Test Plan Document

1. Test Strategy and Approach

The QA strategy ensures comprehensive validation of both backend APIs and frontend applications. The approach includes functional, integration, usability, and regression testing. Testing will validate:

- **Backend (C# APIs):** User authentication, product management, and order processing.
- **Frontend (ReactJS):** User interface elements, data presentation, and workflows.
- **Non-Functional Testing:** Performance, security, and responsiveness.
- **Edge Cases:** Handling unexpected inputs and invalid operations.

2. Test Scope

- **In-Scope:**
  - Backend: All API endpoints defined in Swagger documentation.
  - Frontend: UI workflows such as login, product listing, and order management.
  - Cross-browser compatibility: Chrome, Firefox, and Edge.
- **Out-of-Scope:**
  - Integration with external third-party systems.
  - Stress and load testing.

3. Objectives

1. Validate the functionality and reliability of the backend APIs.
2. Ensure the frontend interacts seamlessly with backend APIs.
3. Test the user experience for responsiveness and accessibility.
4. Identify and document defects with actionable resolutions.

4. Resources

- **Tools:**
  - Postman for API testing.
  - Selenium for UI automation.
  - Browser Developer Tools for manual UI checks.
  - Axe for accessibility testing.
  - JIRA for defect tracking.
- **Test Environment:**
  - Backend: Local environment running on `http://localhost:5000`.
  - Frontend: React app hosted locally on `http://localhost:3000`.
- **Datasets:**
  - Predefined user credentials, test products, and order details.

5. Risks and Mitigation

Risk	Mitigation
API token expiration issues	Include automated token refresh in test scripts.
Stateless app clears data on restart	Preload data through automated scripts before tests.
Browser compatibility issues	Perform cross-browser testing on all major browsers.
Environment inconsistency	Standardize environment setup instructions for testers.

6. Deliverables

1. Comprehensive test plan document.
2. Detailed test cases for backend and frontend testing.
3. Execution report with test results and defect summaries.
4. Final test coverage analysis and recommendations.
5. Automation scripts and instructions for running automated tests.

Test Cases

Backend Test Cases

Test Case ID	Scenario	Steps	Expected Result
API-AUTH-001	Valid login	POST `/login` with valid credentials.	200 OK with token in response.
API-AUTH-002	Invalid login	POST `/login` with invalid credentials.	401 Unauthorized with error message.
API-PROD-001	Create product	POST `/products` with valid data.	201 Created with product ID.
API-PROD-002	Missing product data	POST `/products` with incomplete payload.	400 Bad Request with validation error.
API-PROD-003	Search products	GET `/products` with valid search query.	200 OK with matching products.
API-ORDER-001	Create order	POST `/orders` with valid details.	201 Created with order ID.

Test Case ID	Scenario	Steps	Expected Result
API-ORDER-002	Cancel order	PATCH `/orders/{id}` with status = "cancelled".	200 OK, status updated to "cancelled".
API-ORDER-003	Invalid order cancellation	PATCH `/orders/{id}` for nonexistent ID.	404 Not Found with error message.

Frontend Test Cases

Test Case ID	Scenario	Steps	Expected Result
UI-AUTH-001	Valid login	Enter valid credentials and click Login.	Redirect to dashboard.
UI-AUTH-002	Invalid login	Enter invalid credentials and click Login.	Show error message.
UI-AUTH-003	Logout functionality	Click Logout button from the dashboard.	Redirect to login page.
UI-PROD-001	Product listing display	Navigate to Products page.	Display all products with name, price, and image.
UI-PROD-002	Filter products	Apply a valid filter on the product page.	Display filtered results.
UI-ORDER-001	Create order	Select a product, fill details, and click Order.	Confirmation message displayed.
UI-ORDER-002	Cancel order	Click Cancel for an active order.	Status updated to "cancelled".
UI-RESP-001	UI responsiveness	Resize browser window.	UI adjusts without breaking.

Test Execution Report

Summary

Total Test Cases	Passed	Failed	Blocked
14	12	2	0

Failed Test Cases

Test Case ID	Scenario	Actual Result	Defect ID
API-ORDER-002	Cancel order	500 Internal Server Error.	DEF-001
UI-PROD-001	Product listing display	Product descriptions missing for some items.	DEF-002

Defect Details

Defect ID: DEF-001

- **Title:** Order cancellation fails.
- **Description:** Cancelling an order returns a 500 error.
- **Steps to Reproduce:**
    1. Create an order.
    2. Send PATCH `/orders/{id}` with status = "cancelled".
- **Severity:** Critical

Defect ID: DEF-002

- **Title:** Missing product descriptions on Products page.
- **Description:** Descriptions are not displayed for multiple products.
- **Steps to Reproduce:**
    1. Navigate to Products page.
    2. Observe missing descriptions.
- **Severity:** Major

Diagrams

System Architecture

```
User Browser (Frontend - ReactJS)
       ↓
       HTTP Request
       ↓
Backend API (C#)
       ↓
       Database
```

Workflow Example: Order Cancellation

```
1. **User Action:** Click "Cancel" on an active order.
2. **Frontend:** Sends PATCH `/orders/{id}` request.
3. **Backend:** Updates order status in database.
4. **Response:** Returns success or error message to frontend.
```

Test Workflow Diagram

```
1. Input Data
   ↓
2. Trigger UI or API Action
   ↓
3. Validate Response
   ↓
4. Log Results
   ↓
5. Report Defects (if any)
```
