no prazo. É daí que nasce a ideia de desenvolver uma aplicação semelhante aos *chats* generativos que ganharam popularidade na *internet*.

1.2 Definição do Problema

O processo de elicitação de requisitos feito na empresa faz uso de histórias de usuário, que podem ser entendidas como descrições em alto nível de como um sistema pode ser utilizado em alguma tarefa em particular. As histórias descrevem o que as pessoas fazem, quais informações usam e produzem, e quais sistemas podem adotar nessa etapa [5]. A concepção e escrita dessas histórias, que iriam compor o documento de requisitos a ser validado pelos clientes, era manual, intelectualmente desgastante e morosa. Era necessário pensar nos cenários possíveis que abarcassem as demandas existentes e fazer com que isso se refletisse em algo concreto e tangível do ponto de vista documental.

As histórias de usuário são um elemento importante dentro da metodologia utilizada pela companhia, centrada no *Based Driven Development* (BDD). O BDD, segundo Smart e Molak [6], é uma abordagem de desenvolvimento colaborativo entre as equipes, no qual a comunicação e o trabalho integrativo entre *Product Owner* (PO), analista de qualidade e desenvolvedores buscam estimar o comportamento de determinados entes em determinadas situações. Em suma, trabalha-se com os verbos *given* (dada uma situação), *when* (quando ocorrer algo) e *then* (então se espera algo). Um exemplo seria: quando um usuário fizer o login com as credenciais de gestor, então o redirecione para a página de administração.

Do ponto de vista da gerência, existia um temor referente às políticas de tratamento e armazenamento de dados por parte dos serviços existentes, que não são consistentes. Os modelos de linguagem natural são alimentados a partir de grandes conjuntos de dados, que podem ser textos, artigos ou conversas [7] disponíveis na internet. Não se tem a certeza de que os dados de entrada fornecidos são considerados somente no contexto da conversa ou se são aproveitados para treinamentos incrementais desse modelo. O desejo da gestão era ter em mãos um ambiente hospedado e desenvolvido com infraestrutura própria, customizável, e que permitisse algum nível de auditoria em relação ao uso por parte dos colaboradores.

1.3 Objetivos

O projeto CODATA AI surge com o objetivo de ser uma alternativa confiável e auditável para a utilização de serviços de processamento de linguagem natural por servidores

públicos, usada para automatizar e dar celeridade ao processo de elicitação de requisitos. A aplicação foi concebida para ter escopo pequeno e interface de usuário simples, buscando a adesão por parte dos funcionários, além da possibilidade de ser customizada para ser oferecida às secretarias e órgãos que já são atendidos pela empresa.

Dessa maneira, o CODATA AI pode ser entendido como um artefato da *Design Science Research* (DSR), pois é uma proposta de inovação para resolver um problema do mundo real na área de tecnologia da informação, ao mesmo tempo que é objeto de estudo científico. A documentação desse processo visa não só gerar utilidade pragmática para o setor público por meio de artefatos, mas também contribuir com o fortalecimento das bases teóricas que alimentam a Ciência da Informação e o Design da Informação, áreas que complementam a DSR, segundo Rodrigues [8].

1.4 Estrutura do Relatório

O trabalho está organizado em cinco capítulos. O primeiro, **Introdução**, apresenta o contexto, a motivação, a delineação do problema e os objetivos pretendidos, bem como a estrutura do relatório. O segundo capítulo, **Descrição do Processo**, aborda o referencial teórico e a metodologia que rege este trabalho, além de identificar os atores do processo. O capítulo de **Tecnologias Utilizadas** versa sobre as ferramentas que foram escolhidas para desenvolver o projeto.

O quarto capítulo, **Desenvolvimento da Solução**, é composto pela descrição dos requisitos funcionais e não funcionais, narrativa do processo de desenvolvimento da solução proposta, validação do sistema e análise e discussão dos resultados. Por fim, nas **Conclusões**, há uma síntese das lições aprendidas durante o processo de concepção e desenvolvimento do sistema, assim como são expostas as limitações do projeto e o que se espera para o futuro.

2 DESCRIÇÃO DO PROCESSO

A descrição do processo envolve a identificação dos atores, suas funções e interações, além da metodologia utilizada no desenvolvimento do software. Esse detalhamento assegura que todas as etapas sejam bem definidas, desde a concepção até a validação do sistema. Os atores do processo foram os analistas de requisitos, gerência e diretoria, e os desenvolvedores do sistema. A metodologia foi baseada na *Design Science Research* (DSR) e suas etapas, incluindo a identificação e motivação do problema, definição dos objetivos para uma solução, projeto e desenvolvimento, demonstração, avaliação e comunicação.

2.1 Atores do Processo

Os analistas de requisitos são responsáveis pelo processo de análise e elicitação de requisitos na CODATA. Eles utilizam o padrão BDD para a construção das histórias de usuário, que serão utilizadas tanto para a implementação das funcionalidades do sistema por parte da equipe de desenvolvimento quanto para a contagem dos pontos de função para a cobrança do serviço por parte da empresa. O sistema CODATA AI foi pensado originalmente para automatizar o processo de criação de histórias de usuário, principalmente a partir da análise de protótipos e imagens de documentos.

Apesar do escopo pequeno e definido do projeto, a ideia da gerência e da diretoria, também atores do processo, era tornar o chat customizável e portável, para no futuro ser oferecido a secretarias e órgãos que compõem o rol de serviços da CODATA, com ajustes que se adequassem às necessidades de cada entidade. O sistema foi feito seguindo os seguintes preceitos: ser intuitivo, de fácil uso, de fácil portabilidade dos dados e de fácil manutenção, utilizando tecnologias consolidadas e atuais em relação ao mercado de trabalho. Além disso, o interesse de desenvolver o CODATA AI residia no fato de que o serviço da OpenAI havia sido comprado, mas ainda não utilizado em nenhum sistema a nível de produção. Assim, o projeto também serviu de plataforma para identificar os pontos fortes e as limitações da API do Chat.

Por outro lado, os desenvolvedores são aqueles que vão codificar, validar, manter e evoluir o sistema. Os responsáveis pela gestão do repositório, atualizações, melhorias, correções de bugs, entre outros. A equipe foi formada por dois desenvolvedores: um ficou responsável pelo design e codificação do front-end, utilizando Vue.js e suas bibliotecas, e este autor ficou com a parte de back-end, trabalhando com a modelagem do banco de dados, desenvolvimento e integração da API, a partir do Django e Django REST Framework (DRF).

2.2 Metodologia

A metodologia deste relatório foi baseada na *Design Science Research*, um método para construção de conhecimento que busca unir teorias e métodos científicos à pragmática e lições aprendidas do processo de design e construção de artefatos inovadores que se propõem a resolver um problema do mundo real. A justificativa reside no fato de que, segundo Pimentel e Filippo [22], nossas criações não costumam ser reconhecidas como objetos válidos para uma pesquisa científica, além de não serem abarcadas por métodos tradicionais, como experimento, *survey* ou etnografia. Dessa maneira, a DSR oferece um respaldo metodológico apropriado para pesquisas que versam sobre o desenvolvimento e a avaliação de artefatos [23], sendo assim utilizada para guiar a implementação do software.

A DSR é composta pelas seguintes etapas: identificação e motivação do problema, definição dos objetivos para uma solução, projeto e desenvolvimento, demonstração, avaliação e comunicação.

Na identificação e motivação do problema, discutiram-se os principais temores e indagações que motivaram a gerência da CODATA a iniciar o projeto CODATA AI. A falta de confiabilidade e transparência no tocante ao tratamento e armazenamento de dados em ferramentas existentes no mercado, tais como o ChatGPT e o Gemini, além de ponderações que envolveram a conformidade com a LGPD, o compartilhamento de dados governamentais e de negócios sensíveis pelos funcionários, e o desejo de criar uma ferramenta própria e portável foram razões que induziram a gerência a pensar no desenvolvimento de uma solução para uso interno. Nesse ínterim, a demanda foi apresentada pela chefia a este autor em sua primeira semana de trabalho e iniciaram-se os preparativos para o desenvolvimento da solução.

Na definição dos objetivos para uma solução, desde o início, a aplicação foi pensada para ser o mais simples possível, a fim de ganhar aderência por parte dos colaboradores que iriam utilizá-lo. A ideia era construir uma interface simples, com escopo pequeno, baixa complexidade, e que fizesse proveito das funcionalidades dos modelos pagos da OpenAI, que a CODATA tinha à disposição. Além disso, o sistema seria uma alternativa segura e auditável em relação aos serviços de *chat* que existiam no mercado, com o diferencial centrado no ambiente de administração, que seria utilizado pela gerência.

Na etapa de projeto e desenvolvimento, narrou-se o processo de desenvolvimento do sistema, detalhando as atividades de análise de requisitos, codificação, testagem e homologação do sistema. À princípio, o CODATA AI seria feito totalmente em Django, no entanto, as limitações de design e interação com o usuário presentes no *django template*, um tipo de documento que mescla html a comandos python, acarretaram na escolha de uma *framework* apropriada para o *front-end*, o Vue.js. Por conseguinte, um desenvolvedor foi escalado para auxiliar este autor na codificação, ficando responsável por conceber e

desenvolver a interface de usuário.

