plano de testes

Relatório de testes

Unifacs

Trabalho de qualidade de software

plano de testes

Relatório de testes

Bruno Sales Fiaes Carneiro

Marcelo Miranda Cavalcante

Zanoni Fernandes

Eduardo Bellineli

Diego Ferreira

George Gebers

Levi Dourado

INTRUDUÇÃO

* 1. objetivos

Esse documento tem com objetivo de demonstrar nosso plano de testes com base no trabalho proposto (A3) pelo professor João.

* Identificar informações de projeto existentes e os componentes de software que devem ser testados.
* Listar os Requisitos a Testar recomendados (alto nível).
* Recomendar e descrever as estratégias de teste a serem empregadas
* Identificar os recursos necessários e prover uma estimativa dos esforços de teste.
* Listar os elementos resultantes do projeto de testes.
  1. O projeto
* ( Mercadinho ) tem como objetivo gerir um espaço comercial, com o cadastro de clientes e produtos....

Escopo do projeto

O projeto desta equipe é baseado em um software de um mercadinho, a ideia do projeto é a implementação e execução de testes unitários. Esses testes são fundamentais para garantir a integridade e o funcionamento correto de cada componente do nosso software. Utilizando práticas de desenvolvimento orientado a testes (TDD), o projeto visa aumentar a confiabilidade do código ao identificar e corrigir possíveis problemas de forma precoce no ciclo de desenvolvimento. Com uma arquitetura bem definida e modular.

O software permite a fácil criação e execução de testes unitários para cada unidade isolada de código, garantindo assim a robustez e a qualidade do sistema como um todo.

REQUISITO

A lista abaixo identifica as funcionalidades presente no sistema que já foram ou vão ser implementadas.

* O cliente deve conter : nome, CPF, data de nascimento.
* O produto deve conter: nome do produto.
* Verificar se todos os itens podem ser adicionados ao carrinho de compras.
* Testar o processo de checkout para diferentes formas de pagamento (cartão de crédito, débito, dinheiro).
* Verificar a funcionalidade de devoluções ou cancelamentos de vendas.
* Verifique que as informações úteis obtidas pelo subsistema responsável podem ser atualizadas e que as mesmas podem ser apresentadas.
* Verifique que as informações específicas de cada usuário podem ser acompanhadas.
* Verifique que o sistema pode cadastrar, atualizar e remover um novo clientes de seu banco de dados.
* Verifique que o sistema pode remover categorias de produtos cadastrados.
* Verifique que um novo cliente pode ser cadastrado.
* Verifique que os produtos possam ser cadastrados, removidos e atualizados pelo administrador do sistema.
* Verifique que as informações sobre o sistema possam ser cadastradas, removidas e atualizadas pelo administrador do sistema.
* Verifique que as informações do sistema e dos produtos cadastrados possam ser consultadas pelos usuários.
* Verifique que as informações úteis cadastradas possam ser consultadas.
* Verifique que o usuário cadastrado pode acessar o extrato de sua conta no sistema.
* Verifique que o sistema é capaz de buscar e manter atualizadas as informações de extrato de conta.

**TESTE FUNCIONAL**

* Verifique que as informações úteis obtidas pelo subsistema responsável são automaticamente e periodicamente atualizadas.
* Verifique que qualquer usuário pode acessar sua própria conta através de login e senha.
* Verifique que o relatório da conta do hóspede é correto.
* Verifique que as informações podem ser acessadas em qualquer dos idiomas disponíveis.

**TESTE DE INTERFACE DO USUARIO**

* Não possui

**PERFIL DE PERFORMACE**

* Verificar tempo de resposta do sistema com 10 usuários.
* Verificar tempo de resposta do sistema com 50 usuários.
* Verificar tempo de resposta do sistema com 100 usuários.

**TESTE DE STRESS**

* Ainda não realizado

**TESTE DE VOLUME**

* Ainda não realizado

**TESDE DE SEGURANÇA E CONTROLE DE ACESSO**

* Não implementado.

**TESTE DE INSTALAÇÃO**

* Ainda não implementado.

**ESTRATEGIA USADA PARA ELABORAR OS TESTES**

Neste tópico, adotaremos como os testes foram pensados, para qual intuito e como foram feitos.

Abaixo estão os testes unitários desenvolvidos pela equipe, separados pelo nome da pasta ao qual então vinculador no GitHub.

Esse tópico tem o intuito de explicar de modo geral como cada pasta de testes funciona.

***PASTA CLIENTE***

Nesta pasta o código de teste é verificado o comportamento da classe Cliente do sistema. Ele inclui testes para os métodos de configuração e obtenção de atributos do cliente, como ID, nome, CPF e data de nascimento. Além disso, há testes de validação para o nome, CPF e data de nascimento do cliente, garantindo que as entradas sejam válidas. Esses testes ajudam a assegurar que a classe Cliente funcione corretamente e valide corretamente as informações dos clientes no sistema.

***PASTA DBTESTE***

Esta pasta de código guarda os testes referentes ao comportamento do método **getNomeCompletoCliente()** da classe **ClienteReaderDb**, tanto quando o cliente é ou não é encontrado, garantindo o funcionamento correto da lógica de busca e manipulação de clientes no banco de dados.

