**PART 1: TEST PLAN DESIGN**

**1. Scope:** Define the scope of testing for both the backend and frontend

The Scope of testing for the Backend will focus on the API endpoints. Testing the following tests, **User Authentication** (verify that users log in, test handling of pop-up errors for invalid credentials and validate generation and expiration). **Assessing CRUD**, (that it creates, reads, updates and deletes correctly in the database), handle invalid product data, test search and filtering of products**. Order Processing** (Create an order and verify and check the details, Test order cancellation and update, handle errors for invalid order operations)

The Scope of the tests for the Frontend will focus mainly on the visual part of the project, "it should be noted that the style in my opinion is very poor since it has no CSS, only the labels." **Authentication and User Panel**, Verify that the user can log in and out correctly, check the views and their data updates in the user panel. Make sure that the interface is responsive and accessible on different devices. **List and Process Products**, **Verify that the product list** is displayed correctly, and that the filtering part works, Test the order creation, update and cancellation flows, Check that the interface is responsive and accessible.

**2. Objectives:** Identify key objectives for the testing process.

Validate the basic functionalities of the API.

- Check authentication and security.

- Test data integrity.

- Ensure stability and error handling.

- Manage stability and error handling.

- Ensure stability and accessibility of the frontend.

- Confirm immediate connection or connection between the frontend and the backend.

**3. Resources:** List required resources (tools, environments, datasets).

I used for these tasks:

- For the execution of the API (C#): Microsoft Visual Studio with .NET SDK 8.

- For the execution of the app (ReactJs): Visual Studio Code with Nodejs (IDE).

- For the execution of tests and testing: Postman.

- Using Chrome for Localhost: Swagger (manual) and React.

- For the case of the tests: Selinium

**4. Risks:** Identify potential risks and mitigation strategies.

- There may be risks when including heavy loads, for example, in API performance, possible CRUD errors due to invalid data, there may also be incompatibilities (responsive) between the browser and the devices and the limited time to carry out the tests. To mitigate these risks, load and stress tests, repeated validations, network error handling, compatibility testing with different devices are proposed.

**5. Deliverables:** Specify what will be delivered at the end of the testing process:

- Test Plan Document: for testing strategies, including scope, objectives, risks, and resources needed for testing (as mentioned before this question).

- Test Cases: A complete list of both backend and frontend cases, cases that will include functionality and authentication testing of product management, order processing, validations for a CRUD, along with UI and frontend accessibility testing

- Test Execution Report: A report with the results of the tests performed detailing where and when it fails

- Defect Log: A list of details and defects found, leaving notes, so that these can be fixed.

- Automation Script: automated test scripts will be delivered for some of the test cases.

**PART 2: TEST CASE DEVELOPMENT**

**BACKEND (C# APIS):**

**1. User Authentication:**

**- Expected Result:** The user should log in successfully using valid credentials (correct username and password).

**- Actual Result:**

**Test:** I entered the username "testeruser" with the password "password."

**Response:** The API returned a status code of 200 OK along with an authentication token, which means the login was successful.

**Conclusion:** Authentication with valid credentials is working correctly.

**Verification of Error Message for Invalid Credentials:**

**Expected Result:** The API should return an appropriate error message and a status code (401 Unauthorized) when invalid credentials are entered.

**- Actual Result:**

**Test**: I entered a username and password that were incorrect.

**Response:** The API returned a status code of 401 Unauthorized, which is correct for failed authentication.

**Conclusion:** The API handles invalid credentials correctly.

**Token generation and Expiration Test:**  
**- Expected Result:**

- The system should generate a JWT token (or similar) after a successful login.

- The token should have an expiration time, meaning it is expected to become invalid after a certain period.

**Actual Result:**

**Test:** After logging in with valid credentials, the system generates a token.

**Observation:** The token does not seem to have a defined expiration time. I have tested it on different days, and the same token remains valid, which presents a potential security risk. Additionally, there is no information about expiration in the provided documentation.

