QA Challenge

Part 1: Test Plan Design

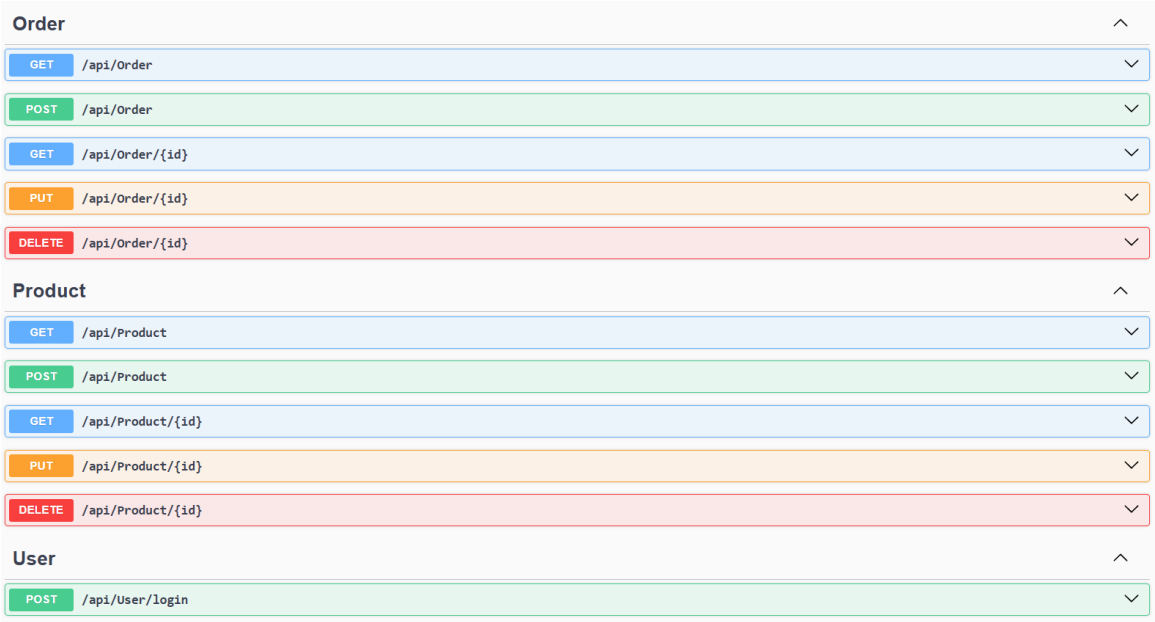
Introduction

This document is aimed to design a test plan for the web application of “SAMO-Technologies” as part of the continuous evaluation process of the GC team. Analyzing the optimal strategies to perform the test plan with a proper understanding of the performance of the web application provided to perform the test cases based on the possible “real” use cases and possible vulnerabilities.

General Analysis of the Application

When verifying the documentation in swagger it can be seen that the only available functions are the “order”, “Product” and “user” functions.

- It should be noted that the “order” and “Product” functions do have the Post, Get, Put and Delete options, so with these we can perform a complete evaluation of the Crud.
- While the “User” only has the post option, which limits the tests to the login attempts without the option to create a new user or close the session
  which was one of the points to evaluate in the frontend.



