Nome: Nilton Dionisio Guerra RA: 1131392213031

mecanismos de manutenção do sistema

Tendo em vista a complexidade do sistema e tempo disponível foi devido que será utilizado as seguintes estratégias, abordagens e tipos de testes que iremos discorrer abaixo.

- Testes de caixa preta

Para o desenvolvimento do projeto, a implementação de testes de integração e de testes de caixa preta será de fundamental importância. Dado que o sistema em questão é de médio porte, é indispensável a realização de testes que verifiquem a integração entre os módulos que serão implementados ao longo do desenvolvimento. Além disso, a adoção de testes de caixa preta se mostra essencial para garantir a verificação da funcionalidade do sistema do ponto de vista do usuário final.

Os testes de caixa preta que serão realizados têm como objetivo principal validar a usabilidade, simplicidade, clareza do sistema, além de identificar possíveis erros. Para isso, diferentes indivíduos executarão esses testes, garantindo que a avaliação seja abrangente e imparcial. No contexto deste projeto, serão aplicadas várias técnicas específicas de teste, cada uma com um foco distinto, para assegurar a qualidade do sistema.

Primeiramente, o teste de caminho será utilizado para verificar se os fluxos de execução dentro do sistema funcionam conforme esperado, garantindo que todas as rotas possíveis no código sejam testadas e que não haja falhas. Em seguida, o teste de tabela de decisão será implementado para analisar cenários de múltiplas condições, assegurando que todas as combinações de entradas resultem nas saídas corretas.

Outro método importante é o teste de transição de estados, que avaliará como o sistema se comporta ao mudar de um estado para outro, verificando a consistência e a estabilidade durante essas transições. Além disso, o teste de valor limite será empregado para identificar erros em limites extremos de entradas, enquanto o teste

de partição de equivalência ajudará a reduzir o número de casos de teste ao agrupar entradas que devem ser tratadas de forma semelhante pelo sistema.

O uso dessas técnicas visa não apenas detectar falhas, mas também aprimorar a segurança e a resiliência do código. A implementação desses testes contribuirá significativamente para uma jornada do usuário mais fluida e satisfatória, assegurando que o sistema atenda aos padrões de qualidade esperados e que funcione corretamente em diversos cenários de uso.

- Testes de integração

Os testes de integração serão conduzidos principalmente no sistema de backend, utilizando uma abordagem de integração bottom-up. Para a realização desses testes, optarei por usar o framework Nest.js, que é baseado no Express.js, construído em Node.js, o qual, por sua vez, utiliza JavaScript. Devido à escolha do JavaScript como a linguagem principal no backend, empregarei as bibliotecas mais populares e eficazes para testes nesse ambiente, a fim de garantir a robustez e a confiabilidade do sistema.

A biblioteca Jest será a principal ferramenta para a execução dos testes de integração, devido à sua ampla aceitação na comunidade JavaScript e sua capacidade de lidar com diversos tipos de testes de forma eficiente. Além do Jest, utilizarei o Supertest, que facilita a simulação de requisições HTTP e a verificação das respostas das APIs, o que é essencial para garantir que os serviços do backend estejam funcionando corretamente quando integrados. Para complementar, a biblioteca Mocha será empregada, especialmente para simular respostas de API de maneira simplificada e flexível, tornando o processo de teste mais ágil e eficaz.

O principal objetivo desses testes de integração é melhorar a segurança do sistema, minimizando a quantidade de problemas que possam surgir durante a integração dos módulos e garantindo que o código seja de alta qualidade. Além disso, a implementação desses testes permitirá uma melhor inserção de outras pessoas no projeto, caso seja necessário no futuro, facilitando a manutenção e a escalabilidade do sistema.

- Testes de caixa branca

Apesar de os testes de caixa branca serem geralmente considerados fundamentais em qualquer projeto de desenvolvimento, optou-se por não implementá-los neste caso. Essa decisão está relacionada à natureza individual do desenvolvimento, que poderia tornar a implementação desses testes especialmente desafiadora e demorada. Dado o contexto específico deste projeto, concluiu-se que os recursos e o tempo disponíveis seriam mais eficazmente aplicados em outras áreas do desenvolvimento, de modo a garantir a conclusão eficiente e dentro dos prazos estabelecidos.