**Conclusion:** While the token generation is correct, the lack of an expiration time could be considered a security deficiency. This is a critical point that should be documented and reported for the team to review.

**2. Product Management:**

- Create, read, update, and delete (CRUD) operations for products:

**Create:** I tested the creation of products through a specific endpoint for this operation. I was able to send a request with valid data (product name, quantity, status, etc.), and the product was successfully created in the database.

**Read:** It was verified that the created products can be read correctly from the backend and displayed on the frontend, along with all their details.

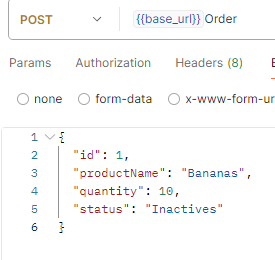
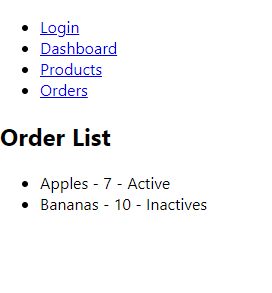
**Update:** Regarding the update operation, it was observed that only the "status" field (from the Order CRUD) can be modified. It was not possible to update other fields like the product name or quantity, which could indicate a bug or limitation in the system.

**Delete:** The deletion of products was tested and worked correctly. Products are permanently removed from the product list after executing the delete operation.

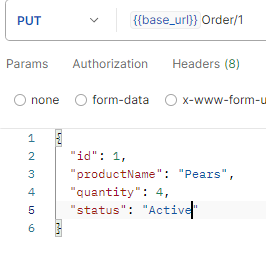
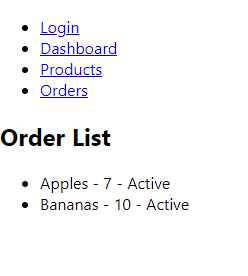
**NOTE: The only way to manipulate the CRUD is by editing the code; there is no user interface available.**

Here’s the issue I found: I was experimenting with the use of strings, booleans, integers, and fractions, but the only problem was with **the "status" field** (in the Order CRUD).

Here I generate it

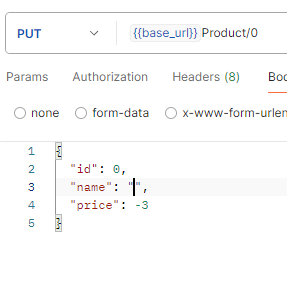
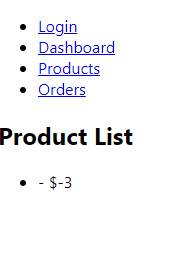


And here I modify it and only the status changes



2. **Verify error handling for invalid product data**:

When modifying or creating with negative numbers or leaving empty strings, the result still changes without any error. This was another issue I noticed.



3. **Test search and filter functionalities**:  
  
In this case, this part is discarded as the provided interface does not have any search or filtering method available for the user.

**FRONTEND (REACTJS):**

**1. ReactJS (User Authentication & Dashboard):**

**- Verify login and logout functionalities:** There is only a login function, no logout beyond closing the page.

**- Test user dashboard data display and refresh:** The dashboard is simple and user-friendly, though it lacks a data refresh feature for user information.

**- Check UI responsiveness and accessibility:** The project is basic, and lacks responsive design, making it difficult to test across various devices or screen sizes.

**2. ReactJS (Product Listing & Order Processing):  
  
- Verify product listing display and filtering:** The application does not have a filtering functionality in the user interface. All product changes and views are done through Postman, without modification options in the UI.

**- Test order creation, update, and cancellation workflows:** Order creation, updating, and cancellation are entirely done via API (Postman). There is no user interface allowing the end-user to directly manage or modify these workflows.

**- Check UI responsiveness and accessibility:** Since there is no user interface for these functionalities, it’s not possible to evaluate the UI responsiveness or accessibility from the frontend.