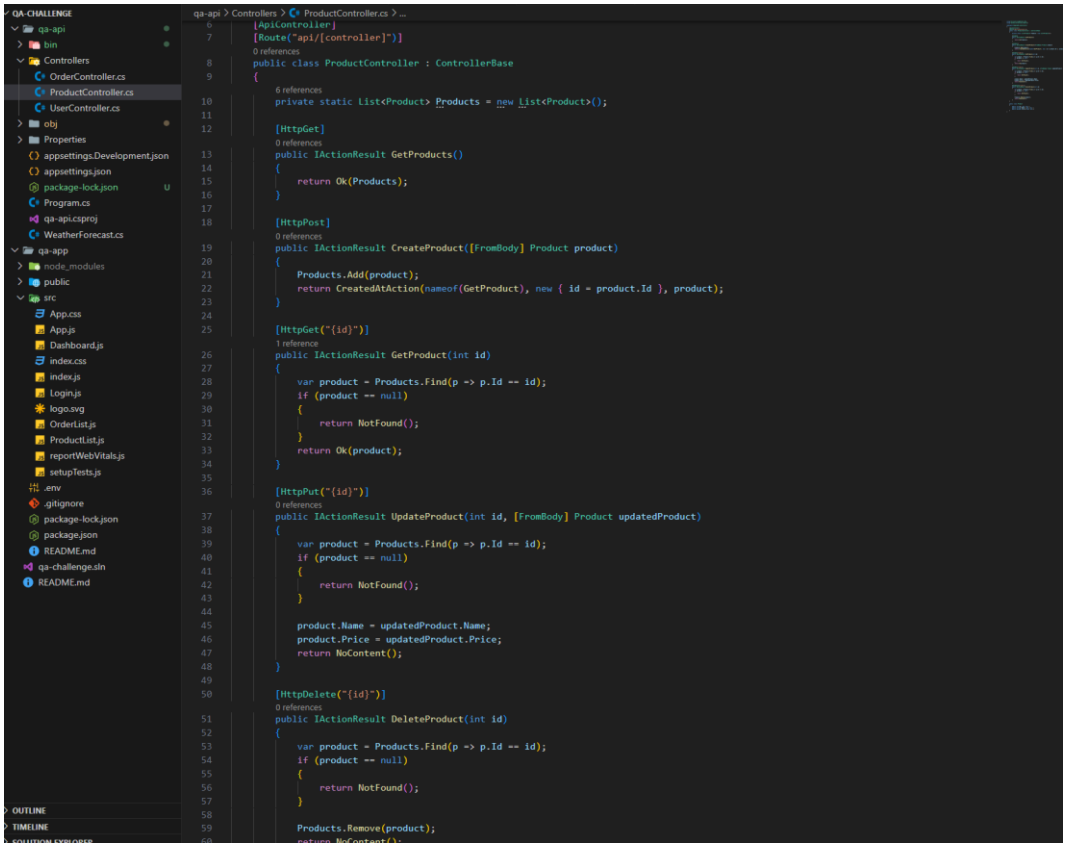
Review of The Code

When verifying the code of the web application we found that the api consists of the 3 controllers mentioned in swagger “OrderController”, “ProductController”, and “UserController”.

Being an implementation of a static list of products without any kind of relational or non-relational database management.

- So we can rule out any kind of SQL injection security test.
- In the same way that extending to perform in-depth performance tests would be a waste of time and effort due to the simplicity of data management.
- Since we would only be evaluating the performance of the computer on which the tests are performed when testing in

postman or Lighthouse.



User Interface

The user interface consists of just an unordered list that act as a navigation bar and a block of content that changes depending on the option the user choose in the navigation bar.

Only the login section has interaction so the rest of the sections are just lists to show the data that is currently stored.

The only way to add or edit entries to these lists is through the api with a tool like postman or Brup.

1. Scope

The testing scope includes validating the **backend APIs** and the **frontend UI** to ensure the web application's functionality, usability, reliability and Basix UX. Testing will focus on:

Backend (C# APIs)

Since none of the components have any validation of maximum attempts. in all backend tests, brute force testing will be performed.

1. **OrderController**:
    a. Full CRUD operations (POST, GET, PUT, DELETE).

   b. Error handling for invalid or incomplete data inputs.
   c. Validation of endpoint responses and status codes.
2. **ProductController**:
   a. Full CRUD operations (POST, GET, PUT, DELETE).
   b. Error handling for invalid operations.
3. **UserController**:
   a. Login functionality through POST.
   b. Error handling for invalid login attempts (e.g., wrong credentials).
   c. Validation of authentication token generation and expiration.

Frontend (ReactJS)

1. **Login Section**:
   a. Authentication flow, including login with valid and invalid credentials.
   b. Validation of the token received from the backend.
   c. Frontend error messages for failed login attempts.
   d. basic validations of enhacements to improve the UX
   e. minimum requirements for passwords
2. **Navbar and Content Rendering**:
   a. Validation of data displayed in the product and order lists.
   b. UI responsiveness and correct rendering based on selected navigation options.
   c. prevent the user from accessing information before starting session

2. Objetives

- **Identify Critical Test Areas:** Discard non-relevant test cases due to the limitations of the sample web application to create an effective test plan.
- **User Experience:** Validate the usability and correct operation of user interfaces and backend flows.
- **System Reliability:** Find errors that may block the regular flow of the application, for prompt correction. as well as finding possible areas of improvement for better growth and scalability of the application functions.
- **Documentation:** record the research process and test results as well as a clear conclusion of what areas need correction and improvement.

