

# Trabalho de Qualidade e Teste de Software - Primeira Entrega

Membros do grupo:

João Victor de Albuquerque Pletsch

Victor Correa da Silva Moreira

Márcio de Amorim Machado Ferreira

Pedro Paulo Sobral de Moraes

Renato Luiz Moura de Azevedo

## 1. Descrição do Escopo do Sistema:

A aplicação é um sistema de gerenciamento de pedidos para um restaurante, que permite que os clientes visualizem o cardápio do restaurante, selecionem produtos disponíveis, adicione-os ao carrinho e finalize o pedido para entrega ou retirada.

Requisitos do Sistema:

- 1 - O sistema deve permitir que os clientes realizem o cadastro para criar uma conta.
- 2 - O cadastro deve solicitar informações como nome, endereço de e-mail e senha.
- 3 - Após o cadastro, os clientes devem poder fazer login usando suas credenciais.
- 4 - Os clientes devem ter a opção de recuperar a senha caso a esqueçam, utilizando um processo seguro de redefinição de senha.
- 5 - O sistema deve permitir que os clientes visualizem o cardápio do restaurante.
- 6 - Cada item do cardápio deve ser acompanhado de uma descrição, preço e imagem ilustrativa.
- 7 - Os clientes devem ser capazes de adicionar produtos ao carrinho de compras.
- 8 - O carrinho de compras deve exibir os itens selecionados, a quantidade de cada produto e o preço total.
- 9 - Os clientes devem ter a opção de remover itens do carrinho antes de finalizar o pedido.
- 10 - O sistema deve permitir que os clientes finalizem o pedido, indicando a forma de pagamento e o endereço de entrega.
- 11 - Após a finalização do pedido, os clientes devem receber uma confirmação do pedido e informações sobre o tempo estimado de entrega.

## 2. Código-fonte original:

O código-fonte do projeto pode ser encontrado no repositório do GitHub:

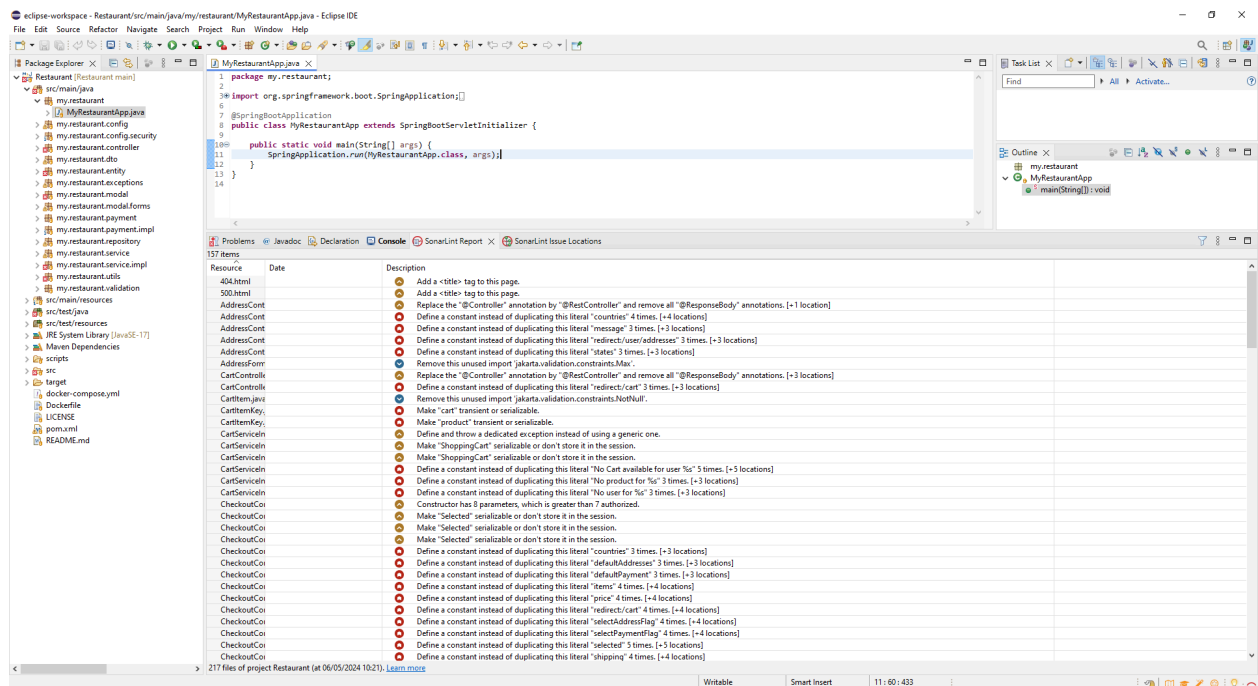
<https://github.com/asdhammu/Restaurant>