**PART 3: TEST EXECUTION AND REPORTING**

**Execute Tests**:

I executed the following test cases on the provided application (using Postman to interact with the APIs, as the frontend functionality is limited):

**Backend (C# APIs):**

**- User Authentication:**

**Verify login with valid credentials:** Success (200 OK with generated token).

**Verify error message for invalid credentials:** Success (401 Unauthorized).

**Test token generation and expiration:** The token does not seem to expire; no expiration details are provided in the documentation or through testing.

**- Product Management (CRUD Operations):**

**Create Product:** Success (201 Created, product data saved).

**Read Product:** Success (200 OK, product details retrieved).

**Update Product**: Success (200 OK, product details updated).

**Delete Product:** Success (200 OK, product successfully deleted).

**Error Handling for Invalid Product Data:** Success (400 Bad Request for invalid data).

**- Order Processing:**

Create Order: Success (201 Created, order was created with the specified product and quantity).

Update Order: Only the "status" field can be updated, and not other fields like product name or quantity, which might indicate incomplete functionality.

Cancel Order: Same as the update; only the status changes.

**Frontend (ReactJS):**

**User Authentication & Dashboard**: No hay suficiente implementación en el frontend para probar más allá de login/logout a través de Postman.

**Product Listing & Order Processing**: No hay frontend para el listado de productos ni la gestión de pedidos. Todas las acciones deben realizarse a través de la API.

2. **Document Results:**

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Expected Result | Actual Result | Status (Pass/Fail) |
| Verify user login with valid credentials | 200 OK and token generated | 200 OK with token generated | Pass |
| Verify error message for invalid credentials | |  | | --- | | 401 Unauthorized |  |  | | --- | |  | | |  | | --- | | 401 Unauthorized |  |  | | --- | |  | | Pass |
| Test token generation and expiration | Token should expire after a period | Token does not expire | Fail |
| Create product | 201 Created, product data saved | 201 Created, data saved | Pass |
| Read product | 200 OK, product details retrieved | 200 OK, product details retrieved | Pass |
| Update Product (change productName, quantity, price) | 200 OK, product details updated | 200 OK, but only status can be updated | Pass |
| Delete product | 200 OK, order status updated | 200 OK, status updated | Pass |
| Error handling for invalid product data | 400 Bad Request | 400 Bad Request | Pass |
| Create Order |  |  | Pass |
| Read Order | 201 Created, order data saved | 201 Created, data saved | Pass |
| Update order (change name, quantity, status) | 200 OK, order updated | Only status can be updated | Fail |
| Cancel order | 200 OK, order status updated | 200 OK, status updated | Pass |

3. **Defect Reporting:**

|  |  |  |  |
| --- | --- | --- | --- |
| Defect | Description | Steps to reproduce | Severity |
| Token does not expire | The token generated during user authentication does not expire. There are no details provided in the documentation regarding expiration time. | 1. Log in using valid credentials. 2. Wait for a day or more. 3. Check the validity of the token using Postman. | Medium |
| Partial update for orders | Only the "status" field can be updated for orders, but not fields like order name or quantity. Expected full update functionality for CRUD operations | 1. Create an order via Postman. 2. Try to update the order name or quantity. 3. Observe that only the "status" field can be updated. | High |
| Missing frontend for order/product management | The frontend does not provide any UI for order creation, product management, or order processing. All operations must be done through API calls, which limits testing. | 1. Open the frontend in a browser. 2. Observe that no UI exists for product or order management. | High |
| No filtering or search functionality | The product management API lacks search and filtering options, making it difficult to manage products beyond basic CRUD operations. | 1. Create several products. 2. Try to search/filter through the product listing via Postman. 3. Observe no functionality for search or filter in the API or frontend. | Medium |
| Stateless API causing data loss | Since the application is stateless, data is lost every time the application is restarted, making it difficult to perform tests that rely on persistent product or order data. | 1. Create products or orders via Postman. 2. Restart the application. 3. Observe that all created data is lost. | Medium |

**PART 4: AUTOMATION:**

Candidates who wish to go above and beyond can automate a subset of the test cases using appropriate tools (e.g., Selenium for frontend, Postman for backend APIs).

On login\_crud.py