QA CHALLENGE

OBJECTIVE:

Evaluate the candidate's ability to design and execute comprehensive testing strategies for a web application with a C# backend and frontends built using ReactJS

ASSIGNMENT OVERVIEW:

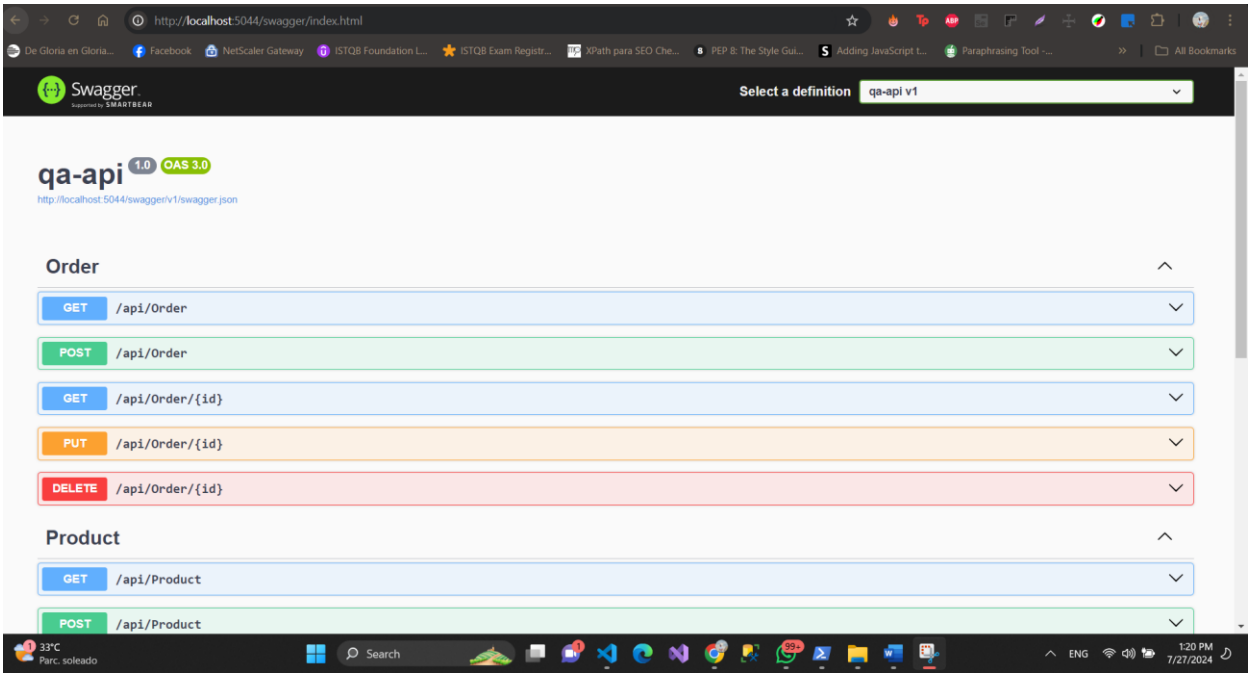
Candidates will be provided with a sample application that includes a set of RESTful APIs built with C# and frontends developed using ReactJS. The goal is to design a test plan, create test cases, and execute these tests, documenting the results and any defects found.

SAMPLE APPLICATION DETAILS:

- **Backend:** C# APIs for user authentication, product management, and order processing.
- **Frontend:**
  - **ReactJS:** User authentication and dashboard.
  - **Aurelia:** Product listing and order processing.
- **Sample Applications***: https://github.com/SAMO-Technologies/qa-challenge*

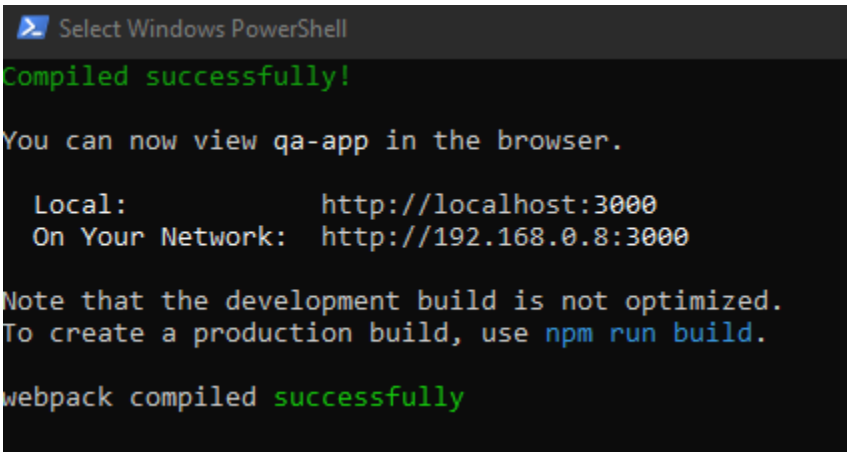
Steps taken to run the project:

