Trackinos

Processo de desenvolvimento

01 Escolher caso de uso

Distribuição de pesquisas via Whatsapp

Objetivo

Implementação de testes automatizados, para que seja possível identificar falhas na plataforma antes do software ser colocado em produção, assim ganhando mais confiança dos clientes da empresa

02 Funcionalidades

1. Upload de planilhas de clientes com validação por linha no banco de dados
2. Distribuição e envio de pesquisas via WhatsApp
3. Recebimento das respostas das pesquisas no banco de dados

03 Construção de componentes

Frontend

Tecnologias

- JavaScript
- TypeScript
- Cypress
- Auth0
- React JS
- Jest JS
- Tailwind CSS

Backend

Tecnologias

- Docker
- SQL
- JavaScript
- TypeScript
- Node JS
- Prisma

Boas práticas

Clean Arq: padrão de projeto que busca organizar o código de forma que ele fique separado em camadas, cada uma com suas responsabilidades específicas.

MVC (Model, View, Controller): padrão de arquitetura de software que separa a aplicação em três componentes principais

04 Suíte de testes

Testes unitários: verificam a funcionalidade de unidades isoladas de código, como funções ou métodos.

Testes de integração: avaliam como diferentes unidades de código, módulos ou serviços funcionam juntos.

Testes de carga: simulam um volume elevado de acessos ou dados para verificar como o sistema se comporta sob condições de alta demanda.

Testes com o cypress: simula a interação do usuário com o sistema, esperando atingir o objetivo do teste.

Backend

13 suites de teste e **39** testes no total

64.4% de coverage

Frontend

8 testes no total

- **4** de interface utilizando cypress
- **4** de renderização

05 Esteira de integração contínua

Conceito: prática de desenvolvimento de software no qual leva a execução dos testes automatizados no projeto, para validar que está tudo funcionando antes de aplicarmos ou removermos algo do nosso projeto.

Funcionamento: quando o pull request for aceito na branch developer a nossa pipeline irá executar testes unitários → testes de integração e por último testes de interface e2e. Caso aconteceça algum problema nesses testes, receberemos uma notificação via webhook falando que a pipeline teve um erro, porém caso não ocorra nenhum erro, o github actions vai colocar a nova aplicação nas nossas máquinas da cloud, nesse caso para o ambiente de desenvolvimento, essa pipeline tem o mesmo propósito na branch main, com a diferença do local do deploy.

Tecnologias utilizadas

1. Sistema de controle de versão: **Git**
2. Plataforma de integração contínua: **Github actions**

Regras implementadas

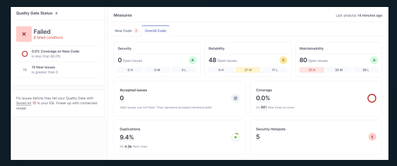
1. Testes automatizados
2. Padrões de codficação

06 Sonarqube

Conceito: A ferramenta Sonarqube se trata de um serviço focado em analisar a qualidade do código presente no sistema, fonecendo uma interface para visualizar a quailidade a partir de um dashboard, que exibe métricas pré-definidas que ajudem na compreenção do que está bom, e o que pode ser melhorado dentro código.

Vantagens: O uso do Sonarqube oferece diversas vantagens, como a rápida obtenção de insights sobre melhorias no código e a facilidade de utilização. Ao contrário de outras ferramentas de análise de código que podem ser difíceis de implementar e atualizar, o Sonarqube é fácil de usar e manter atualizado, o que aumenta a eficiência das análises. Além disso, o uso do Sonarqube facilita a implementação eficiente de futuras melhorias, garantindo a qualidade do código.

Interface do Sonarqube:

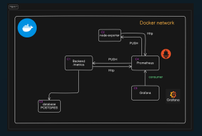


07 Dashboard

Conceito: Com a utilização da plataforma Grafana, um serviço focado para implementar monitoramento de dados dentro de um sistema, constituimos um dashboard baseado em métricas aplicadas dentro do sistema, via a própria biblioteca de código que o serviço oferece.

Arquitetura: Para garantir o funcionamento do Grafana, adaptamos a arquitetura do sistema para encaixa-lo devidamente. A lógica da arquitetura consiste em primeiramente, alocar todos os componentes que o Grafana necessita para funcionar, mais o Backend do sistema, dentro de uma network do docker. Com isso os componentes node-exporter (responsavel por coletar métricas operacionais da maquina) e o Backend (com a coleta das métricas implementadas no código), realizam protocolos de requisição para coletar e enviar informações para o o Prometheus (serviço responsavel por se comunicar e enviar dados para o Grafana), por fim, o Prometheus enviando as devidas métricas para o Grafana de forma que o dashboard possa ser configurado dentro da interface do mesmo.

Imagem da arquitetura implementada:



Métricas: Foram definidas quatro métricas as quais são exibidas no dashboard, sendo elas, quantidade de falhas gerais nas requisições do sistema, quantidade de pesquisas respondidas por dia pelos usuários, tempo médio de resposta do endpoint de que faz upload de planilhas, e quantidade de pesquisas enviadas por dia.

Resultado final do dashboard: