TEST PLAN DESIGN

Scope

The test plan covers the entire application, including both the backend and the frontend, with a comprehensive focus on functional, security, performance and usability validations. Its main objective is to ensure that the modules work correctly individually and as a whole, complying with all the established requirements and guaranteeing an optimal user experience.

- **Backend (APIs in C#):**
    - **User Authentication:** Validates credentials entered, ensuring that only authorised users can access. Generates and manages expiring authentication tokens to prevent unauthorised access. In case of invalid credentials or inactive users, handles errors efficiently.
    - **Product Management:** Product management includes CRUD (Create, Read, Update and Delete) testing, validating entries to ensure they do not contain empty fields, duplicates or negative values. Filtering and search queries are verified to work correctly.
    - **Order Processing:** Order Processing manages the creation, status update (pending, sent, cancelled) and cancellation of orders. It ensures that orders are registered with valid data and that status updates respect the restrictions defined in the system.
- **Frontend (ReactJS):**
    - **User Authentication:**
        - Validation of login and logout flow.
        - Error handling verification in case of invalid credentials.
        - Testing of JWT token storage and expiration in the browser.

- **User Dashboard:**
  - Display of authenticated user data (basic information and custom details).

- **Product Listing:**
  - Display of existing products.
  - Product search by id, name and price.

- **Order Processing:**
  - Creation of new orders from the interface.
  - Updating the status of existing orders (in process, completed, cancelled).
  - Display of success confirmations and error handling from the backend.

Objectives

The main objectives of the test plan are:

- **Functional Validation:**
  - Ensure that each module meets the specified functional requirements.
  - Confirm that key workflows (login, CRUD, commands) work correctly and without interruption.

- **Defect Identification:**
  - Detect bugs in the core and secondary functionalities of the system.
  - Identify integration issues between frontend and backend to ensure flawless operation.

- **Security:**
  - Validate proper handling of authentication and authorisation.
  - Test JWT token generation, expiration and protection to prevent unauthorised access.
  - Identify potential vulnerabilities such as SQL injections, exposure of sensitive data or XSS attack vulnerabilities.

- **Performance and Scalability:**
    - Evaluate the responsiveness of APIs under high loads of concurrent requests to verify their ability to handle heavy traffic.
    - Examine frontend performance when handling large volumes of data and ensure acceptable load times.
    - Ensure stateless architecture is maintained to support scalability.

- **Usability and User Experience:**
    - Verify that the user interface is friendly, accessible and functional on different devices and browsers.
    - Evaluate load and update times of information on the frontend to ensure a smooth experience.

- **Data Consistency:**
    - Validate that data sent and received between the frontend and backend is consistent.
    - Confirm correct synchronisation of order and product statuses to avoid inconsistencies and errors.

Resources

Tools:

- **Backend**:
    - **Postman:** For manual testing of APIs (sending requests, validating responses and handling data).
    - **Swagger**: Interactive documentation and validation of endpoints.

- **Frontend**:
    - **Cypress**: End-to-End test automation to verify the behaviour of complete flows.

- **General**:
    - **Git**: Git: Code version control and test scripts.

- **Excel/Google Sheets**: Documentation and reporting of test cases and results.
- **Visual Studio Code**: Code editor used for both frontend and backend.

Environments:

- **Local Environment:**
  - .NET 8 on local server (qa-api).
  - Node.js running React application (qa-app).

- **Test Data:**
  - **Users**: Valid and invalid data (sample users with passwords).
  - **Products**: Listing with valid data, duplicates and invalid data (negative prices, empty fields).
  - **Orders:** Test data for creation, update and cancellation.

Risks:

Risk	Impact	Mitigation Strategy
Limited time for comprehensive testing	High	Prioritise critical cases, these include: 1.User authentication. 2.CRUD operations. 3.Order or transaction processing
Dependencies on external libraries or frameworks	Media	1.Document specific versions of all libraries and frameworks used in the development. 2.Perform periodic validations to detect compatibility issues between versions. 3.Maintain an update log.
Inconsistent data between backend and frontend	High	1.Implement automated integration tests that verify that data sent and received between frontend and backend are consistent. 2.Perform synchronisation tests to ensure that there are no delays or errors in the data flow. 3.Monitor in real time the transmitted data.
Device or browser specific failures	Media	1.Run cross-compatibility tests. 2.Perform tests on multiple devices 3.Document and prioritise critical issues identified.

Deliverables

1. **Test Plan document:**
    - Detailed scope: Precise specification of the functionalities and scenarios to be covered in the tests.
    - Testing strategy: Methodology and approaches used (manual, automated, unit testing, integration, etc.).
    - Resources and tools: List of tools, environments and resources used during the testing process.
    - Risks and mitigation plans: Identification of potential risks and the strategies defined to mitigate them.

2. **Test cases:**
    - Full coverage: Development of cases covering backend and frontend functional flows.
    - Prioritisation: Ranking of cases by criticality, with emphasis on critical scenarios and edge testing.

3. **Execution Reporting:**
    - Test results: Logging of results obtained in manual and automated tests.
    - Defects identified: Detailed list of the errors found, classified by priority and resolution status.
    - Evidence: Screenshots, error logs and documentation of execution results.

4. **Automation Scripts:**
    - Automated End-to-End Testing: Development and delivery of E2E test scripts using tools such as Cypress, ensuring comprehensive coverage of critical system flows.