UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

BRUNO AUGUSTO COTA SILVA
Orientador: Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti

USO DE GRANDES MODELOS DE LINGUAGEM EM APLICAÇÕES: APLICAÇÃO WEB PARA AUXILIAR PROFISSIONAIS DE QUALIDADE DE SOFTWARE

Ouro Preto, MG
2023

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

BRUNO AUGUSTO COTA SILVA

USO DE GRANDES MODELOS DE LINGUAGEM EM APLICAÇÕES: APLICAÇÃO WEB PARA AUXILIAR PROFISSIONAIS DE QUALIDADE DE SOFTWARE

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti

Ouro Preto, MG
2023

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Bruno Augusto Cota Silva

Uso de grandes modelos de linguagem em aplicações: Aplicação web para auxiliar profissionais de qualidade de software

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 27 de Setembro de 2023.

Membros da banca

Carlos Frederico M. da Cunha Cavalcanti (Orientador) - Doutor - Universidade Federal de Ouro Preto
Vinicius Antonio de Oliveira Martins (Examinador) - Mestre - Universidade Federal de Ouro Preto
Fernando Cortez Sica (Examinador) - Doutor - Universidade Federal de Ouro Preto

Carlos Frederico M. da Cunha Cavalcanti, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 27/09/2023.

Documento assinado eletronicamente por **Carlos Frederico Marcelo da Cunha Cavalcanti**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 06/12/2023, às 10:14, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0587194** e o código CRC **4794D6D6**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Resumo

Este trabalho descreve o desenvolvimento e a implementação do aplicativo QAFlow, que utiliza um modelo de IA nomeado text-davinci-003 para automatizar e aprimorar o processo de geração de histórias de usuário, cenários de teste e códigos de automação no contexto de desenvolvimento de software. A metodologia envolveu a definição clara dos requisitos e escopo do projeto, a seleção criteriosa da API do modelo de inteligência artificial, o design de uma interface de usuário intuitiva e a implementação de um backend eficiente para gerenciar a comunicação com a API. A integração bem-sucedida da API permitiu a formatação adequada das entradas dos usuários e a interpretação das respostas geradas, resultando em histórias de usuário, cenários de teste e códigos de automação coerentes e relevantes. O QAFlow traz benefícios significativos para profissionais de qualidade de software, proporcionando maior eficiência no processo de criação de artefatos de teste e melhorando a colaboração entre equipes de desenvolvimento e qualidade. Além disso, contribui para a padronização e consistência na geração de artefatos, facilitando a comunicação e reduzindo retrabalhos.

Palavras-chave: Aplicativo QAFlow. Modelo text-davinci-003. Desenvolvimento de Software. Processamento linguagem natural. Geração automática. Historias de Usuário. Cenários de Teste. Automação. Produtividade

Abstract

This work describes the development and implementation of the QAFlow application, which uses an AI model named text-davinci-003 to automate and improve the process of generating user stories, test scenarios and automation codes in the context of software development. . The methodology involved clearly defining the requirements and scope of the project, carefully selecting the AI model API, designing an intuitive user interface, and implementing an efficient backend to manage communication with the API. Successful API integration allowed for proper formatting of user input and interpretation of generated responses, resulting in coherent and relevant user stories, test scenarios, and automation code. QAFlow brings significant benefits to software quality professionals, providing greater efficiency in the test artifact creation process and improving collaboration between development and quality teams. Furthermore, it contributes to standardization and consistency in the generation of artifacts, facilitating communication and reducing rework.

Keywords: QAFlow Application. Text-Davinci-003 Model. Software Development. Natural Language Processing. Automatic Generation. User Stories. Test Scenarios. Automation. Productivity.

Sumário

1 Introdução

Nos últimos anos, a qualidade de software tem se tornado cada vez mais crucial para o sucesso e a satisfação dos usuários em diversos setores da indústria. A demanda por sistemas de software confiáveis, eficientes e livres de defeitos tem impulsionado a busca por práticas e técnicas de garantia de qualidade robustas e eficazes (SOMMERVILLE et al., 2011). Nesse contexto, surge a necessidade de ferramentas e soluções inovadoras que auxiliem os profissionais de qualidade de software em suas atividades.

Com o avanço da tecnologia e a crescente demanda por sistemas mais inteligentes e eficientes, a utilização de grandes modelos de linguagem, chamados de *Large Language Models* (LLM), se tornou uma abordagem promissora (TAMKIN et al., 2021). Esses modelos são capazes de compreender e gerar texto de forma natural, tornando-os ideais para tarefas como geração automática de histórias de usuários, cenários de teste e até mesmo códigos de automação.

Neste trabalho, apresentamos a proposta de uma aplicação web inovadora, chamada "QAFlow"(nome da aplicação não tem relação com empresa de mesmo nome, encontrada no mercado), que tem como objetivo facilitar o fluxo de trabalho de um profissional da área de qualidade de software. A aplicação faz uma chamada para um modelo de inteligência artificial pré-treinado, permitindo que o usuário descreva brevemente uma funcionalidade desejada e obtenha automaticamente uma história de usuário completa, cenários de teste associados e *scripts* para automação de testes utilizando *Cypress*.

Com a utilização do "QAFlow", profissionais de desenvolvimento de software poderão economizar tempo e esforço na criação desses artefatos, uma vez que a aplicação automatiza um processo demorado e enfadonho. Além disso, a precisão e qualidade das histórias e cenários gerados pelo modelo de linguagem natural contribuem para uma melhor compreensão das funcionalidades a serem desenvolvidas, facilitando a comunicação entre a equipe de desenvolvimento e os *stakeholders* do projeto.

Ao longo do trabalho, serão apresentados os detalhes da arquitetura da aplicação, bem como as etapas de integração com o modelo text-davinci-003. Além disso serão levantados os benefícios e desafios associados à utilização dessa abordagem. A aplicação "QAFlow"busca contribuir significativamente para o processo de desenvolvimento de software, agilizando a criação de histórias de usuário, cenários de teste e automação de forma precisa e eficiente.

1.1 Justificativa

A justificativa para a criação do aplicativo "QAFlow"baseia-se na necessidade de melhorar e agilizar o processo de qualidade de software, especificamente no que se refere à criação

de cenários de teste, histórias de usuário e códigos para automação. As justificativas para o desenvolvimento desse aplicativo são as seguintes:

1. Eficiência no processo de criação: O processo de criar cenários de teste e histórias de usuário pode ser demorado e complexo, exigindo muita documentação e organização. O QAFlow visa simplificar esse processo, fornecendo uma interface intuitiva e ferramentas adequadas para criar e gerenciar essas informações de forma eficiente.

2. Padronização e consistência: A criação de cenários de teste e histórias de usuário muitas vezes envolve a adesão a padrões e formatos específicos. O "QAFlow"visa seguir o formato *Behavior Driven Development(BDD)* oferecendo padrões e boas práticas, assim garantindo a consistência e a conformidade nos artefatos produzidos.

3. Colaboração e comunicação aprimoradas: O "QAFlow"promove a colaboração entre os membros da equipe de desenvolvimento e de qualidade, permitindo que eles trabalhem de forma conjunta na criação e revisão dos cenários de teste e histórias de usuário. Além disso, o aplicativo facilita a comunicação e o compartilhamento de informações, evitando retrabalhos e garantindo uma compreensão clara dos requisitos de software.

4. Melhoria da qualidade do software: Ao simplificar e aprimorar a velocidade do processo de criação de cenários de teste e histórias de usuário, o "QAFlow"contribui para a melhoria da qualidade do software desenvolvido, pois o testador passa a se preocupara apenas em revisar os conteúdos criados. Com artefatos bem definidos, padronizados e revisados, é possível identificar e corrigir problemas mais cedo no ciclo de desenvolvimento, resultando em produtos finais mais confiáveis e de maior qualidade.

Portanto, o desenvolvimento do aplicativo "QAFlow"se justifica pela necessidade de otimizar o processo de criação de cenários de teste, histórias de usuário e códigos automatizados, promovendo eficiência, padronização, colaboração e consequentemente melhorando a qualidade do software desenvolvido pela equipe.

1.2 Objetivos

Os objetivos levam a criação de uma aplicação web que automatize e aprimore o processo de geração de histórias de usuário, cenários de teste e *scripts* de teste, trazendo benefícios significativos para profissionais de qualidade de software e contribuindo para a melhoria da qualidade dos sistemas de software desenvolvidos. Os objetivos específicos são:

1. Desenvolver um aplicativo web denominado "QAFlow"que faça requisições para um modelo de Linguagem de Aprendizado de Máquina (LLM).

2. Permitir que os usuários do"QAFlow"descrevam características de uma funcionalidade enviem para aplicação.

3. Utilizar o texto fornecido pelo usuário como parte de uma requisição feita para um modelo de linguagem natural, que irá gerar automaticamente uma história de usuário, cenário de teste ou código automatizado.

4. Facilitar a geração de casos de teste abrangentes e detalhados, fornecendo uma cobertura adequada para as funcionalidades da aplicação.

5. Melhorar a qualidade e eficiência do processo de teste de software, permitindo que os profissionais de qualidade de software gerem rapidamente histórias de usuário, cenários de teste e códigos de automação de maneira precisa e padronizada.

6. Contribuir para o avanço da área de teste de software, explorando a integração de modelos de LLM em aplicativos e a automação de outros tipos de atividades de teste.

7. Documentar e relatar o processo de desenvolvimento do "QAFlow", fornecendo uma base sólida para futuros trabalhos e estudos relacionados ao tema.

1.3 Organização do Trabalho

Os capítulos restantes deste documento estão organizados da seguinte forma. No Capítulo 2, é apresentada uma revisão abrangente da literatura e do referencial teórico, incluindo a exploração de conceitos teóricos relevantes e trabalhos relacionados no campo. O Capítulo 3 detalha a metodologia e o processo de desenvolvimento, fornecendo *insights* sobre os materiais, métodos e etapas realizadas para concretizar a aplicação "QAFlow". No Capítulo 4, são detalhados os resultados e discussões decorrentes da implementação e integração do modelo de inteligência artificial text-davinci-003, destacando a qualidade das histórias de usuário, cenários de teste e códigos de automação gerados. As conclusões derivadas da pesquisa e as possíveis direções futuras de estudo são exploradas no Capítulo 5, encapsulando a essência da relevância do trabalho e suas potenciais contribuições para o domínio. Esse formato organizado proporciona uma exploração abrangente da aplicação "QAFlow", desde sua concepção e desenvolvimento até suas implicações potenciais e caminhos para futuras pesquisas.

2 Revisão Bibliográfica

Este capítulo apresenta uma contextualização da pesquisa com um resumo das discussões já feitas por outros autores sobre o assunto abordado e os conceitos principais relativos ao tema.

2.1 Fundamentação Teórica

Nesta seção, são fornecidos uma fundamentação teórica e explicações prévias acerca dos conceitos discutidos ao longo texto.

Para o desenvolvimento de um aplicativo que realiza criação automática de cenários de teste, histórias de usuários e códigos automatizados é essencial compreender os conceitos fundamentais relacionados ao teste de software, histórias de usuários, cenários de testes, automação de software , *Large Language Models (LLM)* e as tecnologias envolvidas na implementação de um aplicativo.

2.1.1 Teste de software

Os desenvolvedores enfrentam o desafio de criar programas que atendam aos requisitos do usuário e funcionem corretamente. No entanto, mesmo os desenvolvedores mais cuidadosos podem permitir que erros passem despercebidos. Testes são necessários para identificar erros e corrigi-los antes que o software seja lançado para o usuário final, garantindo a qualidade do software (MYERS; SANDLER; BADGETT, 2011). A partir de duas definições que encontramos na literatura é possível entender e contextualizar o que é um "teste". A primeira , afirma que o teste é destinado a mostrar o que um programa faz e o que é proposto a fazer, além de servir para descobrir os defeitos do programa antes do uso (SOMMERVILLE et al., 2011). A segunda, define um produto como útil apenas quando fornece o conteúdo, as funções e os recursos que o usuário deseja; além disso, e não menos importante, deve fornecer confiabilidade e isenção de erros. Para verificar se o produto funciona como desejado e aumentar a confiabilidade e isenção de erros, podemos realizar os testes de qualidade (PRESSMMAN, 2011). O processo de teste completo exige duas técnicas de teste, sendo elas: testes caixa-preta e caixa-branca (SOMMERVILLE et al., 2011) e (PRESSMMAN, 2011), 2016).

Os termos "caixa preta"e "caixa branca"referem-se a diferentes abordagens de teste de software, onde a "caixa"se refere ao software e sua funcionalidade, e "preta"e "branca"indicam diferentes níveis de visibilidade interna do código-fonte.

No teste de caixa preta, o testador se concentra apenas na entrada e saída do software, sem considerar sua estrutura interna ou o funcionamento interno. O objetivo principal é avaliar se o software funciona de acordo com as especificações e requisitos, sem se preocupar com a

lógica interna do código. Os testadores criam casos de teste com base nas especificações e na funcionalidade esperada do software, mas não têm conhecimento direto sobre como o software foi implementado. Isso é semelhante a interagir com uma "caixa preta"onde você insere "inputs"e observa as saídas sem saber o que acontece dentro dela.Exemplos de técnicas de teste de caixa preta incluem testes de equivalência, testes de limite, testes de caso de uso, etc. (SOMMERVILLE et al., 2011) (PRESSMMAN, 2011)

No teste de caixa branca, o testador tem acesso ao código-fonte e à estrutura interna do software. O objetivo é avaliar a qualidade do código, a lógica interna, a cobertura de código e identificar possíveis erros de programação. Os testadores criam casos de teste com base na análise do código, visando exercitar diferentes caminhos lógicos, fluxos de controle e estruturas de dados. Isso envolve uma compreensão mais profunda do código e do funcionamento interno do software. Exemplos de técnicas de teste de caixa branca incluem análise de código estática, testes de caminho, testes de ramificação, etc.

Combinando elementos de teste de caixa branca e teste de caixa preta temos outra abordagem de teste de software chamada teste de caixa cinza. Nesse tipo de teste, os testadores têm um conhecimento parcial sobre a estrutura interna do sistema, geralmente mais do que os testadores de caixa preta, mas menos do que os testadores de caixa branca (KHAN; KHAN, 2012).

Em resumo, a principal diferença entre os tipos de teste está na visibilidade do código-fonte. O teste de caixa preta é voltado para a funcionalidade externa, enquanto o teste de caixa branca se concentra tanto na funcionalidade quanto na qualidade do código interno. É importante ressaltar que o teste caixa-preta não é uma alternativa às técnicas caixa-branca, e devem ser utilizados em conjunto para obter um processo de testes completo (SOMMERVILLE et al., 2011) (PRESSMMAN, 2011)

No campo da engenharia de software, deve-se dar importância ao teste de software como uma abordagem profissional. É de extrema necessidade utilizar métodos e técnicas para garantir a qualidade do software em todas as etapas do ciclo de vida do desenvolvimento. Compreender os processos de teste adequados é fundamental para criar um software confiável e eficiente (PRESSMMAN, 2011).

2.1.2 Histórias de Usuário

As histórias de usuários, propostas por Mike Cohn, desempenham um papel fundamental no desenvolvimento ágil de software. Essa abordagem coloca os usuários finais no centro do processo de desenvolvimento buscando compreender suas necessidades e prioridades desde as fases iniciais do projeto (COHN, 2004).

Uma história de usuário é uma descrição curta e concisa de uma funcionalidade desejada do aplicativo, geralmente escrita na perspectiva do usuário. No entanto, além da descrição da

funcionalidade, as histórias de usuários também incluem critérios de aceitação. Os critérios de aceitação definem as condições que devem ser atendidas para que a história seja considerada concluída e aceita pelo usuário (COHN, 2004).

Os critérios de aceitação são uma parte essencial das histórias de usuários, pois ajudam a estabelecer um acordo claro entre a equipe de desenvolvimento e os usuários finais em relação ao que é esperado do software. Eles são especificações detalhadas que podem incluir requisitos funcionais, comportamentais, de desempenho, segurança, entre outros. Esses critérios ajudam a evitar ambiguidades e garantir que as expectativas do usuário sejam atendidas (ADZIC, 2011).

Ao utilizar histórias de usuários com critérios de aceitação, a equipe de desenvolvimento consegue direcionar seus esforços para atender às necessidades dos usuários de maneira mais precisa. Isso contribui para a melhoria da experiência geral do usuário, pois as funcionalidades são desenvolvidas de acordo com suas expectativas e requisitos específicos (ADZIC, 2011).

Em resumo, as histórias de usuários, com seus critérios de aceitação, permitem uma abordagem ágil e orientada ao usuário no desenvolvimento de software. Essa prática ajuda a garantir que o aplicativo atenda às necessidades dos usuários, melhorando sua experiência e satisfação. A inclusão dos critérios de aceitação nas histórias de usuários promove uma comunicação clara e alinhamento entre a equipe de desenvolvimento e os usuários finais, facilitando o desenvolvimento de um software de alta qualidade. Abaixo um exemplo de uma história de usuário referente a funcionalidade de adicionar tarefa à lista:

História de Usuário: Adicionar tarefas à lista

Como um usuário **Quero** poder adicionar novas tarefas à minha lista **Para que** eu possa manter o controle das coisas que preciso fazer.

Critérios de Aceitação:

1. Quando eu abrir o aplicativo, devo ver um campo de entrada de texto onde posso digitar o nome da nova tarefa.

2. Após digitar o nome da tarefa, devo ter uma opção clara para confirmar a adição (por exemplo, um botão "Adicionar"ao lado do campo de entrada).

3. A tarefa recém-adicionada deve ser exibida na lista de tarefas, juntamente com um indicador de conclusão padrão (por exemplo, uma caixa de seleção vazia).

2.1.3 Cenários de teste

Os cenários de testes, utilizando o formato *Behavior-Driven Development - (BDD)*, foram propostos inicialmente por Dan North e são amplamente reconhecidos como uma ferramenta valiosa para o processo de teste de software. (ADZIC, 2011) destaca a importância da utilização

do BDD, demonstrando como equipes de desenvolvimento bem sucedidas entregam o software certo e com excelente qualidade ao utilizar cenários de testes.

O BDD proporciona uma abordagem colaborativa entre os desenvolvedores, testadores e *stakeholders*, facilitando a compreensão dos requisitos e comportamentos esperados do aplicativo. Ao adotar uma linguagem de domínio específica e descrever os cenários em termos de comportamento, o BDD ajuda a garantir que todas as partes interessadas tenham uma compreensão clara das funcionalidades do aplicativo, evitando ambiguidades e proporcionando uma cobertura de testes mais abrangente (KUDRYASHOV, 2015).