3. Resources

- **Tools:**
  - Postman: For backend API testing.
  - Git: For version control and code storage.
  - VS-Code: for text editing and running the web applications locally
    - .NET Install Tool
    - C# for Visual Studio Code
    - Dev Kit
  - Git & GitHub: for version management and control and sharing test results in the public repository.
- **Documentation & Resources:**
  - **Sample Applications:** https://github.com/SAMO-Technologies/qa-challenge
  - **Swagger:** API documentation

4. Risk

From the testing approach, cases of obvious critical risks for the proper functioning of the system in the long term, such as low scalability of the system and the inevitable loss of data, can be evidenced.

In order to correctly identify and prioritize these errors, they will be classified depending on their severity as follows:

- Low: Minor issues that do not affect core functionality.
- Medium: Defects impacting secondary functionalities but not blocking the system.
- High: Failures limiting main functionalities requiring high-priority attention.
- Critical: Defects blocking core functionalities needing immediate resolution.

5. Deliverables

the test report will provide the following items to record the testing and serve as evidence and informational resources for future implementations and tests

- **Test plan design:** A detailed document outlining scope, objectives, resources, risks, and deliverables.
- **Test cases:** A set of cases with descriptions, expected results, preconditions, and detailed steps.
- **Execution reports:** Summaries of successes, failures, and detected defects.

- **scripts:** Configuration files and code for tests with tools like postman.
- **Test repository:** Access to code and documentation generated during the process.

Part 2: Test Case Development

This section details in a structured way the planning of each test case that will be performed to ensure the proper functioning of the web application. Following the next structure:

❖ **ID**: unique identifier of the test case

- **Description:** General information of the test case to be performed.
- **Precondition:** previous steps to be able to perform this test
- **Execution Steps:** steps to follow to perform the test case
- **Expected result:** expected output for the test case

Before continuing it is important to note that in the example repository it is specified that **there is only one user with valid credentials registered and it is the following one:**

- **Username:** testuser
- **Password:** password

BACKEND (C# APIS):

User Authentication

- **ID**: USER-TC01-SuccesfullLogin
  - **Description:** Login with valid data and confirm login token
  - **Precondition:** Set BaseURL on Global variables at the test collection
  - **Execution Steps:** a post request must be made to the api with the credential data of the registered user by making a call to http://localhost:5044/api/User/login.
  - **Expected result:** the user should login correctly and should have as response “token”: “sampletoken”.
    - Status: 200
    - Response time: Immediate because there are no database queries.

- **ID**: USER-TC02-WrongUser
  - **Description:** Login with valid pass but wrong user
  - **Precondition:** Set BaseURL on Global variables at the test collection
  - **Execution Steps:** a post request must be made to the api with the credential data of the registered user but with a wrong user, by making a call to http://localhost:5044/api/User/login.
  - **Expected result:** the user should not login
    - Status: 401
    - Response time: Immediate because there are no database queries.

- **ID**: USER-TC03-WrongPass
  - **Description:** Login with valid user but wrong password
  - **Precondition:** Set BaseURL on Global variables at the test collection
  - **Execution Steps:** a post request must be made to the api with the credential data of the registered user but with a wrong pass, by making a call to http://localhost:5044/api/User/login.
  - **Expected result:** the user should not login
    - Status: 401
    - Response time: Immediate because there are no database queries.

- **ID**: USER-TC04-BlankSpaces
  - **Description:** Login with 1 valid user data and a Blank Space iterating between the possible combinations, between user and password, and both
  - **Precondition:** Set BaseURL on Global variables at the test collection
  - **Execution Steps:** a post request must be made to the api with 1 credential data of the registered user but with Blank Space on the other credential switching between the possible combinations, by making a call to http://localhost:5044/api/User/login.
  - **Expected result:** the user should not login
    - Status: 401
    - Response time: Immediate because there are no database queries.
- **ID**: USER-TC04-Uppercase
  - **Description:** Login with 1 valid user data and a Uppercased Valid Credential iterating between the possible combinations, between user and password, and both
  - **Precondition:** Set BaseURL on Global variables at the test collection
  - **Execution Steps:** a post request must be made to the api with 1 credential data of the registered user but with Uppercased Valid Credential on the other credential switching between the possible combinations, by making a call to http://localhost:5044/api/User/login.
  - **Expected result:** the user should not login
    - Status: 401