Durante a etapa de demonstração e avaliação, o sistema, por ser um projeto experimental e de uso interno na empresa, não dispôs de um ambiente de QA, onde os analistas de qualidade fariam testes automatizados para validar o sistema antes de submetê-lo ao ambiente de produção. Quando a primeira versão do projeto foi disponibilizada no ambiente de homologação, os analistas de requisitos e a gerência, principais atores do processo, foram apresentados ao sistema. Nessa etapa, as primeiras interações e sugestões de melhorias foram propostas pelos usuários.

Nesse cenário, o projeto se limitou aos resultados obtidos em testes do tipo caixa-preta realizados pelos colaboradores da GEDES, nos quais a validação das funcionalidades não leva em consideração a implementação interna ou a estrutura do código. O objetivo do teste de caixa-preta é verificar se as entradas fornecidas produzem as saídas esperadas, conforme os requisitos que foram especificados. Falhas e defeitos identificados durante o processo, bem como sugestões de melhorias, foram repassadas diretamente aos desenvolvedores do projeto.

Por fim, a comunicação desta experiência, destacando os resultados e lições obtidas, à comunidade acadêmica se deu nas seguintes etapas: o *deploy* do sistema em ambiente de produção, a disponibilização para uso interno e a publicação deste relatório, que foi concebido para narrar o processo de desenvolvimento de um artefato inovador proposto para resolver demandas do setor público. Vale salientar que, em conformidade com as normas internas da CODATA, os códigos-fonte dos projetos desenvolvidos na empresa são de propriedade dela, não podendo ser disponibilizados ao público. Dessa forma, este trabalho limita-se a tornar pública a descrição do processo e as imagens que comprovam o desenvolvimento da solução.

3 TECNOLOGIAS UTILIZADAS

3.1 Inteligência Artificial Generativa

A inteligência artificial generativa é uma forma de IA que pode gerar autonomamente novos conteúdos, como texto, imagens, áudio e vídeo [31]. Atualmente, o modelo de linguagem mais popular nesse campo é o ChatGPT, lançado em novembro de 2022 pela OpenAI. Seu funcionamento é baseado em transformadores generativos pré-treinados (GPTs), uma categoria de redes neurais sofisticadas desenvolvidas para lidar com tarefas de processamento de linguagem natural [16]. Esses modelos utilizam a arquitetura de transformadores e passam por um pré-treinamento não supervisionado, utilizando vastos conjuntos de dados sem rotulação.

A OpenAI disponibiliza uma *Application Programming Interface* (API) para a utilização desse serviço. API é um mecanismo que permite que dois componentes de software se comuniquem usando um conjunto de definições e protocolos, análogo a um contrato entre tais aplicações [9]. A precificação desse serviço se dá a partir da contagem dos *tokens* que são processados pelo modelo a cada requisição. Os *Tokens*, nesse contexto, são sequências comuns de caracteres encontradas em um conjunto de texto, utilizadas pelos modelos de GPT em seu processamento [15].

A difusão de ferramentas baseadas em *Large Language Models* (LLMs), modelo de IA que utiliza redes neurais profundas para compreender e gerar linguagem natural de forma autônoma, em uma interface de fácil interação por humanos, tem atraído o interesse de pesquisadores e corporações dos mais diversos nichos para além da ciência da computação, como nas áreas da saúde e educação [32]. Diante disso, a CODATA enxergou uma oportunidade de explorar esse tipo de tecnologia, sob a expectativa de aumentar a produtividade de seus colaboradores.

3.2 Back-end

Neste projeto, a API foi utilizada como intermediário na comunicação entre o banco de dados e a interface de usuário (*front-end*), na qual foi utilizado o serviço de IA da OpenAI, tendo à disposição os seus principais modelos de linguagem generativos (GPT-4o, GPT-4o-mini, o1 e o1-mini, entre outros) [24] na versão paga. Os *endpoints* utilizados foram os do Chat Completions, uma funcionalidade que permite interagir com os modelos de maneira semelhante a uma conversa. Cada mensagem enviada é interpretada pelo modelo, que processa o conteúdo a partir dos tokens e gera uma resposta. Dentre os tipos de mensagem aceitos pelo Chat, foram utilizados no projeto o texto e a imagem.

Por outro lado, o *back-end* é uma parte da aplicação que possui duas responsabilidades principais: executar lógicas mais complexas e armazenar os dados da aplicação [12].

No sistema, essa camada é composta pela API e pelo banco de dados. O Django REST Framework, em conjunto com o Django, foi escolhido para ser o elo entre banco de dados, a API e o usuário, devido à sua robustez, codificação enxuta, autenticação pronta para uso, com suporte aos principais padrões (JWT, OAuth, OAuth2, entre outros), permissões personalizáveis, facilidade de integração com o front-end e suporte a APIs REST bem estruturadas [17]. A experiência profissional deste autor nessa tecnologia e na linguagem de programação Python também foi um fator que propiciou essa decisão.

O banco de dados é uma coleção organizada de informações que servirá a uma aplicação, geralmente armazenada em um computador [10] e gerenciada por Sistemas Gerenciadores de Banco de Dados (SGBDs). Seguindo o padrão da empresa e o escopo do projeto, foi utilizado o banco de dados relacional PostgresSQL, uma plataforma open-source (código aberto e gratuita) baseada no modelo relacional, no qual as entidades são representadas por tabelas e seus relacionamentos. Cada atributo dessa entidade é uma coluna da tabela. A ferramenta de gerenciamento escolhida foi o DBeaver, por oferecer, entre outras coisas, a visualização completa do diagrama entidade-relacionamento da base de dados.

3.3 Front-end

O Front-end está relacionado com a interface gráfica do projeto [11]. Um design eficiente dessa camada garante uma experiência de usuário fluida, prazerosa e intuitiva, dentro dos objetivos do sistema. Nesse contexto, em conformidade com as tecnologias utilizadas pela CODATA, a parte de interação com o usuário ficou a cargo do Vue.js, uma *framework* baseada em Javascript [18] com bibliotecas que facilitam rotinas de autenticação, validação de formulários, gerenciamento de estado, entre outros. A curva de aprendizado do Vue.js é moderada e acessível, sendo intuitiva para desenvolvedores iniciantes e pouco experientes na área de front-end. O gerenciamento de estados, isto é, a forma como os dados de uma aplicação são armazenados e atualizados, foi desenvolvido junto com a biblioteca Pinia. Por outro lado, o design e a estilização ficaram a cargo das bibliotecas PrimeVue e Tailwind CSS.

Também foi realizada nessa camada a etapa de autenticação no sistema, por meio do Single Sign-On (SSO), um esquema de autenticação que permite que os usuários façam login uma vez usando um único conjunto de credenciais e acessem várias aplicações durante a mesma sessão [13]. A adoção do SSO foi possível através do Keycloak, uma ferramenta *open-source* de gerenciamento de autenticação e acesso de usuários [20]. No Keycloak, cada aplicação ou serviço é um cliente que faz parte de um grupo chamado *realm* (reino). O usuário que se autentica em um desses serviços tem acesso a todos os outros que estão dentro desse *realm*. A nível de requisição, a autenticação é realizada por meio da geração de *tokens* de autorização que pertencem ao protocolo OAuth 2.0, uma

versão mais flexível e escalável que o seu antecessor, o OAuth 1.0 [21].

3.4 Virtualização e Controle de Versão

A virtualização do ambiente teve como fim tornar a configuração do sistema flexível e portável entre os sistemas operacionais a nível de desenvolvimento. Para tal, foi utilizado o Docker no processo de criação e gerenciamento de containers, unidades de software que encapsulam microsserviços e suas dependências [28]. O projeto foi segmentado em três containers: api, que encapsulou os códigos do Django, web, que possuía os códigos Vue.js, e sso, no qual rodava o servidor do Keycloak para autenticação. A orquestração dos containers a nível de homologação e produção, dentro da infraestrutura da CODATA, ficou a cargo da ferramenta Kubernetes.

Por outro lado, o versionamento do código foi realizado utilizando o GitLab. Os sistemas de controle de versão são programas de software que ajudam a controlar as alterações feitas no código ao longo do tempo [14]. As versões dos códigos são organizadas em *commits* que são submetidos aos serviços de hospedagem. Versionar o código traz segurança para o trabalho do desenvolvedor, pois cada passo dele estará registrado, podendo ser alterado, revertido ou excluído, de acordo com as demandas do projeto. Versões de lançamento, por sua vez, são catalogadas por *Tags*, que também servem para referenciar pontos específicos do código, marcando o seu versionamento no repositório [29].

4 DESENVOLVIMENTO DA SOLUÇÃO

Neste tópico, encontram-se informações relativas à implementação propriamente dita do sistema, a começar pelo detalhamento do processo de elicitação e listagem de requisitos, tanto funcionais quanto não funcionais. Também é discutida a arquitetura do sistema, detalhando a divisão da aplicação em microsserviços, além de versar sobre a modelagem do banco de dados. A narrativa do processo é a etapa da codificação em si, desde o primeiro versionamento até a disponibilização do ambiente de testes. A validação do sistema oferece um panorama dos tipos e dos casos de teste nos quais a aplicação foi submetida, a fim de mensurar a conformidade dela com os requisitos inicialmente apresentados. Finalmente, o tópico de análise e discussão dos resultados traz uma síntese de todo o processo de desenvolvimento, a partir do que era esperado e do que foi obtido. Também foram discutidas as limitações do sistema, o que foi aprendido e o que se pode esperar no futuro.

