UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CAMPUS PAU DOS FERROS
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA

ALAN ALMEIDA DA SILVA - 2024011434
DALTON FIRMINO CAMPOS - 2024011414
IVERTON EMIQUISON RIBEIRO DE BESSA - 2024011408
PAULINA JÚLIA COSTA DE OLIVEIRA - 2023023644

SISTEMA DE GERENCIAMENTO DE MERCADO

Relatório geral dos testes realizados

PAU DOS FERROS – RN
2024

1. INTRODUÇÃO

Este relatório mostra os resultados dos testes realizados no sistema de gerenciamento de mercado. O foco principal é assegurar que o sistema atenda aos requisitos funcionais e não funcionais, identificando possíveis falhas e verificando o correto funcionamento de todos os componentes, tanto de forma isolada quanto integrada. Durante os testes registramos o fluxo de criação, atualização e exclusão de produtos, estoque e itens de venda. Os testes unitários foram implementados usando o django rest Framework e são executados através das classes como: VendaVIEWSetTest, EstoqueViewSetTeste e ProdutoViewsSetTest que herdam de APITestCase. Já os de sistemas, foram executados a partir do teste no fluxo de sistema, como em realizar cadastros, vendas entre outras.

- **Objetivo dos Testes**: Validar as funcionalidades do sistema de caixa, garantindo que o fluxo de compra esteja correto e sem falhas.

.

2. FLUXO DOS TESTES DE SISTEMA

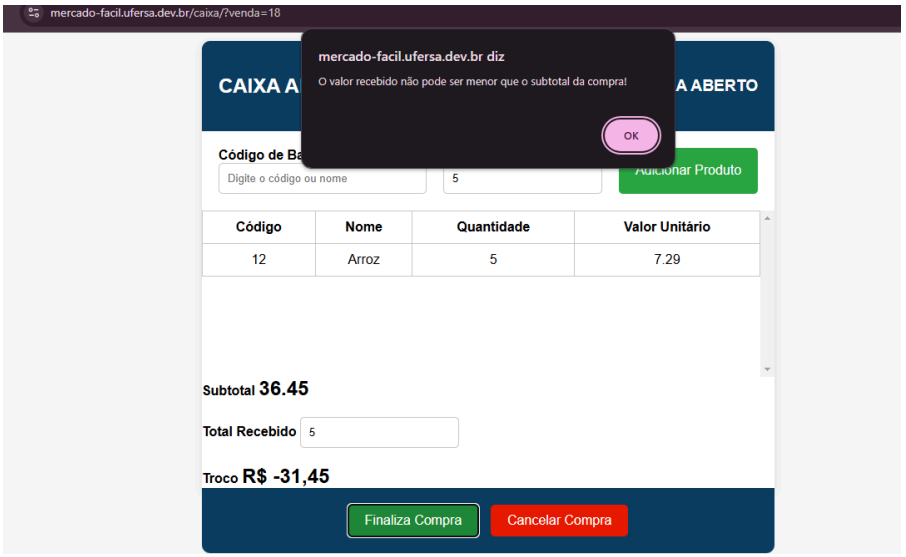
- **Cenário de Teste 1**: Inserindo uma nova compra, calculando descontos e gerando o comprovante de compra com os valores corretos e em pdf.
- **Resultado Esperado**:
- Produtos adicionados corretamente.
- Subtotal e troco calculados com precisão.
- Compra finalizada sem erros.
- Redirecionar para o comprovante de compra em formato de pdf.
- Permitir baixar e imprimir o pdf.
- **Resultado Obtido**: Produtos foram adicionados e calculados corretamente, Ele pega o valor unitário de cada produto e mostra o total, pergunta a opção de valor recebido e já retorna quanto devemos fornecedor de troco. Compra finalizada conforme esperado e redirecionado para o comprovante.
- **Status**: Aprovado
- **Evidências**: As imagens a seguir mostram o fluxo do cadastro de compra

- Imagem 01: Cadastro de compra:

- Imagem 02: Gerando comprovante de compra

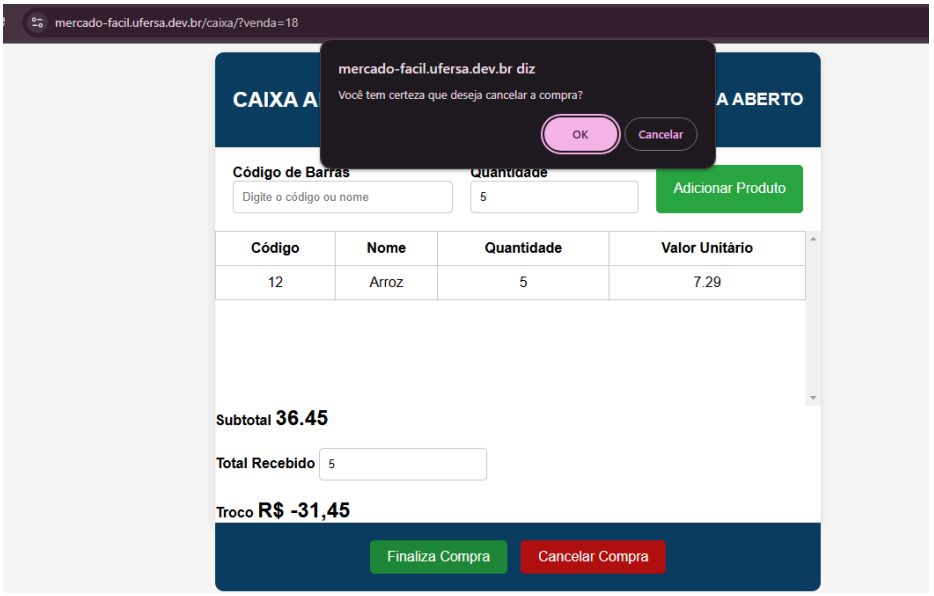
- **Cenário de Teste 2**: Cadastro do valor recebido menor que o valor total calculado da compra.
- **Resultado Esperado**: O sistema deve impedir o cadastro de compra com o valor pago menor que o total.
- **Resultado Obtido**: O sistema retorna uma mensagem de não permitido.
- **Evidências**: As imagens a seguir mostram o fluxo da ação.

- Imagem 03: Mensagem de valor menor que o da compra:

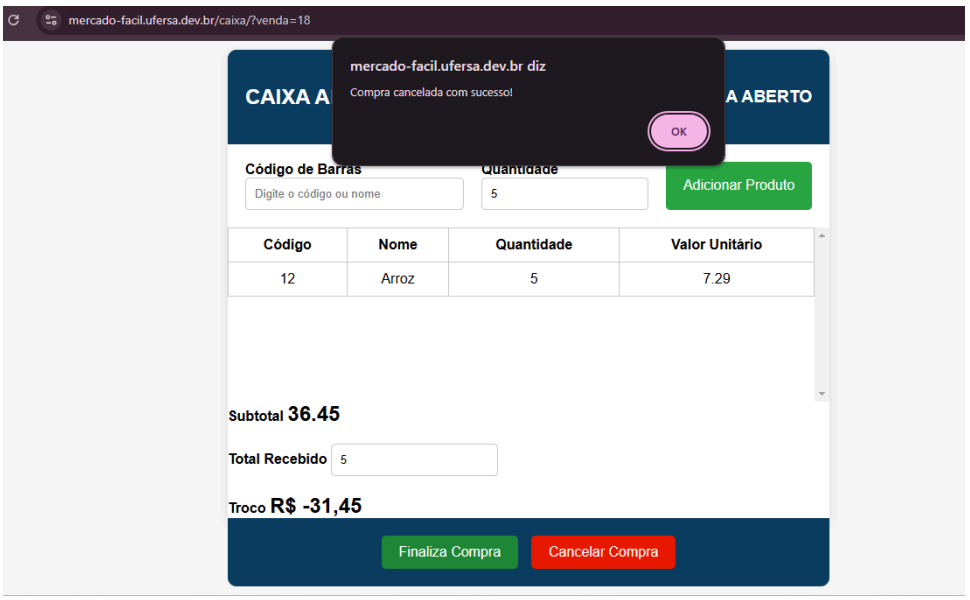


- **Cenário de Teste 3**: Cancelando uma compra
- **Resultado Esperado**: O sistema deve limpar a lista de produtos e voltar à tela de início sem registrar a venda..
- **Resultado Obtido**: O sistema retornou à tela de início, e a compra não foi registrada.
- **Status**: Aprovado
- **Evidências**: As imagens a seguir mostram o fluxo de exclusão de compra