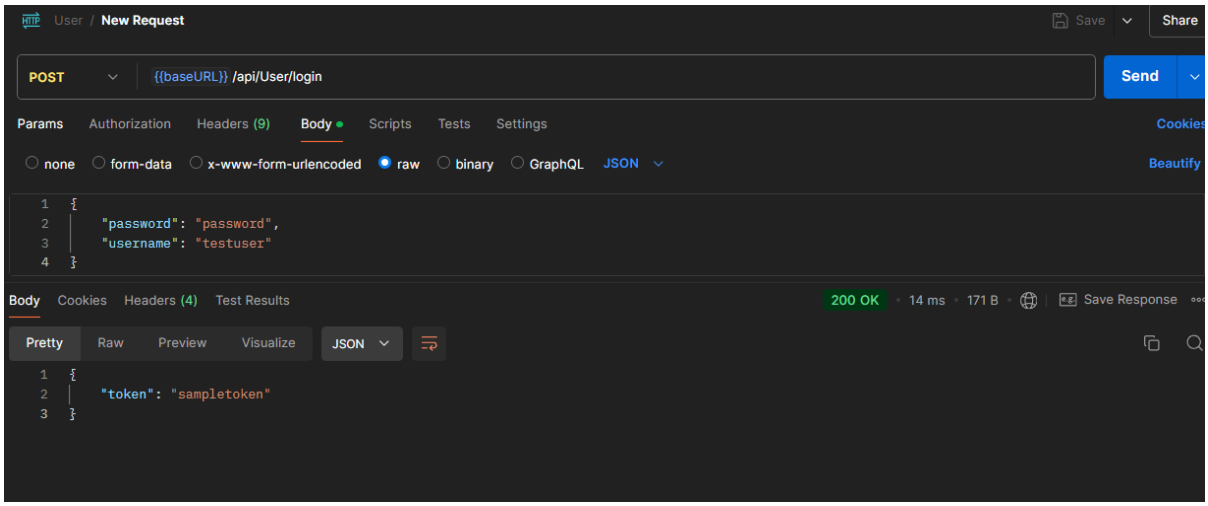
        - Response time: Immediate because there are no database queries.
        -

- **ID**: USER-TC05-WrongCredentials
    - **Description:** try to Login with wrong credentials
    - **Precondition:** Set BaseURL on Global variables at the test collection
    - **Execution Steps:** a post request must be made to the api with a wrong user and pass, by making a call to http://localhost:5044/api/User/login.
    - **Expected result:** the user should not login
        - Status: 401
        - Response time: Immediate because there are no database queries.

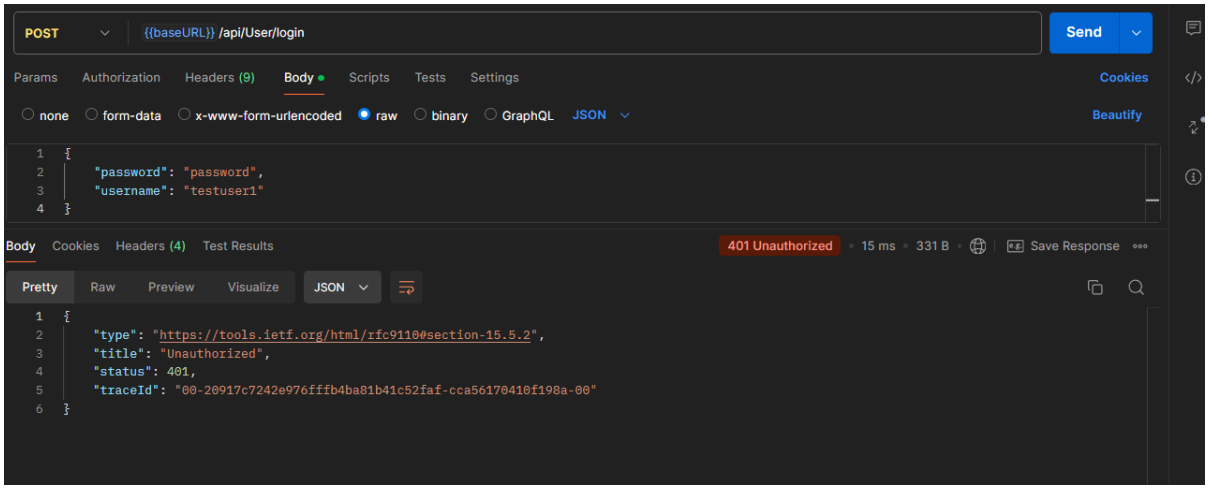
Part 3: Test Cases Execution

BACKEND (C# APIS):