4.1 Elicitação de Requisitos

O processo de elicitação de requisitos consistiu em tomar ciência dos principais interesses da gerência quanto à utilização dos modelos da OpenAI, dentro das capacidades orçamentárias da CODATA. Essa etapa não consumiu uma quantidade grande de tempo, já que o sistema foi pensado para ser simples e funcional, com um escopo relativamente pequeno. Dito isso, foram levantados os seguintes requisitos:

4.2 Requisitos Funcionais

[RF001] - Autenticar com Single Sign-On

A aplicação deve se integrar ao sistema de *login* único utilizado pelos servidores do estado da Paraíba e presente na maioria dos serviços desenvolvidos pela CODATA. Os usuários podem logar tanto pelo CPF cadastrado no estado, quanto pelo Gov.br.

[RF002] - Listar conversas

A interface da aplicação deve conter uma área de listagem das conversas do usuário que está logado.

[RF003] - Listar mensagens

A interface da aplicação deve conter uma área de visualização e listagem das mensagens enviadas pelo usuário e das respostas geradas pela API.

[RF004] - Anexar imagens

Desenvolver um botão ou um espaço para selecionar ou arrastar imagens que serão enviadas junto ao texto no corpo da mensagem.

[RF005] - Página de Administração

Implementar uma página de administração a ser utilizada pelos gestores para auditoria, onde será possível visualizar as atividades dos usuários e a troca de mensagens desses com a API.

[RF006] - Registrar mensagens

O sistema deve salvar todas as mensagens, sejam elas textos ou imagens, geradas no banco de dados, para fins de auditoria.

[RF007] - Registrar atividades de usuário

O sistema deve salvar as atividades do usuário dentro da aplicação: login, criação, edição e exclusão de conversas e mensagens enviadas, para fins de auditoria.

[RF008] - Exportar conversas

A aplicação deve ser capaz de gerar um arquivo com todas as mensagens armazenadas no banco de dados para fins de portabilidade, caso o serviço de GPT seja trocado por outro no futuro.

4.3 Requisitos Não Funcionais

[RNF001] - Sessão de Usuário

O usuário deve ser capaz de realizar *login* e *logout* no sistema, por meio do SSO, usando as credenciais de servidor ou as do Gov.br.

[RNF002] - Responsividade

A interface de usuário deve ser compatível com o tamanho das telas de computador, tablets e telefones, permitindo a navegação fluida e intuitiva em todos esses dispositivos.

[RNF003] - Ações de Usuário

O usuário autenticado no sistema pode criar uma conversa, adicionando um título, que pode ser editável, e excluí-la, bem como pode enviar mensagens com textos e imagens que serão interpretadas pela API.

[RNF004] - Auditoria

O sistema deve ser capaz de armazenar dados de interesse da gestão, para fins de auditoria e futura portabilidade do sistema.

[RNF005] - Comunicação em Tempo Real

O sistema deve garantir que a troca de mensagens aconteça em tempo real, sem atrasos perceptíveis para o usuário. Para isso, as respostas da API não devem ser armazenadas em cache, sendo sempre atualizadas a cada requisição.

[RNF006] - Portabilidade

Os dados das tabelas conversa e mensagem devem ser portáveis para serem reutilizados em outras APIs, por meio de representações de dados amplamente disseminadas e aceitas no mercado.

[RNF007] - Segurança

Por ser uma aplicação de troca de mensagens baseada na entrada do usuário, o sistema deve ter meios de mitigar o principal tipo de ataque para esse nicho: o Cross-Site Scripting (XSS), que manipula as requisições de um servidor através da ingestão de código malicioso que será incorporado à estrutura do site.

4.4 Regras de Negócio

[RN001] - Visibilidade de Conversas

O usuário logado só poderá visualizar e interagir com as conversas que foram criadas por ele. Por outro lado, os gerentes terão acesso a todas as conversas e mensagens presentes no sistema, por meio da página de administração.

[RN002] - Dados para Auditoria

O sistema deve armazenar os seguintes dados: atividades de *login* e *logout*, operações de CRUD (criar, ler, editar e excluir) em todas as tabelas (usuário, conversa e mensagem) e requisições em geral (*logs*).

[RN003] - Representação de Dados

Os dados descritos no RF008 e RNF006 deverão ser armazenados no formato JSON (JavaScript Object Notation).

4.5 Arquitetura do Sistema

Inicialmente, a aplicação foi pensada para rodar em um único serviço. O Django permite criar APIs e templates em um lugar só. Por outro lado, a customização da interface de usuário é custosa e fica mais trabalhosa à medida que o sistema cresce. Dessa maneira, optou-se por decompor o sistema nos seguintes microsserviços: front-end (Vue.js), API (Chat Completions - OpenAI), back-end (Django e Django REST) e banco de dados (PostgreSQL), tornando o projeto flexível e permitindo a divisão de responsabilidades. A Figura 1 apresenta o diagrama de arquitetura do sistema.

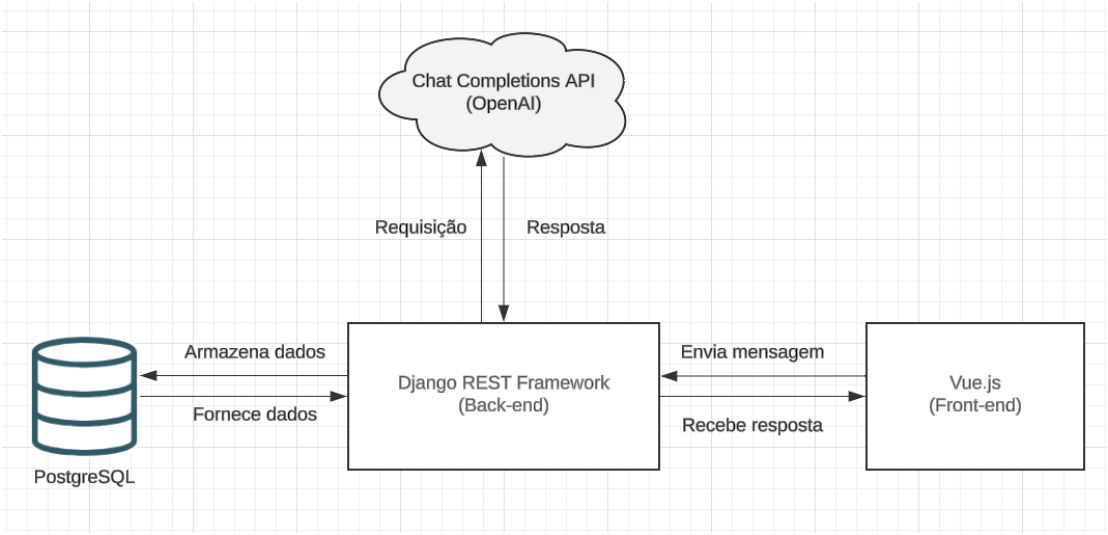

Figura 1: Diagrama de Arquitetura do sistema.

4.6 Modelagem do Banco de Dados

Para o escopo da aplicação, optou-se por um banco de dados do tipo relacional, no qual as informações se organizam em tabelas e se relacionam por meio de chaves primárias, que identificam uma tupla de dados, e estrangeiras, que fazem referência a

uma chave pertencente a outra tabela. O banco de dados escolhido foi o PostgreSQL, sistema *open-source* já utilizado pela CODATA em suas aplicações.

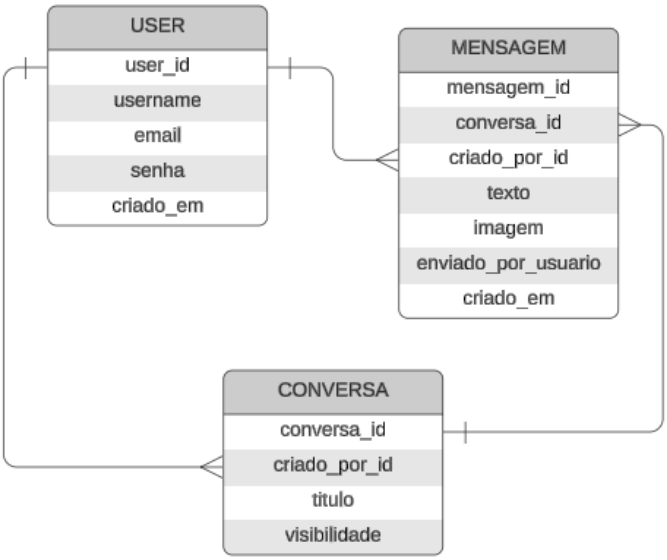

Figura 2: Diagrama Entidade-Relacionamento do sistema.

O esquema é composto por três tabelas: **Usuário**, **Conversa** e **Mensagem**. A tabela Usuário possui os atributos nome de usuário (*username*), e-mail, senha e data de criação. É referenciada nas tabelas Mensagem e Conversa, marcando quem criou a conversa e enviou a mensagem. Por sua vez, a tabela Conversa traz consigo os seguintes dados: o nome da conversa (título), quem criou e se ela é visível. A visibilidade é um dado do tipo booleano (verdadeiro ou falso) e serve para validar se uma conversa será exibida no *front-end* para o usuário ou não, uma vez que, para fins de auditoria, as conversas não serão excluídas do banco, mas serão omitidas da visualização e interação do usuário, podendo ser vistas somente pelos usuários gestores na página de administração, quando a visibilidade estiver configurada como privado (valor 0 para público e valor 1 para privado).

