



CENTRO UNIVERSITÁRIO CENAC  
SANTO AMARO

## Análise de Testes - Sistema Império das Bebidas

### TESTES DE SOFTWARE

- BRUNO HENRIQUE GOUVEIA DA SILVA
- Emanuel Vitor de Holanda Feitosa
- João Vitor Silva Rosa
- Vinicius Tenório Lira
- Vitor Geovani Melo Silva

## Plano de Testes - Funcionalidade: Gestão de Clientes

### Contextualização

O Sistema Império das Bebidas é uma aplicação web desenvolvida com Spring Boot para gerenciamento de vendas de bebidas. O sistema inclui funcionalidades como:

- Gestão de clientes
- Controle de produtos
- Registro de vendas
- Gerenciamento de usuários

### Objetivo da Atividade

Analisar e documentar os testes implementados para garantir a qualidade do software, focando na validação das regras de negócio e na integridade das operações CRUD.

### Análise do Sistema

Os módulos escolhidos para teste incluem:

1. [ClienteServiceTest.java]:
  - Testes de serviços relacionados a clientes
  - Validações de CRUD
  - Verificação de regras de negócio
2. [ClienteResourceTest.java]:
  - Testes de endpoints REST
  - Validação de requisições HTTP
  - Testes de segurança e autenticação

### Definição dos Cenários de Teste

Os testes cobrem os seguintes cenários:

#### Validações de Dados

- Nome: não pode ser vazio
- CNPJ: deve ter 14 dígitos
- Email: deve ser um email válido
- Telefone: formato válido

## Validações de Negócio

- CNPJ: validação de formato
- Cliente: não pode ser duplicado
- ID: deve existir para operações de atualização/exclusão

## Configurações de Segurança

- Todas as operações requerem autenticação
- Usuário deve ter role "ADMIN"
- CSRF protection ativada

## Ferramentas Utilizadas

- JUnit 5 para testes unitários
- Mockito para simulação de dependências
- MockMvc para testes de endpoints REST
- Spring Security Test para testes de autenticação

## Descrição dos Testes

### Escopo dos Testes

- Cadastro de cliente
- Consulta de cliente
- Atualização de cliente
- Exclusão de cliente
- Validações de dados

## Casos de Teste ClienteResourceTest

ID	Cenário	Entrada	Resultado Esperado	Condições	
CT01	Buscar cliente por ID com sucesso	ID: 1	Status: 200 OK Cliente com dados válidos	Cliente existe no sistema	<input checked="" type="checkbox"/>
CT02	Buscar cliente inexistente	ID: 99	Status: 404 (Not Found)	Cliente não existe	<input checked="" type="checkbox"/>
CT03	Criar cliente com sucesso	Nome: "João Silva" CNPJ: "27865757000102" Email: "joao@email.com"	Status: 201 (Created) Cliente criado com dados corretos	Dados válidos	<input checked="" type="checkbox"/>
CT04	Criar cliente com dados inválidos	Nome: "" CNPJ: "123" Email: "email-invalido"	Status: 400 (Bad Request) Mensagens de erro de validação	Dados inválidos	<input checked="" type="checkbox"/>
CT05	Atualizar cliente com sucesso	ID: 1 Dados atualizados do cliente	Status: 200 OK Cliente atualizado	Cliente existe	<input checked="" type="checkbox"/>
CT06	Excluir cliente com sucesso	ID: 1	Status: 204 No Content	Cliente existe	<input checked="" type="checkbox"/>

## Casos de Teste ClienteServiceTest

ID	Caso de Teste	Entrada	Resultado Esperado	Condições	
CT07	Criar cliente no serviço	Nome: "John Doe" CNPJ: "27865757000102" Email: " <a href="mailto:john@example.com">john@example.com</a> "	Cliente criado com sucesso	Dados válidos	<input checked="" type="checkbox"/>
CT08	Buscar cliente por ID no serviço	ID: 1	Cliente encontrado com dados corretos	Cliente existe	<input checked="" type="checkbox"/>
CT09	Buscar cliente inexistente no serviço	ID: 99	ResourceNotFoundException	Cliente não existe	<input checked="" type="checkbox"/>
CT10	Atualizar cliente no serviço	ID: 1 Novos dados do cliente	Cliente atualizado com novos dados	Cliente existe	<input checked="" type="checkbox"/>
CT11	Excluir cliente no serviço	ID: 1	Cliente excluído com sucesso	Cliente existe	<input checked="" type="checkbox"/>
CT12	Criar cliente com CNPJ inválido	Cliente com CNPJ: "12345"	IllegalArgumentException	CNPJ inválido	<input checked="" type="checkbox"/>