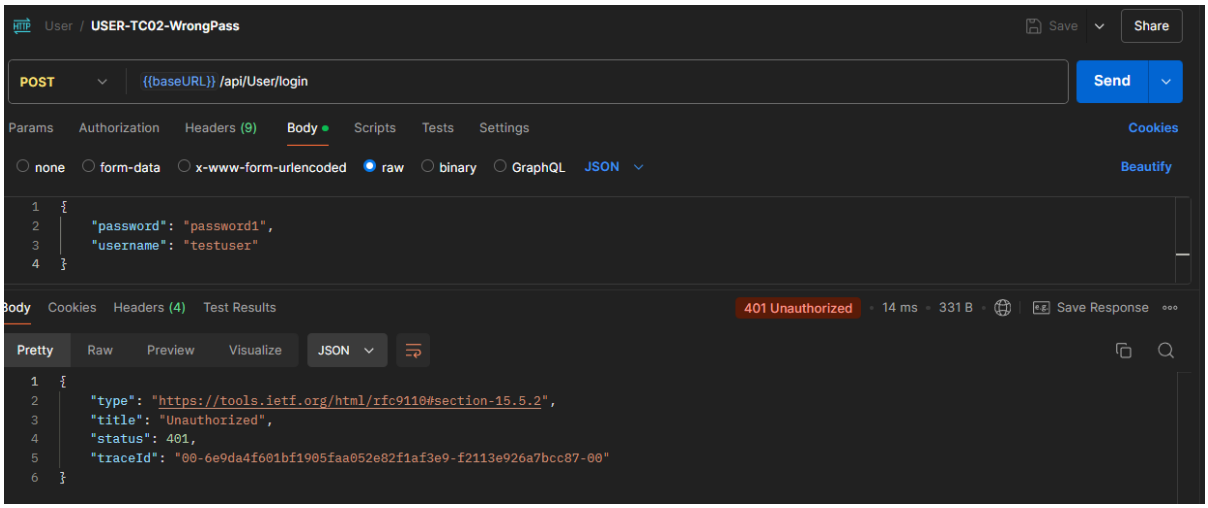
- **ID: USER-TC01-SuccesfullLogin (Succesfull ☑)**
  - **Description:** Login with valid data and confirm login token
  - **Current result:** the user login correctly and have as response “token”: “sampletoken”.
    - Status: 200
    - Response time: 14ms



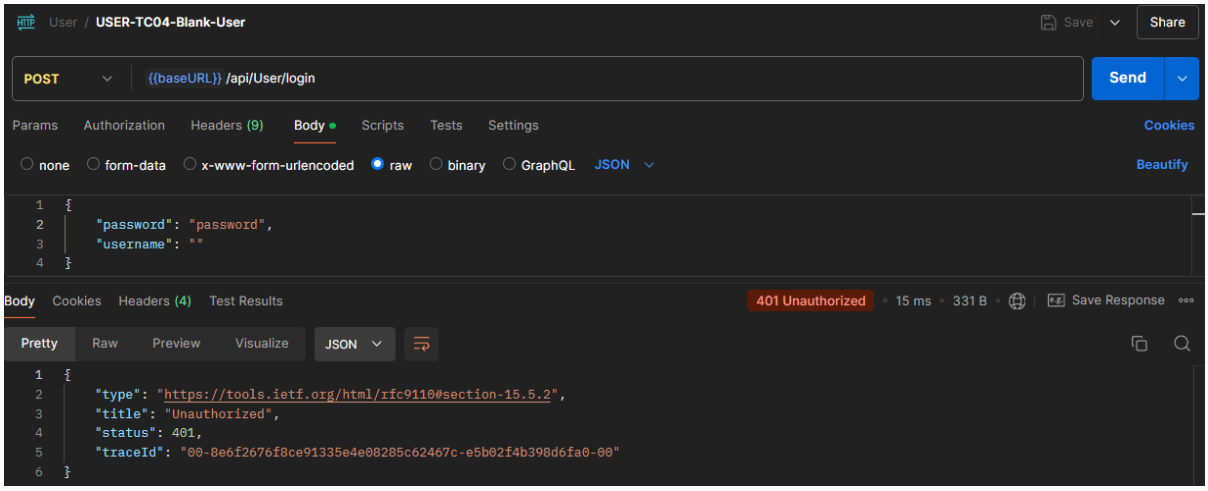
- **ID: USER-TC02-WrongUser (Succesfull ☑)**
  - **Description:** Login with valid pass but wrong user
  - **Current result:** the user did not login
    - **Status:** 401
    - **Response time:** 15ms