Por fim, a tabela Mensagem possui os seguintes atributos: chave estrangeira que faz referência à conversa daquela mensagem, quem a criou, quando criou, o texto, as imagens e se ela foi enviada pelo usuário ou pela IA. Se a coluna enviado_por_usuario estiver com valor 0, significa que aquela mensagem foi enviada pela IA. Se possuir valor 1, ela foi enviada pelo usuário. Essa diferenciação é necessária para fins de exibição no *front-end* e também por causa da auditoria. A Figura 2 apresenta o diagrama de entidade-relacionamento do sistema.

4.7 Narrativa do Processo

Tendo em mãos os principais requisitos, a arquitetura do sistema e o diagrama entidade-relacionamento, iniciou-se o processo de codificação daquilo que, primeiramente, seria uma prova de conceito, isto é, um teste de viabilidade de integração da API Chat Completions com a aplicação Django. Na Figura 3, é possível ver a primeira interface de usuário do projeto, desenvolvida totalmente em Django. Nessa versão, a interação do usuário com a API mostrou-se bem-sucedida, na qual foi possível ainda constatar a capacidade da API de descrever o conteúdo de imagens, com um nível de precisão bastante satisfatório.

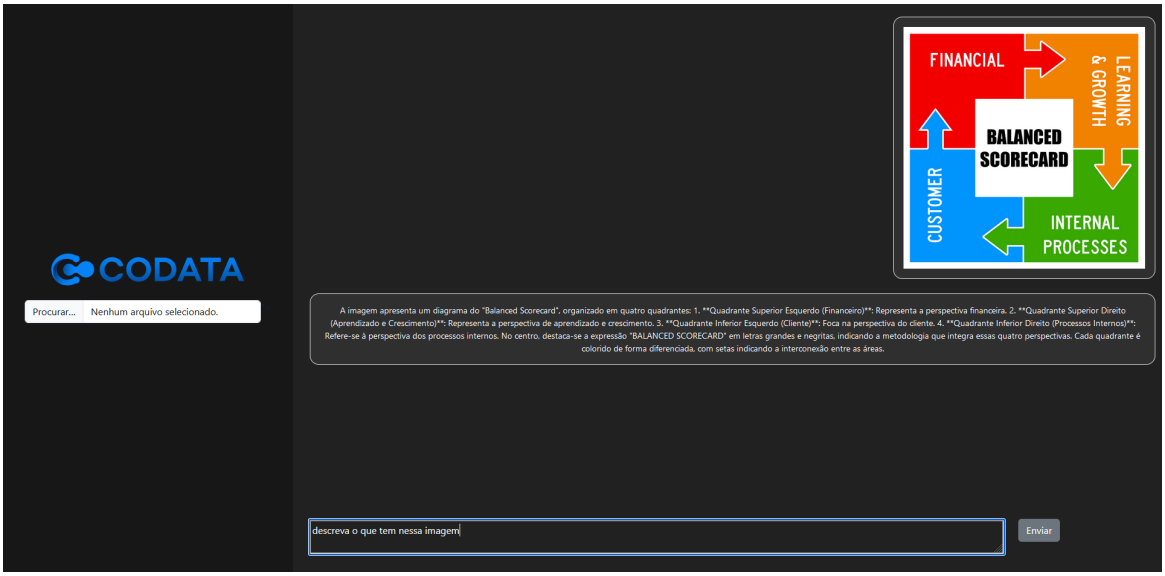

Figura 3: Primeira tela desenvolvida do projeto.

Apesar disso, a experiência de usuário estava aquém do esperado, e os refinamentos de design necessários para uma navegação fluida, intuitiva e prazerosa aumentariam a complexidade do código em Django, tornando o processo de desenvolvimento lento e trabalhoso em termos de manutenção. Foi nesse momento que optou-se por separar o *back-end* do *front-end*, no qual um segundo colaborador entrou no projeto, para codificar a interface do usuário em Vue.js. Com isso, este autor ficou responsável pela codificação do *back-end*, organização do repositório e *deploy* nos ambientes, exercendo os papéis de desenvolvedor e mantenedor do sistema.

Com o projeto tomando forma, foram criados os repositórios para versionamento de código dos serviços API e Web, seguindo os padrões de divisão de pastas para cada microsserviço vigentes na CODATA (Figura 4). Os códigos em Vue.js seriam comitados em Web e os do Django em API, com um desenvolvedor trabalhando em cada tecnologia.

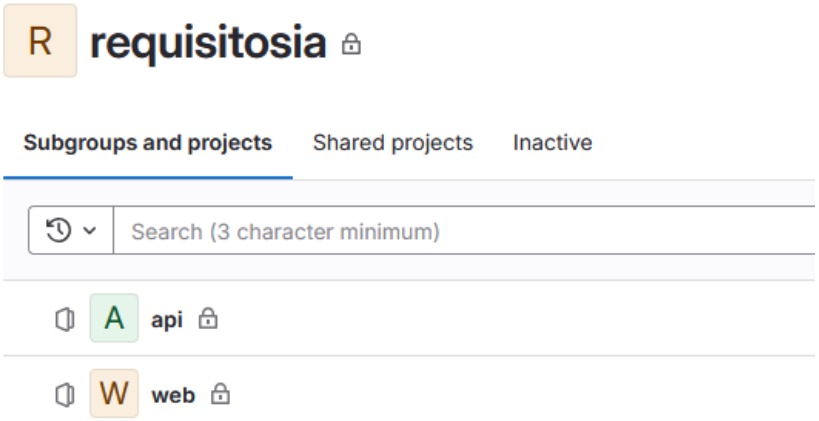

Figura 4: Organização das pastas do projeto no Gitlab.

O primeiro desafio de implementação foi o de integrar os serviços de API e Web aos arquivos *boilerplate* utilizados na empresa. *Boilerplates* são estruturas e arquivos de código reutilizáveis que servem de base para o desenvolvimento de um projeto. A maioria dos ambientes locais de desenvolvimento rodava em sistemas operacionais Linux, enquanto a máquina deste autor rodava um sistema Windows. Nesse sentido, a configuração dos arquivos Dockerfile e docker-compose.yml, necessários para rodar o projeto com *boilerplate* em contêineres do Docker, precisou ser ajustada.

Figura 5: Tela de Login do sistema.

O segundo contratempo residiu na integração com o sistema de login único (SSO). O *Keycloak* utiliza o protocolo *OpenID Connect* (OIDC) para autenticar os usuários e dar-lhes acesso às aplicações pertencentes a um *Realm*. O OIDC faz uso do *JSON Web Token* (JWT) para essa operação, que é gerado cada vez que o usuário requisita acesso a

uma página. Nesse contexto, o processo de login e geração de token ficou para o lado do cliente (*front-end*). A tela de login integrada ao sistema pode ser vista na Figura 5. O *front-end* também ficou responsável por chamar os dados da API no *back-end,* mas isso só foi possível quando as credenciais de login e os *tokens* passados ficaram iguais nos dois ambientes. A Figura 6 exibe a visualização de um dos *endpoints* da API e na Figura 7 é possível ver a primeira versão da interface de usuário desenvolvida em *Vue*.

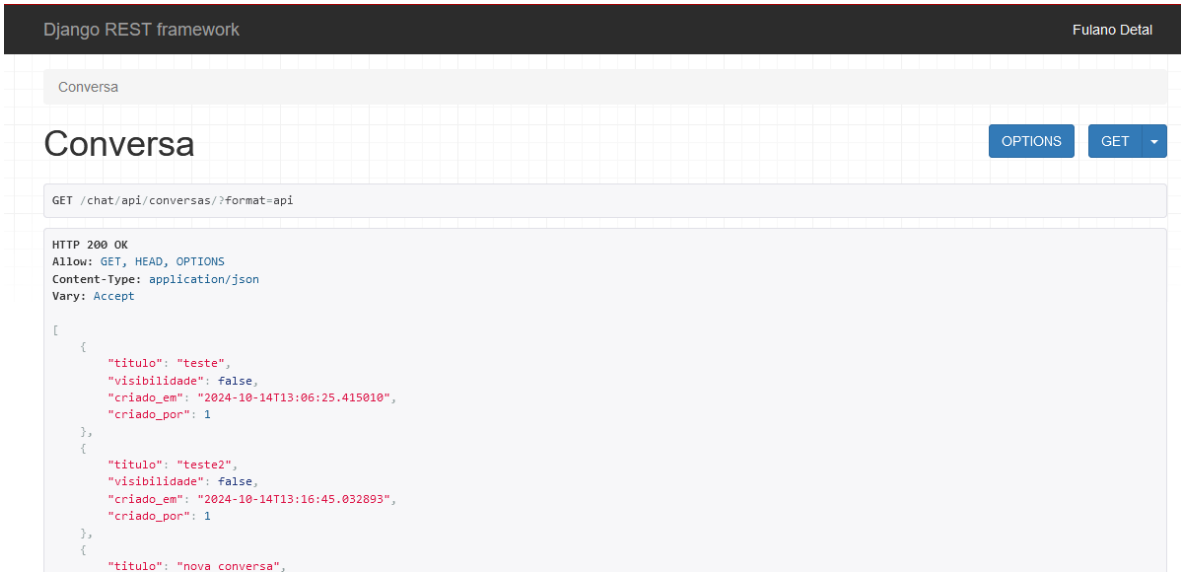

Figura 6: Exemplo de endpoint da API.

Nessa primeira versão, a API processava e respondia uma mensagem de cada vez, sem considerar o contexto da conversa. Era o comportamento padrão esperado em uma aplicação cliente-servidor baseada na comunicação requisição-resposta. Após estudar a documentação da OpenAI [24], viu-se que esse problema seria resolvido enviando todas as mensagens em ordem no corpo da requisição, em formato JSON. Cada mensagem nova seria salva no banco de dados, acrescentada no objeto JSON junto das mensagens já existentes, e então enviadas para a API.

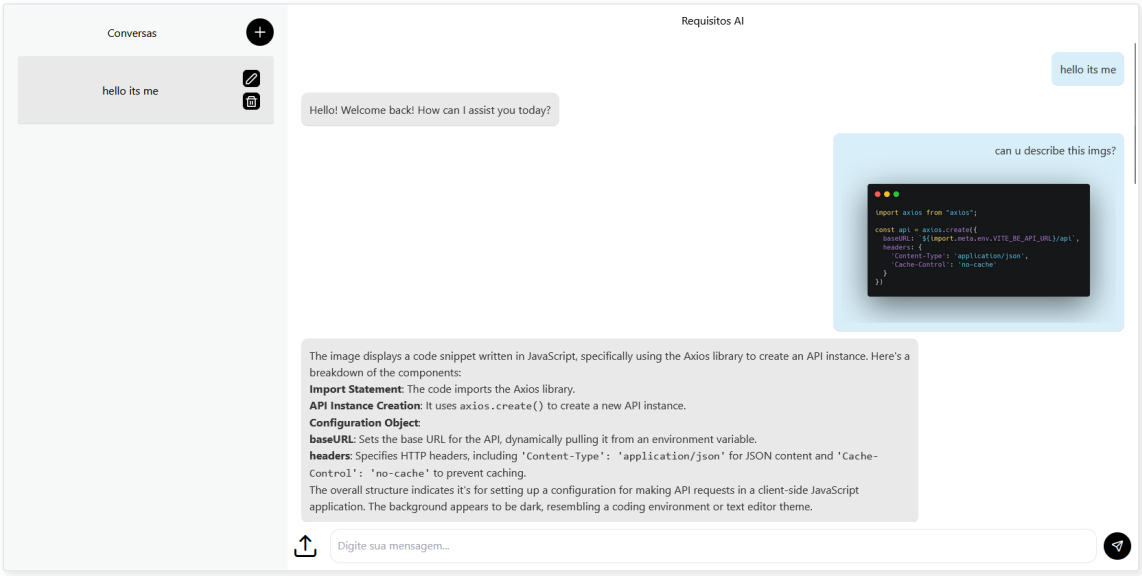

Figura 7: Primeira versão da tela principal do sistema em Vue.js.

A página de administração do Django (Figura 8) é uma funcionalidade já presente no framework, sem necessidade de codificação, e que já veio quase toda configurada no *boilerplate*, bastando alterar o nome do projeto e a versão. Além disso, foram incluídas as funcionalidades de auditoria por meio da biblioteca *django-easy-audit*, que permite monitorar todos os eventos de CRUD, login/logout e requisições feitas no sistema.

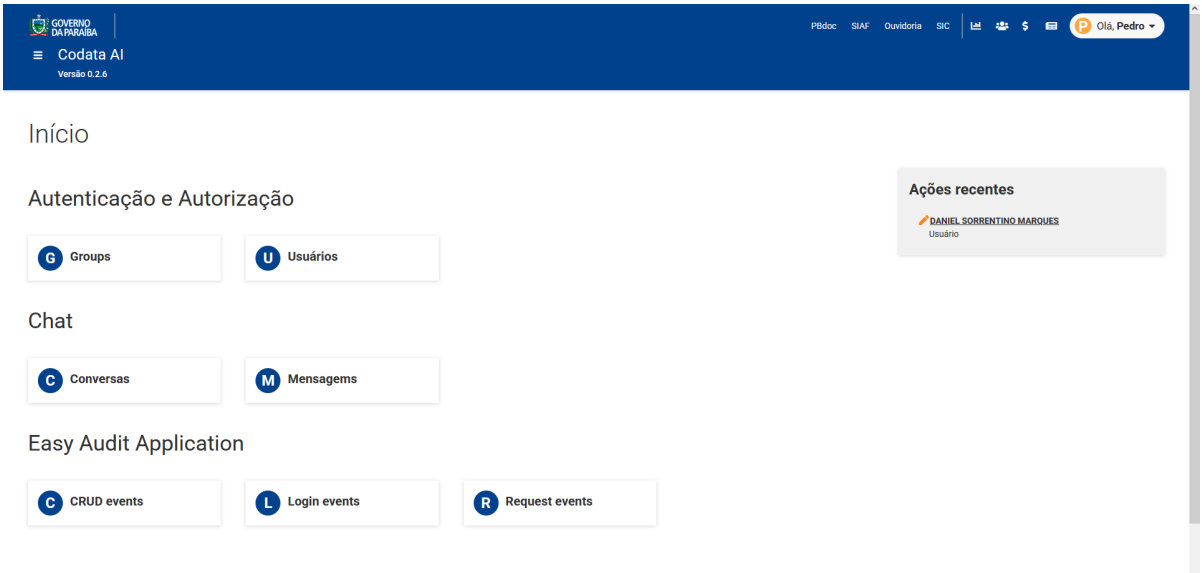

Figura 8: Página de administração do sistema.

Após o desenvolvimento da tela principal do sistema (Figura 6) e integração dela com os dados do *back-end*, a aplicação estava pronta para ser testada em um ambiente de homologação. Devido à natureza experimental do projeto e à indisponibilidade naquele momento de analistas de qualidade para testá-lo, os ambientes configurados pelo time de

DevOps ficaram restritos aos de Homologação e Produção. Normalmente, uma aplicação sob demanda desenvolvida na CODATA dispõe de quatro ambientes, orquestrados com o uso do Kubernetes: Desenvolvimento, QA, Homologação e Produção.

O processo de CI/CD do Gitlab é viabilizado pela configuração de arquivos do tipo .yml que vão instruir o que a plataforma vai fazer em seu *pipeline*. Junto a isso, tem-se à disposição uma infraestrutura em Kubernetes configurada pela CODATA, que consiste na organização dos containers em grupos chamados *Pods*, onde é possível acompanhar o processo de *deployment*, visualizar logs e detectar falhas. Ademais, devido ao sistema trabalhar com imagens, foi necessário criar um volume persistente para armazenar esses arquivos nos dois ambientes de *deploy*.

O disparo de *pipelines* se dá pela criação de *tags* dentro do repositório de cada projeto no Gitlab, condicionado à existência de um arquivo gitlab-ci.yml na pasta raiz. A CODATA utiliza o padrão de versionamento semântico MAJOR.MINOR.PATCH [25] para numerar as *tags*. O número da versão Major é incrementado quando o código tiver mudanças que o deixem incompatíveis com outras versões de serviços ou bibliotecas. O número Minor é alterado quando se adiciona uma nova funcionalidade ao sistema, sem gerar problemas de compatibilidade com outras versões. Por fim, o Patch marca o versionamento de melhorias, correções de bugs e outras alterações que não modifiquem ou adicionem funcionalidades ao sistema.

A última ação antes de subir para homologação foi pedir para os administradores de bancos de dados (DBAs) criarem as bases de dados que seriam utilizadas em homologação e produção. Por questões de segurança, esse processo não é feito pelos desenvolvedores. Os DBAs configuram cada ambiente para apontar para o seu respectivo banco. Outro ponto que diferencia um ambiente do outro são as variáveis de ambiente e seus valores. Na máquina local, é comum armazená-las em um arquivo .txt, mas isso não é uma opção segura e portável. No caso do CODATA AI, as variáveis de ambiente do Django ficaram armazenadas em um servidor denominado *config-manager*, o qual é referenciado pelos *Pods* do *back-end*. Por outro lado, as variáveis do *front-end* sobem junto ao resto do código presente no repositório, em um arquivo denominado *environment.js*.

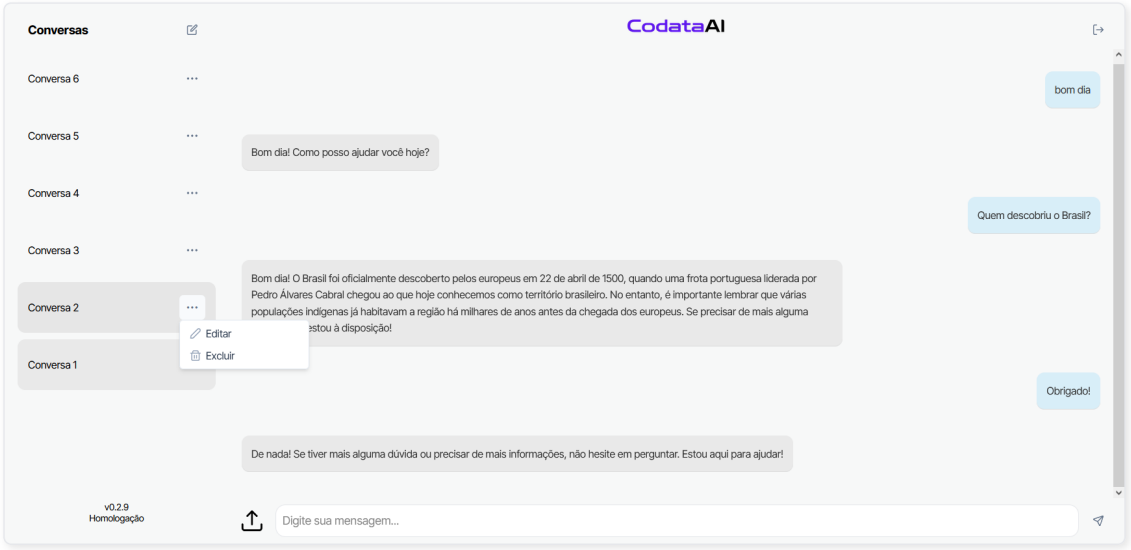

Figura 9: Versão definitiva da tela principal em ambiente de homologação.

