PART 2: Test case development

Guide to interpret the table:

ID: A unique identifier assigned to each test case for easy tracking and organization.

- Example: TC_BACK_01 identifies a test case for the backend.
- Example: TC_FRONT_02 identifies a test case for the frontend.

Test Case Title: A short title describing the purpose of the test case. It serves to get a quick idea of what is being tested.

- Example: “Login with Valid Credentials”.

Description: A more detailed explanation of the purpose of the test case, why it is relevant and what aspect of the system is being evaluated.

- Example: “Verify that users can log in with valid credentials”.

Steps to Execute: The specific steps that must be followed to perform the test. Each step is numbered and clearly written for ease of execution.

- Example:
    1. Navigate to the login page.
    2. Enter a valid username and password.
    3. Click on the “Login” button.

Expected Result: Describes what should happen if the system works correctly. It is the expected result that allows us to validate if the test has passed or failed.

- Example: “An error message appears indicating that the credentials are incorrect.”.

BACKEND (C# APIS):

ID	Test case title	Description	Steps to execute	Expected result
TC_BACK_01	Login with Valid Credentials	Verify that users can successfully log in with valid credentials.	1. Send a POST request to `/api/User/login` with valid credentials. 2. Verify status code 200 and receive an auth token	The API responds with a valid authentication token and status code 200.
TC_BACK_02	Login with invalid user	Validate that the system rejects the login with a user who is not registered	1. Send a POST request to `/api/User/login` with invalid credentials. 2. Verify status code 401.	The API responds with an error message and status code 401.
TC_BACK_03	Login with invalid password	Validate that the system rejects the login with an unregistered password.	1. Send a POST request to `/api/User/login` with invalid credentials. 2. Verify status code 401.	The API responds with an error message and status code 401.
TC_BACK_04	Login with both user and password fields empty	Validate that the system does not authorize login with both fields empty.	1. Send a POST request to `/api/User/login` with no credentials. 2. Verify status code 401	The API responds with an error message and status code 401.
TC_BACK_05	Login with empty user field	Validate that the system does not authorize login with an empty user field.	1. Send a POST request to `/api/User/login` with no credentials. 2. Verify status code 401	The API responds with an error message and status code 401.
TC_BACK_06	Login with empty password field	Validate that the system does not authorize login with an empty password field.	1. Send a POST request to `/api/User/login` with no credentials. 2. Verify status code 401	The API responds with an error message and status code 401.

TC_BACK_07	Create Product	Verify that the API allows creating a product with valid data.	1. Send a POST request to `/api/Product` with valid product data (id name, price).	The API responds with a success message and status code 201.
TC_BACK_08	Consult Product List	Verify that the API allows consulting the list of products.	1. Send a GET request to `/api/Product`. 2. Verify status code 200 and list of products.	The API responds with the complete list of products and status code 200.
TC_BACK_09	Create Product with Existing ID	Verify the API handles creating a product when the product ID already exists.	1. Send a POST request to `/api/Product` with an existing product ID. 2. Verify status code 409.	The API responds with an error message and status code 409, indicating ID conflict.
TC_BACK_10	Edit Product by ID	Verify that the API allows editing a product using its ID.	1. Send a PUT request to `/api/Product/{id}` with updated product details. 2. Verify status code 204.	The API responds with a success message and status code 204.
TC_BACK_11	Delete Product by ID	Verify that the API allows deleting a product by ID.	1. Send a DELETE request to `/api/Product/{id}`. 2. Verify status code 204.	The API responds with a success message and status code 204.
TC_BACK_12	Consult Product by ID	Check that the API returns a specific product when a valid ID is provided.	1. Send a GET request to `/api/Product/{id}`. 2. Verify status code 200.	The API responds with a success message and status code 200.

TC_BACK_13	Create Order with Invalid Quantity	Validate the API rejects order creation when quantity is invalid (0 or negative)	1. Send a POST request to `/api/Order` with invalid quantity (e.g., 0 or negative). 2. Verify status code 400 or 422.	The API responds with an error message and status code 400 or 422.
TC_BACK_14	Create Order with Valid Data	Verify that the API allows creating an order with valid data.	1. Send a POST request to `/api/Order` with valid order details. 2. Verify status code 201	The API responds with a success message and status code 201.
TC_BACK_15	Update Order with Valid Data	Verify that the API allows updating an existing order with valid data.	1. Send a PUT request to `/api/Order/{id}` with updated data (productName, quantity, status). 2. Verify status code 200 or 204.	The API responds with a success message and status code 200 or 204.
TC_BACK_16	Delete Order with Non-Existent ID	Verify the API correctly handles deleting an order with a non-existent ID.	1. Send a DELETE request to `/api/Order/{id}` with a non-existent order ID. 2. Verify status code 404.	The API responds with an error message and status code 404.
TC_BACK_17	Delete Order with Non-Numeric ID	Verify the API handles requests for deleting orders with non-numeric IDs.	1. Send a DELETE request to `/api/Order/{id}` with a non-numeric ID (e.g., string) 2. Verify status code 400.	The API responds with an error message and status code 400.

