PARAM Fundamentals of Quality Assurance Final Project

API AND UI TEST AUTOMATION

Abdallah Osman – 2106433
Nayef Talib - 2359957
Yaseen Yousif - 1906399
Shaded Baaj – 2019451

Summary

This project, part of a Software Quality Assurance Fundamentals course, demonstrates comprehensive software testing techniques. **API testing**, using Postman, automates HTTP requests to check functionality and error handling. **UI testing**, using Selenium, automates user interactions to verify user interface elements and functionality. The combined approach ensures application reliability and quality. A detailed report documents the testing process and results. This project effectively bridges theoretical knowledge with practical application.

API Testing Points

- Base URL : https://a_leibpaqkgx3.v7.demo.nocobase.com/api/order:list

HEADERS :

● AUTHORIZATION: BEARER EYJHBGCIOIJIUZI1NIISINR5CCI6IKPXVCJ9.EYJ1C2VYSWQIOJESINJVBGVOYW1LIJOICM9VDCISIMLHDCI6MTCZNTEZMJY0NCWIZXHWIJOXNZM3NZI0NJQ0FQ.K8WKEX1HPDEPBWXF-T8MJEV91YTM9J9LSFPX7PCCZ8I

● ACCEPT:APPLICATION/JSON

● CONTENT-TYPE: APPLICATION/JSON

We tested 6 scenarios focusing on GET, POST, PUT and DELETE methods to cover positive (success) and negative (error) cases

Workspace Access: You can directly Find it on GitHub “Postman workspace”.

```
https://a_leibpaqkgx3.v7.demo.nocobase.com/api/order:list
```

```
to verify that its endpoints handle both success and failure scenarios correctly.
Wecheckedthe status codes, response formats, error handling, and
performance. We used Postman for automation, placing our test scripts in the
“Scripts” → “Post-res” section (which runs after a response is received).
```

Test Cases Overview

Our testing plan originally focused on Six main scenarios involving Create (POST) and Get (GET).
Also including Update (PUT) and Delete (DELETE), ensuring full CRUD coverage.

1. **Create Order Scenario (200)**
   Verifies successful creation of an order with valid input and proper authentication, returning a 200 OK status.

2. **Create Order Scenario (403)**
   Tests for forbidden access when attempting to create an order without the necessary authorization, returning a 403 Forbidden status.

3. **Get Order Scenario (200)**
   Validates retrieval of an existing order by its ID with proper authorization, ensuring a 200 OK response with the order details.

4. **Get Order Scenario (400**)
   Checks behavior when attempting to fetch a non-existent or malformed order ID, expecting a 400 Bad Request response.

5. **Update Order Scenario (200)**
   Confirms the ability to update an existing order with valid data, returning a 200 OK response with updated order information.

6. **Delete Order Scenario (200)**
   Ensures successful deletion of a specified order with valid input and authorization, expecting a 200 OK confirmation response.

For each scenario, our Postman test scripts checked status codes, response body fields, headers, and performance .

Scenario Case 1 :

Create Order Scenario (200).

Method: POST

A POST request to create an order with valid input and proper authentication, returning a 200 OK status.

Request Body:


```
{
  "customer_id": 123,
  "order_date": "2024-12-25",
  "items": [
    { "product_id": 101, "quantity": 2 },
    { "product_id": 102, "quantity": 1 }
  ]
}
```


Headers:

- **Authorization:** Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInJvbGVOYW1lIjoicm9vdCIsImlhdCI6MTczNTEzMjY0NCwiZXhwIjoxNzM3NzI0NjQ0fQ.k8WkeX1HpDEPBWXF-T8mjeV91YTm9j9LSFPx7pCCz8I
- **Content-Type:** application/json

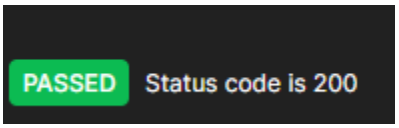
Expected Response : 200 OK

Actual Response :


```
"data": [
    {
        "createdAt": "2024-02-21T22:41:21.293Z",
        "updatedAt": "2024-04-04T02:18:16.113Z",
        "id": 1,
        "total": 12198,
        "company_ID": 3,
        "tax": 0,
        "contact_id": 49,
        "subtotal": 12198,
        "discount": 1,
        "adjustment": 0,
        "currency_ID": null,
        "order_ID": "202402210000",
        "status": "2",
        "sort": 1,
        "createdById": 1,
        "updatedById": 1
```

Post-response Script :