Finda as etapas anteriores, a *tag* de versão 1.0.0 foi criada, e a aplicação foi disponibilizada em ambiente de homologação para uso e validação dos gerentes e colaboradores da gerência de desenvolvimento de sistemas (GEDES) da empresa. A Figura 9 exibe a versão definitiva da interface de usuário principal disponibilizada nessa versão.

4.8 Validação

Como já versado, devido ao escopo experimental do projeto e à indisponibilidade de analistas de qualidade para submeter o sistema a testes automatizados, a validação do CODATA AI ficou restrita a testes do tipo caixa-preta, os quais, segundo Pressman [26], põem em foco os requisitos funcionais do software, não levando em consideração aspectos lógicos e internos da implementação do sistema. Nesse tipo de teste, são fornecidas entradas e avaliadas as saídas geradas para verificar se estão em conformidade com os objetivos especificados [30].

Os casos de teste tentaram criar cenários que pudessem avaliar todos os requisitos funcionais, visando encontrar os seguintes erros: erros de interface de usuário, erros em estruturas de dados ou de acesso a dados, erros de comportamento, erros de inicialização e término. Inicialmente, o sistema em funcionamento foi apresentado para o gerente do setor, que também o mostrou para o diretor. Com o aceite dessas partes, o CODATA AI foi disponibilizado para os colaboradores da GEDES realizarem os testes de aceitação, os quais, à medida que utilizavam a aplicação, reportaram a presença de falhas e sugeriram melhorias para a equipe de desenvolvimento.

Os *feedbacks* foram colhidos pela equipe de desenvolvimento durante duas semanas,

em conversa direta com os colaboradores. Estima-se que o sistema tenha sido utilizado pela maioria dos funcionários nesse período, algo em torno de 30 a 60 pessoas. Os casos de teste foram devidamente escritos e encontram-se listados a seguir:

Título	[T01] Login com SSO
Descrição	Validar se o sistema permite o login bem-sucedido de um usuário utilizando o Single Sign-On (SSO) [RF001].
Passos do teste	1. Acesse o sistema. 2. Insira as credenciais do SSO (CPF e senha). 3. Clique no botão "Entrar"ou aperte a tecla "Enter". 4. Aguarde o redirecionamento para a página inicial do sistema.
Resultado esperado	O usuário é autenticado com sucesso e redirecionado para a página inicial do sistema.
Resultado obtido	Os usuários informaram que o login ocorreu de forma rápida e sem dificuldades.

Tabela 1: Caso de teste [T01].

Título	[T02] Listar Conversas
Descrição	Validar se o sistema exibe um menu lateral com a listagem das conversas do usuário logado [RF002].
Passos do teste	1. Acesse a página inicial do sistema. 2. Após o login, observe a interface principal do sistema. 3. Localize o menu lateral na interface. 4. Verifique se o menu lateral contém a listagem das conversas do usuário logado.
Resultado esperado	O menu lateral é exibido corretamente, contendo a listagem das conversas do usuário logado.
Resultado obtido	Os usuários informaram que foi possível visualizar a listagem de conversas.

Tabela 2: Caso de teste [T02].

Título	[T03] Listar Mensagens
Descrição	Validar se o sistema exibe uma área de visualização com a listagem das mensagens enviadas pelo usuário e das respostas geradas pela API [RF003].
Passos do teste	1. Acesse o sistema com um usuário válido. 2. Após o login, selecione uma conversa na listagem de conversas. 3. Observe a área de visualização de mensagens na interface. 4. Verifique se a área exibe corretamente as mensagens enviadas pelo usuário e as respostas geradas pela API.
Resultado esperado	A área de visualização exibe corretamente a listagem das mensagens enviadas pelo usuário e as respostas geradas pela API.
Resultado obtido	Os usuários informaram que as mensagens foram exibidas corretamente para cada conversa, tanto as deles quanto as da API.

Tabela 3: Caso de teste [T03].

Título	[T04] Anexar Imagens
Descrição	Validar se o sistema permite ao usuário anexar imagens por meio de um botão ou espaço para arrastar arquivos no corpo da mensagem [RF004].
Passos do teste	1. Acesse o sistema com um usuário válido. 2. Selecione uma conversa na listagem de conversas. 3. Localize o botão ou espaço para anexar imagens na área de envio de mensagens. 4. Clique no botão ou arraste uma imagem para o espaço destinado. 5. Verifique se a imagem é anexada corretamente antes do envio.
Resultado esperado	O sistema permite anexar a imagem selecionada ou arrastada, exibindo-a no corpo da mensagem antes do envio.
Resultado obtido	Os usuários informaram que conseguiram anexar as imagens por meio do botão, mas relataram fracasso ao colar a imagem na área de upload.

Tabela 4: Caso de teste [T04].

Título	[T05] Acessar Página de Administração
Descrição	Validar se o sistema implementa uma página de administração que permite aos gestores visualizar as atividades dos usuários e as trocas de mensagens com a API [RF005].
Passos do teste	1. Acesse o sistema como um gestor com permissões administrativas. 2. Navegue até a página de administração no menu principal. 3. Verifique se a página exibe as atividades dos usuários, incluindo logins e interações. 4. Verifique se a página exibe o histórico de trocas de mensagens entre os usuários e a API.
Resultado esperado	A página de administração é exibida corretamente, permitindo a visualização das atividades dos usuários e das trocas de mensagens com a API.
Resultado obtido	Os gerentes informaram que conseguiram acessar a página de administração e visualizar todas as informações requeridas.

Tabela 5: Caso de teste [T05].

Título	[T06] Armazenar Mensagens
Descrição	Validar se o sistema salva todas as mensagens (textos e imagens) geradas no banco de dados para fins de auditoria [RF006].
Passos do teste	1. Acesse o sistema com um usuário válido. 2. Envie uma mensagem de texto em uma conversa. 3. Anexe uma imagem e envie-a na mesma conversa. 4. Acesse o banco de dados e verifique se as mensagens (texto e imagem) foram salvas corretamente.
Resultado esperado	As mensagens de texto e imagem são salvas corretamente no banco de dados para fins de auditoria.
Resultado obtido	Os usuários, ao acessarem novamente a aplicação, confirmaram a persistência de todos os dados, tanto textuais quanto digitais.

Tabela 6: Caso de teste [T06].

Título	[T07] Registrar Atividades de Usuário
Descrição	Validar se o sistema salva as atividades do usuário (login, criação, edição e exclusão de conversas, e envio de mensagens) para fins de auditoria [RF007].
Passos do teste	1. Acesse o sistema com um usuário válido. 2. Realize o login e verifique se a atividade de login é registrada no banco de dados. 3. Crie uma nova conversa e verifique se a criação é registrada. 4. Edite a conversa e verifique se a edição é registrada. 5. Exclua a conversa e verifique se a exclusão é registrada. 6. Envie uma mensagem e verifique se o envio é registrado. 7. Acesse o banco de dados e verifique se todas as atividades foram salvas corretamente.
Resultado esperado	O sistema registra corretamente todas as atividades do usuário no banco de dados para fins de auditoria.
Resultado obtido	Os gerentes informaram que foi possivel visualizar as atividades de todos os usuários do sistema de forma completa dentro da página de administração.

Tabela 7: Caso de teste [T07].

Título	[T08] Exportar Conversas
Descrição	Validar se o sistema é capaz de gerar um arquivo no formato JSON com todas as mensagens armazenadas no banco de dados para fins de portabilidade [RF008].
Passos do teste	1. Acesse a página de administração. 2. Acesse a listagem de conversas clicando no botão de mesmo nome. 3. Selecione as conversas que deseja exportar. 4. Selecione a opção “exportar conversas e mensagens em JSON”. 5. Verifique se o arquivo foi baixado e está no formato JSON.
Resultado esperado	O sistema gera corretamente um arquivo JSON contendo as mensagens de todas as conversas selecionadas.
Resultado obtido	Os gerentes informaram que conseguiram gerar e baixar o arquivo JSON com todas as conversas e suas respectivas mensagens.

Tabela 8: Caso de teste [T08].

Em geral, os testes não acusaram falhas graves no sistema que impedissem o seu funcionamento. A experiência dos usuários durante o período de validação, na maioria dos casos, foi tranquila, com esses não relatando grandes dificuldades em acessar ou interagir com o CODATA AI. A maioria dos *feedbacks* passados à equipe de desenvolvimento versou

a respeito de melhorias de design e interface. Apesar disso, viu-se necessário refatorar o upload das imagens para que ele seja feito de mais de uma maneira. Além disso, para outras versões, deseja-se melhorar a visualização de carregamento das mensagens da API, para deixá-la mais vívida em analogia a alguém do outro lado digitando. Esses pequenos ajustes visam a melhoria da experiência de usuário, um dos pilares do projeto.