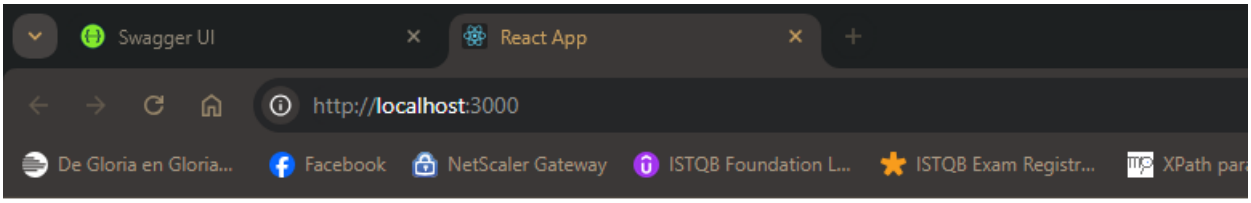
1- I made sure I had the requirements as per the documentations said DOTNET.
2- Clone the repository into my local machine so I can get the app running on my desktop.
3- Since we're using **React** we have to make sure that I have installed **NodeJS** on my machine.
4- Then I followed the instructions to run the project "dotnet run --launch-profile http"
5- We made sure we're using the correct URL for the API and right server to run our Swagger. http://localhost:5044/swagger/index.html



Here we will be able to try and test every end point, look at the schemas, see the responses we should get when running our end point, body etc.

6- We also then installed our NPM in our qa-app folder, then started it so we can compile and see the App in the browser:





- Login
- Dashboard
- Products
- Orders

PART 1: TEST PLAN DESIGN

1. **Scope:** Define the scope of testing for both the backend and frontend.
2. **Objectives:** Identify key objectives for the testing process.
3. **Resources:** List required resources (tools, environments, datasets).
4. **Risks:** Identify potential risks and mitigation strategies.
5. **Deliverables:** Specify what will be delivered at the end of the testing process.

PART 2 & 3: TEST CASE DEVELOPMENT By Candidate Ernesto Castillo.

Develop a comprehensive set of test cases for the following scenarios:

BACKEND (C# APIS):

1. **User Authentication:**
    - Verify user login with valid credentials.
    - Verify error message for invalid credentials.
    - Test token generation and expiration.

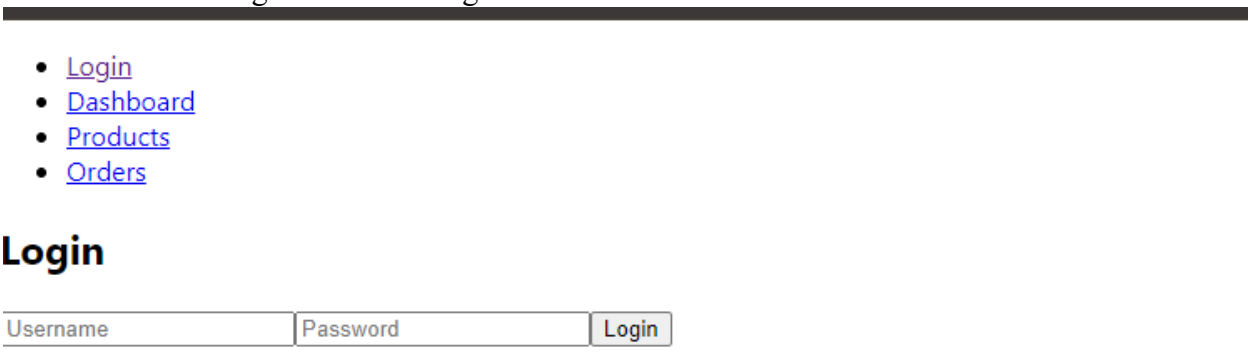
For our User Authentication Test Cases we would build it as the following:

Test Case Name
User Login Functionality
Description
Verify that a registered user can log in to the application using valid credentials and is unable to log in with invalid credentials or at least get some kind of error when using invalid credentials.
Pre-conditions