Código fonte do trabalho: <https://github.com/vaniacourses/Restaurant>

## 3. Relatório de Inspeção do Código-fonte:

### 3.1. Execução da Ferramenta:

Foi utilizada a ferramenta SonarLint para inspeção do código-fonte, através da IDE Eclipse. O resultado da execução pode ser visualizado nos seguintes prints:



157 problemas mostrados no SonarLint.

Foram identificados diversos problemas que requerem atenção e possíveis melhorias para garantir a qualidade do código e o seu bom funcionamento. Alguns desses problemas foram:

1. AddressController.java:
  - Definição de uma constante ao invés de duplicar o literal "countries" em quatro locais diferentes.
2. AddressForm.java:
  - Remoção do import não utilizado 'jakarta.validation.constraints.Max'.
3. CartController.java:
  - Substituição da anotação "@Controller" pela "@RestController" e remoção de todas as anotações "@ResponseBody".
4. CartItemKey.java:
  - Necessidade de tornar "cart" transient ou serializable.
5. CartServiceImpl.java:
  - Definição e lançamento de uma exceção dedicada ao invés de usar uma genérica.
  - Necessidade de tornar "ShoppingCart" serializable ou evitar armazená-lo na sessão.
6. CheckoutController.java:
  - Construtor com 8 parâmetros, o que excede o limite de 7 autorizados.
  - Necessidade de tornar "Selected" serializable ou evitar armazená-lo na sessão.
7. CheckoutControllerTests.java:
  - Remoção do modificador 'public'.
8. Constants.java:
  - Adição de um construtor privado para ocultar o construtor público implícito.

- Renomeação da constante para seguir a expressão regular `^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$`.

9. EmailServiceImpl.java:

- Atualização do logger para usar "EmailServiceImpl.class".

10. HomeController.java:

- Adição de um comentário aninhado explicando por que o método está vazio, lançando uma `UnsupportedOperationException` ou completando a implementação.

11. checkout.html:

- Atributo "aria-hidden" não deve ser definido em elementos focáveis.

12. orderDetails.html:

- Adição de uma tag `<title>` nesta página.

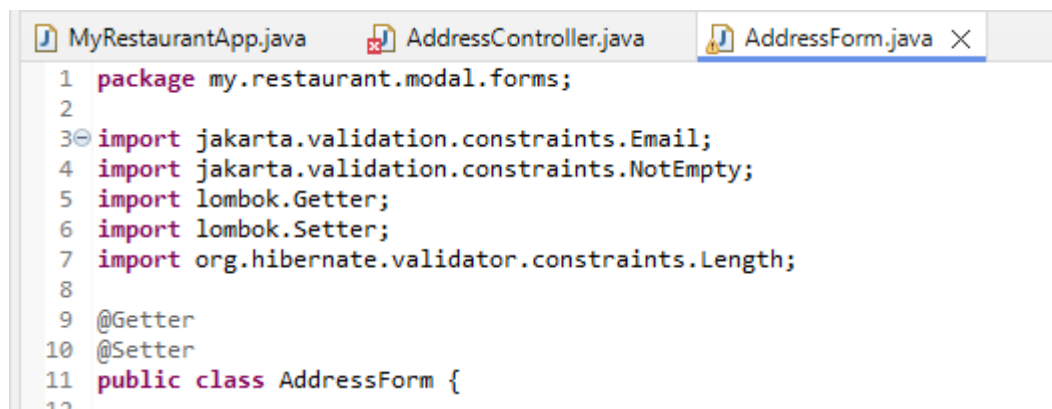
### 3.2. Problemas identificados e correções realizadas:

Alguns exemplos de problemas identificados pelo SonarLint que foram corrigidos:

- Classe: AddressForm.java

Problema identificado: Remove this unused import 'jakarta.validation.constraints.Max'.

Correção: Foi feita a remoção do import.



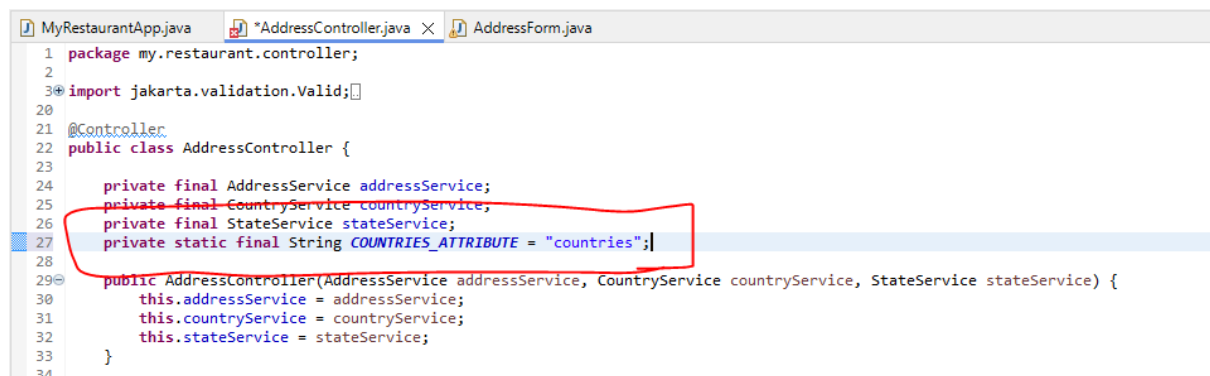
```
1 package my.restaurant.modal.forms;
2
3 import jakarta.validation.constraints.Email;
4 import jakarta.validation.constraints.NotEmpty;
5 import lombok.Getter;
6 import lombok.Setter;
7 import org.hibernate.validator.constraints.Length;
8
9 @Getter
10 @Setter
11 public class AddressForm {
12
```

Print após a correção.

- Classe: AddressController.java

Problema: Define a constant instead of duplicating this literal "countries" 4 times. [+4 locations]

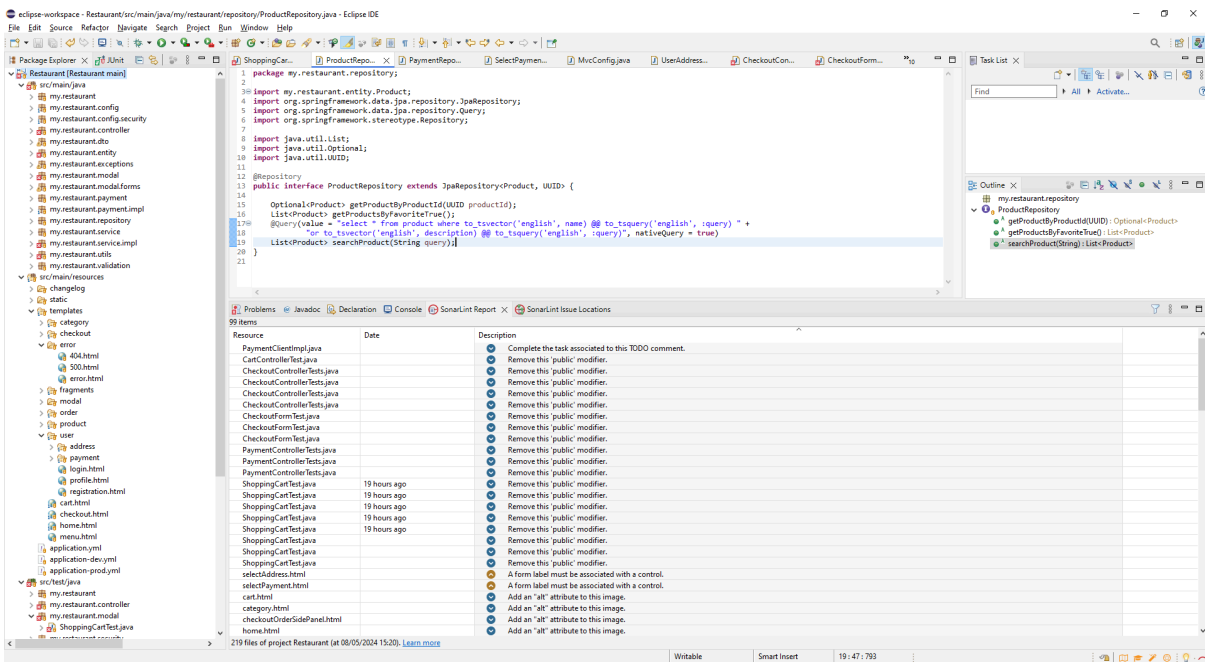
Correção: Foi criada uma constante guardando a string "countries" e passado a utilizar essa constante nos 4 locais na classe que estava especificando a string diretamente.



```
1 package my.restaurant.controller;
2
3 import jakarta.validation.Valid;
4
5 @Controller
6 public class AddressController {
7
8     private final AddressService addressService;
9     private final CountryService countryService;
10    private final StateService stateService;
11    private static final String COUNTRIES_ATTRIBUTE = "countries";
12
13    public AddressController(AddressService addressService, CountryService countryService, StateService stateService) {
14        this.addressService = addressService;
15        this.countryService = countryService;
16        this.stateService = stateService;
17    }
18
```

