1. Test Strategy Document\*\*

#### \*\*Testing Approach Across Different Application Layers:\*\*

To ensure comprehensive testing of the e-commerce application, the following layers will be addressed:

- \*\*Web Frontend\*\*:

- \*\*Functional Testing\*\*: Focus on ensuring that all user interactions, such as adding products to the cart, checking out, and user login/logout, work correctly.

- \*\*Cross-browser Testing\*\*: Test across different browsers (Chrome, Firefox, Safari, Edge) to ensure compatibility.

- \*\*Responsive Design\*\*: Ensure the frontend is fully responsive across different devices (mobile, tablet, desktop).

- \*\*Mobile Apps (iOS/Android)\*\*:

- \*\*Functional Testing\*\*: Validate that the mobile app performs the same key functions as the web frontend, including browsing products, checking out, and payment.

- \*\*UI/UX Testing\*\*: Ensure the mobile UI is intuitive, easy to use, and provides a seamless user experience.

- \*\*Performance and Load Testing\*\*: Ensure mobile apps handle varying load levels.

- \*\*Platform-Specific Testing\*\*: Test iOS and Android-specific behaviors, including push notifications, permissions, and OS-specific functions.

- \*\*RESTful API Backend\*\*:

- \*\*Unit Testing\*\*: Test individual API endpoints for expected behavior.

- \*\*Integration Testing\*\*: Ensure that different services work together as expected (e.g., frontend to backend communication).

- \*\*Load and Performance Testing\*\*: Validate that the API can handle high traffic, ensuring scalability.

- \*\*Security Testing\*\*: Focus on validating security vulnerabilities (e.g., data leakage, SQL injection).

- \*\*Payment Processing Integration\*\*:

- \*\*Functional Testing\*\*: Test different payment flows (credit card, PayPal, etc.) to ensure the payment gateway processes payments accurately.

- \*\*Security and Compliance Testing\*\*: Test against standards such as PCI DSS to ensure the payment process is secure and compliant.

- \*\*Negative Testing\*\*: Simulate failed transactions, network issues, or declined payments to ensure appropriate handling.

- \*\*Inventory Management System\*\*:

- \*\*Data Consistency Testing\*\*: Ensure that the inventory count is consistent across the system.

- \*\*Integration Testing\*\*: Ensure seamless communication between the e-commerce platform and the inventory system.

- \*\*Performance Testing\*\*: Ensure the system scales and performs well with increasing product quantities.

#### \*\*Risk Assessment and Test Prioritization\*\*:

1. \*\*High-Risk Areas\*\*:

- \*\*Payment Processing\*\*: Since payments directly involve financial transactions, this is the highest priority for testing. Ensure security, data integrity, and compliance.

- \*\*Inventory Management\*\*: This is critical to ensure that product availability is correctly reflected, preventing overselling.

- \*\*API Backend\*\*: Any failure in the API layer can cause disruptions across the entire platform.

2. \*\*Medium-Risk Areas\*\*:

- \*\*Mobile Apps\*\*: Ensure compatibility and usability, but this is secondary to payment and inventory.

- \*\*Web Frontend\*\*: While important, the web frontend generally follows functional backend and critical integrations.

3. \*\*Low-Risk Areas\*\*:

- \*\*Cross-Browser Testing\*\*: This can be deprioritized if the core functionality is intact across the main browsers (Chrome/Firefox).

- \*\*Performance Testing\*\*: Load testing should be prioritized once basic functionality has been validated.

#### \*\*Test Environment Requirements\*\*:

- \*\*Web and Mobile Testing\*\*:

- Cloud-based test environments for cross-browser and mobile app testing (e.g., Sauce Labs, BrowserStack).

- Devices/Emulators: Real devices for mobile testing or cloud-based services for both iOS/Android testing.

- \*\*API Testing\*\*:

- Dedicated test server or mock API endpoints to validate the backend independently of the frontend.

- \*\*Payment Integration\*\*:

- Integration with a sandbox version of the payment gateway for testing.

- \*\*Database\*\*:

- Separate test database to avoid data corruption in the production environment.

#### \*\*Test Data Management Approach\*\*:

- \*\*Test Data Generation\*\*: Use tools like Faker or custom scripts to generate realistic test data for users, orders, and payments.

- \*\*Test Data Anonymization\*\*: For security testing, anonymize sensitive data used in test environments.

- \*\*Data Seeding\*\*: Pre-seed the test environment with valid, invalid, and boundary test cases for testing API, inventory, and payment systems.

\*\*2. Test Automation Architecture Diagram\*\*

The following elements will be covered in the automation architecture:

1. \*\*Tool Selection and Justification\*\*:

- \*\*Selenium\*\*: For automating web frontend testing. Selenium is a widely used and robust tool for automating web interactions across multiple browsers.

- \*\*Appium\*\*: For mobile application automation across both Android and iOS platforms. It supports cross-platform testing.

- \*\*Postman/Newman\*\*: For API testing. Postman allows for easy API request creation, and Newman is used to integrate it into the CI/CD pipeline.

- \*\*Jest/Mocha\*\*: For unit and integration testing in both frontend and backend.

- \*\*JMeter\*\*: For performance/load testing across the API and web applications.

2. \*\*Framework Design\*\*:

- \*\*Test Automation Framework\*\*:

- \*\*Modular Architecture\*\*: Separate modules for web, mobile, API, and integration tests, ensuring reusability and ease of maintenance.

- \*\*Page Object Model (POM)\*\* for UI test automation: Helps separate the test logic from the UI structure, improving maintainability.

- \*\*Data-Driven Testing\*\*: Use of external data sources (e.g., CSV, JSON) to run the same tests with different data inputs.

3. \*\*Integration Points Between Different Testing Layers\*\*:

- \*\*API Tests\*\*: Automated API tests (Postman/Newman) will integrate into the CI/CD pipeline, providing continuous feedback on the backend.

- \*\*Web and Mobile Tests\*\*: Selenium/Appium tests are integrated with the pipeline to run automatically during code changes.

- \*\*End-to-End Tests\*\*: Combined test cases across the API, web, and mobile platforms to validate complete user workflows.

4. \*\*CI/CD Pipeline Integration\*\*:

- \*\*Continuous Integration (CI)\*\*: Integration of automated tests into Jenkins/GitLab CI, triggered with every code commit or pull request.

- \*\*Stages\*\*: Code build → Unit tests → Integration tests → UI Tests → Performance tests → Deploy.

- \*\*Reporting and Alerts\*\*: Slack/Email notifications for failed tests or critical issues.

- \*\*Continuous Delivery (CD)\*\*: Automate deployments to a staging environment for validation before production.