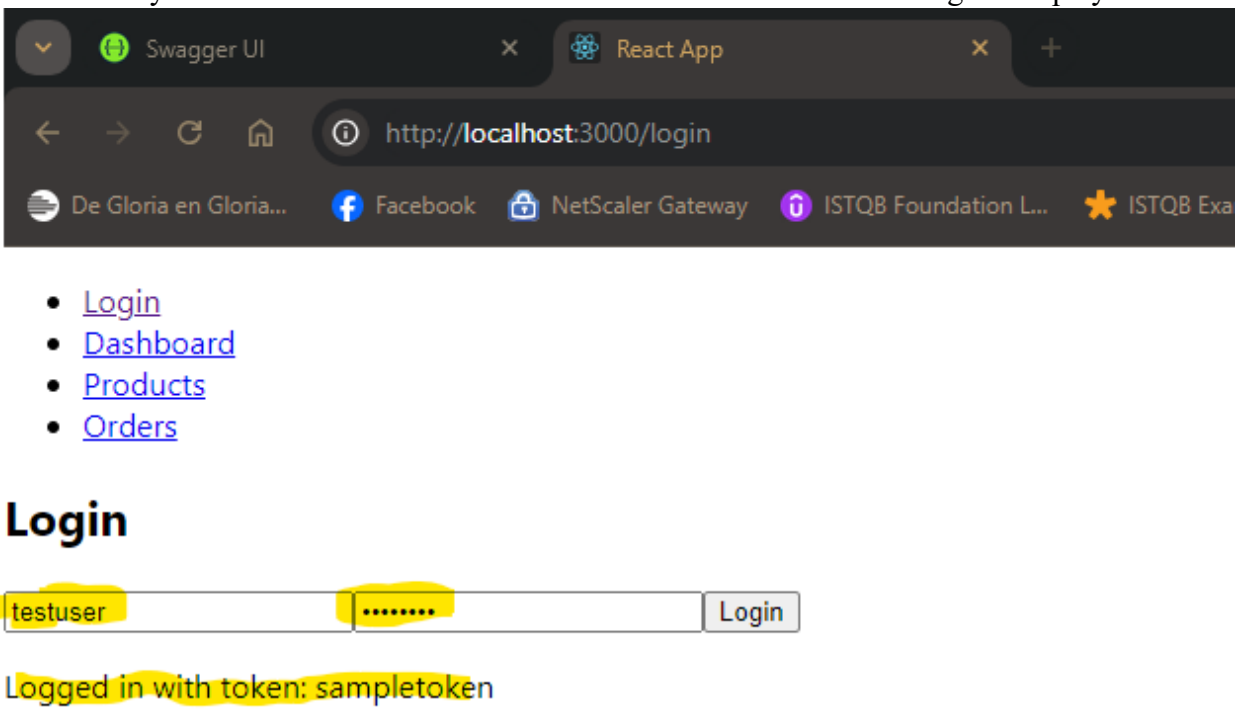
1. The user must be registered and have valid login credentials (username and password). For this scenario username and password has been provided in the documentation.
2. The application must be accessible and the login page should be functional.

Test Steps

1. Open the web browser.
2. Navigate to the login page of the application (http://localhost:3000/).
3. Click on Log in to see the log in form.

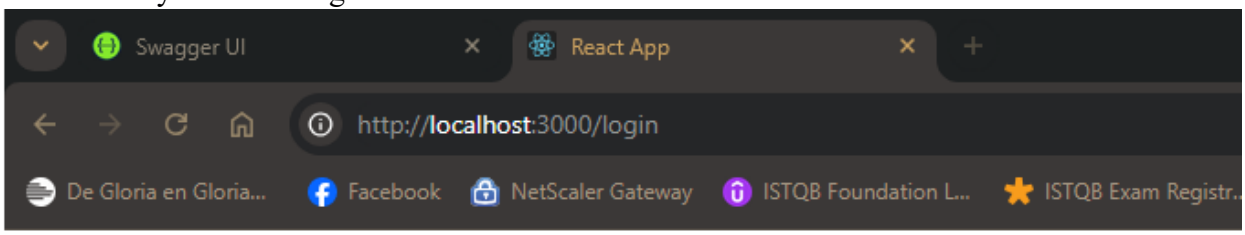


4. Enter the valid username in the username field.
5. Enter the valid password in the password field.
6. Click on the "Login" button.
7. Verify that the user is able to click the button and a welcome message is displayed.



8. Repeat steps 1-5 with an invalid username.
9. Verify that an error message is displayed, indicating invalid credentials.
10. Repeat steps 1-5 with a valid username but an invalid password.
11. Verify that an error message is displayed, indicating invalid credentials.

- Verify error message for invalid credentials.



Test Data.

Test Scenario	Username	Password	Expected Result	RESULT
Valid Login	ValidUser: testuser	Valid Password: password	User the message once he click on the login button.	PASS
Invalid User	Invalid User	Valid Password	Error Message	PASS
Invalid Pass	Valid User	Invalid Password	Error Message	PASS

Expected Results

1. For a valid login, the user should be redirected to the dashboard page and see a welcome message.
2. For invalid login attempts, the user should see an error message indicating invalid credentials.