A criação de cenários de testes adequados, com base no BDD, permite que o aplicativo seja testado de forma abrangente, identificando e corrigindo possíveis falhas e problemas antes do lançamento. Além disso, o uso do BDD auxilia no refinamento das atividades do projeto, melhorando a compreensão do sistema e aumentando a confiança da equipe. A linguagem de negócios simplificada do BDD melhora a comunicação e contribui para uma maior qualidade da aplicação, bem como aperfeiçoa a resiliência dos desenvolvedores ao longo do projeto (SOLIS; WANG, 2011). Abaixo um exemplo do BDD:

Cenário1: Campo de entrada de texto visível ao abrir o aplicativo

Dado que eu abri o aplicativo de lista de tarefas **Então** devo ver um campo de entrada de texto

Cenário2: Adicionar uma tarefa após digitar seu nome

Dado que eu estou no aplicativo de lista de tarefas **E** eu digitei o nome "Comprar leite"na caixa de entrada **Quando** eu clicar no botão "Adicionar"**Então** a tarefa "Comprar leite"deve ser adicionada à lista de tarefas E o campo de entrada de texto deve ser redefinido para vazio

Cenário3:Exibição de tarefa recém-adicionada na lista

Dado que eu adicionei a tarefa "Estudar para o exame"à lista de tarefas **Quando** eu olhar para a lista de tarefas **Então** devo ver a tarefa "Estudar para o exame"na lista **E** a tarefa deve ter um indicador de conclusão padrão

Esses cenários de teste descrevem como a funcionalidade "Adicionar tarefas à lista"deve se comportar em situações diferentes, de acordo com os critérios de aceitação definidos. Cada cenário apresenta uma sequência de ações e resultados esperados para validar o comportamento correto da funcionalidade.

As histórias de usuários e os cenários de teste BDD estão intrinsecamente relacionados no contexto do desenvolvimento ágil de software. As histórias de usuários fornecem uma visão geral das funcionalidades desejadas do aplicativo, enquanto os cenários BDD são usados para detalhar o comportamento esperado dessas funcionalidades. Dessa forma, as histórias de usuários e os cenários BDD se complementam, proporcionando uma abordagem completa e orientada ao usuário para o desenvolvimento de software, garantindo que as funcionalidades sejam im-

plementadas corretamente e atendam às necessidades dos usuários finais.(SMART; MOLAK, 2023).

2.1.4 Automação de testes

Os casos de testes desenvolvidos podem ser testados manualmente, onde o testador executa os cenários de teste manualmente ou pode ser feito de forma automatizada, em que são desenvolvidos *scripts* de automação e o computador executa automaticamente. Na Tabela 2.1, os testes automatizados são comparados com os testes manuais em critérios relevantes para uma empresa de software. A partir dessa comparação percebe-se que os testes automatizados são mais vantajosos na maioria dos quesitos, sendo que possuem desvantagem por serem mais custosos na fase inicial e por serem ineficazes para encontrar erros de interface, (SOMMERVILLE et al., 2011).

Critério	Teste Manual	Automação de Teste
Premissa básica	Precisão menor e não executa tarefas repetidas	Precisão maior e altamente confiável para tarefas repetidas
Tempo de execução	Precisa de muito tempo	É um processo muito rápido
Investimento	Necessário colaboradores	Necessário colaboradores e ferramentas de software
Indicado usar quando	Os casos de testes são executados uma ou duas vezes	Os casos de testes são executados sem limites
Interface com o Usuário	Altamente eficaz, pois, envolve a intervenção humana	Não é eficaz, uma vez que não há intervenção humana
Custo inicial	Pouco	Alto
Verificação	Não recomendado	Altamente recomendado

Tabela 2.1 – Comparação entre testes automatizados e manuais

2.1.5 Modelo de linguagem larga escala

O conceito de Modelos de Linguagem de Grande Escala (LLMs, na sigla em inglês) tem ganhado significativa atenção e exploração nos últimos anos. LLMs, como o *Generative Pre-trained Transformer 3 - (GPT-3*, representam um avanço notável no campo de processamento de linguagem natural e inteligência artificial (TAMKIN et al., 2021). LLMs são modelos densos de linguagem que utilizam enormes quantidades de dados textuais para aprender padrões e estruturas da linguagem humana. Esses modelos são treinados em diversas modalidades de dados, abrangendo uma ampla gama de tópicos, estilos e contextos.

LLMs como o GPT-3 são conhecidos por suas impressionantes capacidades de geração de linguagem, frequentemente produzindo texto coeso e contextualmente relevante. Eles são construídos sobre arquiteturas de transformadores, permitindo capturar conexões complexas dentro da linguagem, tornando-os adequados para várias tarefas de compreensão e geração de linguagem natural. Sua escala notável permite que gerem texto semelhante ao produzido por humanos, explorando o que é entendido da linguagem.

A utilização dos *Large Language Models (LLMs)* é um componente central na implementação do aplicativo proposto. O artigo de (ZIEGLER et al., 2019) explora como os grandes modelos de linguagem são aprendizes multitarefas não supervisionados. Os LLM's oferecem uma capacidade impressionante de processar e gerar texto em larga escala, tornando-os ideais para geração automática de cenários de teste, histórias de usuários e códigos para automação de testes.

2.1.6 Framework React

Para a criação do aplicativo, é necessário considerar a escolha da tecnologia de desenvolvimento. O React, um *framework* JavaScript popular para a construção de interfaces de usuário, desempenha um papel importante nesse processo. (RAWAT; MAHAJAN, 2020) aborda o uso do React para desenvolver aplicativos da web, fornecendo orientações práticas e exemplos para a criação de interfaces interativas e responsivas.

O ReactJS é uma biblioteca JavaScript usada para desenvolver componentes de interface do usuário (UI) reutilizáveis. De acordo com a documentação oficial do React, a seguinte é a definição: React é uma biblioteca para construir interfaces de usuário modulares.

O React permite o desenvolvimento de aplicativos web grandes e complexos que podem alterar seus dados sem atualizações subsequentes da página. Ele é usado como a View (V) no Modelo-Visão-Controlador (MVC). O React abstrai o *Document Object Model*, também conhecido como "Modelo de Objeto de Documento" (DOM), oferecendo uma experiência de desenvolvimento de aplicativos simples, eficiente e robusta. DOM é uma representação lógica padrão de uma página web conforme o *World Wide Web Consortium*. DOM é implementada na maioria dos navegadores (*browsers*) modernos e permite que a página seja modificada facilmente por uma linguagem de script, como JavaScript visto a representação estruturada da página como uma árvore.

O React renderiza principalmente no lado do servidor usando o NodeJS e o suporte para aplicativos móveis nativos é fornecido pelo React Native. O React implementa um fluxo de dados unidirecional, simplificando a reutilização de código (conhecida também como *boilerplate*) e é considerado mais fácil do que as abordagens tradicionais de vinculação de dados (*data binding*) (RAWAT; MAHAJAN, 2020).

Principais Características do *framework* React:

A) DOM Leve para Melhor Desempenho: O React fornece um modelo de objeto de documento muito eficiente e leve. Ele não interage diretamente com o DOM gerado pelo navegador, mas reage ao modelo de objeto de documento armazenado na memória. Isso resulta em um desempenho rápido e robusto do aplicativo. O React usa algo chamado de DOM virtual. Comparativos entre o DOM virtual e o DOM real são feitos usando um algoritmo de verificação de diferenças, genericamente denominados como "diff()", em analogia ao comando `diff` do linux, e apenas as mudanças nos nós são refletidas de volta na árvore do modelo de objeto de documento.

B) Curva de Aprendizado Fácil: A natureza fácil e não complexa do React permite que os desenvolvedores se familiarizem rapidamente com o framework. A curva de aprendizado é extremamente fácil, tornando a adoção do React sem complicações. A arquitetura é intuitivamente simples, e a ideia de usar JSX parece natural e agradável para os desenvolvedores.

C) JSX: JavaScript XML, ou simplesmente JSX, é uma extensão da linguagem JavaScript que permite a inserção de elementos HTML em um código . Embora não seja obrigatório usá-lo ao desenvolver um aplicativo baseado em React, JSX é altamente popular entre os desenvolvedores, pois é uma forma simplificada de escrever HTML diretamente no código Javascript.

D) Desempenho: O ReactJS é conhecido por ser altamente eficiente. Isso é possível devido ao uso do DOM virtual. O React mantém um modelo de objeto de documento virtual na memória. Quando uma alteração precisa ser refletida na página atual, em vez de atualizar instantaneamente o DOM do navegador, as alterações são feitas primeiro no DOM virtual. Depois disso, um algoritmo diff() compara o DOM virtual com o DOM do navegador e apenas os nós relevantes e desejados são atualizados, resultando em um desempenho rápido do aplicativo.

E) Fluxo de Dados Unidirecional: ReactJS foi projetado de forma a suportar fluxos de dados unidirecional. O conceito de fluxo de dados unidirecional vem da programação funcional que significa que os componentes filhos não podem atualizar os dados que vêm do componente pai. Isso faz muito sentido lembrando que páginas Web são documentos DOM estruturados em forma de árvore, que a unidirecionalidade garante que os componentes sejam imutáveis e que os dados dentro deles não mudem em nenhuma circunstância. Portanto, somente o fluxo de dados que vem em uma direção é permitido, não o contrário.

F) DOM Virtual: Outra característica fundamental do ReactJS é o DOM virtual. Ele é semelhante ao DOM gerado pelo navegador mas é armazenado em memória. Quando uma solicitação de mudança de conteúdo da página é feita, as alterações são refletidas no DOM virtual residente na memória. Em seguida, um algoritmo diff() verifica as diferenças, comparando o DOM virtual e o DOM do navegador e reflete somente as mudanças necessárias no DOM do navegador, em vez de renderizar novamente todo o DOM. Isso melhora significativamente o desempenho do aplicativo, especialmente em cenários com muitas alterações de dados.

Considerando os objetivos do projeto objeto desta monografia, com base em suas ca-

racterísticas e facilidades, o *framework* escolhido para o aplicativo foi o ReactJS. A escolha se fundamentou na capacidade do ReactJS de desenvolver interfaces de usuário modulares e reutilizáveis de maneira eficiente. Sua abstração do Document Object Model (DOM) e a implementação do DOM virtual possibilitam um desempenho mais rápido e robusto do aplicativo. Além disso, a facilidade de aprendizado e a ampla adoção da sintaxe JSX simplificam o processo de desenvolvimento e melhoram a legibilidade do código.

A natureza unidirecional do fluxo de dados do ReactJS e sua capacidade de renderização no lado do servidor usando NodeJS alinham-se bem com as necessidades do projeto. Isso permitirá a criação de uma experiência de usuário fluida e responsiva, evitando atualizações de página desnecessárias e proporcionando interações mais dinâmicas.

Outro fator determinante para a escolha do ReactJS foi sua comunidade ativa e o vasto ecossistema de bibliotecas e ferramentas disponíveis. Isso facilita a integração de funcionalidades avançadas, como manipulação de estados complexos e gerenciamento eficiente de componentes. Com a capacidade de se adaptar tanto a projetos pequenos quanto a aplicações escaláveis, o ReactJS emerge como uma escolha sólida para construir um aplicativo que atenda aos requisitos do projeto de maneira eficaz e eficiente.

2.1.7 Modelo LLM text-davinci-003

O "Text-Davinci-003"é uma variante intermediária do modelo GPT (Generative Pre-trained Transformer), situada entre o GPT-3 e o GPT-4. Ele representa uma evolução em relação ao GPT-3, apresentando um desempenho aprimorado por meio de ajustes de instruções mais específicas. Essa variante serve como uma ponte entre as duas versões, permitindo análises comparativas e oferecendo um equilíbrio entre recursos e complexidade.(ZHONG et al., 2023)

Ao criar histórias de usuário e cenários de teste, é fundamental ter um modelo de linguagem que seja capaz de entender instruções específicas, gerar texto coerente e se ajustar aos requisitos e nuances das histórias e cenários. O "Text-Davinci-003"é projetado para oferecer um equilíbrio ideal nesse sentido. Ele possui uma compreensão refinada de linguagem natural, permitindo a geração de conteúdo coeso e relevante para histórias de usuário e cenários de teste (ZHONG et al., 2023).

Além disso, o "Text-Davinci-003"foi escolhido considerando a natureza das tarefas envolvidas. Histórias de usuário e cenários de teste muitas vezes exigem descrições detalhadas, consistência lógica e coerência no texto gerado. O "Text-Davinci-003"é capaz de cumprir esses requisitos de maneira competente.

Ao mesmo tempo, a escolha do "Text-Davinci-003"também leva em consideração a praticidade e a acessibilidade. Embora o GPT-4 possa oferecer um desempenho ainda mais avançado, o "Text-Davinci-003"proporciona um balanço adequado entre qualidade e recursos, tornando-o uma escolha viável para projetos que buscam gerar histórias de usuário e cenários de

teste de maneira eficaz e eficiente.

2.2 Trabalhos Relacionados

Abaixo serão descritos os principais trabalhos realizados por outros autores sobre a temática escolhida para ser desenvolvida, apresentando os conceitos mais importantes, justificativas e características sobre o tema, do ponto de vista da análise feita pelos autores.

2.2.1 Revisão sobre ferramentas, técnicas e desafios no teste de software

A partir de uma revisão sistemática da literatura sobre ferramentas, técnicas e desafios em teste de software foi possível destacar algumas ferramentas existentes, as técnicas utilizadas e os desafios enfrentados no campo do teste de software entre os anos de 2010 e 2019 (ABDULLAHI et al., 2020). Nesse artigo foi realizada uma revisão sistemática da literatura, onde foram selecionados 70 artigos relevantes que abordavam o tema. Os critérios de inclusão e exclusão foram definidos para garantir que apenas os estudos pertinentes fossem considerados. A coleta de dados foi realizada por meio da análise dos artigos selecionados, com ênfase nas ferramentas mencionadas, nas técnicas propostas e nos desafios identificados pelos autores.

Para a metodologia foram conduzidos procedimentos rigorosos para garantir a qualidade e a abrangência da revisão. Os pesquisadores buscaram identificar as principais tendências e avanços na área de teste de software, com o intuito de fornecer um panorama atualizado sobre o tema.

A conclusão da revisão dos estudos destaca os principais desafios enfrentados no teste de software, como apresentado na Figura 2.1. Primeiramente, foi observado que a geração manual de casos de teste foi abordada por 42% dos estudos revisados. Essa abordagem manual tem se mostrado ineficaz no processo de teste, ressaltando a necessidade de soluções mais eficientes e automatizadas nessa área. Em seguida, 14% dos estudos propuseram soluções de automação de testes para superar os desafios do teste manual. A automação de tarefas de teste de software tem se mostrado promissora e tem o potencial de melhorar a eficiência dos testes. Diversas técnicas de automação foram propostas e exploradas nos estudos selecionados, demonstrando a importância de abordagens automatizadas para otimizar o processo de teste. Além disso, 18% dos estudos se concentraram em desafios relacionados à localização de falhas em programas de software. Identificar e corrigir falhas de forma eficiente é fundamental para garantir a qualidade do software. Esses estudos abordaram técnicas avançadas de depuração e análise de código para melhorar a precisão e a eficácia na localização de falhas. Outros desafios mencionados incluíram a melhoria da efetividade do teste, com 8% dos estudos buscando aprimorar a eficácia global do processo de teste, e a obtenção de cobertura de teste, abordada por 4% dos estudos. (ABDULLAHI et al., 2020)

A aplicação irá abordar esses principais problemas enfrentados. Com as história de usuário e cenários de teste BDD sendo gerados automaticamente torna a geração de casos de teste mais rápida e eficiente. Quanto a parte de testes manuais a geração automática de códigos para automação ajuda a ter mais casos de testes automatizados que melhoram a velocidade do processo de testes e o torna mais confiável. Além disso, com o "QAFlow"será possível o testador dedicar mais tempo a outras tarefas como a localização de erros, pois terá alguns processo otimizados.

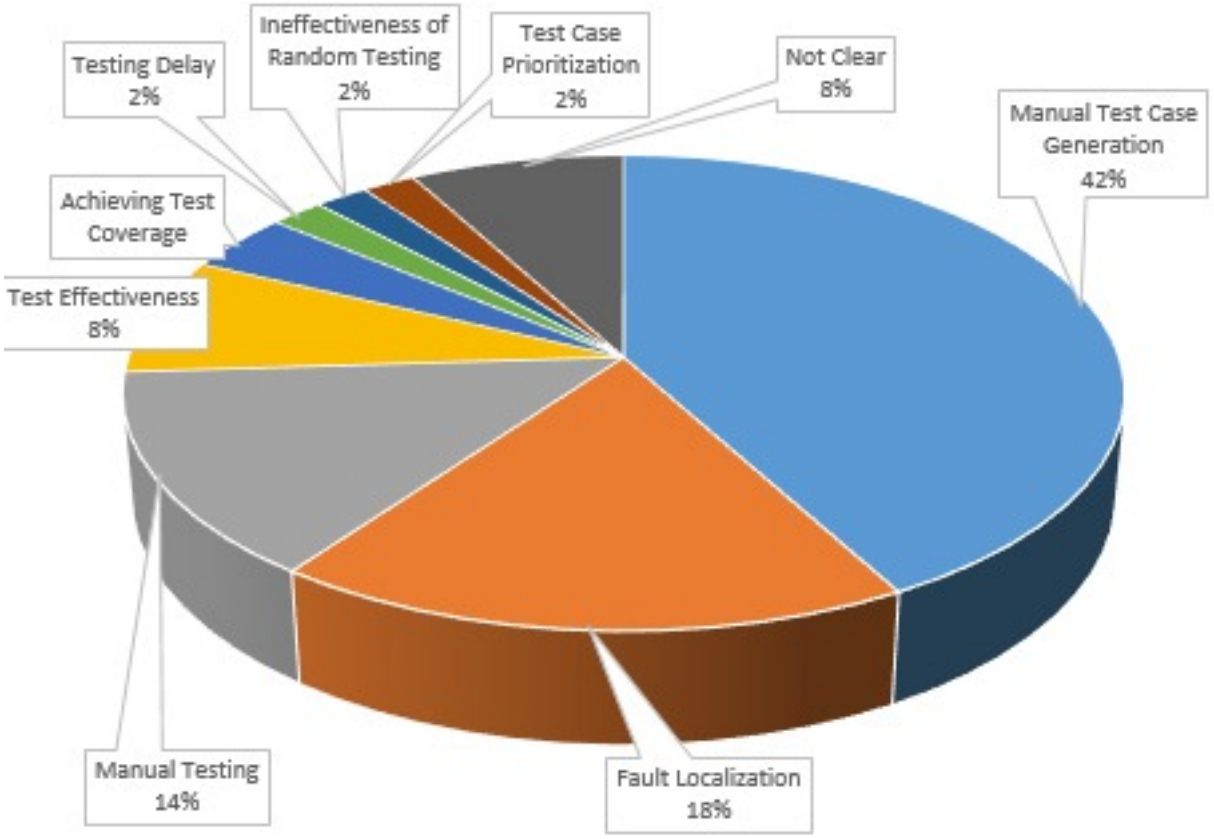

Figura 2.1: Principais desafios enfrentados no teste de software

Fonte: (ABDULLAHI et al., 2020)

Esses resultados evidenciam a importância de enfrentar os desafios no teste de software por meio de abordagens inovadoras, como o uso de inteligência artificial para aumentar a eficiência do processo de qualidade de software. A pesquisa e o desenvolvimento contínuos nessa área são fundamentais para aprimorar a qualidade do software e atender às demandas crescentes dos sistemas cada vez mais complexos.