Print após a correção.

Esses e diversos outros problemas identificados foram corrigidos, e agora após essas correções, ao fazer uma nova execução do SonarLint o estado é o seguinte:



Resource	Date	Description
PaymentClientImpl.java		Complete the task associated to this TODO comment.
CartControllerTest.java		Remove this 'public' modifier.
CheckoutControllerTests.java		Remove this 'public' modifier.
CheckoutControllerTests.java		Remove this 'public' modifier.
CheckoutControllerTests.java		Remove this 'public' modifier.
CheckoutControllerTests.java		Remove this 'public' modifier.
CheckoutFormTest.java		Remove this 'public' modifier.
CheckoutFormTest.java		Remove this 'public' modifier.
PaymentControllerTests.java		Remove this 'public' modifier.
PaymentControllerTests.java		Remove this 'public' modifier.
PaymentControllerTests.java		Remove this 'public' modifier.
PaymentControllerTests.java		Remove this 'public' modifier.
ShoppingCartTest.java	19 hours ago	Remove this 'public' modifier.
ShoppingCartTest.java	19 hours ago	Remove this 'public' modifier.
ShoppingCartTest.java	19 hours ago	Remove this 'public' modifier.
ShoppingCartTest.java	19 hours ago	Remove this 'public' modifier.
ShoppingCartTest.java	19 hours ago	Remove this 'public' modifier.

99 problemas mostrados no SonarLint. Portanto, 58 problemas já foram resolvidos.

#### 4. Projetar Casos de Testes Unitários:

##### Testes usando JUnit:

- Classe: ShoppingCart.java

Classe de teste: ShoppingCartTest.java

Foram criados métodos de teste para verificar diferentes cenários do funcionamento dos itens no carrinho, são eles: adição, remoção, atualização, se está sendo esvaziado corretamente, preço sendo calculado corretamente, se o carrinho fica vazio após remover