- Testes de regressão

Optou-se por não incluir testes de regressão neste projeto, embora sejam amplamente recomendados, especialmente em projetos de médio e grande porte. Esses testes desempenham um papel crucial na garantia de que novas alterações no código não comprometam funcionalidades já existentes. Contudo, a implementação de testes de regressão eficazes exigiria um investimento significativo de tempo e recursos, o que não se alinha com as restrições impostas a este projeto. Após uma análise cuidadosa, concluiu-se que, dadas as limitações atuais, o foco deve ser direcionado para os aspectos mais críticos e viáveis do desenvolvimento.

- Testes de aceitação

É importante destacar que o projeto não contará com testes de aceitação pois seria necessário disponibilizar o sistema ao público para uso. Dado que o sistema lida com transações financeiras, isso poderia representar um risco significativo, incluindo possíveis perdas financeiras, caso algum problema fosse identificado apenas após a disponibilização ao público mesmo em pequena escala.

- Testes de sistema

Quanto aos testes de sistema, eles foram excluídos devido à limitação de tempo disponível para o desenvolvimento do projeto. Embora esses testes sejam essenciais para a validação completa do sistema como um todo, a sua implementação demandaria um tempo que, neste momento, não é viável, considerando o cronograma e os recursos disponíveis.

- Ferramentas utilizadas para a criação dos testes

Para a realização dos testes unitários, uma abordagem essencial nos testes de integração, escolhemos as bibliotecas Jest, Supertest, e mock no Backend. Essas ferramentas foram selecionadas pela sua eficiência em conjunto com o Nest.js, que é o framework utilizado no projeto. O Jest foi escolhido por ser uma das bibliotecas mais populares para testes em JavaScript, oferecendo suporte robusto para testes unitários e de integração. Supertest foi selecionado para testar requisições HTTP, permitindo verificar se os endpoints funcionam corretamente. Já o mock foi integrado para simular dependências e isolar os testes, assegurando que possamos testar partes individuais do código sem influências externas, alguns dos motivos que nos levaram a usar essas ferramentas são:

Jest: A escolha do Jest foi motivada pela sua flexibilidade e ampla aceitação na comunidade de desenvolvimento JavaScript. Ele se integra perfeitamente com o Nest.js, além de oferecer uma sintaxe simples para criar testes e relatórios abrangentes, o que facilita a identificação de erros.

Supertest: A decisão de usar o Supertest está diretamente relacionada à sua capacidade de simular e testar endpoints de forma eficiente. Ao utilizá-lo em conjunto com o Jest, conseguimos realizar testes de integração robustos, garantindo que as APIs do Backend funcionem conforme esperado.

mock: Esta biblioteca foi selecionada para criar cenários de testes onde dependências externas (como chamadas a APIs de terceiros) não afetem os resultados, garantindo a precisão dos testes unitários.

Para os testes de API, optamos pelo uso do Postman, uma ferramenta amplamente reconhecida pela sua facilidade em simular requisições HTTP. A escolha do Postman foi baseada em sua capacidade de testar rapidamente os endpoints do Backend, simulando as interações que o Frontend terá com o Backend. Com ele, pudemos verificar se todas as rotas e respostas estavam corretas, facilitando a integração entre as duas camadas e identificando rapidamente possíveis falhas.

No Frontend, que é focado em um aplicativo mobile desenvolvido majoritariamente em Flutter, decidimos utilizar o Appium para os testes. O Appium foi escolhido por sua compatibilidade com aplicações móveis híbridas e nativas, sendo uma ferramenta ideal para automatizar testes em múltiplas plataformas. Como nosso

projeto se baseia fortemente em dispositivos móveis, o Appium se mostrou fundamental para testar a interface do usuário e as funcionalidades em diferentes ambientes, garantindo a qualidade e usabilidade do aplicativo.