2.2.2 O impacto da inteligência artificial no teste de software

A Inteligência Artificial (IA) desempenha um papel importante em nossas vidas e impacta a maioria de nossas aplicações e sistemas ao redor (HOURANI; HAMMAD; LAFI, 2019). Enormes quantidades de dados são geradas todos os dias a partir de diversas fontes que precisam ser monitoradas, analisadas adequadamente e ter seus resultados reportados e ações tomadas.

À medida que aplicativos de software mais complexos são desenvolvidos, o tempo está se tornando um fator crítico para lançar aplicativos que devem ser completamente testados e estar em conformidade com os Requisitos de Negócios. A IA desempenha um papel fundamental no Teste de Software, proporcionando resultados mais precisos e economizando tempo. Este artigo discute os pilares-chave da Inteligência Artificial que podem ser utilizados no Teste de Software. Ele também lança uma visão sobre como o futuro será em termos de Inteligência Artificial e Teste de Software. Os resultados mostram que a IA pode alcançar melhores resultados no Teste de Software e que os testes impulsionados pela IA liderarão a nova era do trabalho de garantia de qualidade (QA) em um futuro próximo. O Teste de Software baseado em IA reduzirá o tempo de lançamento no mercado e aumentará a eficiência da organização para produzir software mais sofisticado, criando testes automatizados mais inteligentes.

A Inteligência Artificial tem a capacidade de analisar automaticamente dados complexos usando modelos e algoritmos inteligentes. A IA já demonstrou ser capaz de obter melhores resultados no teste de software. Os testes impulsionados pela IA liderarão a nova era do trabalho de QA em um futuro próximo. Eles gerenciarão e controlarão a maioria das áreas de teste, agregando grande valor ao resultado dos testes e produzindo resultados mais precisos em um prazo competitivo. Não há dúvida de que a IA influenciará a indústria de QA e teste e liderará esse caminho no futuro. A automação inteligente do teste de software melhorará a qualidade do software e terá um grande impacto na experiência do cliente, fornecendo aplicativos e soluções sólidas e livres de defeitos.

É esperado que a IA desempenhe um papel fundamental no teste de software no futuro. O novo papel e escopo dos testadores se concentrarão em aprimorar os modelos de IA, algoritmos e técnicas para que se tornem mais inteligentes. Algoritmos de Teste de IA também se conectarão a novas tecnologias no futuro (como tecnologia de nuvem, IoT, Big Data e outras) e extrairão as melhores práticas e técnicas que se adequem à aplicação do cliente para obter casos de teste mais precisos e inteligentes e gerar resultados perfeitos.

Existe a possibilidade de pesquisas futuras ao explorar outras áreas na interseção entre a IA e o teste de software. A aprendizagem profunda é uma área promissora na IA que pode oferecer melhores resultados do que os algoritmos tradicionais de IA. Essa área pode ser investigada para avaliar como a aprendizagem profunda pode desempenhar um papel no teste de software. Outra área a ser explorada é a cobertura de mais estudos para investigar outras áreas de teste que ainda não foram abordadas nesta pesquisa. Isso ampliaria o entendimento do impacto da IA no teste de software em diversos contextos (HOURANI; HAMMAD; LAFI, 2019).

2.2.3 Revisão da literatura Inteligência Artificial Aplicada a Testes de Software

Embora inteligência artificial (IA) e *machine learning (ML)* tenham sido aplicados com sucesso em vários cenários do mundo real, sua aplicação em teste de software ainda é em sua maioria acadêmica (LIMA; CRUZ; RIBEIRO, 2020). O artigo analisa o progresso dos métodos de IA e ML usados no teste de software nos últimos três anos, categorizando-os com base nos tipos de teste. Ele conclui que o teste de caixa preta é o método preferido quando a IA é aplicada, e todos os três métodos de ML (supervisionado, não supervisionado e de reforço) são comumente usados no teste de caixa preta. O artigo também identifica algoritmos específicos de IA, como agrupamento, redes neurais artificiais e algoritmos genéticos, que são aplicados a testes de fuzzing e regressão.

Os LLMs são uma subcategoria especializada de IA que se concentra principalmente em tarefas de processamento de linguagem natural, como tradução automática, geração de texto e análise de sentimentos em texto. Eles estão trazendo mudanças significativas para o campo da Engenharia de Software. O potencial desses modelos para lidar com tarefas complexas pode remodelar fundamentalmente muitas práticas e ferramentas da engenharia de software. Foi analisado a crescente utilização de LLMs na engenharia de software, abrangendo trabalhos publicados desde o surgimento do primeiro LLM (BERT) (HOU et al., 2023).

Foram analisados diversos LLMs que têm sido empregados em tarefas da engenharia de softwate e foram exploradas suas características e aplicações distintas. Em seguida, foram investigados os processos envolvidos na coleta de dados, pré-processamento e uso, destacando o papel significativo de conjuntos de dados bem curados no êxito da aplicação de LLMs para solucionar tarefas da área (HOU et al., 2023).

Essas perspectivas representam uma indicação promissora do impacto revolucionário futuro de LLMs na Engenharia de Software. Elas apontam para uma transformação significativa na forma como o desenvolvimento de software é conduzido, incorporando de forma mais profunda e inteligente a compreensão da linguagem natural em todo o ciclo de desenvolvimento (HOU et al., 2023).

2.2.4 Desenvolvendo um aplicativo móvel com ChatGPT

Foi desenvolvido um aplicativo com o propósito abordar de maneira inovadora a problemática do declínio do interesse pela leitura, especialmente entre as gerações mais jovens. Em resposta ao aumento do uso de tecnologia por parte dos jovens e a possíveis consequências negativas para a saúde cognitiva, é proposto um aplicativo móvel para dispositivos Android, que se utiliza justamente da tecnologia para revitalizar o interesse pela leitura (CORRÊA et al., 2023).

Uma das características distintivas desse aplicativo é o uso da inteligência artificial, representada pelo ChatGPT, como base para suas funcionalidades. O uso da inteligência artificial

adiciona um toque de inovação ao projeto, permitindo uma experiência de exploração de histórias altamente adaptável e envolvente para os usuários. Através do uso do ChatGPT, o aplicativo oferecerá uma plataforma que não apenas captura a atenção do público-alvo, composto por crianças e jovens leitores, mas também estimula o enriquecimento cognitivo, criativo e linguístico (ZHONG et al., 2023).

Ao combinar a tecnologia de um aplicativo móvel com a inteligência artificial, este trabalho proporciona uma solução inovadora para incentivar a leitura entre os jovens, fazendo uso das ferramentas tecnológicas de forma responsável e benéfica para a saúde cognitiva. Este projeto reforça como o uso de LLM em um software pode ser uma estratégia valiosa em diversas ocasiões.

2.2.5 Estudo sobre o Assistente Jurídico Inteligente

Existem diversos estudos que exploraram o uso de IA na pesquisa jurídica e na automação de tarefas legais, esta pesquisa expandiu o escopo, aplicando modelos de linguagem de grande porte (GPT-3) para oferecer suporte a operadores do sistema judiciário. No entanto, em contraste com alguns desses trabalhos anteriores que se concentraram em pesquisa e processamento de documentos jurídicos, o "Assistente Jurídico Inteligente"se destaca ao se concentrar na identificação de crimes, tipificação de condutas e respostas específicas para casos hipotéticos (SILVA, 2023).

Este estudo sobre o "Assistente Jurídico Inteligente"e o estudo relacionado ao aplicativo "QAFlow"compartilham algumas semelhanças e diferenças notáveis, apesar de estarem em domínios distintos. Segue uma breve comparação:

Semelhanças:

1. Aplicação de Modelos de Linguagem de Grande Porte (LLMs): Ambos os estudos empregam LLMs, como o GPT-3, para criar assistentes virtuais ou aplicativos de inteligência artificial. Eles demonstram o uso da tecnologia de LLMs para fornecer respostas relevantes e úteis aos usuários em seus respectivos domínios.

2. Automatização de Tarefas Específicas: Tanto o "Assistente Jurídico Inteligente"quanto o aplicativo de teste de software visam automatizar tarefas específicas em seus respectivos campos. O primeiro lida com a análise de documentos legais e a identificação de crimes, enquanto o segundo se concentra na geração de cenários de teste, histórias de usuário e códigos de automação.

3. Eficiência e Precisão: Ambos os estudos buscam melhorar a eficiência e a precisão das tarefas relacionadas aos seus campos de aplicação. O "Assistente Jurídico Inteligente"procura economizar tempo na pesquisa de leis e análise de processos judiciais, enquanto o aplica-

tivo de teste de software almeja agilizar o processo de criação de artefatos para teste de software.

Diferenças:

1. Domínio de Aplicação: A diferença mais óbvia é o domínio de aplicação. O "Assistente Jurídico Inteligente"está relacionado ao campo jurídico, com foco na análise de documentos legais e processos judiciais. Por outro lado, o aplicativo de teste de software está inserido no contexto da engenharia de software, com ênfase na criação de cenários de teste e histórias de usuário.

2. Entrada e Saída de Dados: As entradas e saídas de dados são diferentes. O "Assistente Jurídico Inteligente"recebe consultas ou informações legais e fornece respostas baseadas em leis e jurisprudência. Enquanto isso, o aplicativo de teste de software recebe requisitos e especificações de software e gera cenários de teste ou histórias de usuário.

3. Natureza das Respostas: As respostas geradas pelos LLMs em cada contexto são diferentes. No aplicativo jurídico, as respostas podem ser uma interpretação jurídica, enquanto no aplicativo de teste, as respostas são cenários ou histórias de usuário que descrevem o comportamento do software.

4. Uso de Dados de Treinamento: Os aplicativos podem usar diferentes conjuntos de dados de treinamento. O "Assistente Jurídico Inteligente"pode ser treinado em documentos legais e casos judiciais, enquanto o aplicativo de teste de software pode se beneficiar do uso de especificações de software e casos de teste existentes.

Em resumo, embora os dois estudos aproveitem o poder dos LLMs para automatizar tarefas e melhorar a eficiência em seus respectivos domínios, eles são aplicados em contextos muito diferentes e atendem a necessidades específicas de cada área. Ambos representam exemplos interessantes de como a inteligência artificial pode ser aplicada para resolver problemas complexos em diversos campos.

3 Desenvolvimento

Neste capítulo é definido o tipo de pesquisa e como foi realizada. São detalhados todos os instrumentos e procedimentos metodológicos, apresentados na fundamentação teórica, que serão utilizados para alcançar o objetivo proposto.

Nesta seção, exploraremos em detalhes o processo de desenvolvimento do aplicativo que utiliza a API do modelo de inteligencia artificial text-davinci-003 para gerar histórias de usuário, cenários de teste e automações com base nas descrições de funcionalidades fornecidas pelos usuários. O desenvolvimento do aplicativo foi dividido em etapas bem definidas para garantir um processo organizado e eficaz.

3.1 Definição de Requisitos e Escopo

No início do desenvolvimento do QAFlow, foi realizado uma análise minuciosa dos requisitos do projeto, levando em consideração diversos aspectos essenciais para o sucesso da aplicação. Este processo de definição de requisitos e escopo foi fundamental para estabelecer as bases sólidas do QAFlow. Aqui estão as principais etapas e considerações:

1. **Compreensão das Necessidades dos Usuários:** Primeiramente, buscamos entender profundamente as necessidades dos usuários do QAFlow. Isso foi realizado através de pesquisas que permitiram identificar as lacunas nas ferramentas existentes e definir requisitos que realmente atendessem às necessidades da comunidade de qualidade de software.

2. **Saídas Desejadas Claras:** Definimos com precisão os tipos de saídas desejadas pelo QAFlow. Isso englobou a geração automática de três tipos principais de artefatos: histórias de usuário, cenários de teste e códigos de automação. Esses artefatos são cruciais para o planejamento e execução de projetos de desenvolvimento de software, tornando-os uma parte central da funcionalidade do QAFlow.

3. **Exploração dos Recursos da API Davinci 003:** Uma parte fundamental do processo de definição de requisitos foi a exploração dos recursos específicos da API Davinci 003. Esta API de modelo de linguagem de aprendizado de máquina ofereceu a capacidade de gerar texto de maneira coesa e relevante. Avaliamos cuidadosamente como esses recursos poderiam ser aproveitados para atender aos requisitos do QAFlow.

4. **Escopo Claramente Definido:** Uma vez que compreendemos as necessidades dos usuários e as saídas desejadas, definimos claramente o escopo do QAFlow. Isso significa que concentramos nossos esforços nas funcionalidades principais que proporcionariam o maior valor aos profissionais de qualidade de software. Estabelecer um escopo claro nos permitiu

direcionar nossos recursos de maneira eficaz e garantir o desenvolvimento de um aplicativo útil e eficiente.

Em resumo, a fase de definição de requisitos e escopo no desenvolvimento do QAFlow foi conduzida de maneira abrangente, considerando as necessidades dos usuários, os tipos de saídas desejadas e os recursos da API Davinci 003. Isso resultou em um aplicativo que se concentra nas funcionalidades essenciais para melhorar o processo de qualidade de software, contribuindo significativamente para o sucesso de projetos de desenvolvimento.

3.2 Escolha do modelo text-davinci-003

A seleção do modelo de linguagem de aprendizado de máquina (LLM) é um passo crítico no desenvolvimento do QAFlow. Nesse contexto, a escolha recaiu sobre o modelo "Text-Davinci-003". Essa decisão foi baseada em uma cuidadosa avaliação das necessidades específicas do projeto e das características do modelo. A escolha do "Text-Davinci-003"foi fundamentada nas seguintes considerações:

1. **Compreensão Aprimorada de Linguagem Natural:** O "Text-Davinci-003"demonstrou uma compreensão avançada da linguagem natural. Sua capacidade de interpretar descrições de funcionalidades fornecidas pelos usuários e gerar texto coeso e relevante tornou-o uma escolha ideal para um projeto voltado para a criação de histórias de usuário, cenários de teste e scripts automatizados. Isso é fundamental para garantir a qualidade e a utilidade das saídas geradas pelo QAFlow.

2. **Equilíbrio entre Recursos e Complexidade**: Em comparação com modelos mais recentes, como o GPT-4, o "Text-Davinci-003"oferece uma combinação equilibrada de recursos e complexidade. Isso é especialmente importante para garantir que o QAFlow seja eficiente e acessível. Optar por um modelo mais avançado poderia aumentar a complexidade desnecessariamente, sem fornecer benefícios substanciais adicionais para os usuários.

3. **Acessibilidade:** A escolha do "Text-Davinci-003"também considerou a acessibilidade. Modelos mais recentes podem ser mais caros ou exigir recursos computacionais significativamente maiores. Ao escolher o "Text-Davinci-003", buscamos garantir que o QAFlow seja uma solução viável para uma ampla gama de usuários, incluindo pequenas equipes de desenvolvimento e projetos com recursos limitados.

Em resumo, a escolha do modelo "Text-Davinci-003"para o QAFlow foi cuidadosamente ponderada, visando atender às demandas específicas do projeto. Sua capacidade aprimorada de compreensão da linguagem natural, equilíbrio entre recursos e acessibilidade foram fatores-chave na decisão. Essa escolha desempenha um papel fundamental na capacidade do QAFlow de gerar

saídas precisas e úteis para profissionais de qualidade de software, contribuindo assim para a eficácia do processo de desenvolvimento de software.

3.3 Design da Interface de Usuário

O design da interface de usuário desempenha um papel fundamental na usabilidade e na experiência do usuário do QAFlow. Foi uma etapa crucial do processo de desenvolvimento, buscando foco em ter clareza, simplicidade e eficácia. Abaixo, são destacados os principais aspectos do design da interface:

1. Usabilidade Intuitiva: A usabilidade do QAFlow era uma prioridade máxima. Para garantir que os usuários pudessem interagir facilmente com o aplicativo, criamos uma interface intuitiva. O processo de inserção de descrições de funcionalidades foi projetado para ser claro e direto, permitindo que os usuários fornecessem informações de maneira simples e eficaz.

2. Organização Lógica: A organização da interface foi planejada cuidadosamente. As saídas geradas pelo modelo de IA, como histórias de usuário e cenários de teste, são exibidas de maneira organizada e estruturada. Isso facilita a compreensão e a visualização das informações, ajudando os usuários a acompanhar o progresso de seus projetos de desenvolvimento de software.

3. Minimização de Complexidade: Evitamos a sobrecarga de informações e a complexidade desnecessária na interface. Isso foi essencial para garantir que os usuários pudessem se concentrar nas tarefas principais, como a descrição de funcionalidades, sem distrações ou confusões.

4. Feedback Visual: Foi implementado um feedback visual para que os usuários saibam quando suas ações foram concluídas com sucesso. Isso inclui indicadores visuais claros para confirmar a aceitação de descrições de funcionalidades e a exibição imediata das saídas geradas.

5. Responsividade: Foi criado uma interface responsiva, que se adapta a diferentes tamanhos de tela e dispositivos. Isso permite que os usuários acessem o QAFlow em computadores, tablets e smartphones, tornando-o versátil e acessível.

6. Avaliação de Usuários: Durante o processo de desenvolvimento, foram realizados testes de usabilidade manuais. Tais testes foram essenciais para aprimorar ainda mais a interface, tornando-a mais alinhada com as necessidades e expectativas dos usuários.

Em resumo, o design da interface de usuário do QAFlow foi cuidadosamente planejado para proporcionar uma experiência agradável e eficiente aos usuários. A interface intuitiva, a

organização lógica, a minimização da complexidade e a responsividade são elementos-chave que contribuíram para tornar o QAFlow uma ferramenta amigável e poderosa para profissionais de qualidade de software. Podemos observar a interface final da aplicação na Figura 3.

Figura 3 - Interface da aplicação web QAFlow

Fonte: App QAFlow

3.4 Desenvolvimento do *Backend*:

O desenvolvimento do backend do QAFlow desempenhou um papel crucial na garantia de que o aplicativo pudesse funcionar de maneira eficaz e oferecer resultados precisos aos usuários. Aqui estão alguns detalhes essenciais sobre o desenvolvimento do *backend*:

1. **Tecnologias Utilizadas:** Optamos por tecnologias de *backend* que fossem robustas e escaláveis para lidar com a comunicação com a API Davinci 003 e o processamento das respostas. O backend foi desenvolvido utilizando Node.js, Express.js, CORS e Openai API o que proporcionou uma base sólida e confiável para a aplicação.

2. **Integração com a API Davinci 003:** Uma parte central do desenvolvimento do *backend* envolveu a integração perfeita com a API Davinci 003. Foram implementados *endpoints* cuidadosamente projetados para facilitar o envio de solicitações para a API, receberem as respostas e processarem os dados retornados. Isso incluiu a formatação adequada das entradas dos usuários para atender aos requisitos da API e a interpretação das respostas geradas pelo modelo de IA.