- Imagem 04: Cancelando compra :



- Imagem 05: Após o primeiro alerta ao confirmar aparece a mensagem de confirmação da exclusão de compra:



- Imagem 06: Após acessar no “ok” somos redirecionados para a página inicial e a compra é cancelada..

- **Cenário de Teste 4**: Fluxo de Atualização do Estoque.
- **Resultado Esperado**: O sistema deve atualizar corretamente o estoque após a realização de uma venda e impedir vendas que excedam a quantidade disponível no estoque.
- **Resultado Obtido**: O sistema atualiza corretamente o estoque e não deixou cadastrar uma nova venda que fosse uma quantidade maior que a disponível no estoque, retornando uma mensagem de alerta.
- **Status**: Aprovado
- **Evidências**: As imagens a seguir mostram o fluxo da atualização de estoque.

- Imagem 07: Imagem da quantidade disponível no estoque antes de um cadastro de compra.

- Imagem 08: Após uma venda cadastrada o produto é automaticamente atualizado no estoque:

2. FLUXO DOS TESTES DE UNIDADE

Objetivo: Validar classes do código para garantir que elas funcionem corretamente de forma isolada.

- Imagem A:

```python
def test_create_produto(self):
        """Testa se um novo produto pode ser criado"""
        data = {
            "nome": "Produto Novo",
            "codigo": 456,
            "descricao": "Novo produto",
            "valor_venda": "200.00",
        }
        response = self.client.post(self.url_list, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Produto.objects.count(), 2)
```

- Imagem B: Testa a criação de um novo produto, verificando se a resposta é 201 e se o banco contém 2 produtos.

```python
def test_delete_produto(self):
        """Testa se um produto pode ser deletado"""
        url_detail = f'/api/produtos/{self.produto.pk}/'
        response = self.client.delete(url_detail)
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
        self.assertEqual(Produto.objects.count(), 0)
```

- Imagem C: Testa a busca de um produto, confirmando o status 200 e que 1 produto é retornado com base no termo pesquisado

```python
def test_search_produto(self):
        """Testa se a busca de produto está funcionando"""
        response = self.client.get(self.url_list, {'search': 'Produto Teste'})
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data), 1)
```

- Imagem D: Testa a atualização de um produto existente, confirmando a resposta 200 e que o nome foi alterado corretamente no banco.

```python
def test_update_produto(self):
        """Testa se um produto pode ser atualizado"""
        url_detail = f'/api/produtos/{self.produto.pk}/'
        data = {
            "nome": "Produto Atualizado",
            "codigo": 789,
            "descricao": "Descrição atualizada",
            "valor_venda": "150.00",
        }
        response = self.client.put(url_detail, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.produto.refresh_from_db()
        self.assertEqual(self.produto.nome, "Produto Atualizado")
```

- Imagem E: Verifica se uma movimentação de estoque pode ser deletada, garantindo o status 204 e que o banco fica sem movimentações.

```python
def test_delete_estoque(self):
    """Testa se uma movimentação de estoque pode ser deletada"""
    url_detail = f'/api/estoque/{self.estoque.pk}/'
    response = self.client.delete(url_detail)
    self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
    self.assertEqual(Estoque.objects.count(), 0)
```