Actual Results:

- When Clicking on the “login” button response seems to be working fine, using the valid user’s name and the valid password it will let you through.
- Same happening when using the other scenario with an invalid user and password

2. **Product Management:**
    - Create, read, update, and delete (CRUD) operations for products
    - Verify error handling for invalid product data.
    - Test search and filter functionalities.

Test Case: Product Management - CRUD Operations

Test Case ID

TC-002

Test Case Name

Product Management - CRUD Operations

Description

Verify that a user can create, read, update, and delete products. Additionally, verify error handling for invalid product data and test search and filter functionalities.

Pre-conditions

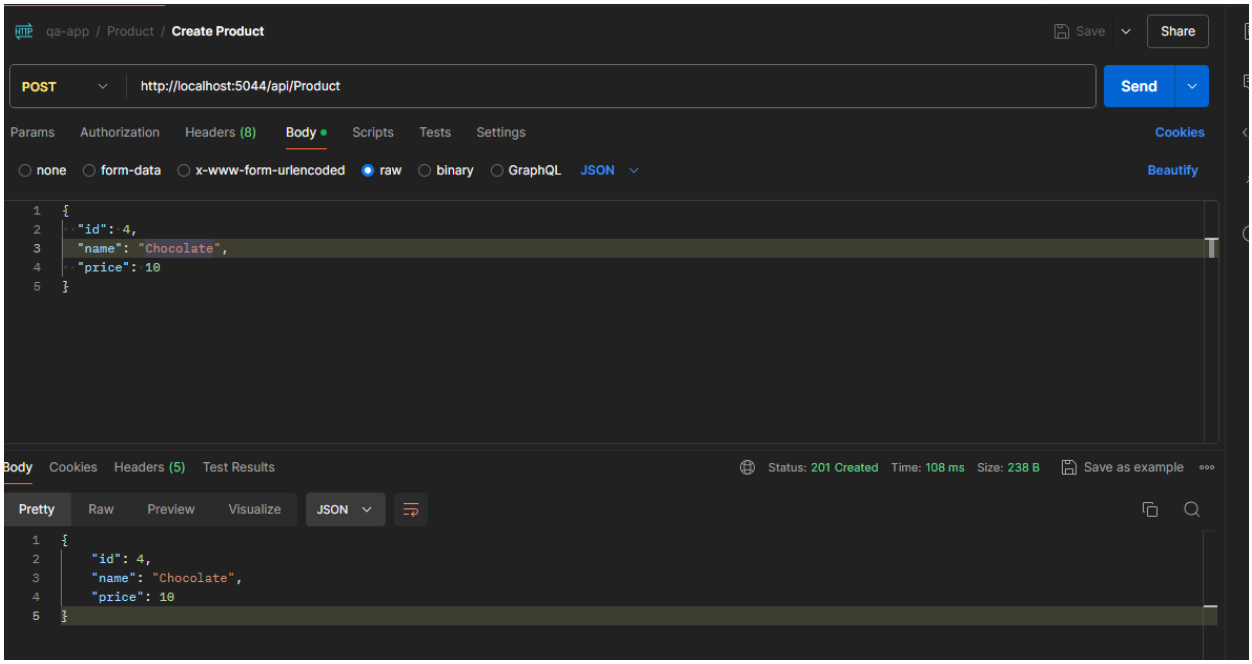
1. The user must be logged in with appropriate permissions to manage products.
2. The application must be accessible and the product management interface should be functional.

Note: for the following steps we will be adding new products or elements to the Products tab but using "POSTMAN" to do these actions since there is NO UI to actually "Add" a new product in the page.