3. **Validação de Entradas e Saídas:** A segurança e a consistência eram prioridades na construção do *backend*. Foi implemetado um sistema de validação de entradas dos usuários para garantir que as descrições de funcionalidades fornecidas estivessem no formato correto

e fossem adequadas para a chamada à API. Além disso, as saídas geradas tiveram que ser formatadas, garantindo que as histórias de usuário, cenários de teste e automações fossem coerentes e relevantes para o contexto do desenvolvimento de software.

4. **Integração com o *Frontend*:** O *backend* foi projetado para funcionar em perfeita harmonia com o *frontend* do QAFlow. A comunicação entre essas duas partes foi cuidadosamente sincronizada para oferecer uma experiência de usuário contínua e eficaz.

O desenvolvimento do *backend* do QAFlow desempenhou um papel fundamental na transformação de uma ideia inovadora em uma aplicação funcional e eficiente. Suas capacidades de comunicação com a API Davinci 003, validação de entradas, formatação de saídas e integração com o *frontend* são elementos essenciais para o sucesso da aplicação, contribuindo significativamente para a automação e melhoria da qualidade no processo de desenvolvimento de software.

3.5 Integração com a API text-davinci-003

Nesta etapa, a integração com a API text-davinci-003 desempenhou um papel central na capacidade do QAFlow de gerar histórias de usuário, cenários de teste e automações de forma precisa e eficiente. Abaixo estão os detalhes sobre como essa integração foi realizada:

1. **Formatação de Descrições de Funcionalidades:** Uma parte fundamental da integração foi a criação de lógica para formatar as descrições de funcionalidades fornecidas pelos usuários. Essas descrições foram preparadas em um formato adequado para a entrada do modelo de IA text-davinci-003. Isso envolveu a estruturação clara das informações para garantir que a API pudesse compreender os requisitos do usuário.

2. **Requisições à API:** Após a formatação das entradas, a aplicação enviou solicitações para a API text-davinci-003. Essas solicitações incluíram as descrições de funcionalidades preparadas e um texto pré-escrito que ajudava a orientar o modelo de linguagem na geração das saídas desejadas.

3. **Processamento de Respostas:** Quando as respostas foram recebidas da API, a aplicação foi capaz de interpretar os dados retornados. Isso incluiu a extração das histórias de usuário, cenários de teste e automações geradas pelo modelo de IA. As respostas da API foram processadas para garantir que o conteúdo fosse coerente e relevante para o contexto do desenvolvimento de software.

4. **Integração Perfeita com o Frontend:** A integração entre o *backend* e o *frontend* foi projetada para funcionar de maneira harmoniosa. Os resultados da API text-davinci-003 foram transmitidos de volta ao frontend, onde foram apresentados de maneira compreensível aos usuários.

5. **Trabalho em Tempo Real:** A integração permitiu que o QAFlow funcionasse em tempo real, permitindo que os usuários vissem imediatamente as histórias de usuário, cenários de teste e automações geradas com base em suas descrições de funcionalidades.

Essa integração com a API text-davinci-003 desempenhou um papel essencial na capacidade do QAFlow de automatizar e melhorar o processo de geração de artefatos de teste, proporcionando aos profissionais de qualidade de software uma ferramenta eficaz para suas atividades.

3.6 Testes e Validação

Para garantir a confiabilidade e a eficácia do QAFlow, foram realizados extensos testes manuais em todas as etapas do desenvolvimento. Esses testes abrangeram a integração perfeita com a API Davinci 003 e a geração de saídas em uma variedade de cenários de uso. Durante os testes, foi avaliado a qualidade das histórias de usuário, cenários de teste e automações geradas pelo aplicativo.

Os testes manuais desempenharam um papel fundamental na identificação e correção de possíveis problemas, garantindo que o QAFlow pudesse oferecer resultados precisos e confiáveis. Os *feedbacks* obtidos durante essa fase de teste contribuíram significativamente para aprimorar a aplicação, tornando-a mais robusta e eficiente.

O processo de teste e validação foi crucial para assegurar que o QAFlow atendesse às expectativas dos profissionais de qualidade de software, proporcionando um ambiente confiável para a geração de artefatos essenciais no desenvolvimento de software.

Dado que as saídas produzidas pela aplicação podem apresentar variações e não necessariamente coincidirem com o exemplo de referência, os testes manuais tornaram-se uma opção mais vantajosa. Isso ocorre porque a observação humana pode facilmente identificar se os artefatos gerados se aproximam do que é necessário, enquanto a automação busca uma correspondência exata com o exemplo fornecido.

Em conclusão, os resultados da bateria de testes realizados revelaram que a nossa ferramenta demonstrou capacidade de gerar corretamente os artefatos desejados. Estas validações, demonstradas nas subseções seguintes, reforçam a confiabilidade e eficácia da ferramenta na automatização de tarefas cruciais no ciclo de desenvolvimento de software. A capacidade de produzir com precisão os resultados desejados não apenas economiza tempo e recursos, mas também aumenta significativamente a qualidade e a consistência dos artefatos gerados, contribuindo assim para um processo de desenvolvimento mais eficiente e confiável.

3.6.1 Testes da geração das histórias de usuário

Abaixo são exemplificados como foram feitos alguns dos testes da página de geração de histórias de usuário. Primeiramente é mostrada a história criada manualmente, que seria o resultado esperado e depois como foi a história de usuário gerada pela aplicação "QAFlow".

Historia de usuário feita manualmente: Realizar Login EU COMO usuário GOSTARIA DE realizar meu login PARA poder ter acesso a funcionalidade exclusivas do aplicativo

Resposta do app ao digitar *Accomplish Login*: *As a user, I want to be able to log into my account So that I can access my account information Acceptance Criteria: - User must provide a valid username and password - User must be authenticated before being allowed access - If authentication is successful, the user is granted access to their account information*, mostrado na Figura 3.1.

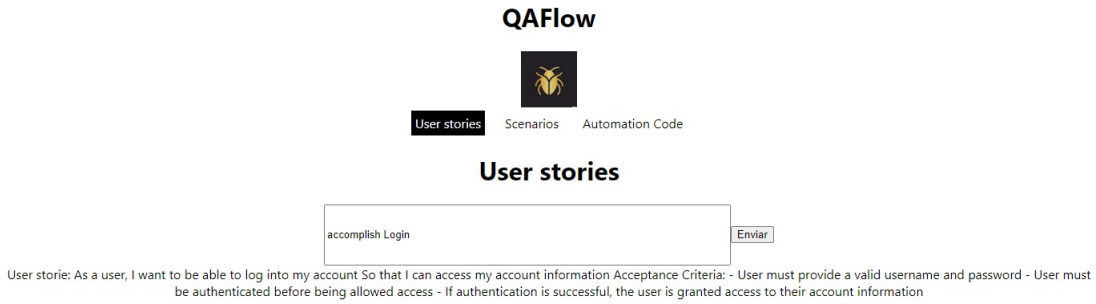

Figura 3.1: Resultado história de usuário para requisição *Accomplish Login*

Fonte: App QAFlow

Historia de usuário feita manualmente: Realizar Logout EU COMO usuário GOSTARIA DE realizar logout PARA poder me desvincular do aplicativo

Resposta do app ao digitar *Accomplish Logout*: *As a user, I want to log out of my account so that my account is secure. Acceptance Criteria: 1. The user should be able to log out of their account with a single click. 2. The user should be taken to the log in page after they log out. 3. The user should be logged out securely*, mostrado na Figura 3.2.

Historia de usuário feita manualmente: Recuperar Senha EU COMO usuário cadastrado GOSTARIA DE recuperar a minha senha. PARA recuperar o meu acesso no aplicativo Mobile.

Resposta do app ao digitar *Recover Password*: *As a user, I want to be able to recover my password So that I can access my account Acceptance Criteria: 1. The user should be able to input their email address 2. The user should be able to receive an email with instructions on how to reset their password 3. The user should be able to reset their password to a new one.*, como mostrado na Figura 3.3.

Historia de usuário feita manualmente: Cadastramento de usuário EU COMO usuário não cadastrado GOSTARIA DE realizar meu cadastro PARA poder ter acesso a funcionalidades

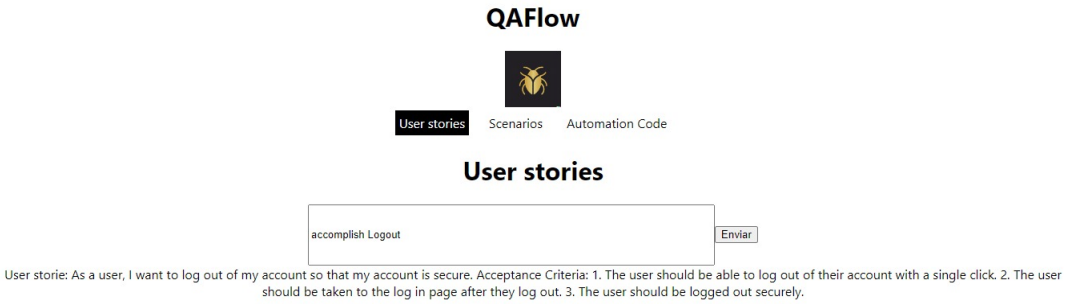

Figura 3.2: Resultado história de usuário para requisição *Accomplish Logout*

Fonte: App QAFlow

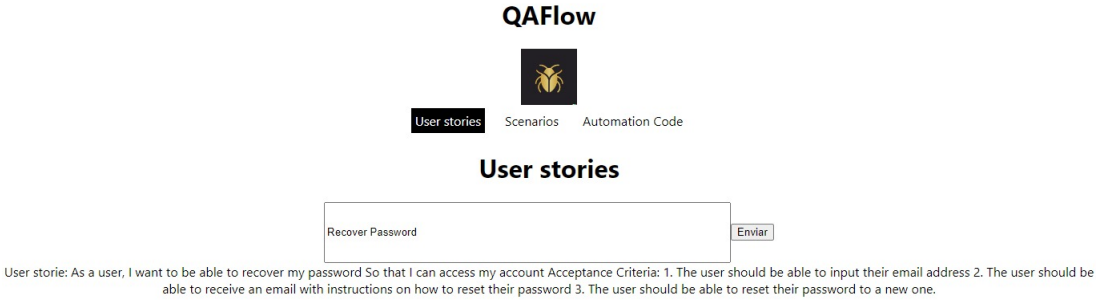

Figura 3.3: Resultado história de usuário para requisição *Recover Password*

Fonte: App QAFlow

exclusivas do aplicativo

Resposta do app ao digitar *User registration*: *As a user, I want to be able to register for an account So that I can access the website features Acceptance Criteria: -The user must provide a valid email address -The user must provide a unique username -The user must provide a secure password -The user must provide their first and last name -The user must agree to the terms and conditions of the website -The user must be able to confirm their registration via email*, como mostrado na Figura 3.4.

Historia de usuário feita manualmente: Visualizar Perfil - Mulher EU COMO mulher GOSTARIA DE visualizar meu perfil PARA poder ver minhas informações no aplicativo

Resposta do app ao digitar *View Profile - Woman*: *As a user, I want to view a profile of a woman So that I can learn more about her Acceptance Criteria: - The profile should include the woman's name - The profile should include a photo of the woman - The profile should include the woman's age - The profile should include the woman's interests - The profile should include the woman's location*, como mostrado na Figura 3.5.

Historia de usuário feita manualmente: Listar denúncias EU COMO usuário GOSTARIA DE visualizar uma lista com todas as minhas denúncias PARA poder ver minhas denúncias

Resposta do app ao digitar *List reports*: *As a user, I want to be able to list reports So that*

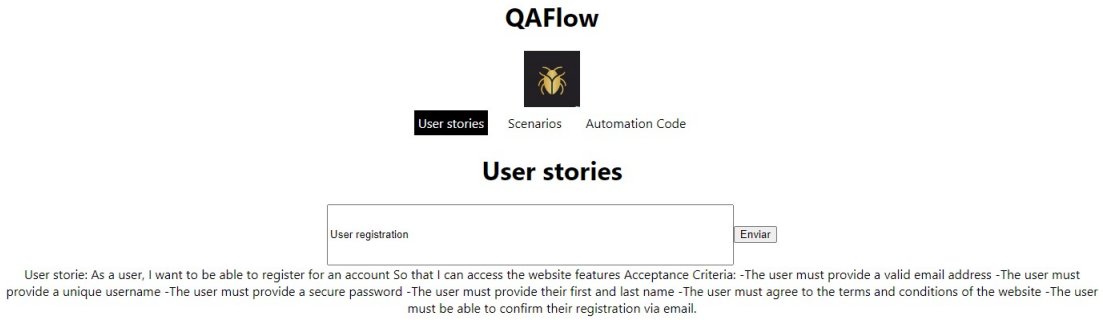

Figura 3.4: Resultado história de usuário para requisição *User registration*

Fonte: App QAFlow

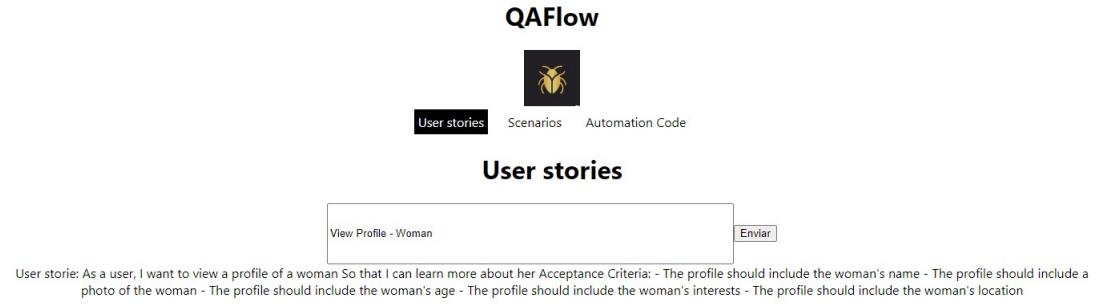

Figura 3.5: Resultado história de usuário para requisição *View Profile - Woman*

Fonte: App QAFlow

I can easily access and view the reports Acceptance criteria: - The list of reports should be in chronological order - The list of reports should be able to be filtered by date range - The list of reports should be able to be filtered by report type - The list of reports should be able to be searched by keyword - The list of reports should be able to be sorted by report type or date, como mostrado na Figura 3.6.

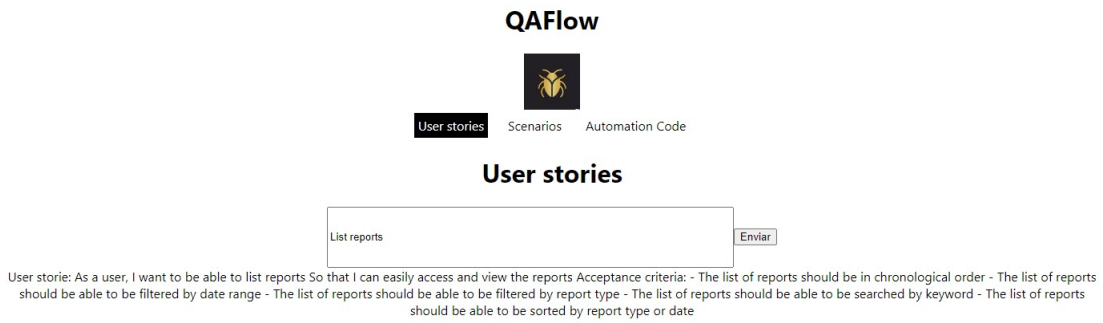

Figura 3.6: Resultado história de usuário para requisição *List reports*

Fonte: App QAFlow

Historia de usuário feita manualmente: Excluir denúncia EU COMO Mulher GOSTARIA DE que minha denúncia fosse excluida PARA que eu possa desistir de uma denúncia

Resposta do app ao digitar *Delete report*: *As a user, I want to delete a report So that I can remove outdated or irrelevant information Acceptance criteria: 1. The report should be successfully deleted from the system 2. A confirmation message should appear to confirm the deletion 3. Users should not be able to delete reports that they do not own*, como mostrado na Figura 3.7.

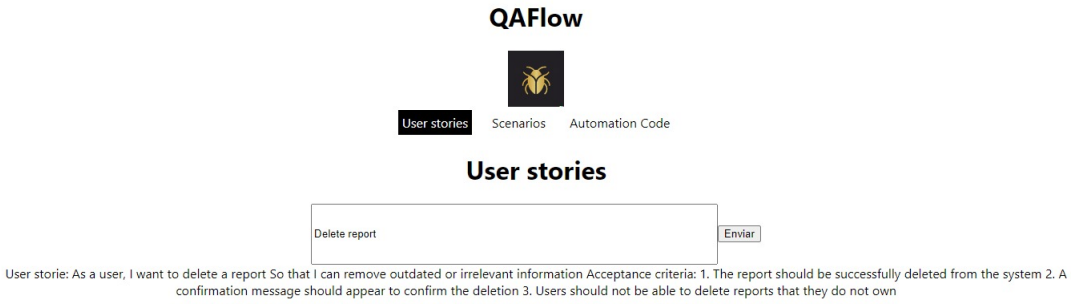

Figura 3.7: Resultado história de usuário para requisição *Delete report*

Fonte: App QAFlow

Historia de usuário feita manualmente: Visualizar mapa EU COMO usuário GOSTARIA DE visualizar o mapa PARA poder visualizar o mapa com as denúncias

Resposta do app ao digitar *View map*: *As a user, I want to view a map So that I can visualize my location and get directions. Acceptance Criteria: - The map should be interactive and easy to use - The map should be up-to-date and accurate - The map should be able to display directions from one location to another*, como mostrado na Figura 3.8.

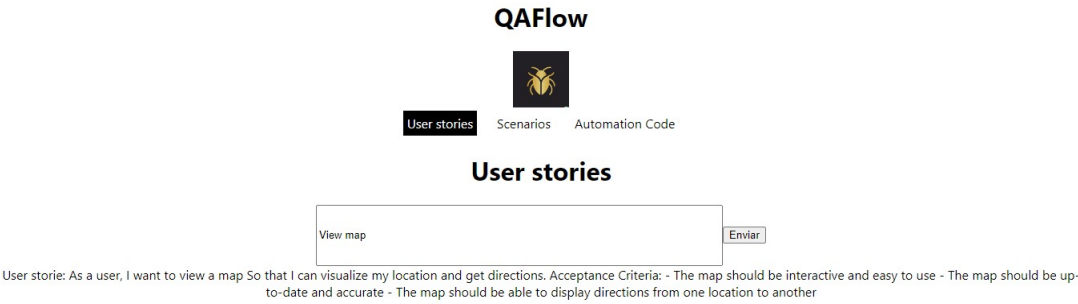

Figura 3.8: Resultado história de usuário para requisição *View map*

Fonte: App QAFlow

Historia de usuário feita manualmente: Visualizar detalhes da denúncia EU COMO usuário não cadastrado GOSTARIA DE visualizar as informações da denúncia PARA que eu possa saber mais sobre uma denúncia

Resposta do app ao digitar *View report details*: *As a user, I want to view report details So that I can make informed decisions Acceptance Criteria: - The report details should be clearly visible on the screen - The report should include all relevant information - The user should be able to access the report details quickly and easily - The user should be able to filter the report details by date, category, etc.*, como mostrado na Figura 3.9.