- Imagem F: Testa a atualização de uma movimentação de estoque existente, confirmando a resposta 200 e que os atributos de peso e quantidade foram atualizados corretamente.

```python
def test_update_estoque(self):
        """Testa se uma movimentação de estoque pode ser atualizada"""
        url_detail = f'/api/estoque/{self.estoque.pk}/'
        data = {
            "produto": self.produto.pk,
            "peso": "15.00",
            "valor_compra": "60.00",
            "quantidade": 80,
            "vencimento": "2025-01-01"
        }
        response = self.client.put(url_detail, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.estoque.refresh_from_db()
        self.assertEqual(self.estoque.peso, 15.00)
        self.assertEqual(self.estoque.quantidade, 80)
```

- Imagem G: **`test_list_estoques`** - Verifica se a listagem de estoques retorna o status 200 e confirma que há 1 movimentação de estoque cadastrada. **`test_create_estoque`** - Testa a criação de uma nova movimentação de estoque, verificando a resposta 201 e se há 2 movimentações no banco.

```python
def test_list_estoques(self):
        """Testa se a listagem de estoques está funcionando"""
        response = self.client.get(self.url_list)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data), 1)

    def test_create_estoque(self):
        """Testa se uma nova movimentação de estoque pode ser criada"""
        data = {
            "produto": self.produto.pk,
            "peso": "5.75",
            "valor_compra": "25.00",
            "quantidade": 50,
            "vencimento": "2024-12-31"
        }
        response = self.client.post(self.url_list, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Estoque.objects.count(), 2)
```

- Imagem H: Testa a exclusão de um item de venda, garantindo que o valor total da venda é atualizado para 0 após a remoção.

```python
def test_delete_item_venda(self):
        """Testa se um item de venda pode ser deletado e o valor total da venda atualizado"""
        item_venda = ItemVenda.objects.create(
            venda=self.venda,
            produto=self.produto,
            quantidade=2,
            valor_unitario=100.00
        )
        url_detail_item_venda = f'/api/item-venda/{item_venda.pk}/'

        response = self.client.delete(url_detail_item_venda)
        self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)

        self.venda.refresh_from_db()
        self.assertEqual(self.venda.valor_total, 0.00)
```

- Imagem I: Verifica se o valor total da venda é atualizado corretamente para 200.00 após adicionar um item.

```python
def test_valor_total_atualizado_apos_item_venda(self):
        """Testa se o valor total da venda é atualizado corretamente após adicionar um item"""
        # Verifica o valor total inicial da venda
        self.assertEqual(self.venda.valor_total, 0)

        # Dados do item de venda
        data = {
            "venda": self.venda.pk,
            "produto": self.produto.pk,
            "quantidade": 2,
            "valor_unitario": "100.00"
        }

        response = self.client.post(self.url_list_itens_venda, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)

        self.venda.refresh_from_db()

        self.assertEqual(self.venda.valor_total, 200.00)
```

- Imagem J: Testa a atualização de um item de venda, verificando se o valor total é recalculado corretamente para 300.00 após a alteração da quantidade.

```python
def test_update_item_venda(self):
        """Testa se um item de venda pode ser atualizado e o valor total da venda recalculado"""
        item_venda = ItemVenda.objects.create(
            venda=self.venda,
            produto=self.produto,
            quantidade=2,
            valor_unitario=100.00
        )
        url_detail_item_venda = f'/api/item-venda/{item_venda.pk}/'

        data = {
            "venda": self.venda.pk,
            "produto": self.produto.pk,
            "quantidade": 3,
            "valor_unitario": "100.00"
        }
        response = self.client.put(url_detail_item_venda, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)

        self.venda.refresh_from_db()
        self.assertEqual(self.venda.valor_total, 300.00)
```

- Imagem K: Verifica se um item de venda pode ser criado e se o valor total da venda é atualizado, retornando status 201 e o valor correto de 200.00.

```python
def test_create_item_venda(self):
        """Testa se um novo item de venda pode ser criado e o valor total da venda atualizado"""
        data = {
            "venda": self.venda.pk,
            "produto": self.produto.pk,
            "quantidade": 2,
            "valor_unitario": "100.00"
        }
        response = self.client.post(self.url_list_itens_venda, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(ItemVenda.objects.count(), 1)

        self.venda.refresh_from_db()
        self.assertEqual(self.venda.valor_total, 200.00)
```