TC_BACK_18	View all orders	Verify that the API correctly returns all existing orders.	1. Send a GET request to`/api/Order/`with out additional parameters. 2. Verify status code 200.	The API responds with a complete list of commands and a status code 200.
TC_BACK_19	Search for an order using an existing ID.	Verify that the API returns a specific order when a valid ID is provided.	1. Send a DELETE request to `/api/Order/{id}` 2. Verify status code 201	The API responds with the details of the specific order and a 201 status code
TC_BACK_20	Create order with empty productName	Validate that the API properly handles the creation of an order when the productName is empty.	1. Send a POST request to`/api/Order/` with empty productName in the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422.
TC_BACK_21	Create order with quantity at 0	Verify that the API rejects the creation of an order when quantity is equal to 0, since it does not make sense to create an order without products.	1. Send a POST request to`/api/Order/` with quantity equal to 0 in the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422
TC_BACK_22	Create order with empty status	Verify that the API properly handles the creation of an order when status is empty.	1. Send a POST request to `/api/Order/` with empty status in the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422

TC_BACK_23	Create order with duplicate ID	Validate that the API rejects the creation of an order when the id already exists.	1. Send a POST request to`/api/Order/` with an id that already exists. 2. Verify status code 409.	The API responds with an error message and a 409 status code, indicating ID duplication.
TC_BACK_24	Create order with all empty fields	Validate that the API rejects the creation of an order when all fields are empty.	1. Send a POST request to`/api/Order/` with all empty fields in the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422
TC_BACK_25	Update the ID of an order	Verify that the API does not allow you to modify the id of an existing order.	1. Send a PUT request to `/api/Order/{id}` with a body that includes a different id than the current one 2. Verify status code 400 or 409.	The API responds with an error message and a status code 400 or 409, indicating that the id is not editable.
TC_BACK_26	Update the order with an empty status	Verify that the API properly handles the update of an order when status is empty.	1. Send a PUT request to `/api/Order/{id}` with empty status in the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422.
TC_BACK_27	Prevent deletion of existing order	Verify that the API prevents the deletion of an existing order when a valid order ID is provided.	1. Send a DELETE request to `/api/Order/{id}` with a valid id in the URL. 2.Verify status code 403 or 405	The API responds with an error message indicating that the deletion is not allowed, and returns a status code 403 or 405

TC_BACK_28	Update order with empty productName	Validate that the API properly handles the update of an order when the productName is empty.	1. Send a PUT request to `/api/Order/{id}` with empty productName in the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422.
TC_BACK_29	Update order with negative quantity	Validate that the API rejects the update of an order when quantity is negative.	1. Send a PUT request to `/api/Order/{id}` with negative quantity on the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422
TC_BACK_30	Update order with all fields empty	Validate that the API rejects the update of an order when all fields are empty.	1. Send a PUT request to `/api/Order/{id}` with all empty fields in the body. 2. Verify status code 400 or 422.	The API responds with an error message and a status code 400 or 422

Frontend (ReactJS)

ID	Test case title	Description	Steps to execute	Expected result
TC_FRONT_01	Login with Valid Credentials	Verify that users can successfully log in with valid credentials.	1. Navigate to the login page. 2. Enter a valid username and password. 3. Click on "Login".	The user is redirected to the Dashboard page and a welcome message is displayed.
TC_FRONT_02	Login with Invalid Credentials	Validate that the system rejects login with incorrect credentials.	1. Navigate to the login page. 2. Enter an invalid username and password. 3. Click on "Login".	An error message appears indicating that the credentials are incorrect.

TC_FRONT_03	Login with Empty Username and Password	Verify that the login form prevents submission when both the username and password fields are empty.	1. Navigate to the login page. 2. Leave the username and password fields empty. 3. Click the "Login" button. 4. Observe the behavior of the form.	The login form displays validation error messages indicating that the fields are required. The login request is not sent to the server.
TC_FRONT_04	View Product List	Verify that the product list is displayed correctly.	1. Login to the application. 2. Navigate to the product list page. 3. Verify that all products are displayed with correct details (name, price, etc.).	The product list page displays all available products with accurate information
TC_FRONT_05	Navigation between Sections	Verify that it is possible to navigate between available sections	1. Log in to the application. 2. click on the various sections (e.g. “Products”, “Orders”).	The user is redirected to the correct section, and the content is displayed as expected.
TC_FRONT_06	Responsive Design	Validate that the UI displays correctly on different screen sizes.	1. Open the application on a desktop browser. 2. Resize the browser window to simulate different devices. 3. Observe layout changes.	The layout adjusts properly for desktop, tablet, and mobile screen sizes.
TC_FRONT_07	Logout Functionality	Verify that clicking the "Logout" button logs out the user and redirects them to the login page.	1. Log in to the application with valid credentials. 2. Navigate to a protected section of the app (e.g., Dashboard or Product). 3. Click on the "Logout" button in the navigation bar or menu. 4. Observe the system behavior after the action.	The user is logged out, their session is cleared, and they are redirected to the login page. Any attempt to access protected pages without logging in redirects back to the login page.