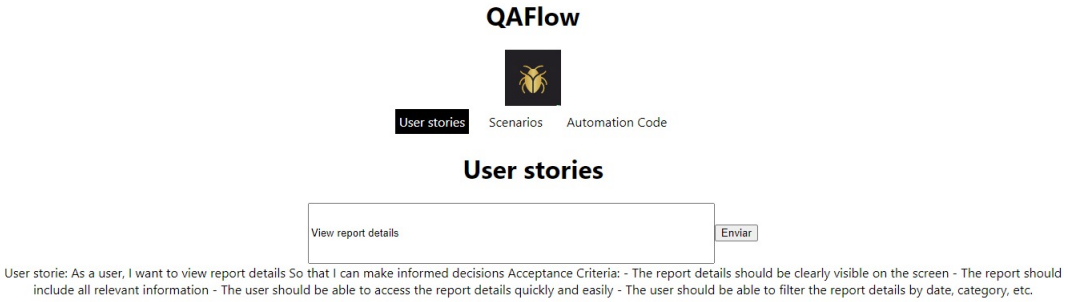

Figura 3.9: Resultado história de usuário para requisição *View report details*

Fonte: App QAFlow

Historia de usuário feita manualmente: Criar denúncia EU COMO mulher GOSTARIA DE criar uma denúncia PARA indicar meu descontentamento sobre um local

Resposta do app ao digitar *Create report*: *As a Report Analyst, I want to create a report So that I can provide a comprehensive analysis of the data Acceptance Criteria: - The report should include data from all relevant sources - The report should be clearly organized and easy to understand - The report should include all relevant information in a concise format - The report should be completed within the specified timeframe*, como mostrado na Figura 3.10.

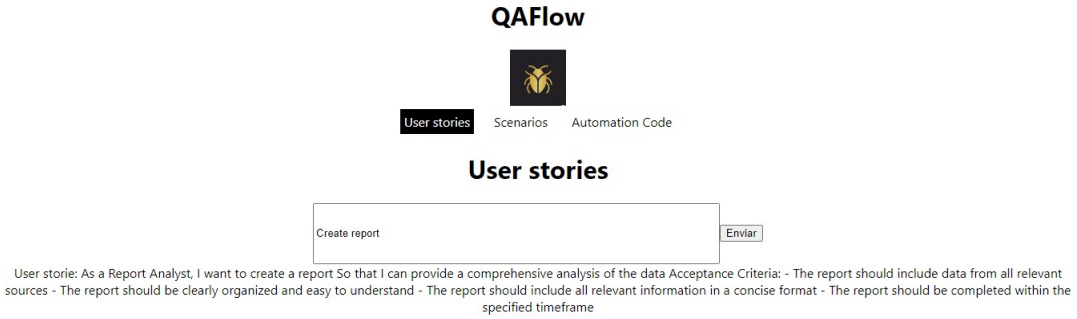

Figura 3.10: Resultado história de usuário para requisição *Create report*

Fonte: App QAFlow

Historia de usuário feita manualmente: Enviar mensagem EU COMO usuário, GOSTARIA DE enviar mensagens de texto para meus contatos PARA me comunicar com amigos e colegas de forma eficaz e instantânea.

Resposta do app ao digitar *Send Menssage*: *As a user I want to be able to send messages So that I can communicate with other users Acceptance Criteria: 1. The user should be able to*

type in the message 2. The user should be able to select the recipient of the message 3. The user should be able to send the message 4. The recipient should receive the message 5. The message should be displayed in the recipient's inbox 6. The user should be able to delete the message if needed, como mostrado na Figura 3.11.

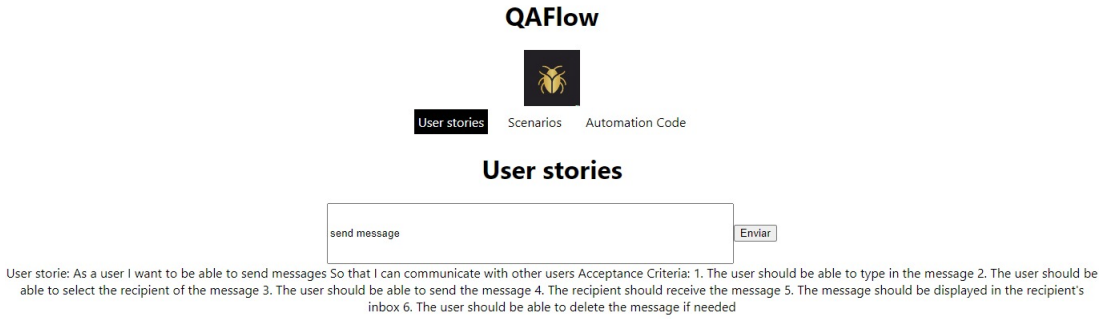

Figura 3.11: Resultado história de usuário para requisição *send message*

Fonte: App QAFlow

Historia de usuário feita manualmente Publicar status: EU COMO usuário GOSTARIA de publicar atualizações de status PARA compartilhar momentos da minha vida e interagir com minha rede de contatos.

Resposta do app ao digitar *Post Status*: *As a user, I want to be able to post my status on a social media platform So that I can share my thoughts with my friends Acceptance Criteria: - I should be able to post my status in the form of text, images, or videos - I should be able to post my status to a specific group of people - I should be able to see the number of people who have viewed my post - I should be able to delete my post if I choose to do so*, Figura 3.12..

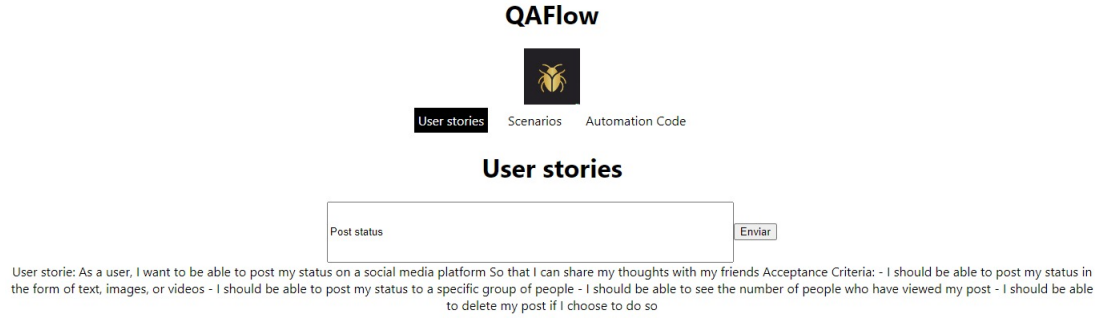

Figura 3.12: Resultado história de usuário para requisição *Post Status*

Fonte: App QAFlow

Historia de usuário feita manualmente Adicionar produtos ao carrinho de compras: EU COMO comprador, GOSTARIA DE adicionar produtos ao meu carrinho de compras, PARA selecionar os itens que desejo comprar.

Resposta do app ao digitar *Add products to shopping cart*: *As a shopper, I want to add products to my shopping cart, so that I can purchase them. Acceptance Criteria: 1. The shopper should be able to view a list of available products. 2. The shopper should be able to select the products they wish to purchase. 3. The shopper should be able to add the selected products to their shopping cart. 4. The shopping cart should display the correct quantity and total cost of the products added. 5. The shopper should be able to update the quantity and remove products from their shopping cart.*, como mostrado na Figura 3.13.

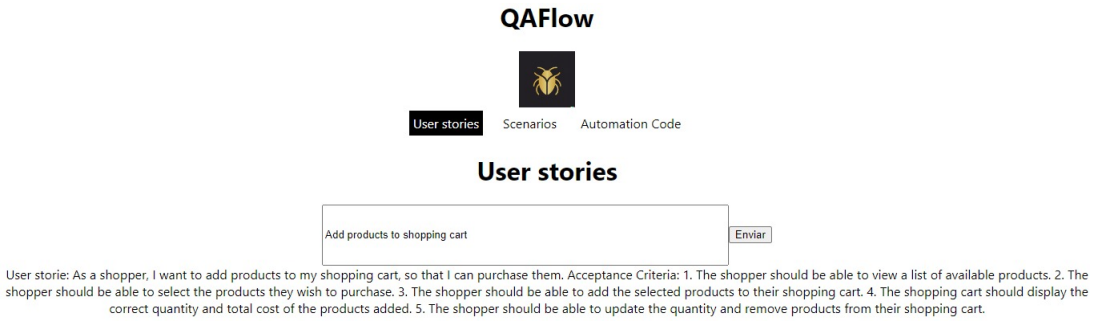

Figura 3.13: Resultado história de usuário para requisição *Add products to shopping cart*

Fonte: App QAFlow

Historia de usuário feita manualmente Categorizar minhas postagens: Eu, como criador de conteúdo, gostaria de marcar e categorizar minhas postagens, para facilitar a organização e pesquisa de meu conteúdo.

Resposta do app ao digitar *Categorize my posts*: *User storie: As a user, I want to categorize my posts So that I can easily find them later Acceptance Criteria: - I should be able to assign different categories to my posts - I should be able to view my posts by category - I should be able to filter posts by multiple categories - I should be able to edit the categories assigned to my posts*, como mostrado na Figura 3.14.

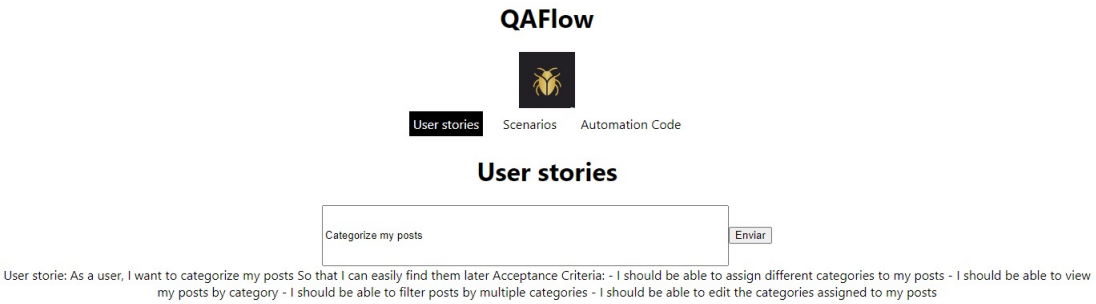

Figura 3.14: Resultado história de usuário para requisição *Categorize my posts*

Fonte: App QAFlow

Historia de usuário feita manualmente Agendar consulta Eu, como cliente, gostaria de agendar uma consulta online, para garantir um horário conveniente com um profissional de saúde.

Resposta do app ao digitar *Schedule appointment*: *As a patient, I want to schedule an appointment with my doctor So that I can receive medical care. Acceptance Criteria: - The patient should be able to view a list of available appointment times - The patient should be able to select a time for the appointment - The patient should be able to confirm the appointment - The doctor should be notified of the appointment - The appointment should be added to the doctor's calendar*, como mostrado na Figura 3.15.

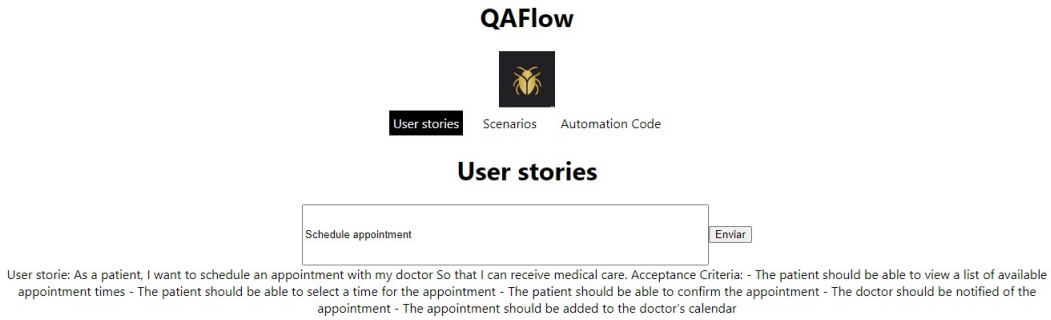

Figura 3.15: Resultado história de usuário para requisição *Schedule appointment*

Fonte: App QAFlow

Historia de usuário feita manualmente rastrear despesas Eu, como usuário, gostaria de rastrear minhas despesas e receitas, para manter controle sobre minha situação financeira e fazer orçamentos.

Resposta do app ao digitar *Track expenses*:*As a user, I want to be able to track my expenses So that I can manage my finances effectively Acceptance Criteria: - The user should be able to enter expenses into the system - The user should be able to view a summary of expenses - The user should be able to categorize expenses - The user should be able to set up budgets and track progress against them - The user should be able to view a history of expenses over time - The user should be able to set up notifications for when expenses exceed a certain amount*, como mostrado na Figura 3.16.

Historia de usuário feita manualmente Reprodução de vídeo aulas Eu, como aluno, gostaria de assistir aulas em vídeo e fazer testes online, para aprender e avaliar meu conhecimento de forma interativa.

Resposta do app ao digitar *Video lesson reproduction*: *As a student, I want to be able to reproduce video lessons So that I can learn the material more effectively Acceptance Criteria: - The video lessons must be clear and of high quality - The video lessons must have accompanying audio - The video lessons must be able to be paused and rewinded - The video lessons must be*

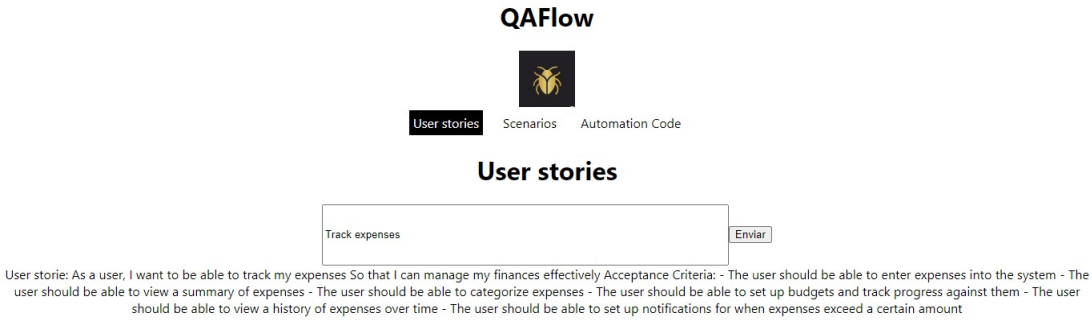

Figura 3.16: Resultado história de usuário para requisição *Track expenses*

Fonte: App QAFlow

able to be played on multiple devices - The video lessons must have subtitles for accessibility, como mostrado na Figura 3.17.

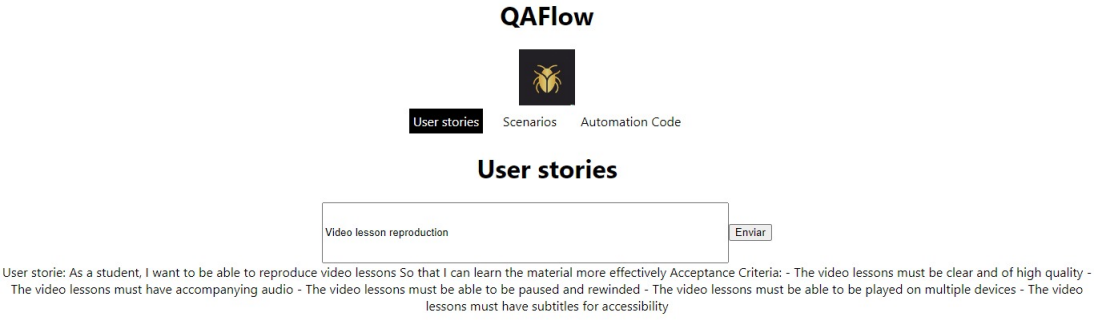

Figura 3.17: Resultado história de usuário para requisição *Video lesson reproduction*

Fonte: App QAFlow

Historia de usuário feita manualmente registrar meus exercícios Eu, como usuário, gostaria de registrar meus exercícios e dieta, para acompanhar meu progresso em direção a metas de saúde.

Resposta do app ao digitar *record my exercises*: *As a fitness enthusiast, I want to record my exercises so that I can track my progress and health. Acceptance Criteria: -I should be able to easily add new exercises to my record -I should be able to view my past exercises -I should be able to edit or delete existing exercises from my record -I should be able to view detailed information about my exercises such as duration, intensity, type, etc. -I should be able to access my exercise record from any device*, como mostrado na Figura 3.18.

Historia de usuário feita manualmente Criar tarefas do projeto eu, como gerente de projeto, gostaria de criar tarefas e atribuir responsabilidades aos membros da equipe, para gerenciar efetivamente o progresso do projeto.

Resposta do app ao digitar *Create project tasks*:*User storie: As a project manager, I want to create project tasks So that I can assign and track tasks to team members. Acceptance Criteria:*

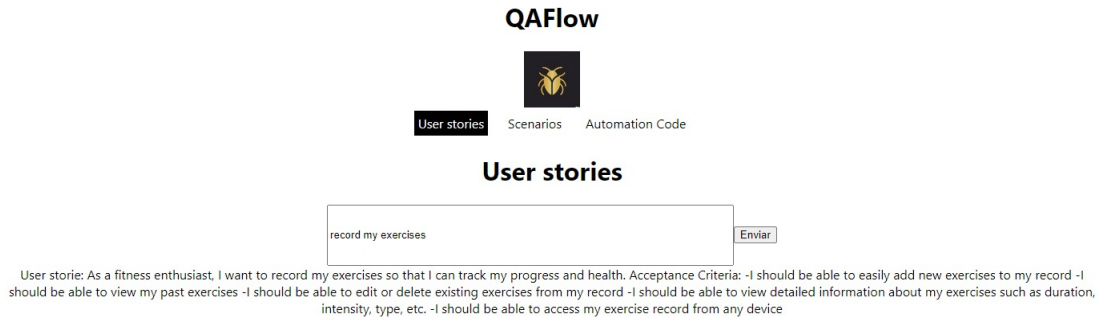

Figura 3.18: Resultado história de usuário para requisição *record my exercises*

Fonte: App QAFlow

- Tasks should be able to be assigned to multiple team members - Tasks should have a due date - Tasks should be able to be marked as completed - Tasks should be able to be edited or deleted, como mostrado na Figura 3.19.