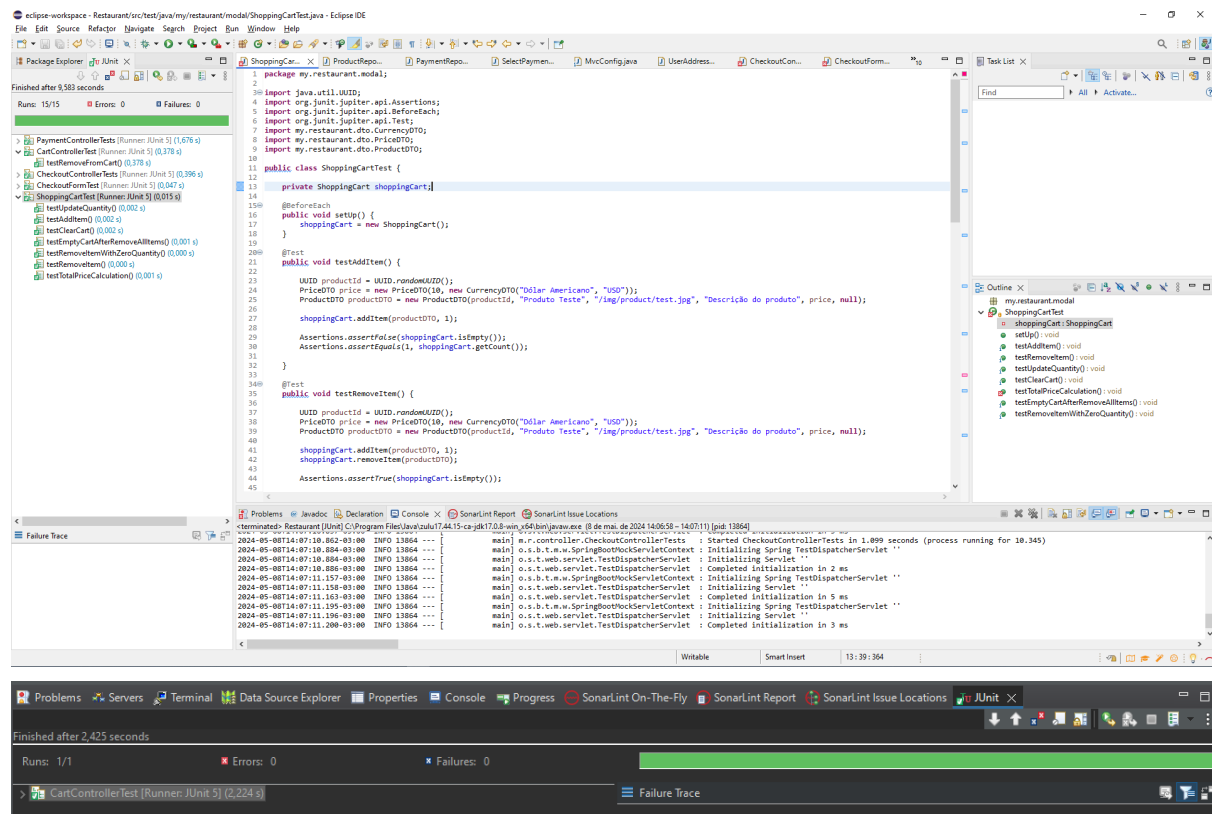
todos os itens, e se atualização da quantidade de um item altera na quantidade de itens do carrinho.

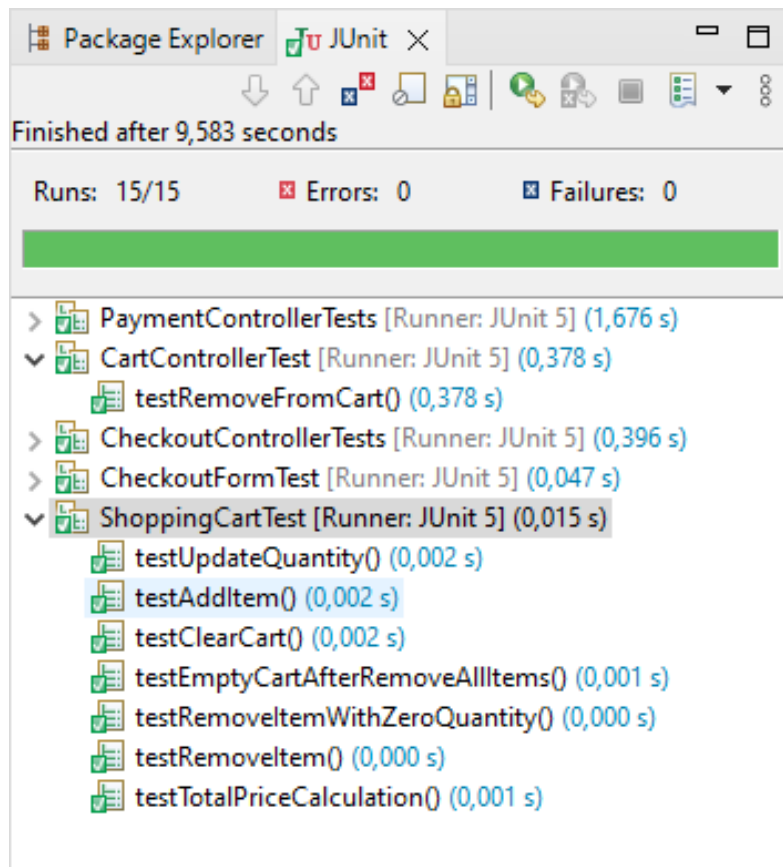
- Classe: CartController.java

Classe de teste: CartControllerTest.java

Realiza o teste da remoção de um item do carrinho pelo usuário.