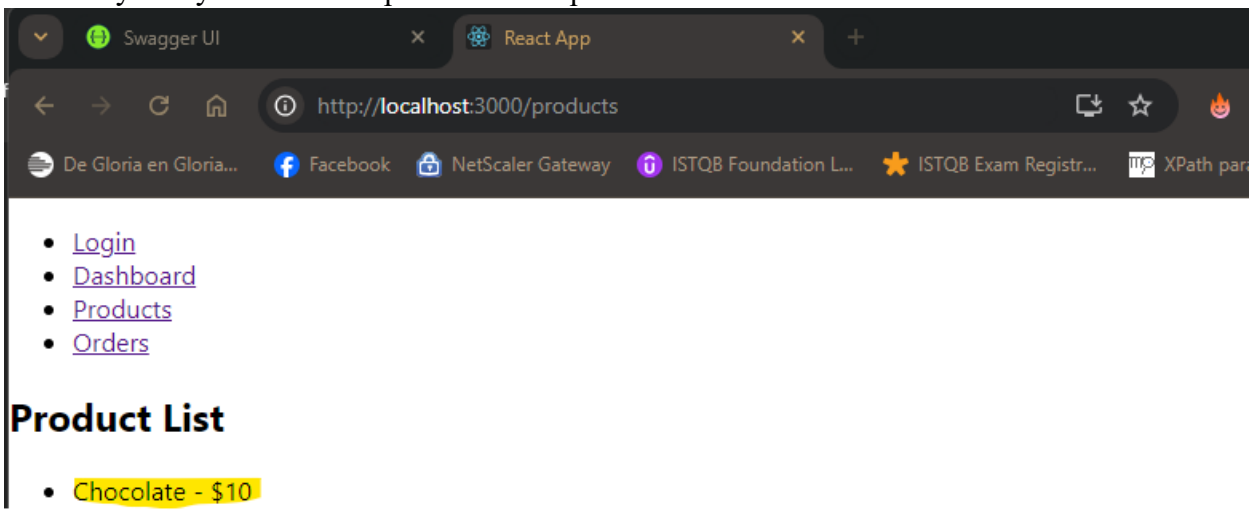
Test Steps

Create Product (Using Postman)

1. Go to Swagger with the following URL: http://localhost:5044/swagger/index.html
2. Scroll down to the "Products" section.
3. Click on "GET" to see the parameters, responses & body.
4. Open Postman.
5. From Swagger we will use our URL in postman so we can send "data" to our product tab "http://localhost:5044/api/Product "
6. We're using the method "POST" to send our new product.
7. Enter valid information for the body in all required fields (e.g., product name, description, price, category).
8. In postman we click on Send



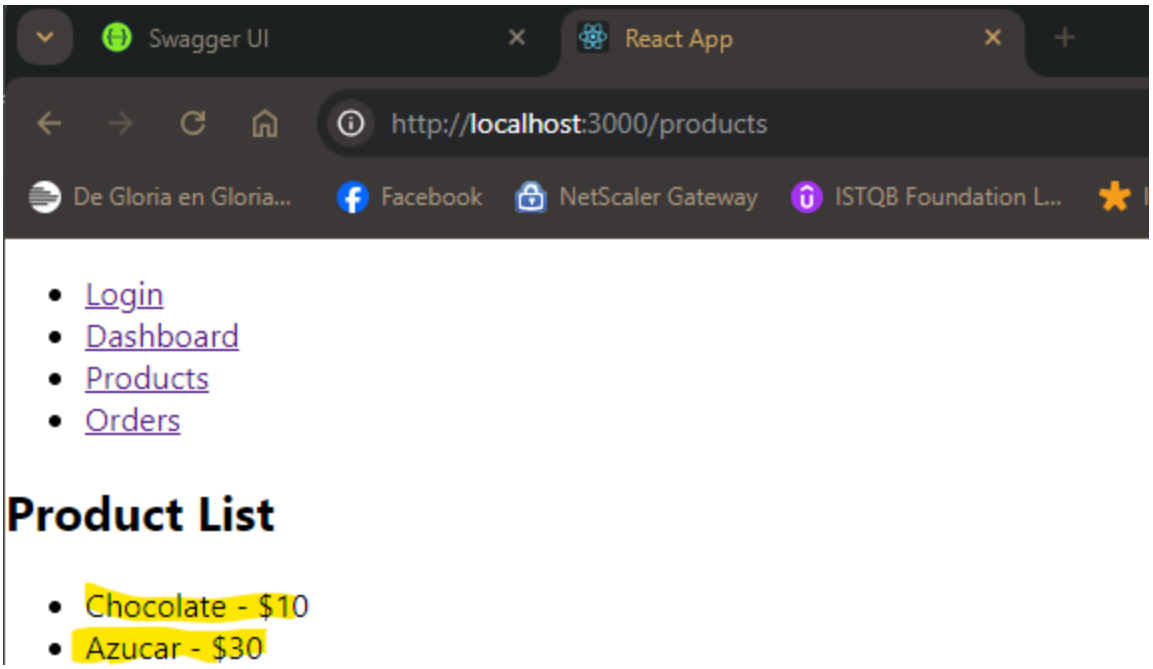
9. Navigate to the product management page http://localhost:3000/
10. Then click on Products.
11. Verify that you see a new product in the product section.



12. Verify that the new product is listed in the product list and a success message is displayed.

Read Product

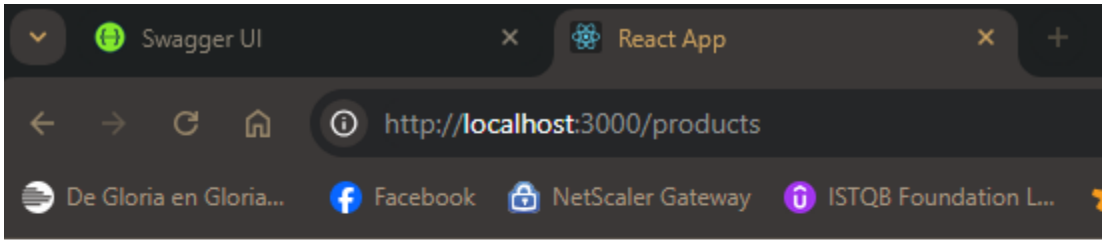
7. Navigate to the product management page.
8. Locate the newly created product in the product list.
9. Verify that all product details are displayed correctly.