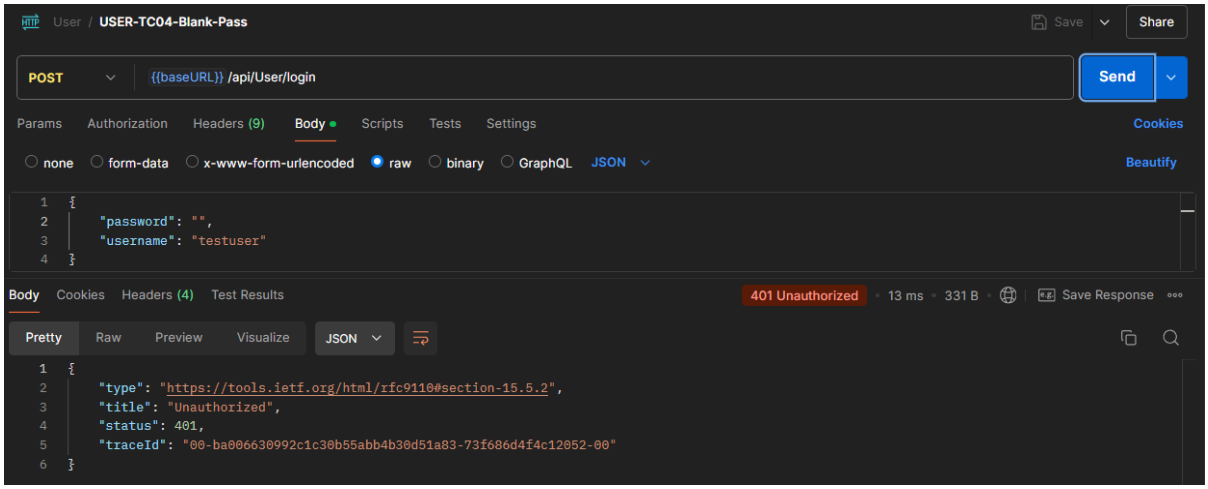
- **ID: USER-TC03-WrongPass (Succesfull ☑)**
  - **Description:** Login with valid user but wrong pass
  - **Current result:** the user did not login
    - **Status:** 401
    - **Response time:** 1ms



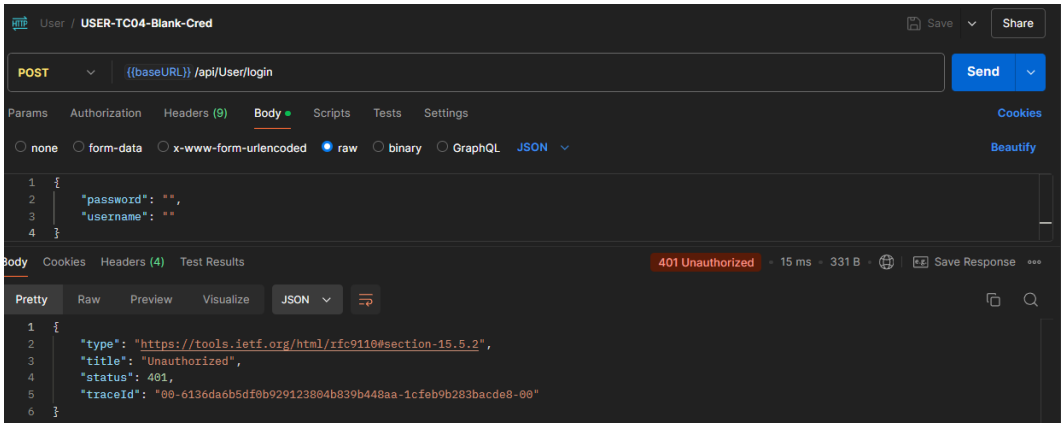
- **ID: USER-TC04-BlankSpaces (Succesfull ☑)**
  - **Description:** Login with 1 valid credential and a Blank Space iterating between the possible combinations, between user and password, and both
  - **Current result:**
    - BlankUser: the user did not login
    - Status: 401
    - Response time: 15ms