4.9 Demonstração

Com o sistema funcional, tornou-se possível reproduzir uma das demandas que motivaram a sua concepção: a geração automatizada de histórias de usuário no padrão BDD para alimentar um documento de requisitos fictício. Ao logar no sistema e criar uma nova conversa, iniciamos a interação com o modelo. A partir de sua resposta, fornecemos um *prompt* de texto junto com uma imagem da página inicial do Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA) da UFPB (Figura 10), no qual foi pedido ao modelo histórias com base nas funcionalidades presentes na imagem fornecida.

Figura 10: Imagem da página inicial do SIGAA fornecida ao modelo.

Na Figura 11, é possível ver o comando dado ao modelo. Por sua vez, a Figura 12 exibe as histórias de usuário geradas por ele, nas quais pode-se constatar a associação delas com a imagem disponibilizada. A experiência completa foi gravada e disponibilizada no endereço *https://youtu.be/ieGGbotP3uA*. A partir desse contexto, um analista de requisitos pode rapidamente copiar essas histórias de usuário e colocá-las em um documento, dispensando o trabalho manual e oneroso de pensar nos cenários e escrevê-los um a um no padrão desejado.

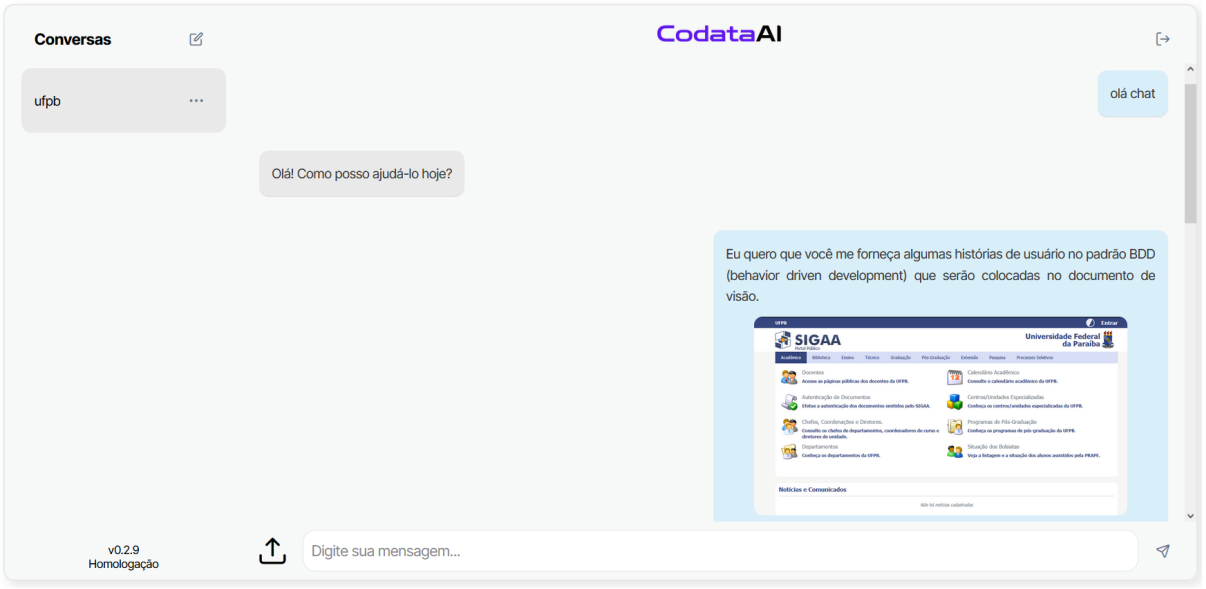

Figura 11: Prompt de entrada dado ao modelo.

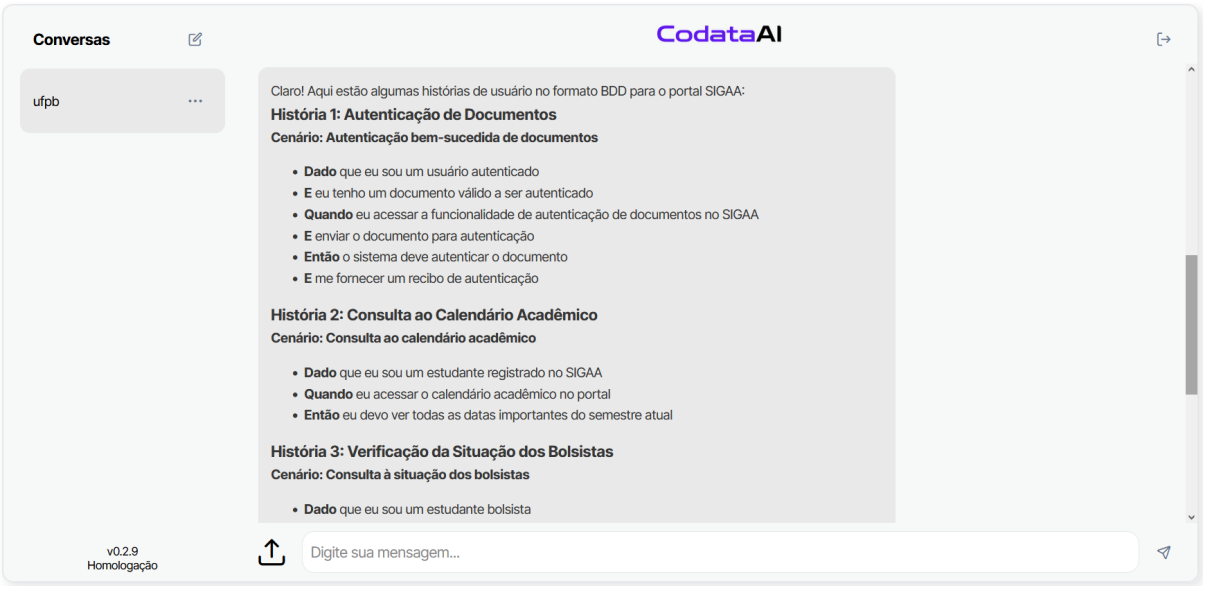

Figura 12: Histórias de usuário geradas na resposta.

4.10 Análise e Discussão dos Resultados

Aquilo que a princípio era um projeto experimental se tornou uma aplicação completa e pronta para uso. Apesar do projeto demandar testes complementares que o validem para o ambiente de produção, como os do tipo caixa-branca, os casos em que o sistema foi submetido nos testes de caixa-preta ofereceram um *feedback* — pelos funcionários e pela gerência da GEDES — satisfatório de suas funcionalidades, alinhados aos requisitos funcionais especificados. Inicialmente, o CODATA AI destinava-se a atender às demandas de um público pequeno e definido dentro da CODATA: os analistas de requisitos. Todavia, a

ratificação da viabilidade do projeto trouxe consigo diferentes possibilidades de expansão do seu escopo, atraindo o interesse da gerência.

Nesse contexto, o projeto foi feito de maneira a não escalar a complexidade em termos de codificação, manutenção e experiência de usuário. Tudo foi pensado para ser simples e escalável. Apesar da fragmentação do sistema em microsserviços (*back-end* e *front-end*) ter implicado no aumento do custo de desenvolvimento e gerenciamento, os ganhos de produtividade, desempenho e usabilidade contrabalançaram esse impacto. O CODATA AI procurou se apresentar ao mundo como sendo uma plataforma concebida dentro de uma arquitetura e filosofia de projeto robusta e atual. As tecnologias adotadas procuraram mitigar os impactos da rotatividade de desenvolvedores em projetos, por serem populares no mercado de trabalho e possuírem uma curva de aprendizado relativamente pequena, em conformidade com as ferramentas utilizadas na empresa.

O desenvolvimento do *back-end* em Django trouxe flexibilidade e entregou um sistema completo de fácil acesso e gerenciamento de API, sem abrir mão da robustez e da segurança. O código é de fácil manutenção, customização e expansão. Por outro lado, o Vue.js, fazendo uso de suas bibliotecas de gerenciamento de estado, propiciou uma experiência de usuário baseada em uma navegação simples de tela única, sem a necessidade de recarregamentos ou redirecionamentos e sem fechar os elos com o Django REST. Além disso, a hospedagem do sistema na infraestrutura da CODATA garantiu sua disponibilidade em 99,9% do tempo desta experiência.

A maior expectativa levantada durante o processo de desenvolvimento do sistema foi a capacidade do modelo de processar e interpretar o conteúdo de imagens, visto que essa funcionalidade era um recurso pago na maioria dos serviços que existiam na *internet* naquele momento. A equipe se surpreendeu com o nível de precisão do modelo em descrever as imagens. Isso se aliou ao processo de treinamento da IA, por parte dos gerentes, que possibilitou a geração de respostas condizentes ao trabalho dos analistas de requisitos, os quais esperavam saídas dentro dos padrões técnicos e metodológicos utilizados por eles, como o *Behavior-Driven Development*. A aplicação se mostrou pronta o suficiente para atender às demandas desse público, automatizando o processo de escrita de histórias de usuário e dos documentos de visão dos projetos.

A página de administração deu ao projeto mais uma camada de segurança, uma vez que tornou possível o acompanhamento e a auditoria de tudo o que aconteceu dentro da plataforma, dando garantias não só à gerência, como também aos usuários colaboradores. Tudo passou a ser registrado e persistido no banco de dados, que está sob custódia da CODATA e sua infraestrutura. Por ser uma aplicação de *chat* simples, não houve restrição quanto ao seu uso por outros tipos de colaboradores. Dessa forma, o CODATA AI estabeleceu-se como uma alternativa interna, segura e gerenciável que pode ser demonstrada e comercializada pela CODATA dentro do seu rol de clientes.