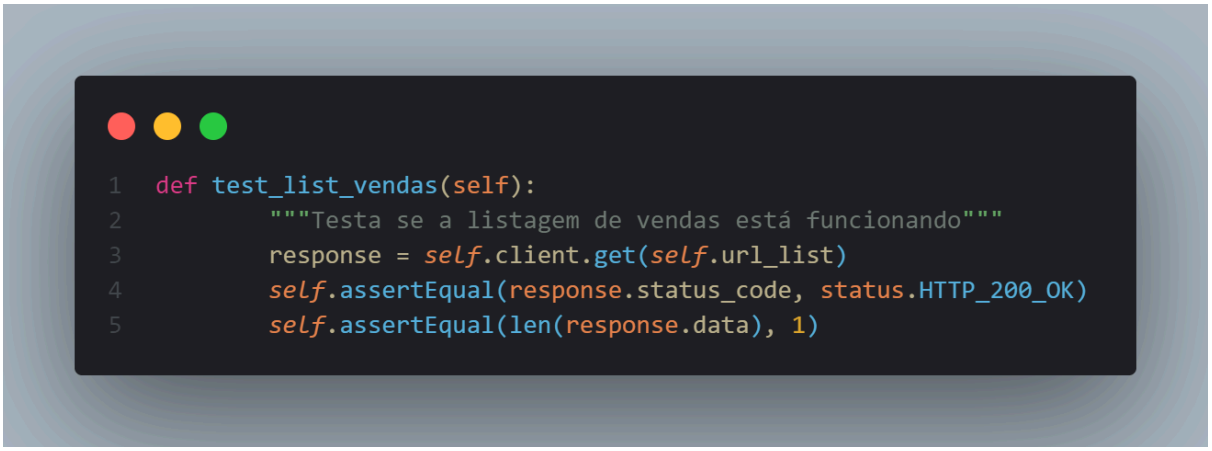
- Imagem L: Testa a rejeição de um item de venda com valor unitário inválido, retornando status 400 e nenhuma inserção no banco.

```python
def test_create_venda_valor_invalido(self):
        """Testa se a criação de uma venda com valor inválido é rejeitada"""
        data = {
            "vendedor_content_type": self.vendedor_content_type.id,
            "vendedor_object_id": self.user.id,
            "valor_total": "-50.00"
        }
        response = self.client.post(self.url_list, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
        self.assertIn('valor_total', response.data)
        self.assertEqual(Venda.objects.count(), 1)
```

- Imagem M: Testa a criação de uma nova venda, verificando o status 201 e se há duas vendas registradas.

```python
def test_create_venda(self):
        """Testa se uma nova venda pode ser criada"""
        data = {
            "vendedor_content_type": self.vendedor_content_type.id,
            "vendedor_object_id": self.user.id,
            "valor_total": "200.00"
        }
        response = self.client.post(self.url_list, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Venda.objects.count(), 2)
```

- Imagem N: Verifica se a listagem de vendas funciona corretamente, retornando status 200 e uma venda cadastrada.



```python
def test_list_vendas(self):
        """Testa se a listagem de vendas está funcionando"""
        response = self.client.get(self.url_list)
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data), 1)
```

- Imagem O: Setup inicial dos testes de venda

```python
class VendaViewSetTest(APITestCase):

    def setUp(self):
        self.user = User.objects.create_user(username='vendedor', password='12345')
        self.client.login(username='vendedor', password='12345')

        self.vendedor_content_type = ContentType.objects.get_for_model(self.user)

        self.produto = Produto.objects.create(
            nome="Produto Teste", codigo=123, valor_venda=100.00
        )
        self.estoque = Estoque.objects.create(
            produto=self.produto,peso=1,valor_compra=10,quantidade=100,vencimento=datetime.datetime.now()
        )

        self.venda = Venda.objects.create(
            vendedor_content_type=self.vendedor_content_type,
            vendedor_object_id=self.user.id,
            valor_total=0
        )

        self.url_list = '/api/venda/'
        self.url_list_itens_venda = '/api/item-venda/'
```