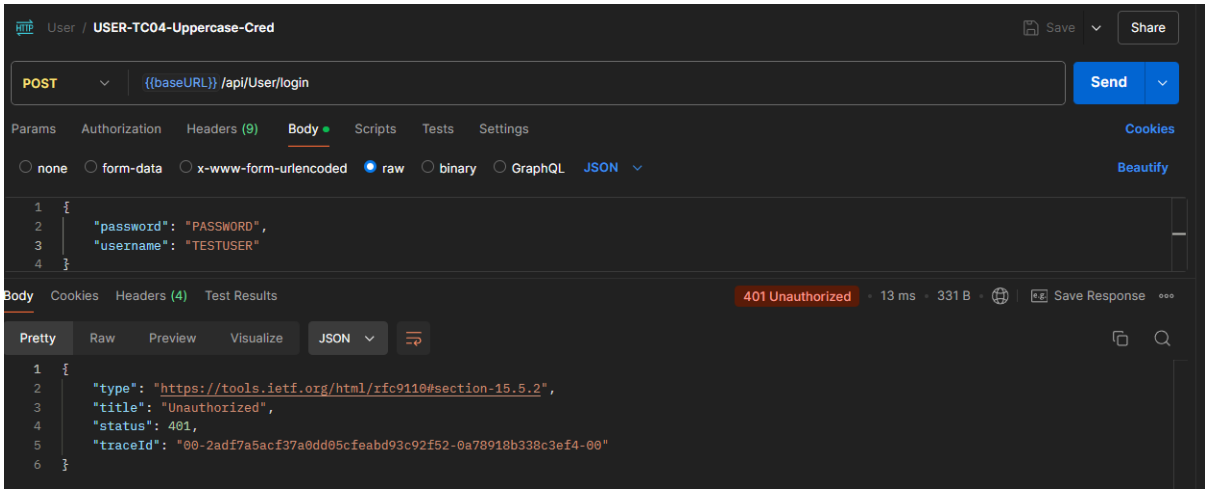
  - BlankPass: the user did not login
  - Status: 401
  - Response time: 13ms



- BlankCredentials: the user did not login
- Status: 401
- Response time: 15ms

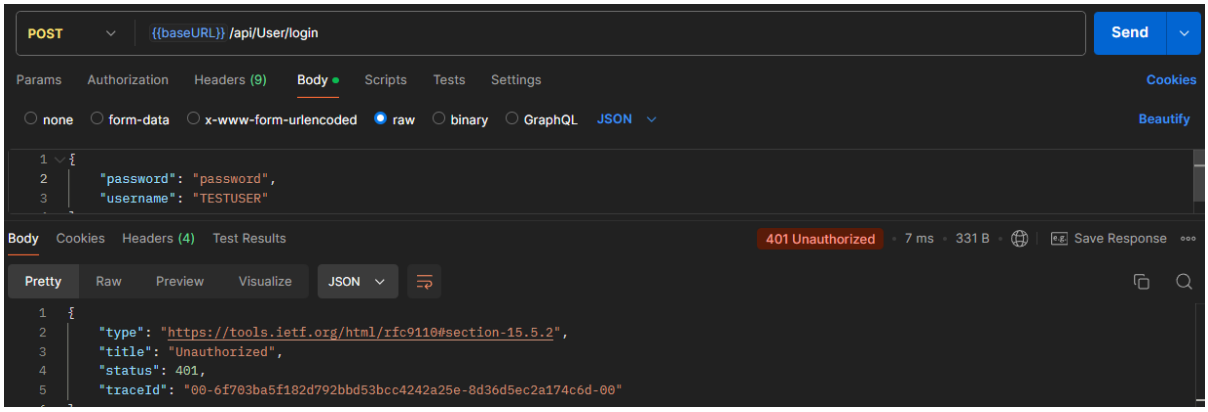


- ❖ USER-TC04-Uppercase **(Succesfull ☑)**
  - ➢ **Description:** Login with 1 valid user data and a Uppercased Valid Credential iterating between the possible combinations, between user and password, and both
  - ➢ **Current result:**
    - UppercasedCred: the user did not login
    - Status: 401
    - Response time: 13ms