*Configuração Inicial:*

O teste está dentro da classe DbTest e utiliza a anotação @ExtendWith(MockitoExtension.class) para integrar o Mockito com o JUnit.Importações de classes necessárias, incluindo asserções estáticas do JUnit (Assertions), recursos de simulação do Mockito e outras classes relacionadas.

*Inicialização de Objetos:*

No método setup(), que é executado antes de cada teste (@BeforeEach), são inicializados objetos necessários para os testes:clienteReaderDb: Uma instância de ClienteReaderDb.controladorEntidades: Um objeto simulado (mock) da classe ControladorEntidades, criado usando o Mockito.

**Teste de Caso de Sucesso (clienteReaderDbFoundTest):**

Neste teste, um objeto Cliente é criado e configurado com um nome ("Maria Antônia").O Mockito é usado para simular o comportamento do método find() do controladorEntidades, que deve encontrar um cliente com ID 1 e retornar o objeto cliente criado anteriormente.É chamado o método getNomeCompletoCliente(1) do clienteReaderDb para obter o nome do cliente com ID 1.A asserção assertEquals() verifica se o nome retornado é "Maria Antônia", conforme esperado.

**Teste de Caso de Falha (clienteReaderDbNotFoundTest):**

Neste teste, o Mockito é usado para simular o comportamento do método find() do controladorEntidades, que deve retornar null para indicar que o cliente com ID 1 não foi encontrado.É chamado o método getNomeCompletoCliente(1) do clienteReaderDb para obter o nome do cliente com ID 1.A asserção assertNull() verifica se o nome retornado é null, indicando que o cliente não foi encontrado.

**PASTA PEDIDO**

Este código consiste em testes unitários escritos em Java usando o framework de testes JUnit.

*Teste validaContagemItensPedidoTest:*

Este teste verifica se a contagem de itens em um pedido está sendo validada corretamente.Inicialmente, é criado um objeto Pedido e um objeto ProdutoPeso (umProdutoPeso).O método pushItemLista() é utilizado para adicionar o produto ao pedido.O método validaContagemItensPedido() é chamado para validar a contagem de itens no pedido.O teste verifica se a validação é bem-sucedida utilizando o método assertTrue().

*Teste totalizaPedidoTest:*

Este teste verifica se o cálculo do valor total de um pedido está correto.É criado um objeto Pedido e dois objetos de produtos (umProdutoPeso e umProdutoUnidade).Ambos os produtos são adicionados ao pedido usando o método pushItemLista().O método validaContagemItensPedido() é chamado para validar a contagem de itens no pedido.Se a validação for bem-sucedida, o método totalizaPedido() é chamado para calcular o valor total do pedido.Finalmente, o teste verifica se o valor total calculado está correto usando o método assertEquals().Esses testes verificam se a lógica relacionada à contagem de itens e ao cálculo do valor total em um pedido está funcionando conforme o esperado. Esses testes são importantes para garantir que o sistema de pedidos do mercado esteja funcionando corretamente, evitando erros de contagem ou cálculos incorretos que poderiam afetar o funcionamento do sistema.

**PASTA PRODUTO**

Este código consiste em uma série de testes unitários escritos em Java utilizando o framework de testes JUnit.

*Testes setEGetTest:*

Cada um desses testes verifica se os métodos set e get de diferentes atributos de produtos (fabricante, nome, preco, dataValidade, quantidade) estão funcionando corretamente.Para cada atributo, o teste configura um valor usando o método set e então verifica se o valor retornado pelo método get é o mesmo que foi configurado usando o método assertEquals().Além disso, alguns testes incluem mensagens personalizadas no caso de falha.

*Teste validaStringsProdutoTest:*

Estes testes verificam se o método validaStringsProduto() retorna true quando uma string não está vazia e false quando está vazia.Um caso de teste inclui uma string válida (Nestlé) e o outro inclui uma string vazia.

*Teste validaDataValidadeTest:*

Estes testes verificam se o método validaDataValidade() retorna true quando a data de validade é futura e false quando é passada.Um caso de teste inclui uma data futura (2024-04-25) e o outro inclui uma data passada (2023-04-25).

*Testes validaProdutoNonZeroTest:*

Esses testes verificam se os métodos validaProdutoFloatNonZero() e validaProdutoIntNonZero() retornam true quando o valor é não zero e false quando é zero.Cada teste inclui um caso onde o valor é não zero e outro caso onde o valor é zero.Em resumo, esses testes verificam diversas funcionalidades dos objetos ProdutoPeso e ProdutoUnidade, incluindo a configuração e recuperação de atributos, validação de strings, validação de datas de validade e validação de valores não zero. Esses testes são importantes para garantir que a lógica relacionada aos produtos esteja funcionando corretamente no sistema do mercadinho.

RECURSOS

Este tópico apresenta os recursos recomentados para o projeto ( Mercadinho ), suas principais responsabilidades.

CONOGRAMA

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Data de Início** | **Data de Término** |
| Planejar Teste | 15/04/24 | 18/04/24 |
| Projetar Teste | 19/04/24 | 22/04/24 |
| Implementar Teste | 22/04/24 | 24/04/24 |
| Executar Teste | 24/04/24 | 24/04/24 |
| Avaliar Teste | Próxima entrega | Próxima entrega |