PLANO DE TESTES — Toshiro-Shibakita Market Online

(Conforme modelo ISTQB – estruturado e profissional)

---

1. Introdução

Este Plano de Testes descreve a estratégia, o escopo, os recursos, o cronograma e os processos que serão utilizados para testar a plataforma de e-commerce “Toshiro-Shibakita Market Online”.
O objetivo é garantir que o sistema atenda aos requisitos funcionais e não funcionais, assegurando qualidade, confiabilidade e segurança antes da publicação.

---

2. Escopo dos Testes

2.1 Incluído

Os testes abrangem:

- Catálogo digital de produtos
- Carrinho de compras
- Cálculo de frete e agendamento
- Checkout (pagamento e finalização)
- Integração com estoque
- Notificações (email/SMS)
- Perfil do cliente e histórico de pedidos
- Painel administrativo de produtos e pedidos

- APIs expostas (REST)

2.1 Excluído

- Infraestrutura física das lojas
- Sistemas legados internos não integrados
- Testes de performance avançados (serão executados em ciclo separado)
- Testes de segurança avançados (pentest externo)

---

3. Abordagem de Testes

A abordagem adotada será orientada por risco (Risk-Based Testing), contemplando vários níveis de testes:

3.1 Testes Unitários

- Garantir corretude de serviços, regras e módulos isolados
- Executados automaticamente via CI
- Frameworks: JUnit, Mockito

3.2 Testes de Integração

- Fluxo entre módulos internos (estoque ↔ catálogo ↔ checkout)
- Banco de dados real ou containerizado
- Foco em falhas de comunicação

3.3 Testes de API

- Endpoints REST

- Autenticação/token
- Validação de status codes, payload e performance básica
- Ferramentas: Postman/Newman, RestAssured

3.4 Testes Funcionais (UI)

- Navegação
- Carrinho, checkout, agendamentos
- Comportamento responsivo
- Selenium/Playwright

3.5 Testes de Aceitação

- Baseados nas histórias de usuário
- Via BDD (Cucumber)

3.6 Testes Não Funcionais

- Usabilidade (heurísticas + UX)
- Segurança (OWASP Top 10 básica)
- Performance (carga média)

---

4. Critérios de Aceitação dos Testes

4.1 Critérios para Começar

- Requisitos validados
- Ambiente de teste disponível

- Dados de teste preparados
- Pipelines de CI funcionando

4.2 Critérios para Concluir

- 100% dos testes críticos executados
- 95% de testes funcionais aprovados
- 0 defeitos críticos ou altos em aberto
- Documentação atualizada
- Aprovação do PO

---

5. Itens de Teste

- Página inicial da loja
- Catálogo e busca
- Filtros por categoria
- Página de detalhes do produto
- Carrinho de compras
- Cálculo de frete
- Agendamento de entrega
- Login e cadastro
- Checkout (Pix, cartão, carteiras digitais)
- Painel administrativo
- API de estoque

- API de pedidos

---

6. Itens *Não* Testados

- Relatórios avançados do sistema administrativo
- Sistemas internos não integrados
- Promos automáticas com IA (futuro)

---

7. Ambiente de Testes

7.1 Arquitetura

- Backend: Java 17 + Spring Boot
- Banco: PostgreSQL (container Docker)
- Front: React + Vite
- Integração estoque: API REST

7.2 Infraestrutura

- Ambiente exclusivo “test”
- Containers via Docker Compose
- Testes automatizados via GitHub Actions

---

8. Responsabilidades

● QA Engineer

- Planejar todas as atividades de testes.
- Executar testes funcionais, integração e regressão.
- Registrar e acompanhar defeitos.
- Garantir que critérios de aceitação estejam atendidos.

● Desenvolvedores

- Corrigir defeitos reportados pelo QA.
- Criar e manter testes unitários automatizados.
- Apoiar testes de integração quando necessário.
- Implementar melhorias relacionadas à qualidade.

● Product Owner (PO)

- Validar requisitos antes do desenvolvimento.
- Aprovar histórias concluídas.
- Priorizar backlog considerando riscos e valor de negócio.
- Decidir sobre aceitação final do produto.

● DevOps

- Configurar e manter pipelines de CI/CD.
- Gerenciar ambientes de desenvolvimento, testes e homologação.
- Assegurar observabilidade (logs, monitoramento, alertas).
- Garantir versionamento adequado e automações do fluxo.

---

9. Riscos e Mitigações

- Falhas na integração de estoque → *monitoramento + fallback*
- Alta demanda sem testes de carga → *rodar stress tests antes do lançamento*
- Checkout falhar → *gateways redundantes*
- Falhas de comunicação API → *timeouts + retentativas*

---

10. Estratégia de Reporte de Defeitos

- Ferramenta: GitHub Issues
- Severidade (Critical, High, Medium, Low)
- Defeitos terão SLA:
    - Críticos: 24 horas
    - Altos: 48 horas
    - Médios/baixa prioridade: backlog

---

11. Métricas de Teste

- % de casos de teste executados
- % de casos aprovados
- Taxa de defeitos regressivos
- Tempo médio para correção
- Cobertura de código (objetivo 80%)

---

12. Cronograma Geral

● Planejamento

- Período: **Semana 1**

● Desenvolvimento + Testes Unitários

- Período: **Semanas 2 a 5**

● Testes de Integração

- Período: **Semana 6**

● Testes Funcionais

- Período: **Semanas 7 e 8**

● Testes de Aceitação (UAT)

- Período: **Semana 9**

● Homologação Final

- Período: **Semana 10**

---

13. Aprovação

- PO
- Líder Técnico
- QA Lead

- DevOps