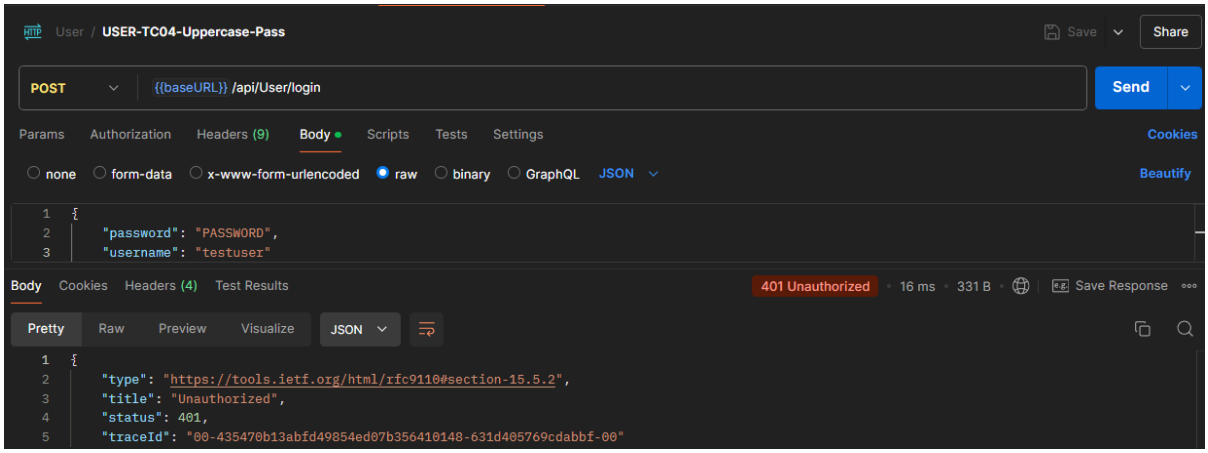


- UppercasedUser: the user did not login

- Status: 401
- Response time: 7ms



- UppercasedPass: the user did not login
- Status: 401
- Response time: 16ms

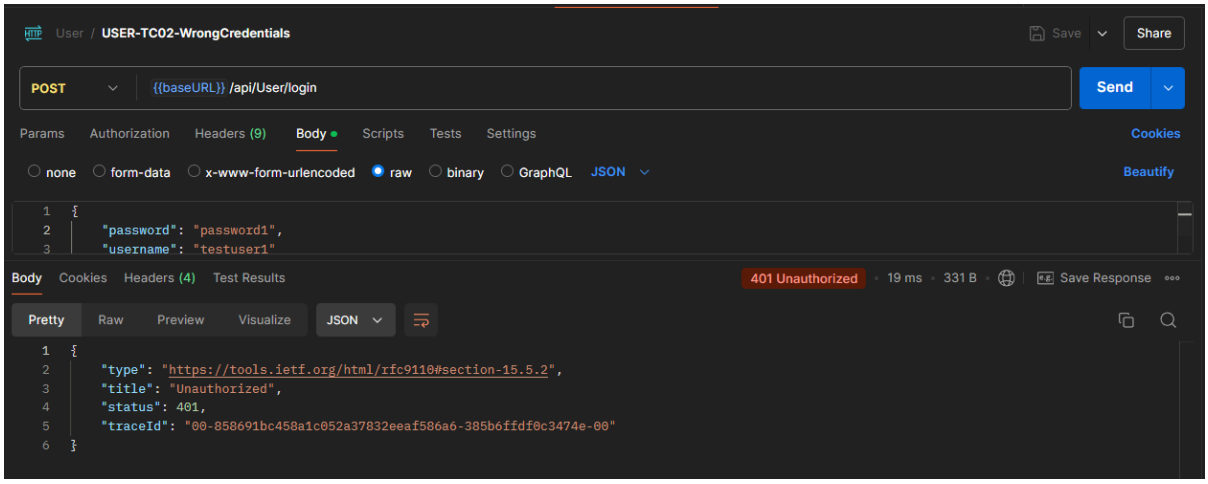


❖ **ID: USER-TC05-WrongCredentials (Succesfull ☑)**

➢ **Description: try to** Login with wrong credentials

➢ **Current result:** the user did not login

- **Status:** 401
- **Response time:** 19ms


User / USER-TC02-WrongCredentials
Save
Share
POST
{{baseURL}} /api/User/login
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"password": "password1",
"username": "testuser1"
Body
Cookies
Headers (4)
Test Results
401 Unauthorized
19 ms
331 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"type": "https://tools.ietf.org/html/rfc9110#section-15.5.2",
"title": "Unauthorized",
"status": 401,
"traceId": "00-858691bc458a1c052a37832eeaf586a6-385b6ffdf0c3474e-00"
}