**Test Plan for Order Management System (OMS)**

**Project Name**: Order Management System (OMS) for E-commerce

**Test Plan Version**: 1.0

**Test Plan Author**: Avital Z

**Test Plan Date**:16.3.2025

**Test Plan Approved By**: QA Team Leader

### **1. Introduction**

This document outlines the test strategy and scope for the Order Management System (OMS) of the e-commerce platform. The system facilitates users in placing, processing, and tracking their orders, as well as allowing admins to update order statuses and manage inventory.

**2.Scope**

* **API Testing:** Validate the REST API for login authorization, authentication, order placement, order status updates, and MongoDB interaction.
* **UI Testing:** Validate the customer and admin user interfaces.
* **MongoDB Testing:** Ensure data integrity and consistency with CRUD operations.
* **Authentication & Authorization:** Verify login, token-based authentication (JWT), and role-based access control (RBAC).
* **Order Placement & Workflow:** Test end-to-end order placement, processing, shipping, and delivery. Verify email notification delivery to the customer for each status change.
* **Race Conditions:** Test scenarios involving simultaneous updates and order deletions by admins.
* **Cross-Browser Testing:** Ensure UI compatibility across multiple browsers and devices.
* **Performance Testing:** Ensure the system can handle a high volume of concurrent orders and user actions without performance degradation.

**3.Test Scenarios**

The scenarios outline the key workflows to test, including CRUD operations, UI status updates, login flows, race conditions, and edge cases.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **Test Scenario** | **Description** | **Test Type** | **Test Method** | **Assessment Time** | **Cross-Browser Testing** |
| 1 | API: CRUD Operations on Orders | Verify that order creation, retrieval, update, and deletion APIs work as expected. | Automated | pytest + Requests + MongoDB validation | 45 min | No |
| 2 | API: Order Status Update Validation | Test order status update from "Pending" to "Shipped" and validate MongoDB updates. | Automated | pytest (API request + MongoDB validation) | 20 min | No |
| 3 | API: Error Handling (Invalid Order ID, Token Expiry) | Ensure proper error handling for invalid order IDs, expired tokens, and malformed requests. | Automated | pytest (API testing) | 20 min | No |
| 4 | Authentication: Customer vs Admin Login | Verify login functionality for both customers and admins, ensuring correct access levels. | Automated/Manual | pytest (API testing), manual UI testing | 15 min | Yes |
| 5 | UI: Order Status Update after Admin Action (Reflect on UI) | Ensure the UI correctly reflects order status updates after an admin action. | Automated | Selenium (UI Testing) + MongoDB validation | 30 min | Yes |
| 6 | Admin Access Control | Confirm that only admins can access the admin panel while customers are restricted. | Manual | Manual (UI testing) | 20 min | Yes |
| 7 | Race Condition - Status Update | |  |  | | --- | --- | | Test multiple admin updates to the same order simultaneously for data integrity. |  | | Manual | Manual (Simultaneous UI/API actions, timestamp validation) + MongoDB check | 40 min | No |
| 8 | Order Placement with Invalid Stock | Ensure the system prevents order placement when stock is unavailable. | Manual | UI + Backend validation (MongoDB checks) | 30 min | Yes |
| 9 | UI: Order Pagination and Filters | Test order history pagination and filtering functionality, including edge cases. | Manual | Manual (UI testing, API validation for pagination) | 15 min | Yes |
| 10 | Cross-Browser: UI Compatibility Check | Verify UI functionality across multiple browsers and mobile devices. | Manual | Manual (UI testing on Chrome, Firefox, Safari, Edge) | 40 min | Yes |

**4. Test Data**

Test Data will include the following:

* **Pre-created Orders:** Test orders with different statuses: "Pending", "Processing", "Shipped", "Delivered".
* **Products:** Sample products with different stock quantities (e.g., Laptop, Mouse, etc.).
* **Users:** A set of test users (both customer and admin) for login and order placement.
* **Test User Roles:** Include customer users, admin users, and invalid users for authentication and authorization testing.

### **5. Test Tools**

**Test Automation:**

* **Pytest:** For automated testing of API endpoints.
* **Postman:** For manual API testing and validation.
* **MongoDB Compass:** For verifying data in MongoDB.
* **Selenium:** For UI testing of React-based front end.
* **Jenkins:** For CI/CD integration.

**Bug Reporting Tools:**

* **Jira:** For tracking bugs and test results.

### **6. Automation vs Manual Testing**

**Automated Testing:**

* **API Testing:**
  + CRUD Operations on orders: Test login authorization, order creation, fetching, updating, and deletion.
  + Pytest parametrize example for creating orders with multiple sets of test data.
  + Updating Order Status: Test updating an order status from "Pending" to "Processing", "Shipped", and "Delivered".
  + MongoDB Data Consistency: Validate that data in MongoDB remains consistent after updates and deletions.
* **UI Testing:**
  + Verifying UI Updates after Status Change: Test that the order status change in the admin panel reflects correctly on the user interface using Selenium.
  + Login Flow: Test login for multiple user roles (customer, admin). pytest parametrize example for login with different roles.