Update Product

10. In Postman, using the method PUT.
11. Confirm your list of elements in products.
12. Enter valid information for the body in all required fields (e.g., product name, description, price, category).
13. In postman use the correct URL to use the method PUT, using the parameters necessary to update an element in our products section (ID) http://localhost:5044/api/Product/6

Price Before:

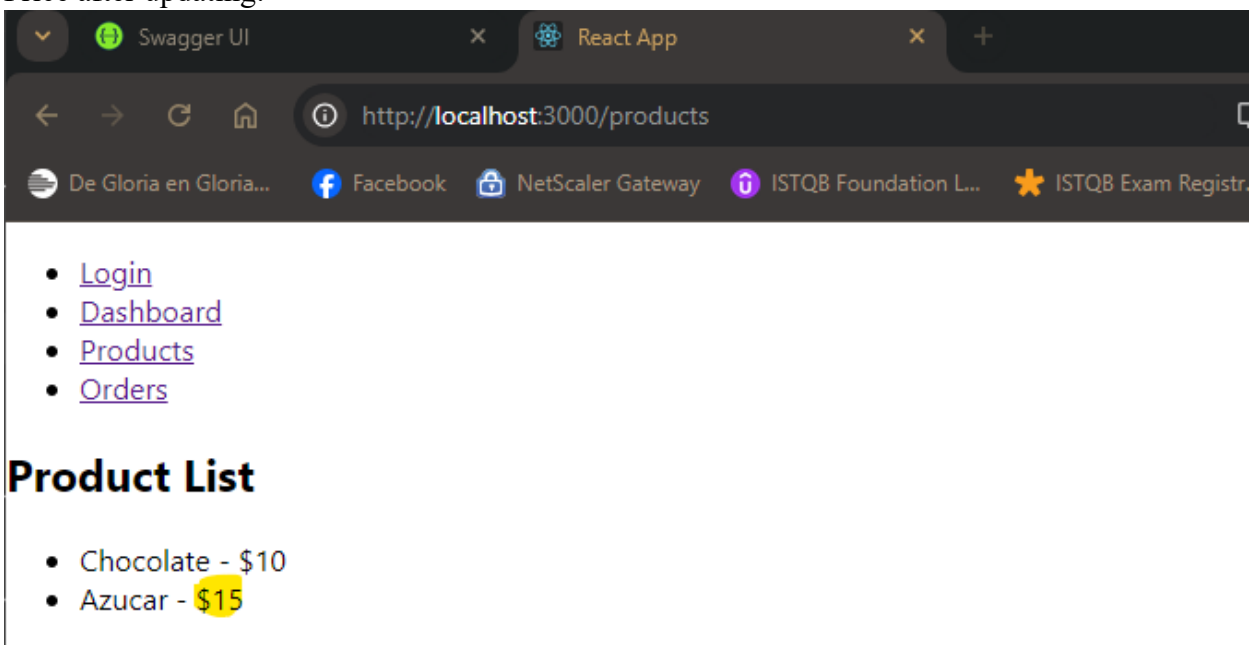


- Login
- Dashboard
- Products
- Orders

Product List

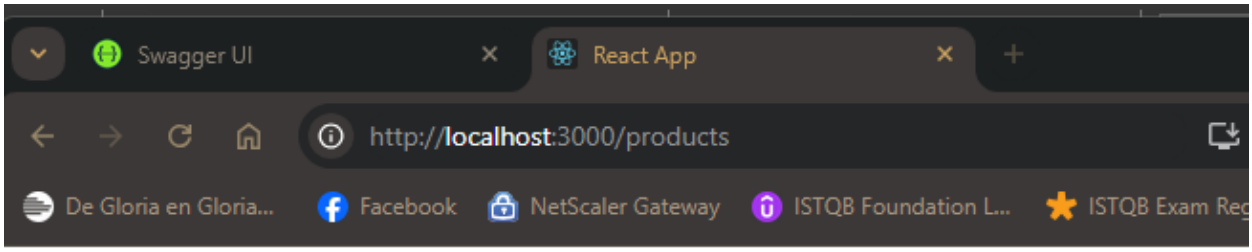
- Chocolate - $10
- Azucar - $30

Price after updating:



Delete Product
1-We'll be using the same steps as an adding, updating a new product in postman but with the method DELETE.
2-For the parameters we're using the ID of the product we're deleting.
http://localhost:5044/api/Product/6

