QA Challenge - Test PLAN

Nicolás Sira

# 1.1 Scope

This test plan covers both **backend API testing** and **frontend UI validation** for the QA Challenge application.

## **Backend Testing (C# REST APIs)** Performed using Postman. The test collection includes scenarios for:

* + Valid and invalid login attempts.
  + Product CRUD operations with edge cases (e.g., missing fields, invalid types, duplicates).
  + Order creation and validation with both possitive and negative paths.
  + Automation scripts include assertions for status codes, response structure, and error handling.

## **Frontend Testing (ReactJS App)** Automated using Cypress. Scenarios include:

* + Login and logout flows.
  + Product rendering.
  + Order rendering
  + Dashboard rendering
  + UI responsiveness

## **Out of Scope**:

* + Performance testing (load/stress) of APIs or frontend.
  + Cross-browser compatibility testing.
  + Security testing such as token expiration or XSS attacks.
  + Full end-to-end integration with real databases or third-party services.

This plan focuses on **functional testing** and **error handling validation**, along with automation proof using widely adopted QA tools.

# 1.2 Objectives

The main objective of this QA effort is to verify that the application meets the functional requirements by executing both manual and automated test cases on the backend (API) and frontend (UI) layers. The specific objectives are divided as follows:

## General Objectives

* Validate the core functionality of the system from both the backend and frontend perspectives.
* Ensure that user flows behave correctly with valid and invalid inputs.
* Detect and report functional defects, including edge cases and negative scenarios.
* Provide automated test coverage to improve regression reliability and speed.

## Backend API Objectives

* Verify that all RESTful endpoints respond with appropriate status codes and data formats.
* Test CRUD operations for users, products, and orders with various data inputs.
* Ensure proper handling of errors such as:
  + Missing or malformed request bodies.
  + Business rule violations (e.g., duplicate IDs, invalid price types).
* Confirm that security mechanisms (e.g., login token generation) work as expected.
* Automate key API flows using Postman to enable fast and repeatable execution.

## Frontend UI Objectives

* Validate the functionality and responsiveness of UI components using Cypress.
* Confirm that:
  + Users can log in successfully.
  + The product rendering work correctly.
  + The orders rendering work correctly.
* Check for basic accessibility and layout consistency across the app.
* Automate high-impact user flows to verify business logic through the UI layer.

# 1.3 Resources

## Backend Testing Resources

* .NET 8 Environment
* Qa-api Sample Application
* Swagger
* Postman Application

## Frontend Testing Resources

* ReactJS qa-app
* Cypress.io
* VS Code + Node.js
* Google Chrome (Browser)

## Reporting

* VSCODE
* Office (Excel, Word)

# 1.4 Risks and Mitigation

|  |  |  |
| --- | --- | --- |
| Risk | Risk Level | Mitigation Strategy |
| Incomplete understanding of API endpoints or parameters | Medium | Utilized the provided Swagger documentation to clarify expected input/output structures. |
| Limited time to explore and test additional/custom endpoints | High | Focused on core scenarios defined in the challenge; prioritized endpoints based on impact. |
| Inconsistent data between UI and API layers | High | Added extra Cypress validations to compare UI elements against API data responses. |
| Unexpected behavior or bugs in the provided sample application | Medium | Logged and documented application-level issues without modifying source code. |
| Missing validation in backend for invalid data (e.g., string for price) | High | Covered edge cases in Postman with negative test cases and documented test failures. |
| UI layout or rendering issues affecting automation scripts | Medium | Cypress selectors were scoped carefully; retries and waits added where needed. |
| Lack of authentication/session state persistence across tests | Low | Token-based authentication was tested in isolation, and Postman environment variables were used. |

# 1.5 Deliverables

- This test documentation (docx)

-Test case development document (Pdf)

-Test run defects encountered (Pdf)

## For the API testing

* + Postman collection (.json)
  + Postman test run (.json)
  + Summary (.MD)
  + Test results (xlsx)
  + Test results (pdf)
  + Testing screenshots

## For the UI testing

* + Cypress Scripts (.ZIP)
  + Summary (.MD)
  + Test results (xlsx)
  + Test Results (pdf)
  + Testing screenshots