**Manual Testing:**

* **Admin Panel Access:**
  + Verifying Access Restrictions for Customers vs. Admins: Test that customers cannot access the Admin Panel, and admins can access restricted sections like order management and status updates.
  + For admin access testing: Try logging in as a customer and attempt to access the admin panel. Ensure the system denies access and shows a proper error message.
* **Role/Authentication:**
  + Test User Role Access: Test that only admin users can update the order status, and customers can only view their orders.
  + Manual test steps: Login as a customer. Attempt to access order status updates in the admin panel. Ensure access is denied, and the system responds with a "Forbidden" or "Unauthorized" error.
* **Race Conditions:**
  + Manually test race condition scenarios, such as multiple admins trying to update the same order status or delete the same order at the same time.
* **Edge Cases:**
  + Test order placement under various edge cases:
    - Invalid stock: Trying to place an order for more quantity than available in stock.
    - Missing user info: Place an order without providing mandatory user details (e.g., missing address or payment details).
    - Negative stock: Try to place an order with a negative or zero quantity in stock.

### **7. Test Execution Strategy**

**Test Execution Phases:**

* **Initial Setup:**
  + Set up a test environment (API, UI, and MongoDB).
  + Configure testing tools (Pytest, Postman, etc.).
* **Test Case Execution:**
  + Execute functional and regression tests.
  + Verify API responses and MongoDB data consistency.
  + Execute UI tests to ensure front-end correctness.
* **Bug Reporting:**
  + Document any issues found during testing and report them to the dev team.
* **Post-Execution:**
  + Validate fixes for any bugs and re-execute tests.

### **8. Exit Criteria**

Testing will be considered complete when:

* All critical and high-priority test scenarios have passed.
* No major bugs are found in the system's core functionalities.
* Critical high priority bugs have been fixed and verified in the test environment.

### **9. Risk Mitigation**

* **Race Condition Handling:** Implement locks or other mechanisms to prevent simultaneous updates or deletions by multiple admins.
* **User Error Handling:** Ensure that invalid actions (e.g., invalid order status update) are caught and appropriate error messages are shown.
* **UI Consistency:** Test under different environments (browsers, mobile, desktop) to ensure that UI is responsive and consistent.
* **Security Testing:** Ensure the system is protected against common security vulnerabilities, such as SQL injection, cross-site scripting (XSS), and data leakage.

**10. Bug Reporting Format**

The following details should be included in the bug report:

* **Title:** Descriptive bug title, for example "UI Not Reflecting Order Status Change on Firefox"
* **Environment:**
  + **Operating System:** Windows/Mac/Linux/iOS/Android.
  + **Browser(s) Tested:** List the browsers where the bug was reproduced (Chrome Firefox).
  + **Version:** The version of the application where the bug was reproduced.
  + **Environment Configuration:** Development/Staging/Production
  + **URL/Path**
  + **Database**
* **Steps to Reproduce:** A clear set of steps to reproduce the issue.
* **Expected Behavior:** A description of what should happen.
* **Actual Behavior:** A description of what actually happened.
* **Severity:** Blocker, Critical, Major, Minor.
* **Priority:** Critical, High, Medium, Low.
* **Status:** Open, In Progress, Resolved, Closed.
* **Assigned To:** Developer responsible for fixing.
* **Regression:** Yes/No (Was this issue introduced in the current release?).
* **Reproduction Rate:** (Always, Sometimes, Rarely).
* **Additional Information:** Any other relevant information (screenshots, screen recordings, logs).
  + **Screenshots:** Attach any relevant screenshots.
  + **Screen Recording:** Include screen recording for better context, if applicable.
  + **logs** - attach logs from backend FastAPI logs, database, Network Logs: API request/response logs (captured via Postman or browser DevTools).

**Example Of A Bug Report Of Order Placement Fails with Invalid Stock on Mobile Chrome:**

* **Title:** Order Placement Fails with Invalid Stock on Mobile Chrome
* **Environment:**
  + **Operating System:** Android v11 Chrome
  + **Browser Tested:** Chrome, FireFox,Safari (Mobile - Bug reproduced).
  + **Environment Configuration:** Staging.
  + **Version:** 1.0.0
* **Steps to Reproduce:**
  + Go to the URL http://localhost:3000/user
  + Log in as a customer.
    - email [123@gmail.com](mailto:123@gmail.com)
    - password 123password
  + Add a product with insufficient stock to the cart.
  + Attempt to place an order by navigating to the URLhttp://localhost:3000/cart/checkout
  + Observe that the order does not proceed and an error message is shown.
* **Expected Behavior:** The system should display an appropriate error message, and the order should not proceed.
* **Actual Behavior:** No error message is shown, and the order fails silently.
* **Severity:** Critical
* **Priority:** Medium
* **Status:** Open
* **Assigned To:** Developer X
* **Regression:** No (New feature)
* **Reproduction Rate:** Always
* **Additional Information:**
  + **Screenshots:** Screenshot1, Screenshot2
  + **Screen Recording:** Recording1
  + **Browser Tested:** Android Chrome.
  + **Environment that the issue does not reproduce:** **OS:** Windows 10, macOS
    - Browsers: Chrome (Desktop), Firefox (Desktop), Safari (Desktop)
    - API URL: http://localhost:8000
    - Frontend URL: http://localhost:3000