Current products list before deleting:

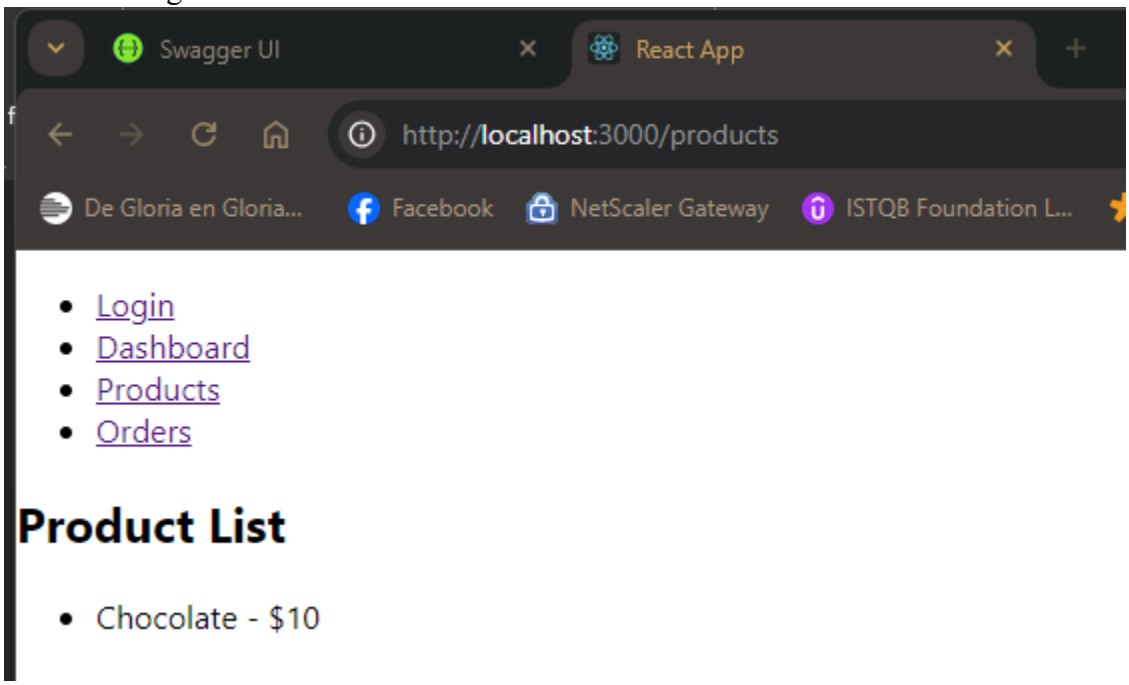


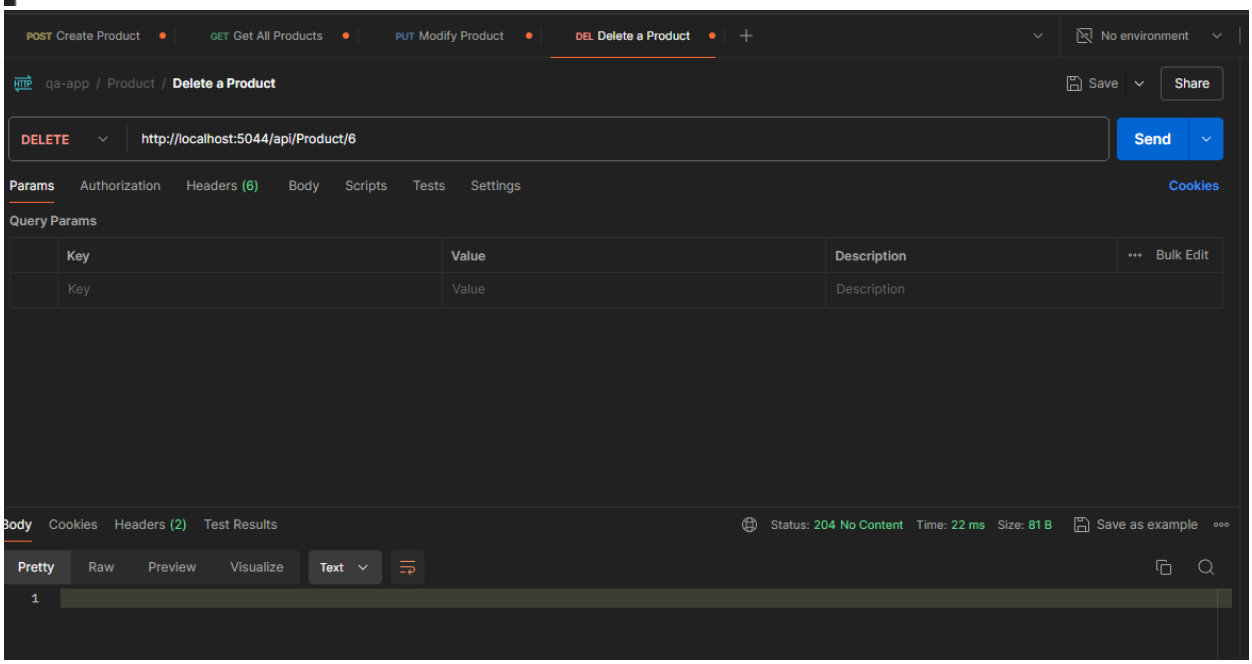
- Login
- Dashboard
- Products
- Orders

Product List

- Chocolate - $10
- Azucar - $15

After deleting:





Error Handling for Invalid Product Data

*We noticed you're able to add more than one product with the same ID, this is an issue that needs to be ASAP, no product should have the same ID number.

Search and Filter Functionalities
There is no search or filter functionality included for this test.

3. **Order Processing:**
    - Create an order and verify order details.
    - Test order cancellation and status updates. (There is no order cancellation or status update functionality)

Verify error handling for invalid order operations.

*When adding a new order we're obtaining the same result, it's letting us to add a new order with the same ID.

FRONTEND (REACTJS):

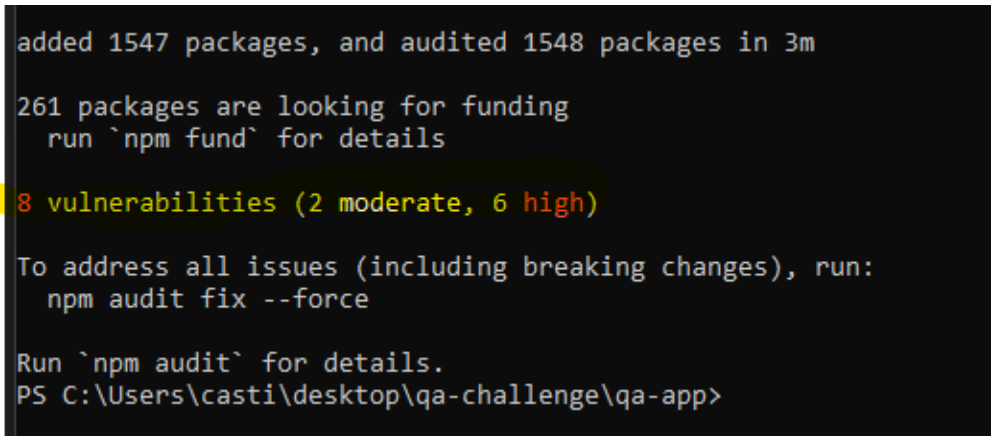
1. **ReactJS (User Authentication & Dashboard):**
    - Verify login and logout functionalities.
    - Test user dashboard data display and refresh.
    - Check UI responsiveness and accessibility.
2. **ReactJS 2. (Product Listing & Order Processing):**
    - Verify product listing display and filtering.
    - Test order creation, update, and cancellation workflows.
      Check UI responsiveness and accessibility.

PART 3: TEST EXECUTION AND REPORTING

1. **Execute Tests:** Run the test cases on the provided application.
2. **Document Results:** Record test results, including pass/fail status and any defects found.
3. **Defect Reporting:** Log defects with detailed descriptions, steps to reproduce, and severity.