Por fim, a documentação desse processo não só contribuiu para a ampliação da literatura na comunidade científica, por ser um material baseado na *Design Science Research*, como também trouxe expertise para os desenvolvedores envolvidos diretamente no projeto. A adoção de uma API baseada em IA e o desenvolvimento de uma aplicação em formato de *chat* foram experiências inéditas na companhia, trazendo desafios e aprendizados. Desenvolver é muito mais do que codificar; é dominar a arte de resolver problemas reais que impactem na vida das pessoas. Ao final de tudo, a CODATA ganhou mais uma equipe de desenvolvedores capacitada para expandir projetos futuros na área de inteligência artificial, que podem ser convertidos em receita e vantagem competitiva para a empresa, contribuindo para a digitalização dos serviços a serem prestados para o estado da Paraíba e seus cidadãos.

5 CONCLUSÕES E TRABALHOS FUTUROS

O projeto CODATA AI surgiu para colocar em xeque os pontos altos e as limitações da API *Chat Completions*. Antes da incorporação em um sistema, o controle do serviço estava nas mãos dos gerentes, que tinham disponível apenas a página fornecida pela OpenAI para reger os parâmetros da API (modelo, grau de precisão, número máximo de *tokens* a serem processados, entre outros). O projeto se tornou viável e foi implementado. Embora a OpenAI ofereça suporte para o consumo de suas APIs utilizando a linguagem de programação *Python*, na qual o Django e o Django REST foram escritos, a documentação deixa a desejar em exemplos concretos e integrados a algum tipo de projeto ou tecnologia populares. A sua assimilação pode ser difícil para desenvolvedores iniciantes.

O sistema da OpenAI é pago pela CODATA e a precificação se dá pelo número de tokens processados pelo modelo. Dessa forma, quanto mais pessoas usam, mais caras são as despesas com o serviço, o que implica a possibilidade de alteração futura para alternativas *open-source*, como o Ollama [27]. A arquitetura do CODATA AI foi pensada para mitigar os impactos dessa portabilidade, de modo que as intervenções sejam feitas apenas no *back-end*. Apesar disso, urge definir uma regra de negócio que torne a comercialização do projeto viável e vantajosa para a empresa, seja com ferramentas pagas, seja com ferramentas gratuitas.

O ciclo de testes poderia ser mais abrangente, aumentando a cobertura das validações e mitigando a presença de falhas, que só foram detectadas em ambiente de homologação. Isso pouparia o retrabalho e ofereceria uma visão mais holística da corretude do sistema para os desenvolvedores. A qualidade do software independe de ele ser feito para um cliente ou para uso interno, devendo seguir os mesmos padrões de excelência e boas práticas de desenvolvimento.

Ainda que a aplicação esteja funcional e pronta para rodar em ambiente de produção, há de se estudar meios de otimizar e dinamizar as requisições, saindo da rigidez do modelo cliente-servidor. Também é necessário estudar meios de mitigar a geração de respostas alucinadas pelo modelo. A experiência do usuário é outro ponto que pode ser aprimorado, com um design mais convidativo e adição de mais funcionalidades. Por fim, em relação a trabalhos futuros, espera-se que o CODATA AI seja a base para sistemas maiores, distribuídos e multiusuário, devidamente otimizados e tolerantes a falhas. O governo digital é uma realidade, e espera-se que o uso da inteligência artificial catalise esse processo.

REFERÊNCIAS

[1] SOARES, Margarida. Impacto do Chat GPT na sociedade. The Trends Hub, n. 3, 2023.

[2] MCCALLUM, Shiona. ChatGPT banido na Itália por questões de privacidade. BBC. 2023. Disponível em: https://www.bbc.com/news/technology-65139406. Acesso em: 19 jan. 2025.

[3] BRASIL. Lei nº 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados Pessoais (LGPD). Diário Oficial da União, Brasília, DF, 15 ago. 2018. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: 19 jan. 2025.

[4] VALENTE, Marco Tulio. Engenharia de software moderna. Princípios e Práticas para Desenvolvimento de Software com Produtividade, v. 1, n. 24, cap 3. 2020.

[5] SOMMERVILLE, Ian. Engenharia de Software, 10ª Edição. Editora Pearson, 2019.

[6] SMART, John Ferguson; MOLAK, Jan. BDD in Action: Behavior-driven development for the whole software lifecycle. Simon and Schuster, 2023.

[7] TEXAS STATE UNIVERSITY. Plagiarism, AI and ChatGPT: ChatGPT: How does work?. Disponível em: https://guides.library.txstate.edu/c.php?g=1321038&p=9718369. Acesso em: 21 jan. 2025.

[8] RODRIGUES, Diogo Duarte. Design Science Research como caminho metodológico para disciplinas e projetos de Design da Informação— Design Science Research as methodological path for Information Design subjects and projects. InfoDesign-Revista Brasileira de Design da Informação, v. 15, n. 1, p. 111-124, 2018.

[9] AMAZON WEB SERVICES. O que é uma API (interface de programação de aplicações)?. Disponível em: https://aws.amazon.com/pt/what-is/api/. Acesso em: 21 jan. 2025.

[10] ORACLE. O que é um Banco de Dados?. Disponível em: https://www.oracle.com/br/database/what-is-database/. Acesso em: 21 jan. 2025.

[11] TOTVS. Front end: O que é, como funciona e qual a importância. Disponível em: https://www.totvs.com/blog/developers/front-end/. Acesso em: 21 jan. 2025.

[12] ALURA. Back-End: o que é e um guia para iniciar na área. Disponível em: https://www.alura.com.br/artigos/backend?srsltid=AfmBOorPiyQWOR4pSt1q4o-ZXhXQfYLENbzs-Q5dkwiokQWPvmLYJ4o8. Acesso em: 22 jan. 2025.

[13] SCAPICCHIO, Mark. FORREST, Amber. O que é SSO?. IBM. Disponível em: https://www.ibm.com/br-pt/topics/single-sign-on. Acesso em: 22 jan. 2025.

[14] MICROSOFT. O que é controle de versão?. Disponível em: https://learn.microsoft.com/pt-br/devops/develop/git/what-is-version-control. Acesso em: 22 jan. 2025.

[15] OPENAI. Tokenizer. Disponível em: https://platform.openai.com/tokenizer. Acesso em: 22 jan. 2025.

[16] BELCIC, Ivan. STRYKER, Cole. O que é GPT (transformador generativo pré-treinado)?. IBM. Disponível em: https://www.ibm.com/br-pt/think/topics/gpt. Acesso em: 22 jan. 2025.

[17] DJANGO REST FRAMEWORK. Disponível em: https://www.django-rest-framework.org/. Acesso em: 22 jan. 2025.

[18] VUE. Introduction. Disponível em: https://vuejs.org/guide/introduction.html. Acesso em: 22 jan. 2025.

[19] OPENAI. Models. Disponível em: https://platform.openai.com/docs/models. Acesso em: 23 jan. 2025.

[20] KEYCLOAK. Disponível em: https://www.keycloak.org/. Acesso em: 23 jan. 2025.

[21] NARANPANAWA, Rashmini. OAuth 1.0 Vs OAuth 2.0. Medium. Disponível em: https://medium.com/identity-beyond-borders/oauth-1-0-vs-oauth-2-0-e36f8924a835. Acesso em: 23 jan. 2025.

[22] PIMENTEL, Mariano; FILIPPO, Denise; DOS SANTOS, Thiago Marcondes. Design Science Research: pesquisa científica atrelada ao design de artefatos. RE@ D-Revista de Educação a Distância e eLearning, v. 3, n. 1, p. 37-61, 2020.

[23] LACERDA, Daniel Pacheco et al. Design Science Research: método de pesquisa para a engenharia de produção. Gestão & produção, v. 20, p. 741-761, 2013.

[24] OPENAI API DOCS. API Reference. Disponível em: https://platform.openai.com/docs/api-reference/chat. Acesso em: 26 jan. 2025.

[25] SEMVER. Versionamento Semântico 2.0.0. Disponível em: https://semver.org/lang/pt-BR/. Acesso em: 26 jan. 2025.

[26] PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de software-9. McGraw Hill Brasil, 2021.

[27] ARIFFUD, Muhammad. What is Ollama? Understanding how it works, main features and models What is Ollama? Understanding how it works, main features and models. Hostinger. Disponível em: https://www.hostinger.com/tutorials/what-is-ollama. Acesso em 27 jan. 2025.

[28] COSTA, Holliver de Oliveira. Microsserviços e orquestração de contêineres: uma abordagem prática. 2023.

[29] SCHMITT, Jacob. Git tags vs branches: Differences and when to use them. Circle CI. Disponível em: https://circleci.com/blog/git-tags-vs-branches/. Acesso em 13 fev. 2025.

[30] DELAMARO, Marcio; JINO, Mario; MALDONADO, Jose. Introdução ao teste de software. Elsevier Brasil, 2013.

[31] LV, Zhihan. Generative artificial intelligence in the metaverse era. Cognitive Robotics, v. 3, p. 208-217, 2023.

[32] RAMOS, Anatália Saraiva Martins. Inteligência Artificial Generativa baseada em grandes modelos de linguagem-ferramentas de uso na pesquisa acadêmica. 2023.