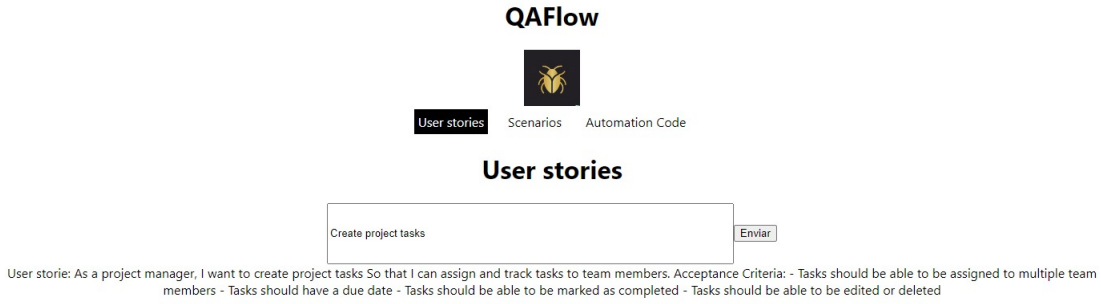

Figura 3.19: Resultado história de usuário para requisição *Create project tasks*

Fonte: App QAFlow

Historia de usuário feita manualmente aumentar fonte para acessibilidade Eu, como usuário com necessidades especiais, gostaria de aumentar o tamanho da fonte e ativar recursos de acessibilidade, para tornar o aplicativo mais utilizável para mim.

Resposta do app ao digitar *increase font for accessibility*: *As a user with accessibility needs, I want to increase the font size on the website so that I can read the content more easily. Acceptance Criteria: - The font size should be adjustable - The font should be legible and easy to read - The font size should be increased to a minimum of 16px - The font size should be adjustable across the entire website*, como mostrado na Figura 3.20.

3.6.2 Testes da geração dos cenários

Abaixo são exemplificados como foram feitos alguns dos testes da página de geração de cenários de teste BDD. Primeiramente é mostrado o cenário criado manualmente, que seria o

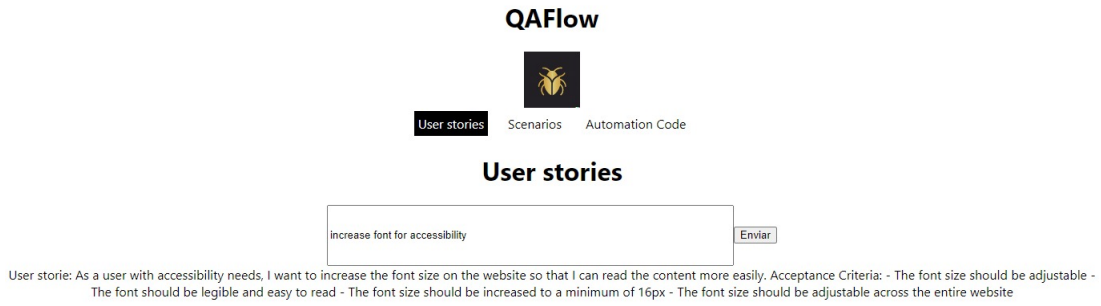

Figura 3.20: Resultado história de usuário para requisição *increase font for accessibility*

Fonte: App QAFlow

resultado esperado e depois como foi o cenário de teste criado pela aplicação "QAFlow".

Cenário de Teste Feito manualmente: Realizar Login DADO estou na tela inicial E já realizei cadastro QUANDO preencho o campo e-mail corretamente E preencho o campo senha corretamente ENTÃO serei redirecionado para a tela perfil de usuário E estarei logado

Resposta do app ao digitar *accomplish Login*: *Scenario 1: Login with valid credentials Given I am on the login page When I enter valid username and password Then I should be logged in successfully Scenario 2: Login with invalid credentials Given I am on the login page When I enter invalid username and password Then I should get an error message informing me of invalid credentials Scenario 3: Login with empty credentials Given I am on the login page When I enter no username and password Then I should get an error message informing me to enter valid credentials*, como mostrado na Figura 3.21.

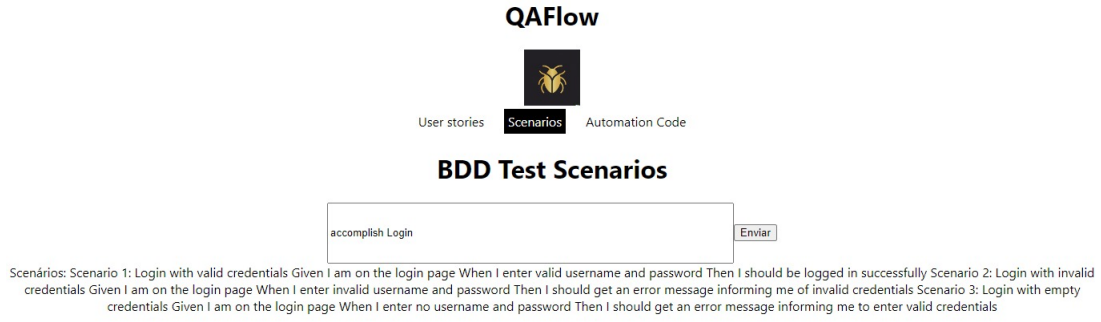

Figura 3.21: Resultado cenário de teste para requisição *accomplish Login*

Fonte: App QAFlow

Cenário de Teste Feito manualmente: Realizar Logout Cenário: Usuário logado realiza logout DADO Estou na tela inicial e logado E Clico no botão "BotaoProfileLogin"QUANDO Clico no botão "SAIR DA CONTA"E Confirmo que desejo sair da conta ENTÃO deve ser feito logout do aplicativo

Resposta do app ao digitar *accomplish Logout*: *Scenario 1: Logout using the Logout button Given a user is logged in When the user clicks the Logout button Then the user should be logged out and redirected to the login page Scenario 2: Logout using the browser's close button Given a user is logged in When the user closes the browser Then the user should be logged out and redirected to the login page Scenario 3: Logout using the browser's back button Given a user is logged in When the user clicks the browser's back button Then the user should be logged out and redirected to the login page*, como mostrado na Figura 3.22.

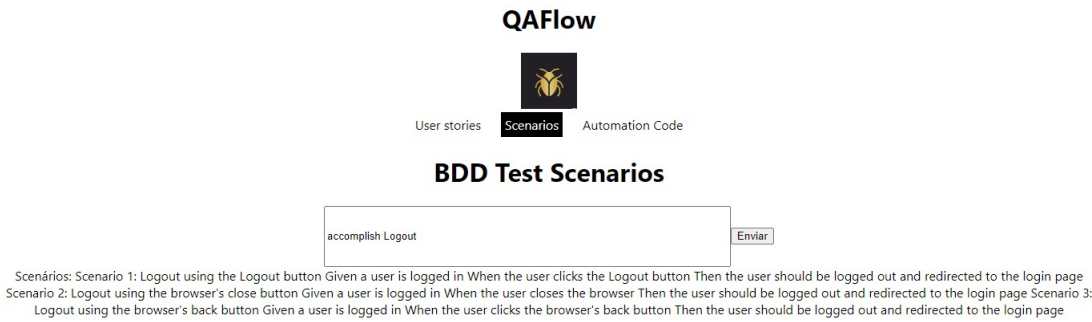

Figura 3.22: Resultado cenário de teste para requisição *accomplish Logout*

Fonte: App QAFlow

Cenário de Teste Feito manualmente: Recuperar Senha Cenário: recuperar a senha da conta do usuário no aplicativo Mobile DADO QUE: eu estou na página principal do aplicativo Mobile QUANDO: clico no botão "user-profile-button"E: clico na opção "Esqueci minha senha"E: preencho o campo "Email"com "EMAIL"E: clico no botão "ENVIAR"E: aparecer a mensagem "Redefina sua senha no email enviado para voce"ENTÃO: deverei clicar no botão escrito "OK"

Resposta do app ao digitar *Recover Password*: *Scenario 1: Recover Password by Email Given a user has an account When the user requests a password reset Then they should receive an email with a link to reset their password Scenario 2: Reset Password Given a user has an account When the user clicks the link in the password reset email Then they should be able to enter a new password Scenario 3: Password Reset Successful Given a user has an account When the user successfully resets their password Then they should be able to login with their new password*, como mostrado na Figura 3.23.

Cenário de Teste Feito manualmente: Cadastramento de usuário DADO estou na tela de login E clico no botão "cadastrar-se"QUANDO preencho todos os dados corretamente E clico em "cadastrar"ENTÃO uma nova conta será criada com os dados preenchidos

Resposta do app ao digitar *User registration*: *Scenario 1: User Registration Given a user is on the registration page When the user enters their name, email address, password and confirms the password Then the user should be successfully registered and redirected to the login page. Scenario 2: User Registration with Invalid Email Given a user is on the registration page When the user enters their name, an invalid email address, password and confirms the password*

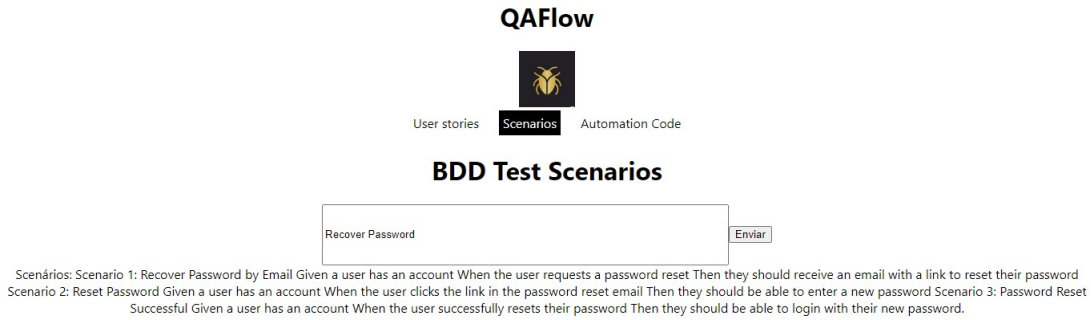

Figura 3.23: Resultado cenário de teste para requisição *Recover Password*

Fonte: App QAFlow

Then an error message should be displayed informing the user that the email address is invalid. Scenario 3: User Registration with Password Mismatch Given a user is on the registration page When the user enters their name, email address, password and confirms the password with a mismatched password Then an error message should be displayed informing the user that the passwords do not match, como mostrado na Figura 3.24.

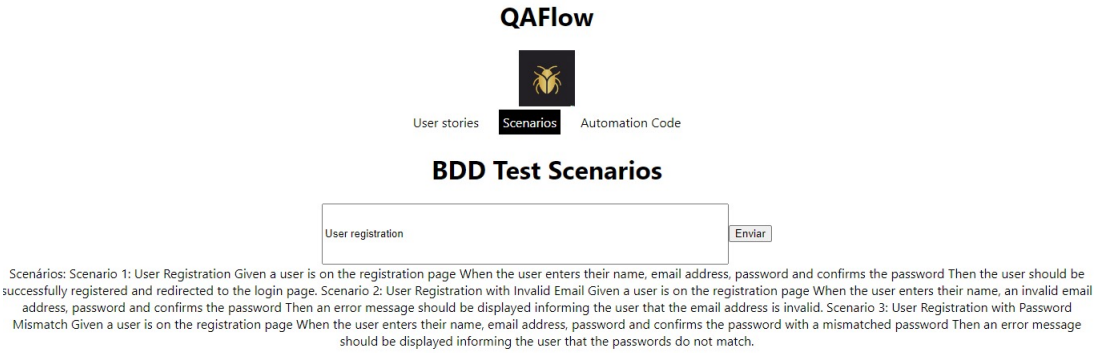

Figura 3.24: Resultado cenário de teste para requisição *User registration*

Fonte: App QAFlow

Cenário de Teste Feito manualmente: Visualizar Perfil - Mulher DADO estou logado QUANDO clico no ícone perfil ENTÃO deve ser possível visualizar meu dados

Resposta do app ao digitar *View Profile - Woman*: *Scenario 1: View Profile - Woman Given a user is on the profile page When the user selects the "Woman"option Then the profile page should display the female profile information* , como mostrado na Figura 3.25.

Cenário de Teste Feito manualmente: Listar denúncias DADO estou logado QUANDO clico no ícone de perfil ENTÃO devo visualizar uma lista com todas as minhas denúncias

Resposta do app ao digitar *List reports*: *Scenario 1: List all reports Given the user is on the main page When the user clicks on the 'Reports' tab Then all available reports should*

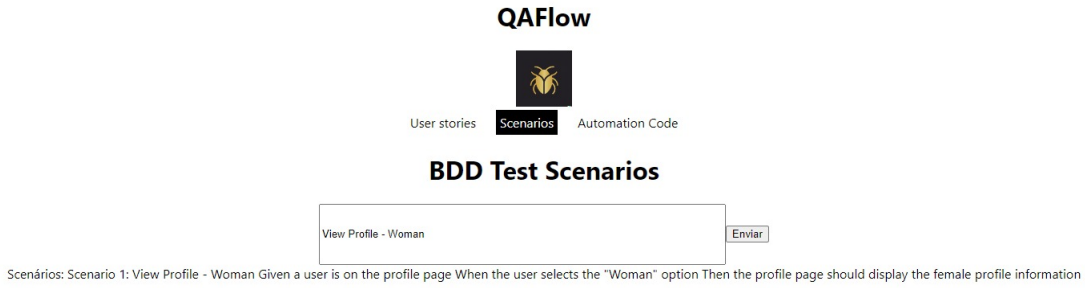

Figura 3.25: Resultado cenário de teste para requisição *View Profile - Woman*

Fonte: App QAFlow

be listed on the page Scenario 2: Filter reports Given the user is on the main page When the user clicks on the 'Reports' tab And selects a filter option Then only the reports associated with the selected filter should be listed on the page Scenario 3: View report Given the user is on the main page When the user clicks on the 'Reports' tab And clicks on a listed report Then the report should be displayed on the page, como mostrado na Figura 3.26.

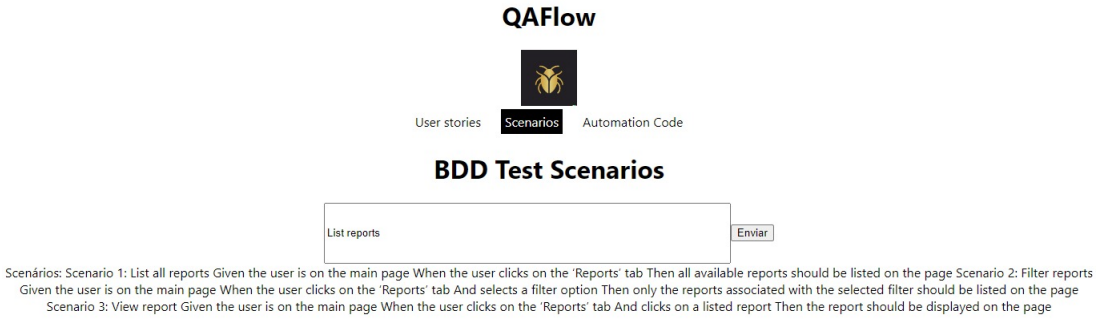

Figura 3.26: Resultado cenário de teste para requisição *List reports*

Fonte: App QAFlow

Cenário de Teste Feito manualmente: Excluir denúncia DADO Estou na tela inicial E estou logado QUANDO clico no ícone do perfil E clico no ícone de lixeira em frente a denúncia desejada ENTÃO a denúncia escolhida deve ser deletado

Resposta do app ao digitar *Delete report*: *Scenario 1: Delete report Given the user is logged into the system When they select the report to delete Then the report should be deleted from the system Scenario 2: Verify deleted report Given the user has deleted a report When they check their list of reports Then the deleted report should not be present in the list Scenario 3: Verify successful deletion message Given the user has deleted a report When they check for a successful deletion message Then the message should confirm the report was successfully deleted*, como mostrado na Figura 3.27.

Cenário de Teste Feito manualmente: Visualizar detalhes da denúncia Cenário Usuário visualiza mais detalhes sobre um local DADO estou na tela inicial (mapa do aplicativo) QUANDO

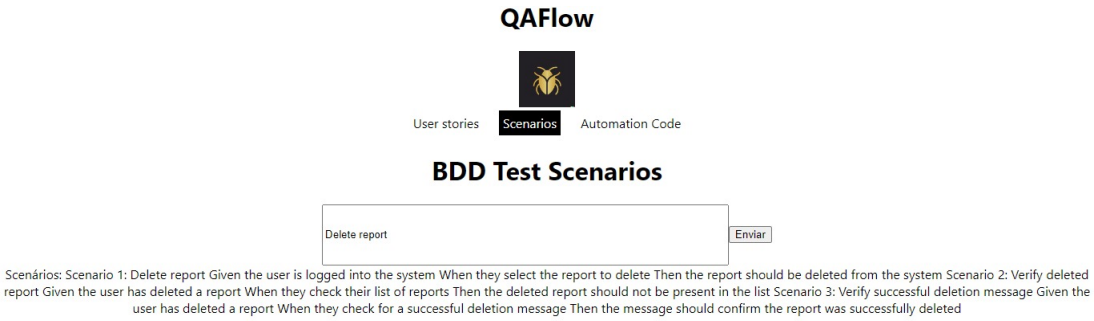

Figura 3.27: Resultado cenário de teste para requisição *Delete report*

Fonte: App QAFlow

eu clico no ícone de uma denúncia E e clico no callout que aparece ENTÃO os detalhes daquele local serão exibidos

Resposta do app ao digitar *View report details*: *Scenários: Scenario 1: View Report Details Given a user is on the Report Details page When the user clicks on the View Report button Then the Report Details page should be displayed with all the report information Scenario 2: Verify Report Details Given a user is on the Report Details page When the user views the report information Then the report details should be accurate and up to date Scenario 3: Verify Report Format Given a user is on the Report Details page When the user views the report Then the report should be displayed in the correct format*, como mostrado na Figura 3.28

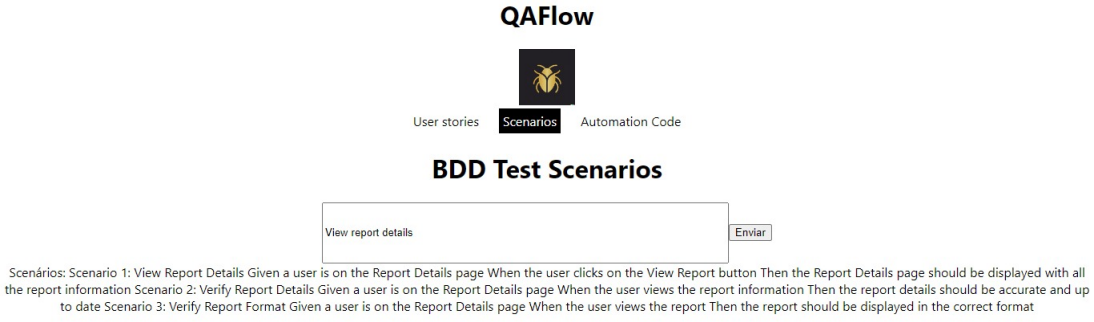

Figura 3.28: Resultado cenário de teste para requisição *View report details*

Fonte: App QAFlow

Cenário de Teste Feito manualmente: Visualizar Mapa DADO estou na tela inicial ENTÂO será possível eu visualizar o mapa com as denúncias

Resposta do app ao digitar *View map*: *Scenários: Scenario 1: Given the user is on the homepage When they click the "View Map"button Then the map should be displayed correctly Scenario 2: Given the user is on the homepage When they click the "View Map"button Then the map should be interactive Scenario 3: Given the user is on the homepage When they click the "View Map"button Then the map should be zoomable*, como mostrado na Figura 3.29.