RISKS:

- When I installed the NPM I noticed there are 8 vulnerabilities, this is something that should be fixed and look at to see how we can improve the security of our project.



- The log in does not have a token. The API is open, the APIS are exposed without the need for authentication, this is not a defect but this is something we should definitely fix or something to pay attention as a future reference.

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).

DELIVERABLES:

Public repository in Github.com.

1. **Test Plan Document:** Outline the strategy and approach for testing. Diagrams are appreciated.
2. **Test Cases:** Comprehensive list of test cases for both backend and frontend.
3. **Test Execution Report:** Results of the executed tests with documented defects.
4. **Automation Scripts (if any):** Provide scripts and instructions for running automated tests.

For our automation test I'm using Playwright with JavaScript.

What was done?

1. From PowerShell we created a Playwright folder.
2. We installed playwright.
3. We went to our VS code, tests folder and in the folder, we will find our doc js where we run our scripts for the automated test.

Please see the script will be attached.

EVALUATION CRITERIA:

1. **Completeness and Clarity:** How well the test plan and test cases cover the application's functionalities.
2. **Attention to Detail:** Identification and documentation of edge cases and potential issues.
3. **Defect Identification:** Ability to find and document defects accurately. Critically think about the normal, happy and horrible path.
4. **Automation:** Quality and effectiveness of automation scripts.