## Codigo ClienteResourceTest

```
class ClienteResourceTest {  
    @Autowired  
    private MockMvc mockMvc;  
    @MockBean  
    private ClienteService clienteService;  
    @Autowired  
    private ObjectMapper objectMapper;  
    private Cliente cliente;  
    private static final String VALID_CNPJ = "27865757000102";  
  
    @BeforeEach  
    void setUp() {  
        cliente = new Cliente();  
        cliente.setId(1L);  
        cliente.setNome("João Silva");  
        cliente.setCnpj(VALID_CNPJ);  
        cliente.setEmail("joao@email.com");  
        cliente.setTelefone("11999999999");}  
  
    // CT01 testGetClienteSuccess: Verifica se retorna 200 (OK) quando encontra o cliente  
    @Test  
    @WithMockUser(roles = "ADMIN")  
    void testGetClienteSuccess() throws Exception {  
        when(clienteService.findById(1L)).thenReturn(cliente);  
  
        mockMvc.perform(get("/api/clientes/1"))  
            .andExpect(status().isOk())  
            .andExpect(jsonPath("$.nome").value("João Silva"))  
            .andExpect(jsonPath("$.cnpj").value(VALID_CNPJ));}  
  
    // CT02 testGetClienteNotFound: Verifica se retorna 404 (Not Found) quando não encontra o cliente  
    @Test  
    @WithMockUser(roles = "ADMIN")  
    void testGetClienteNotFound() throws Exception {  
        when(clienteService.findById(99L))  
            .thenThrow(new ResourceNotFoundException("Cliente não encontrado"));  
  
        mockMvc.perform(get("/api/clientes/99"))  
            .andExpect(status().isNotFound());}  
  
    // CT03 testCreateClienteSuccess: Verifica se retorna 201 (Created) ao criar cliente válido  
    @Test  
    @WithMockUser(roles = "ADMIN")  
    void testCreateClienteSuccess() throws Exception {  
        when(clienteService.save(any(Cliente.class))).thenReturn(cliente);  
        mockMvc.perform(post("/api/clientes")  
            .with(csrf())  
            .contentType(MediaType.APPLICATION_JSON)  
            .content(objectMapper.writeValueAsString(cliente)))  
            .andExpect(status().isCreated())  
            .andExpect(jsonPath("$.nome").value("João Silva"))  
            .andExpect(jsonPath("$.cnpj").value(VALID_CNPJ));}
```

```

// CT04 testCreateClienteValidation: Verifica se retorna 400 (Bad Request) ao tentar criar cliente
inválido
@Test
@WithMockUser(roles = "ADMIN")
void testCreateClienteValidation() throws Exception {
    Cliente clienteInvalido = new Cliente();
    clienteInvalido.setId(1L);
    clienteInvalido.setNome(""); // nome vazio para validação
    clienteInvalido.setCnpj("123"); // CNPJ inválido
    clienteInvalido.setEmail("email-invalido"); // email inválido

    when(clienteService.save(any(Cliente.class)))
        .thenThrow(new IllegalArgumentException("CNPJ inválido"));

    mockMvc.perform(post("/api/clientes")
        .with(csrf())
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(clienteInvalido)))
        .andExpect(status().isBadRequest())
        .andExpect(content().contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath("$.error").value("CNPJ inválido"));
}

```

```

// CT05 testUpdateClienteSuccess: Verifica se retorna 200 (OK) ao atualizar cliente válido
@Test
@WithMockUser(roles = "ADMIN")
void testUpdateClienteSuccess() throws Exception {
    when(clienteService.update(eq(1L), any(Cliente.class))).thenReturn(cliente);

    mockMvc.perform(put("/api/clientes/1")
        .with(csrf())
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(cliente)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.nome").value("João Silva"))
        .andExpect(jsonPath("$.cnpj").value(VALID_CNPJ));
}

```

```

//CT06 testDeleteClienteSuccess: Verifica se retorna 204 (No Content) ao deletar cliente
@Test
@WithMockUser(roles = "ADMIN")
void testDeleteClienteSuccess() throws Exception {
    doNothing().when(clienteService).delete(1L);

    mockMvc.perform(delete("/api/clientes/1")
        .with(csrf()))
        .andExpect(status().isNoContent());
}

```

## Código ClienteServiceTest

```
class ClienteServiceTest {  
    @Mock  
    private ClienteRepository clienteRepository;  
    @InjectMocks  
    private ClienteService clienteService;  
    private Cliente cliente;  
    // CNPJ válido para testes (27.865.757/0001-02)  
    private static final String VALID_CNPJ = "27865757000102";  
  
    // Configuração inicial do cliente para os testes  
    @BeforeEach  
    void setUp() {  
        cliente = new Cliente();  
        cliente.setId(1L);  
        cliente.setNome("John Doe");  
        cliente.setCnpj(VALID_CNPJ);  
        cliente.setEmail("john@example.com");  
        cliente.setTelefone("11999999999");}  
  
    // CT07 Verifica a criação de um novo cliente Valida nome, CNPJ e chamada do repository  
    @Test  
    void testCreateCliente() {  
        // Arrange  
        when(clienteRepository.save(any(Cliente.class))).thenReturn(cliente);  
  
        // Act  
        Cliente createdCliente = clienteService.createCliente(cliente);  
  
        // Assert  
        assertNotNull(createdCliente);  
        assertEquals("John Doe", createdCliente.getNome());  
        assertEquals(VALID_CNPJ, createdCliente.getCnpj());  
        verify(clienteRepository, times(1)).save(any(Cliente.class));  
    }  
  
    // CT08 Verifica a busca de um cliente por ID, valida o retorno e chamada do repository  
    @Test  
    void testGetClienteById() {  
        // Arrange  
        when(clienteRepository.findById(1L)).thenReturn(java.util.Optional.of(cliente));  
  
        // Act  
        Cliente foundCliente = clienteService.getClienteById(1L);  
  
        // Assert  
        assertNotNull(foundCliente);  
        assertEquals(cliente.getId(), foundCliente.getId());  
        assertEquals(cliente.getNome(), foundCliente.getNome());  
        verify(clienteRepository, times(1)).findById(1L);}
```

```

// CT09 Verifica a busca de um cliente por ID não encontrado, valida a exceção lançada
@Test
void testGetClienteById_NotFound() {
    // Arrange
    when(clienteRepository.findById(99L)).thenReturn(java.util.Optional.empty());

    // Act & Assert
    assertThrows(ResourceNotFoundException.class, () -> {
        clienteService.getClienteById(99L);});}

// CT10 Verifica a atualização de um cliente, valida os dados atualizados e chamada do repository
@Test
void testUpdateCliente() {
    // Arrange
    Cliente updatedInfo = new Cliente();
    updatedInfo.setId(1L);
    updatedInfo.setNome("John Smith");
    updatedInfo.setCnpj(VALID_CNPJ);
    updatedInfo.setEmail("john.smith@example.com");

    when(clienteRepository.findById(1L)).thenReturn(java.util.Optional.of(cliente));
    when(clienteRepository.save(any(Cliente.class))).thenReturn(updatedInfo);

    // Act
    Cliente result = clienteService.update(1L, updatedInfo);

    // Assert
    assertEquals("John Smith", result.getNome());
    assertEquals("john.smith@example.com", result.getEmail());
    verify(clienteRepository).save(any(Cliente.class));}

// CT11 Testa exclusão de cliente Verifica se repository.delete é chamado
@Test
void testDeleteCliente() {
    // Arrange
    when(clienteRepository.findById(1L)).thenReturn(java.util.Optional.of(cliente));
    doNothing().when(clienteRepository).delete(cliente);

    // Act
    clienteService.deleteCliente(1L);

    // Assert
    verify(clienteRepository).delete(cliente);}

// CT12 Testa validação de CNPJ inválidoVerifica se lança IllegalArgumentException
@Test
void testCreateClienteWithInvalidCNPJ() {
    // Arrange
    Cliente invalidCliente = new Cliente();
    invalidCliente.setNome("Test");
    invalidCliente.setCnpj("12345"); // CNPJ inválido

    // Act & Assert
    assertThrows(IllegalArgumentException.class, () -> {
        clienteService.createCliente(invalidCliente); });

}

```

## Dependências de Teste

```
<!-- JUnit 5 -->

<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.9.3</version>
    <scope>test</scope>
</dependency>

<!-- Spring Boot Test -->

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

<!-- Spring Security Test -->

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>

<!-- Mockito -->

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <scope>test</scope>
</dependency>
```

## Execução dos Testes

- Todos os testes do ClienteService foram executados com sucesso
- Os testes de validação identificaram corretamente dados inválidos
- Os testes de autenticação funcionaram conforme esperado

## Cobertura de Código

- Alta cobertura nas operações CRUD básicas
- Boa cobertura nas validações de negócio
- Cobertura adequada nos endpoints REST

## Análise Crítica

Pontos Fortes: Testes bem organizados e estruturados, Boa cobertura das regras de negócio e Validações robustas de dados.

Limitações: Ausência de testes para cenários complexos, Cobertura limitada de casos de erro e Falta de testes de integração mais abrangentes.

## Conclusão

### Aprendizados

- Implementação efetiva de testes unitários com JUnit 5
- Uso adequado de mocks para isolar dependências
- Importância da validação de dados em diferentes camadas

## Desafios e Soluções

1. **Desafio:** Teste de validações complexas
  - o **Solução:** Uso de @Valid e validações personalizadas
2. **Desafio:** Simulação de dependências
  - o **Solução:** Implementação efetiva do Mockito

## Recomendações Finais

1. Implementar mais testes de integração
2. Aumentar cobertura de casos de erro
3. Adicionar testes de performance
4. Incluir testes para cenários de concorrência

## Referências

- Documentação do Spring Boot Testing
- Documentação do JUnit 5
- Documentação do Mockito
- Spring Security Testing Guide
- Java Bean Validation Documentation