## Execução dos testes no JUnit:





As classes ShoppingCartTest.java e CartController.java foram criadas com métodos de teste para serem executadas pelo JUnit, e as duas tiveram todos os testes com sucesso. As outras classes de testes eram classes que já existiam no código original da aplicação.

### **Outros Casos de Teste:**

Também foram feitos casos de testes executando a aplicação manualmente e mapeando os passos realizados em uma planilha. Por exemplo:

- Classe: UserController.java

Caso de Teste 1: Testar o cadastro usando um e-mail que nunca foi registrado no sistema.

Caso de Teste 2: Testar o cadastro usando um e-mail de uma conta já existente no sistema.

Os dois testes passaram com sucesso. No segundo caso, quando já existe uma conta com o e-mail digitado o campo de e-mail aparece em vermelho e também um erro: "account exists for email".

- Classe: UserController.java

Caso de Teste 1: Testar o login com credenciais válidas.

Caso de Teste 2: Testar o login com credenciais inválidas.

- Classe: CartController.java

Caso de Teste 1: Testar a adição de um item ao carrinho.

Caso de Teste 2: Testar a remoção de um item ao carrinho.

Ambos os testes obtiveram êxito.

Planilha com estes e demais Casos de Testes: [📄 Casos de Teste](#)

## 5. Plano de Teste:

### - Escopo e objetivo:

O escopo dos testes inclui todas as funcionalidades principais da nossa aplicação escolhida, visando garantir a sua qualidade e maior confiabilidade. O objetivo principal dos testes é validar o funcionamento correto do sistema, desde a autenticação do usuário até a finalização do pedido pelo usuário.

### - Características do produto a serem testadas:

Funcionalidade: Garantir que todas as funcionalidades do sistema estejam operando conforme especificado.

Confiabilidade: Verificar a estabilidade e robustez do sistema em diferentes cenários de uso.

Usabilidade: Avaliar a facilidade de uso e a experiência do usuário durante a interação com o sistema.

### - Abordagem a ser utilizada:

A abordagem de teste adotada será uma combinação de testes de unidade, testes de integração e testes de sistema, que poderá garantir uma cobertura bastante abrangente do sistema em diferentes níveis de granularidade. As ferramentas de teste utilizadas até o momento foram o Junit e SonarLint, mas outras ainda estão sendo analisadas para serem utilizadas na continuidade dos testes pelo sistema, como por exemplo o Selenium, que poderá nos ajudar para automação de testes pela interface do usuário nas páginas web do sistema.

### - Principais itens a serem testados:

Autenticação de usuário.

Seleção de itens do cardápio.

Adição ao carrinho.

Remoção de itens do carrinho.

Finalização do pedido.

### - Tarefas e Artefatos de Teste:

- Desenvolvimento de casos de teste para cada funcionalidade do sistema.

- Execução dos casos de teste e documentação dos resultados.

- Geração de relatórios de teste para avaliar a cobertura e a qualidade dos testes realizados.

### - Cronograma para o Teste:

Planejamento dos testes: 06/2024.

Execução dos testes: mês 06 e 07 de 2024.

Revisão dos resultados e ajustes: 07/2024.

Pessoal responsável pelas atividades de teste: todos os membros do grupo.

As atividades de teste serão realizadas por todos os membros do grupo, com cada um sendo responsável por cada funcionalidade específica do sistema. A responsabilidade pela execução e documentação dos testes será atribuída conforme a distribuição de tarefas ainda for definida.

### **Riscos associados aos testes:**

Os principais riscos associados aos testes incluem atrasos na entrega de artefatos de teste, falta de cobertura adequada dos casos de teste e bugs não detectados durante os nossos testes. Algumas estratégias de mitigação desses problemas poderão ser implementadas para lidar com esses riscos, como revisões regulares dos resultados dos testes e ajustes conforme necessário.

### **Responsabilidades e contribuições na primeira entrega:**

Definição do Escopo do sistema: João Victor, Victor, Pedro Paulo, Marcio

Criação de Casos de Testes: João Victor, Pedro Paulo, Marcio

Resolução de problemas no código do sistema: João Victor, Pedro Paulo, Victor

Plano de teste: João Victor, Marcio, Pedro Paulo, Victor

Elaboração da documentação: João Victor, Victor, Pedro Paulo