INFOGRÁFICO DA SOLUÇÃO

SOLUÇÃO GRUPO 2

CASOS DE USO

1. Distribuição via Whatsapp
2. Distribuição via SMS Link

Objetivos

Implementar **testes automatizados** em funcionalidades críticas, para:

- Reduzir regressões
- Detectar problemas rapidamente
- Aumentar a confiança do cliente na plataforma

DADOS DE DESENVOLVIMENTO

+5000	12	+80%	8
linhas de código	endpoints	cobertura de testes	contribuidores

Requisitos Funcionais

Criação de distribuição de Pesquisas via SMS Link

Eu, enquanto empresa contratante da Track.co, quero distribuir uma pesquisa de satisfação para meus clientes através do canal SMS Link, para obter insights sobre meu produto/serviço.

Criação de distribuição de Pesquisas via Whatsapp

Eu, como gerente de pesquisa, quero automatizar a distribuição de pesquisas, aos clientes selecionados, via Whatsapp, para agilizar e simplificar o envio de pesquisas, tornando o processo mais eficiente e escalável.

Requisitos Não Funcionais

Tempo de resposta

Eu, como usuário, quero que a minha jornada na aplicação seja rápida e eficiente.

O tempo de resposta da aplicação deve ser de, no máximo, 3 segundos.

Solução proposta

Duração média de **1,05s** em um cenário com **100 usuários simultâneos**

Segurança

Eu, como empresa que distribui as pesquisas, quero que meus dados e o de meus clientes estejam seguros.

Solução proposta

Autenticação de usuários utilizando a ferramenta Google Firebase, garantindo que apenas usuários permitidos acessem o sistema.

Construção dos componentes do frontend

Frameworks utilizados

O Angular, com uma arquitetura baseada em componentes, foi utilizado para o desenvolvimento da interface. Já o Jasmine e Karma foram utilizados para a realização de testes.

Tecnologias utilizadas

O Typescript oferece verificação estática de tipos, melhorando a robustez do código. Sua capacidade de lidar com projetos de grande porte e sua integração com o Angular o tornou uma opção atraente para utiliza-la.

Padrões de projeto implementados

Arquitetura de Componentes

Seguindo as boas práticas recomendadas do Angular, adotamos uma abordagem baseada em componentes para organizar e modularizar o código. Dessa forma, possibilitando a reutilização e modularização do código de forma limpa e eficaz.

Construção dos componentes do backend

Frameworks utilizados

Utilizamos o Jest framework para realizar testes dentro da nossa aplicação, aplicando desde testes unitários até testes de integração.

Tecnologias utilizadas

Optamos por utilizar o TypeScript principalmente por ser uma tecnologia que possui alta tipagem, o que será necessário para seguir com o padrão arquitetural e integrar com o nosso banco de dados relacional.

Padrões de projeto implementados

SOLID PATTERN

Utilizamos o padrão arquitetural SOLID. A partir dele conseguimos desenvolver a aplicação promovendo códigos mais limpos, autoexplicativos e de fácil manutenção. Principalmente pela fato da independência entre as camadas do app.

SUÍTE DE TESTES

TIPOS DE TESTES

Testes unitários backend — 89% de cobertura de código

Testes unitários frontend — 100% dos componentes testados

Testes de integração — 48% dos endpoints testados

Testes End to End frontend — 100% de funcionalidades testadas

TESTES DE CARGA

5 Endpoints

6 Cenários de teste para cada

Início das falhas

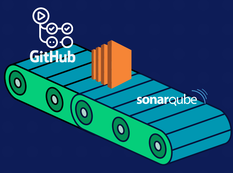
- 1000 usuários simultâneos
- 15 minutos
- 0,12% de falhas em requisições
- 237 mb recebidos
- 83 mb enviados

Limite da aplicação

- 10000 usuários simultâneos
- 15 minutos
- 84,38% de falhas em requisições
- 33 mb recebidos
- 17 mb enviados

ESTEIRA DE INTEGRAÇÃO CONTÍNUA

1. **PROVISIONAMENTO DA INSTÂNCIA EC2**
   Ao optarmos por configurar uma instância EC2 na AWS para hospedar nossos serviços de integração, buscamos garantir um ambiente altamente controlado e flexível. Essa escolha nos permite ter total autonomia sobre a configuração do ambiente, desde a escolha do sistema operacional até a instalação de ferramentas específicas, como o SonarQube.
2. **IMPLEMENTAÇÃO SONARQUBE**
   O SonarQube desempenha um papel fundamental em nossa esteira de integração contínua ao realizar análises estáticas de código. Essa ferramenta nos permite identificar uma ampla gama de problemas de qualidade de código, como vulnerabilidades de segurança, bugs, dívidas técnicas e padrões de código inadequados. Ao integrar o SonarQube à nossa instância EC2, garantimos que cada alteração de código seja submetida a uma verificação de qualidade, resultando em um código mais limpo, seguro e sustentável.
3. **INTEGRAÇÃO GITHUB ACTIONS**
   Ao integrar nossa esteira com o GitHub Actions, alcançamos um novo nível de automação em nosso processo de desenvolvimento. Configuramos ações personalizadas para automatizar tarefas como compilação, teste e análise de código, garantindo que cada alteração no main seja submetida a verificações antes de ser integrada ao código principal. Essa integração contínua nos permite identificar e corrigir problemas de forma ágil, reduzindo significativamente o tempo e os esforços necessários para manter a qualidade do código.



REGRAS IMPLEMENTADAS

- #1 Análise Estática de Código com o SonarQube
- #2 Execução automática de testes sempre que uma alteração é submetida na main
- #3 Regra de exclusão de arquivos e diretórios não essenciais durante o processo de implantação
- #4 Automação do Fluxo de Trabalho com o GitHub Actions