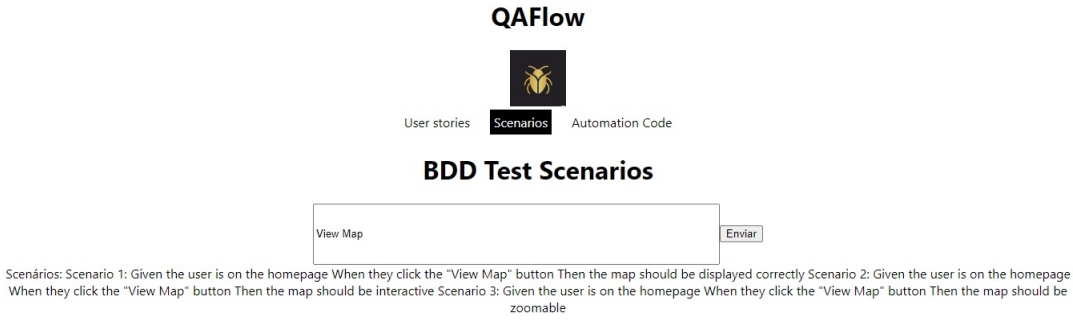

Figura 3.29: Resultado cenário de teste para requisição *View map*

Fonte: App QAFlow

Cenário de Teste Feito manualmente: Cenário: Usuário com cadastro cria denúncia com sucesso DADO Eu estou na tela do mapa E estou logado QUANDO Clico no botão "+"no canto inferior esquerdo da tela E Seleciono um local no mapa E Clico em "Proximo"E Verifico se o local desejado já possui uma denúncia E Clico no local desejado (se existir) E Clico em "Proximo"E Respondo todas as perguntas do questionário E Clico no botão "BotaoFinalizar"ENTÃO Uma nova denúncia deve ser criada

Resposta do app ao digitar *create report*: *Scenários: Scenario 1: Generating a Report Given the user has all the necessary data and access rights When the user selects the "Create Report"option Then the user should be able to generate the report in the specified format with the correct data. Scenario 2: Accessing a Report Given the user has the necessary access rights When the user selects the "View Report"option Then the user should be able to view the report in the specified format with the correct data. Scenario 3: Editing a Report Given the user has the necessary access rights When the user selects the "Edit Report"option Then the user should be able to edit the report in the specified format with the correct data*, como mostrado na Figura 3.30.

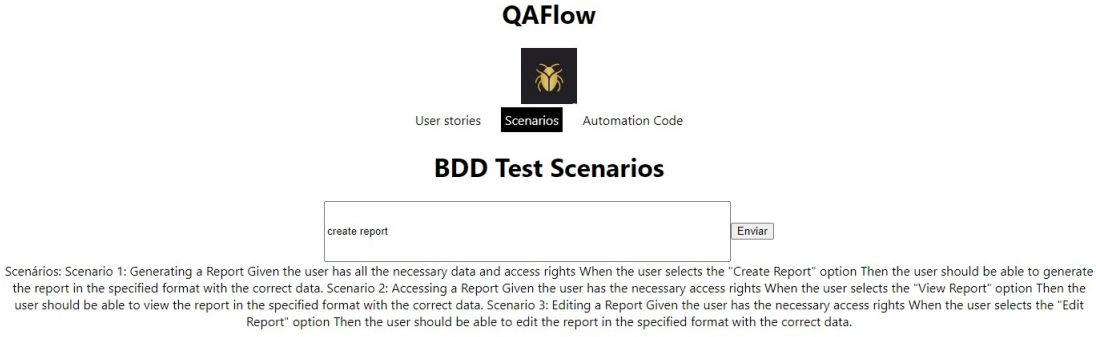

Figura 3.30: Resultado cenário de teste para requisição *create report*

Fonte: App QAFlow

Cenário de teste feito manualmente Envio de mensagem de texto

Dado que estou em uma conversa com um contato Quando digito uma mensagem de texto E clico no botão de envio Então meu contato deve receber a mensagem com sucesso

Resposta do app ao digitar *Sending a text message*: *Scenários: Scenario 1: Sending a text message Given I have a mobile phone When I compose a text message Then I should be able to send the message to the intended recipient Scenario 2: Receiving a text message Given I have a mobile phone When I receive a text message Then I should be able to view the message and respond as necessary Scenario 3: Deleting a text message Given I have a mobile phone When I select a text message Then I should be able to delete the message*, como mostrado na Figura 3.31.

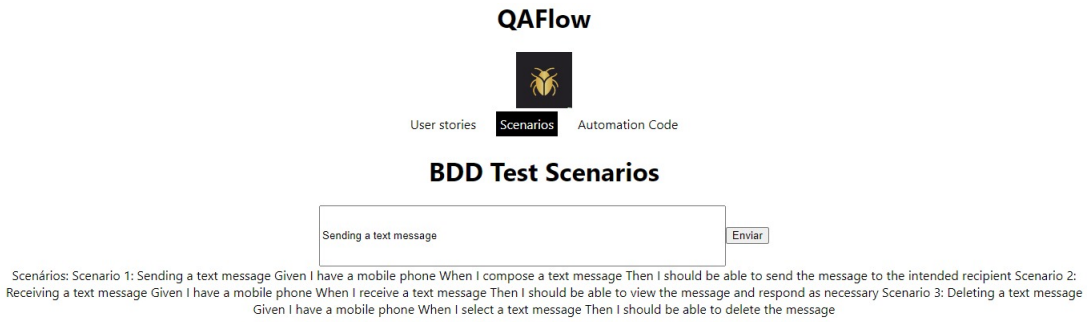

Figura 3.31: Resultado cenário de teste para requisição *Sending a text message*

Fonte: App QAFlow

Cenário de teste feito manualmente: Publicação de atualização de status Dado que estou na minha página de perfil Quando escrevo uma atualização de status E clico no botão de publicar Então minha atualização de status deve ser visível para meus amigos e seguidores

Resposta do app ao digitar *Status update post*: *Scenários: Scenario 1: Given the user is on the status update page When the user enters a post Then the post should be successfully displayed on the page Scenario 2: Given the user is on the status update page When the user does not enter a post Then an error message should be displayed Scenario 3: Given the user is on the status update page When the user enters a post with a hashtag Then the post should be successfully displayed on the page with the hashtag link*, como mostrado na Figura 3.32.

Cenário de teste feito manualmente: Adição de produto ao carrinho Dado que estou na página de detalhes de um produto Quando clico no botão "Adicionar ao carrinho"Então o produto deve ser adicionado ao meu carrinho de compras

Resposta do app ao digitar *Adding product to cart*: *Scenários: Scenario 1: Adding a product to cart Given the user is on the product page When they select a product and click "Add to Cart"Then the product is added to their shopping cart Scenario 2: Adding a product with multiple quantities Given the user is on the product page When they select a product and click "Add to Cart"multiple times Then the product is added to their shopping cart with the correct quantity Scenario 3: Adding a product with an invalid quantity Given the user is on the product page When they select a product and enter an invalid quantity and click "Add to Cart"Then an*

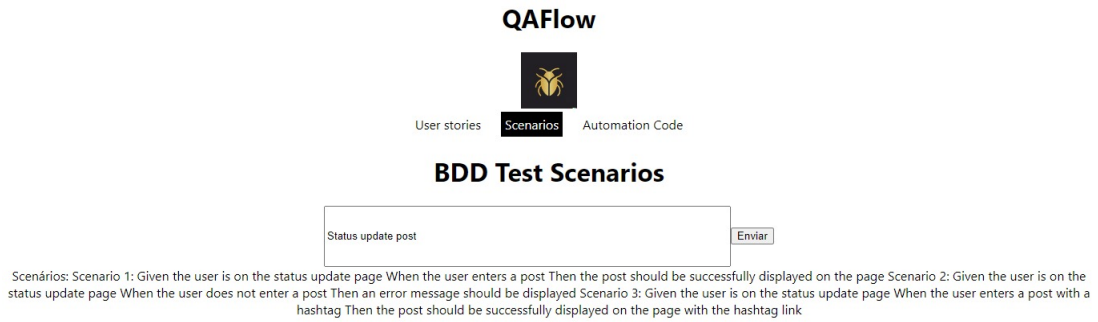

Figura 3.32: Resultado cenário de teste para requisição *Status update post*

Fonte: App QAFlow

error message is displayed and the product is not added to their shopping cart, como mostrado na Figura 3.33

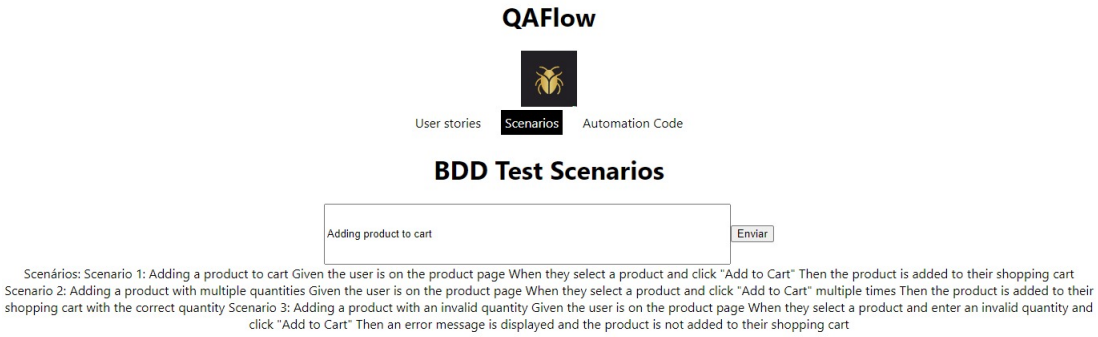

Figura 3.33: Resultado cenário de teste para requisição *Status update post*

Fonte: App QAFlow

3.6.3 Testes das automações

Abaixo são exemplificados como foram feitos alguns dos testes da página de geração de códigos de automação. Primeiramente é mostrada o código que foi criado manualmente, que seria o resultado esperado e depois como foi código gerado pela aplicação "QAFlow".

Código gerado manualmente Login

```
describe('Login bem-sucedido', () => {
  it('Deve redirecionar para a pagina de perfil apos o login', () => {
    cy.visit('sua_url_de_login');
    cy.get('input[name="username"]').type('seu_nome_de_usuario');
    cy.get('input[name="password"]').type('sua_senha');
    cy.get('button[type="submit"]').click();
    cy.url().should('include', 'pagina_de_perfil');
```

```
  });
});
```

Listing 3.1 – Código gerado manualmente Login

Código gerado pela aplicação com requisição accomplish Login

```
//This code would be used to test a login page using Cypress
describe('Login Page', () => { beforeEach(() => { cy.visit('http://
   localhost:3000/login'); });
it('should display the login page', () => {
cy.get('h1').should('contain', 'Login'); });
it('should accept a valid username and password', () => { cy.get('#
   username').type('testuser'); cy.get('#password').type('
   password123');
cy.get('#submit').click(); cy.url().should('include', '/home'); });
    it('should not accept an invalid username', () => { cy.get('#
   username').type('invaliduser'); cy.get('#password').type('
   password123');
cy.get('#submit').click(); cy.url().should('not.include', '/home');
    }); it('should not accept an invalid password', () => { cy.get(
   '#username').type('testuser'); cy.get('#password').type('
   invalidpassword'); cy.get('#submit').click(); cy.url().should('
   not.include', '/home'); }); });
```

Listing 3.2 – Código gerado pela aplicação com requisição accomplish Login

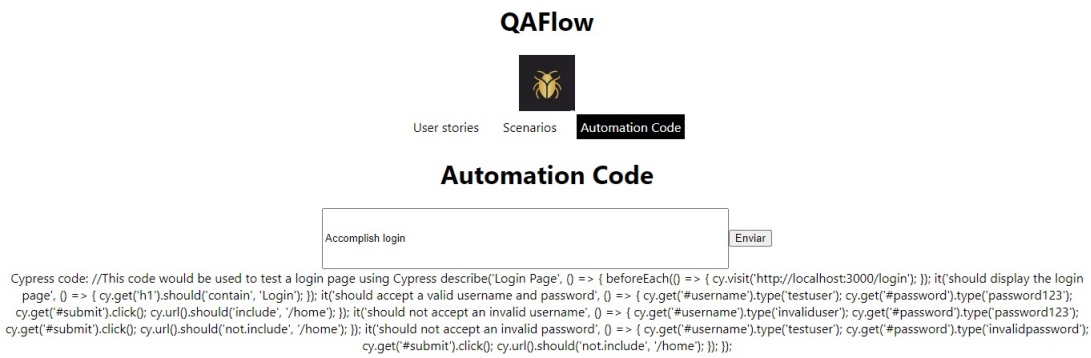

Figura 3.34: Resultado código de automação para requisição *Accomplish Login*

Fonte: App QAFlow

Código gerado manualmente Adição de produto ao carrinho

```
describe('Adicao de produto ao carrinho', () => {
  it('Deve adicionar o produto ao carrinho com sucesso', () => {
    cy.visit('URL_do_produto');
    cy.get('botao_adicionar').click();
```

```
    cy.get('carrinho').should('contain', 'Nome do Produto');
  });
});
```

Listing 3.3 – Código gerado manualmente Adição de produto ao carrinho

Código gerado pela aplicação com requisição *Adding product to cart*

```
describe('Adding product to cart', () => { beforeEach(() => {
// Visit the page where the product will be added cy.visit('http://
   example.com/products'); });
it('should add the product to the cart', () => {
// Find the product in the list
cy.get('[data-test="product-list"]') .find('[data-test="product-
   item"]').contains('Product Name') .click();
// Add the product to the cart cy.get('[data-test="add-to-cart"]').
   click(); // Check that product was added to the cart
cy.get('[data-test="cart-items"]') .should('contain', 'Product Name
   '); }); });
```

Listing 3.4 – Código gerado pelo app para *Adding product to cart*

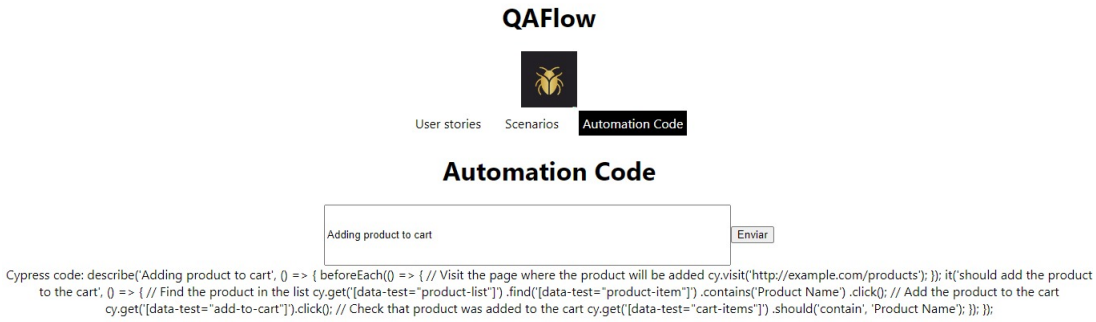

Figura 3.35: Resultado código de automação para requisição *Adding product to cart*

Fonte: App QAFlow

Código gerado manualmente Publicação de atualização de status

```
describe('Publica atualizacaoo de status', () => {
  it('Deve publicar a atualizacao de status com sucesso', () => {
    cy.visit('URL_da_pagina_de_perfil');
    cy.get('textarea[name="atualizacao_de_status"]').type('Minha-
       nova atualizacao-de-status');
    cy.get('button[name="botao_publicar"]').click();
    cy.get('.atualizacao-de-status').should('contain', 'Minha-nova
       atualizacao-de-status');
  });
```

```
8 });
```

Listing 3.5 – Código gerado manualmente atualização de status

Código gerado pela aplicação com requisição *Status update post*

```
1 // Here is an example code in JavaScript for test automation using
     Cypress:
2 describe('Status Update Post', function() {
3 it('should be able to post a status update', function() {
4 cy.visit('/home');
5 cy.get('#status-update-button').click();
6 cy.get('#status-update-textarea').type('This is a status update
     post!'); cy.get('#status-update-submit-button').click();
7 cy.get('#status-update-confirmation').should('contain', 'Status
     update posted!'); }); });
```

Listing 3.6 – Código gerado pelo app requisição *Status update post*

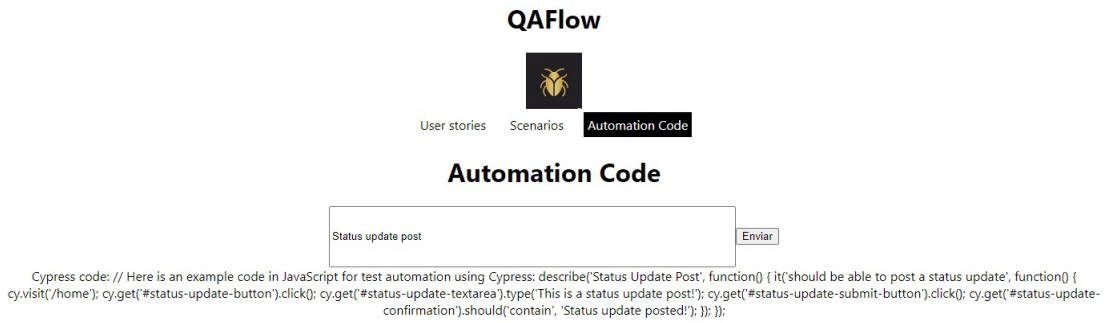

Figura 3.36: Resultado código de automação para requisição *Status update post*

Fonte: App QAFlow

3.7 Otimização e Melhorias

Após a conclusão das funcionalidades principais do aplicativo, foi dada atenção a otimização e aprimoramento do desempenho geral do QAFlow. Foram identificadas as áreas que poderiam se beneficiar de melhorias de eficiência e desempenho, trazendo ajustes a lógica do *backend* e aprimoramentos para a chamada à API. Essas otimizações foram essenciais para garantir tempos de resposta rápidos e proporcionar uma experiência de usuário suave e ágil.

O desenvolvimento do QAFlow, que utiliza a API do modelo text-davinci-003 para gerar histórias de usuário, cenários de teste e automações com base em descrições de funcionalidades, representou um desafio complexo de engenharia de software. Foi necessária uma abordagem minuciosa para integrar a lógica do *frontend* com a API e a formatação das saídas de forma eficiente.

Portanto, o resultado final é uma aplicação altamente eficaz e inovadora que capacita os usuários a criar, planejar e visualizar projetos de desenvolvimento de software com maior facilidade e eficiência. As otimizações implementadas permitiram que o QAFlow oferecesse resultados rápidos e precisos, atendendo às demandas dos profissionais de qualidade de software

4 Resultados

Neste capítulo são apresentados, interpretados e analisados os resultados alcançados com o desenvolvimento do aplicativo QAFlow. A análise será conduzida de maneira a evidenciar o cumprimento dos objetivos específicos estabelecidos no trabalho. Além disso, quando possível, será realizada uma comparação com os resultados encontrados na literatura, ressaltando a relevância da pesquisa realizada no contexto acadêmico.

4.1 Resultados do Aplicativo QAFlow

O desenvolvimento e implementação do aplicativo QAFlow resultaram em conquistas notáveis, validando sua eficácia e utilidade para a geração de histórias de usuário, cenários de teste e códigos de automação. O aplicativo demonstrou alta precisão e padronização na geração das saídas, superando os métodos tradicionais de criação manual. A seguir, destacamos os principais resultados obtidos:

4.1.1 Geração Precisa de Histórias de Usuário, Cenários de Teste e Automação

O QAFlow revelou sua capacidade excepcional para gerar não apenas histórias de usuário e cenários de teste precisos, mas também para criar código de automação de maneira coerente e relevante. As saídas geradas pelo aplicativo refletiram de forma precisa as informações de entrada fornecidas pelos usuários, abrangendo múltiplas dimensões do processo de desenvolvimento de software. A figura 4.1 mostra um exemplo da geração de história de usuário.

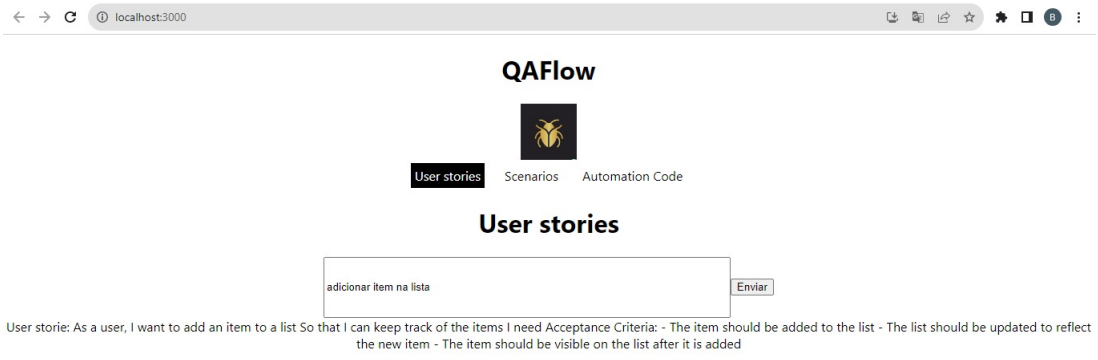

Figura 4.1: Geração História de usuário

Fonte: QAFlow

A habilidade do QAFlow em gerar código de automação adiciona um nível adicional de valor ao aplicativo, permitindo a criação automatizada de casos de teste funcionais e cenários de execução. Isso não apenas acelera o processo de teste, mas também melhora a consistência e a confiabilidade dos testes realizados. A geração automatizada de código de automação abre novas perspectivas para a eficiência e qualidade no ciclo de desenvolvimento de software. A figura 4.2 exemplifica a geração de cenários de teste.

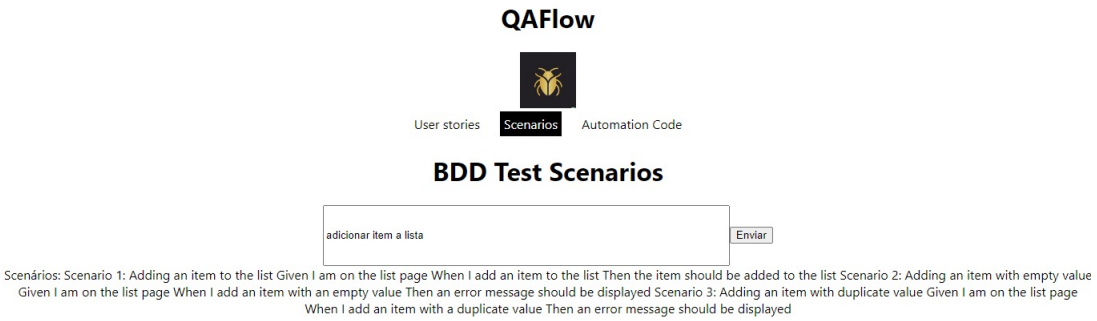

Figura 4.2: Geração Cenário de teste BDD

Fonte: QAFlow

O resultado positivo da geração de histórias de usuário, cenários de teste e código de automação reforça a robustez e versatilidade do QAFlow como uma ferramenta abrangente para planejamento e execução de projetos de desenvolvimento de software. A capacidade de gerar múltiplos tipos de saída com alta precisão evidencia o potencial do aplicativo em diversas etapas do ciclo de desenvolvimento e do processo de teste. Como mostrado ao gerar o código para automação no exemplo da Figura 4.3.

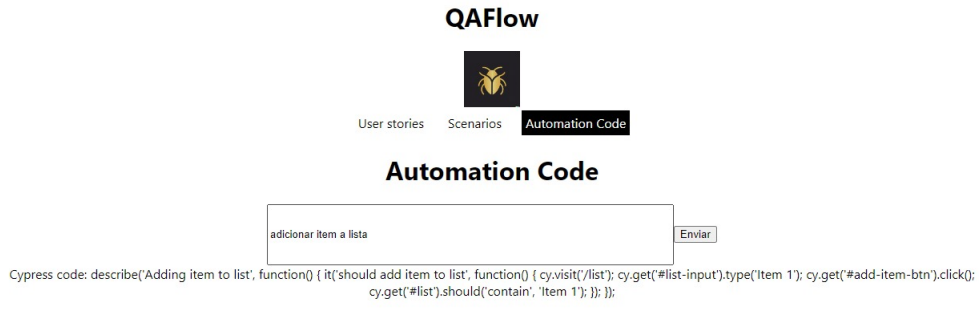

Figura 4.3: Geração código automação

Fonte: QAFlow

4.1.2 Eficiência em Relação a Métodos Tradicionais

A análise comparativa entre o QAFlow e os métodos tradicionais de criação de histórias de usuário e cenários de teste demonstrou uma notável eficiência do aplicativo. Enquanto os métodos manuais exigem um tempo considerável e esforço humano, o QAFlow reduziu substancialmente o tempo necessário para gerar saídas padronizadas e coesas. Isso representa um ganho significativo de produtividade para equipes de desenvolvimento.

5 Considerações Finais

Neste capítulo é analisado se todos os objetivos descritos na introdução foram atingidos, além de ressaltar a contribuição do trabalho para o meio acadêmico. Os resultados obtidos e um fechamento de todo trabalho desenvolvido são apresentados de forma sucinta.

5.1 Conclusão

Em resumo, esta seção proporciona uma visão abrangente das conclusões alcançadas no decorrer deste trabalho. Foram recapitulados e avaliados os objetivos específicos delineados previamente, bem como os resultados obtidos.

O QAFlow foi concebido com a visão de se tornar uma ferramenta inovadora para auxiliar no processo de geração de histórias de usuário, cenários de teste e automações no contexto de desenvolvimento de software. As etapas de definição de requisitos, escolha da API do modelo de IA Davinci 003, design de interface de usuário, desenvolvimento de backend e integração da API foram realizadas. A metodologia adotada revelou-se eficaz para viabilizar a implementação desse aplicativo que busca simplificar e otimizar o trabalho de profissionais da área de qualidade de software.

O desenvolvimento do QAFlow proporcionou insights valiosos sobre a aplicação de modelos de IA, como o Davinci 003, na automação de tarefas relacionadas à criação de artefatos de teste em projetos de desenvolvimento de software. A integração bem-sucedida da API e a geração de histórias de usuário, cenários de teste e automações ilustram a promissora aplicabilidade dessa abordagem para melhorar a eficiência e padronização do processo de qualidade de software.

Com base nas realizações deste trabalho e nas lições aprendidas, recomenda-se que esforços futuros se concentrem na etapa de lançamento e avaliação prática do QAFlow com uma amostra representativa de usuários. Essa validação prática forneceria insights fundamentais para refinar e ajustar a aplicação, aprimorando sua usabilidade e eficácia em cenários reais.

Em última análise, o trabalho realizado representa um passo importante em direção ao desenvolvimento de uma ferramenta que pode ter um impacto positivo significativo no campo da qualidade de software.

O aplicativo QAFlow é uma ferramenta inovadora que tem o potencial de revolucionar o processo de teste de software. O aplicativo é capaz de automatizar e aperfeiçoar a geração de histórias de usuário e cenários de teste, trazendo benefícios significativos para profissionais de qualidade de software e contribuindo para a melhoria da qualidade dos sistemas de software desenvolvidos.

5.2 Trabalhos Futuros

As propostas de continuidade relacionados ao aplicativo QAFlow incluem:

1. Lançamento do aplicativo para um grupo limitado de usuários iniciais. Reaizar coleta de *feedback* para entender como o aplicativo é utilizado e identificar possíveis melhorias adicionais.

2. Com base em *feedback* dos usuários, implementar atualizações regulares para aprimorar a experiência e corrigir possíveis problemas. Mantendo uma abordagem iterativa, continuando a monitorar o desempenho e a usabilidade do aplicativo após o lançamento.

3. O desenvolvimento de novos recursos e funcionalidades, como a integração com outras ferramentas de teste de software e de gestão de projetos.

4. A avaliação da eficácia e usabilidade do aplicativo em diferentes projetos de desenvolvimento de software.

5. Fazer uma comparação da eficiência do "QAFlow"ao utilizar outros modelos de linguagem grande escala.

6. A publicação de artigos científicos e a apresentação de palestras sobre o aplicativo QAFlow em conferências e eventos relacionados à qualidade de software.

7. Procurar maneiras de monetizar a ferramenta, como solicitar pagamentos mensais aos usuários que utilizarem a ferramenta em seu ambiente de trabalho.

O aplicativo QAFlow tem o potencial de se tornar uma ferramenta essencial para profissionais de qualidade de software em todo o mundo.

Referências

ABDULLAHI, S.; ZAKARI, A.; ABDU, H.; NURA, A.; ZAYYAD, M. A.; SULEIMAN, S.; ADAMU, A.; MASHASHA, A. S. Software testing: Review on tools, techniques and challenges. *International Journal of Advanced Research in Technology and Innovation*, v. 2, n. 2, p. 11–18, 2020.

ADZIC, G. *Specification by example: how successful teams deliver the right software*. [S.l.]: Simon and Schuster, 2011.

COHN, M. *User stories applied: For agile software development*. [S.l.]: Addison-Wesley Professional, 2004.

CORRÊA, D. R.; LAMIM, G. B.; SOUZA, P. de C.; LIMA, L.; JUNIOR, M. M. Contando histórias: desenvolvendo um aplicativo móvel com chatgpt para aumentar o hábito de leitura em jovens. 2023.

HOU, X.; ZHAO, Y.; LIU, Y.; YANG, Z.; WANG, K.; LI, L.; LUO, X.; LO, D.; GRUNDY, J.; WANG, H. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620*, 2023.

HOURANI, H.; HAMMAD, A.; LAFI, M. The impact of artificial intelligence on software testing. In: IEEE. *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. [S.l.], 2019. p. 565–570.

KHAN, M. E.; KHAN, F. A comparative study of white box, black box and grey box testing techniques. *International Journal of Advanced Computer Science and Applications*, Citeseer, v. 3, n. 6, 2012.

KUDRYASHOV, K. The beginner's guide to bdd. *Dan North Q & A. https://inviqa.com/blog/bdd-guide*, 2015.

LIMA, R.; CRUZ, A. M. R. da; RIBEIRO, J. Artificial intelligence applied to software testing: A literature review. In: IEEE. *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2020. p. 1–6.

MYERS, G. J.; SANDLER, C.; BADGETT, T. *The art of software testing*. [S.l.]: John Wiley & Sons, 2011.

PRESSMMAN, R. *Engenharia de software*. [S.l.]: Porto Alegre: AMGH, 2011.

RAWAT, P.; MAHAJAN, A. N. Reactjs: A modern web development framework. *International Journal of Innovative Science and Research Technology*, v. 5, n. 11, p. 698–702, 2020.

SILVA, C. A. A. d. Desenvolvimento de assistente jurídico inteligente utilizando o modelo gpt-3. Universidade Federal da Bahia, 2023.

SMART, J. F.; MOLAK, J. *BDD in Action: Behavior-driven development for the whole software lifecycle*. [S.l.]: Simon and Schuster, 2023.

SOLIS, C.; WANG, X. A study of the characteristics of behaviour driven development. In: IEEE. *2011 37th EUROMICRO conference on software engineering and advanced applications*. [S.l.], 2011. p. 383–387.

SOMMERVILLE, I. et al. Engenharia de software.[sl]. *Pearson Education*, v. 19, p. 60, 2011.

TAMKIN, A.; BRUNDAGE, M.; CLARK, J.; GANGULI, D. Understanding the capabilities, limitations, and societal impact of large language models. *arXiv preprint arXiv:2102.02503*, 2021.

ZHONG, W.; CUI, R.; GUO, Y.; LIANG, Y.; LU, S.; WANG, Y.; SAIED, A.; CHEN, W.; DUAN, N. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*, 2023.

ZIEGLER, D. M.; STIENNON, N.; WU, J.; BROWN, T. B.; RADFORD, A.; AMODEI, D.; CHRISTIANO, P.; IRVING, G. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Apêndices

APÊNDICE A – Principais trechos de código da aplicação

Código fonte *backend*

```
1
2  // Importacao das bibliotecas necessarias
3  const express = require('express');
4  const app = express();
5  app.use(express.json()); // Habilita o uso de JSON no corpo das
       solicitacoes
6  app.listen(3333); // Inicia o servidor na porta 3333
7
8  // Habilita o CORS (Cross-Origin Resource Sharing) para permitir
       solicitacoes de diferentes origens
9  const cors = require('cors');
10 app.use(cors());
11 app.options('*', cors());
12
13 // Configuracao da API OpenAI
14 const { Configuration, OpenAIApi } = require('openai');
15 const config = new Configuration({
16     apiKey: 'minha_chave_secreta_da_API',
17 });
18 const openai = new OpenAIApi(config);
19
20 // Rota para lidar com solicitacoes POST
21 app.post('/api/call', async (req, res) => {
22
23     // Funcao para executar a chamada para API da OpenAI
24     const runPrompt = async () => {
25         const response = await openai.createCompletion({
26             model: 'text-davinci-003',
27             prompt: req.body.prompt, // A entrada e fornecida no
                   corpo da solicitacao
28             max_tokens: 2000,
29             temperature: 0.5,
30         });
31         return response.data;
```

```
32      };
33
34      // Executa a funcao para obter a resposta da API
35      const responseFromAPI = await runPrompt();
36
37      console.log(responseFromAPI);
38
39      // Envia a resposta da API de volta como resposta para a
            solicitacao
40      res.send(responseFromAPI.choices[0].text);
41  });
```

Listing A.1 – Código fonte *backend*

Pré texto para geração de histórias de usuários

```
1   return (
2     <div>
3       <h1>User stories</h1>
4
5       {/* API call */}
6       <div>
7         <input
8           type="text"
9           name=""
10          id=""
11          placeholder="Descreva a funcionalidade..."
12          style={{ width: "500px", height: "70px" }}
13
14          // Esta entrada permite ao usuario descrever a
                funcionalidade desejada.
15          // A descricao inserida aqui adicionada ao texto pre
                definido sera usada como prompt para a chamada a API
                Davinci 003.
16          // A funcao onChange e usada para atualizar a variavel de
                estado 'prompt' com a descricao inserida.
17          onChange={(e) =>
18            setPrompt(`
19              Com esta descricao do problema: ${e.target.value}
20              Crie uma historia de usuario BDD com criterios de
                    aceitacao.
21              Exemplo: Como um(a) ...
22                       Eu quero...
23                       Para que...
```

```
                    Criterios de aceitacao...
          `)}
      />
      <button onClick={handleClick}>Enviar</button>

      {/* Esta secao do codigo exibe um botao "Enviar" que o
         usuario pode clicar para iniciar o processo de geracao de
         historias de usuario. */}

      <div>
        Historia de usuario: {result}
      </div>

      {/* Aqui, a historia de usuario gerada com base na descricao
         fornecida pelo usuario e exibida. */}
    </div>
  </div>
);
```

Listing A.2 – Código fonte *frontend* com pré texto para gerar histórias

Pré texto para geração de cenários de teste

```

return (
  <div>
    <h1>BDD Test Scenarios</h1>

    {/* API call */}
    <div>
      <input
        type="text"
        name=""
        id=""
        placeholder="Descreva a funcionalidade..."
        style={{ width: "500px", height: "70px" }}

        // Esta entrada permite ao usuario descrever a
           funcionalidade desejada.
        // A descricao inserida aqui somada com o texto pre
           definido sera usada como prompt para a chamada a API
           Davinci 003.
        // A funcao onChange e usada para atualizar a variavel de
           estado 'prompt' com a descricao inserida.
```

```
        onChange={(e) =>
          setPrompt(`
            Com esta descricao do problema: ${e.target.value}
            Crie cenarios BDD para cada caso de teste
            Exemplo: Cenario1, Cenario2, Cenario3
                     Dado...
                     Quando...
                     Entao...
          `)}
      />
      <button onClick={handleClick}>Enviar</button>

      {/* Esta secao do codigo exibe um botao "Enviar" que o
         usuario pode clicar para iniciar o processo de geracao de
         cenarios BDD. */}

      <div>
        Cenarios: {result}
      </div>

      {/* Aqui, os cenarios BDD gerados com base na descricao
         fornecida pelo usuario sao exibidos. */}
    </div>
  </div>
);
```

Listing A.3 – Código fonte *frontend* com pré texto para gerar cenários

Pré texto para geração de códigos de automação

```

return (
  <div>
    <h1>Automation Code</h1>

    {/* Esta secao do codigo define o titulo "Automation Code" na
       pagina. */}

    {/*chama api */}
    <div>
      <input
        type="text"
        name=""
        id=""
```

```
14          placeholder="Descreva a funcionalidade..."
15          style={{ width: "500px", height: "70px" }}
16
17          // Esta entrada permite ao usuario descrever a
               funcionalidade desejada.
18          // A descricao inserida aqui adicionada com o texto
               predefinido sera usada como prompt para a chamada a API
               Davinci 003.
19          // A funcao onChange e usada para atualizar a variavel de
               estado 'prompt' com a descricao inserida.
20          onChange={(e) =>
21            setPrompt(`Com esta descricao do problema:  ${e.target.
                 value}
22              e com base nos possiveis casos de teste, crie um codigo
                     em JavaScript para automacao de teste usando o
                   Cypress`)
23          }
24        />
25        <button onClick={handleClick}>Enviar</button>
26
27        {/* Esta secao do codigo exibe um botao "Enviar" que o
             usuario pode clicar para iniciar o processo de geracao do
             codigo de automacao. */}
28
29        <div>
30          Cypress code: {result}
31        </div>
32
33        {/* Aqui, o codigo de automacao gerado com base na descricao
             fornecida pelo usuario e exibido. */}
34      </div>
35    </div>
36  );
```

Listing A.4 – Código fonte *frontend* com pré texto para gerar códigos automatizados