Além disso, decidimos utilizar a biblioteca Color Contrast Checker para melhorar a acessibilidade da plataforma. Essa biblioteca permite verificar o contraste de cores dentro do aplicativo, garantindo que as combinações de cores sejam acessíveis para todos os usuários, especialmente aqueles com deficiência visual. Com isso, buscamos tornar a experiência mais inclusiva e conforme os padrões de acessibilidade.

- Avaliação das ferramentas

Jest e Supertest supriram as nossas necessidades, oferecendo uma maneira prática e eficiente de escrever e manter testes de Backend. A integração fluida entre estas bibliotecas e o Nest.js ajudou a aumentar a confiabilidade e cobertura dos testes, permitindo detectar problemas de maneira rápida e precisa.

Postman também atendeu nossas expectativas ao facilitar a simulação de requisições API. Sua interface gráfica permitiu uma configuração rápida de cenários de testes e ajudou a verificar se os endpoints criados funcionavam corretamente, acelerando a fase de integração com o Frontend.

Appium, por sua vez, se provou eficiente na automação de testes para dispositivos móveis, fornecendo uma plataforma robusta para garantir que o aplicativo desenvolvido em Flutter funcionasse adequadamente em diferentes dispositivos.

A biblioteca Color Contrast Checker supriu nossas necessidades, oferecendo uma solução eficaz para garantir a acessibilidade visual dentro do aplicativo. Sua integração foi simples e rápida, permitindo uma verificação eficiente do contraste de cores entre texto e fundo. A ferramenta nos ajudou a garantir que as combinações de cores atendam aos padrões exigidos pelas diretrizes de acessibilidade WCAG, facilitando a adequação aos níveis de conformidade AA e AAA.

Além disso, a análise automática proporcionada pela biblioteca foi crucial para identificar possíveis problemas de contraste, permitindo ajustes rápidos na interface do usuário. Isso acelerou o processo de desenvolvimento e garantiu que o aplicativo fosse mais inclusivo para usuários com deficiências visuais.

- Pontos de melhoria e observações após o uso

Jest: Durante o uso, observamos que o Jest oferece uma excelente documentação e ferramentas para testes de componentes, porém, em casos de testes mais complexos envolvendo muitos mocks, houve uma pequena curva de aprendizado.

Supertest: Suas funcionalidades se mostraram extremamente úteis para testar APIs, mas em alguns casos mais avançados, a criação de testes de integração pode se tornar um pouco trabalhosa.

Postman: Embora o Postman seja bastante intuitivo, ele exige a criação manual de algumas requisições, o que pode tornar o processo de teste repetitivo para grandes volumes de endpoints. Uma solução poderia ser integrar o Postman com um sistema de automação de testes.

Appium: Funcionou bem para nossos testes móveis, mas notamos que configurar corretamente os ambientes para testes em múltiplos dispositivos pode ser demorado. Isso pode ser um ponto de atenção em projetos futuros, onde a variedade de dispositivos pode crescer.

Color Contrast Checker: Durante o uso, a biblioteca atendeu bem às nossas expectativas, permitindo a verificação rápida e precisa do contraste de cores. No entanto, um ponto de melhoria seria a inclusão de uma funcionalidade que oferecesse sugestões automáticas de ajuste de cores quando as combinações não atendem aos critérios de acessibilidade. Atualmente, a ferramenta apenas indica se o contraste está inadequado, mas cabe ao desenvolvedor buscar manualmente as alternativas. Isso poderia acelerar o processo de ajuste em projetos com muitas combinações de cores.

Em resumo, as ferramentas escolhidas foram fundamentais para garantir a qualidade e eficiência dos testes tanto no Backend quanto no Frontend, atendendo bem às necessidades do projeto até o momento. Embora algumas pequenas limitações tenham sido identificadas, todas as ferramentas proporcionaram ganhos significativos em termos de produtividade e confiança na entrega final do projeto.