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```



Scenario Case 2 :
Create Order Scenario (403).
Method: POST

A POST request to create an order with valid input with no authentication, returning a 403 Forbidden status.

```
{
  "customer_id": 123,
  "order_date": "2024-12-25",
  "items": [
    { "product_id": 101, "quantity": 2 },
    { "product_id": 102, "quantity": 1 }
  ]
}
```

Headers:

- **Authorization:** Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInJvbGVOYW1lIjoicm9vdCIsImlhdCI6MTczNTEzMjY0NCwiZXhwIjoxNzM3NzI0NjQ0fQ.k8WkeX1HpDEPBWXF-T8mjeV91YTm9j9LSFPx7pCCz8I
- **Content-Type:** application/json

Expected Response : 403 Forbidden

Actual Response :

```
{
    "errors": [
        {
            "message": "No permissions"
        }
    ]
}
```

Post-response Script :

```
pm.test("Response status code is 403", function () {
    pm.expect(pm.response.code).to.equal(403);
});


pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});


pm.test("Response has the required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.errors).to.be.an('array');
    responseData.errors.forEach(function(error) {
        pm.expect(error).to.have.property('message');
    });
});


pm.test("Content type is application/json", function () {
    pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});


pm.test("Errors array is present and contains expected number of elements", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.errors).to.exist.and.to.be.an('array');
    pm.expect(responseData.errors).to.have.lengthOf(1);
});
```

Scenario Case 3 :

Get Order Scenario (200).

Method: GET

A GET validates retrieval of an existing order by its ID with proper authorization, ensuring a 200 OK response with the order details.

```
{
  "order_id": 1,
  "customer_id": 123,
  "status": "created",
  "items": [
    { "product_id": 101, "quantity": 2 },
    { "product_id": 102, "quantity": 1 }
  ]
}
```


Headers:

- **Authorization:** Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInJvbGVOYW1lIjoicm9vdCIsImlhdCI6MTczNTEzMjY0NCwiZXhwIjoxNzM3NzI0NjQ0fQ.k8WkeX1HpDEPBWXF-T8mjeV91YTm9j9LSFPx7pCCz8I
- **Content-Type:** application/json

Expected Response : 200 OK

Actual Response :

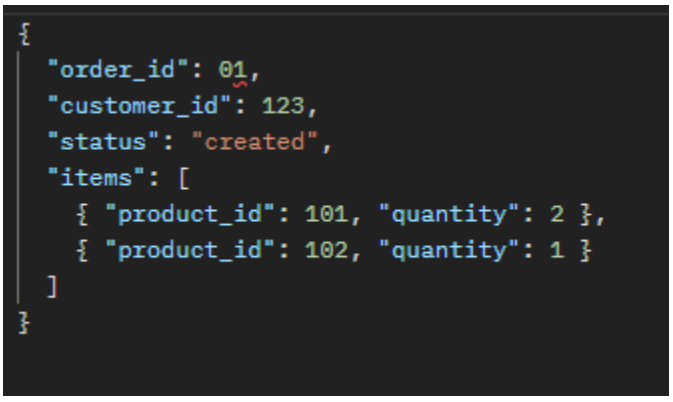

```
{
    "data": [
        {
            "createdAt": "2024-02-21T22:41:21.293Z",
            "updatedAt": "2024-04-04T02:18:16.113Z",
            "id": 1,
            "total": 12198,
            "company_ID": 3,
            "tax": 0,
            "contact_id": 49,
            "subtotal": 12198,
            "discount": 1,
            "adjustment": 0,
            "currency_ID": null,
            "order_ID": "202402210000",
            "status": "2",
            "sort": 1,
            "createdById": 1,
            "updatedById": 1
        },
```

Post-response Script :

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response status code is 200", function () {
  pm.response.to.have.status(200);
});


pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});


pm.test("Validate the data schema attributes", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.data).to.be.an('array');

    responseData.data.forEach(function(order) {
        pm.expect(order.createdAt).to.exist.and.to.be.a('string');
        pm.expect(order.updatedAt).to.exist.and.to.be.a('string');
        pm.expect(order.id).to.exist.and.to.be.a('number');
        pm.expect(order.total).to.exist.and.to.be.a('number');
        pm.expect(order.company_ID).to.exist.and.to.be.a('number');
        pm.expect(order.tax).to.exist.and.to.be.a('number');
        pm.expect(order.contact_id).to.exist.and.to.be.a('number');
        pm.expect(order.subtotal).to.exist.and.to.be.a('number');
        pm.expect(order.discount).to.exist.and.to.be.a('number');
        pm.expect(order.adjustment).to.exist.and.to.be.a('number');
        pm.expect(order.currency_ID).to.be.a('null');
        pm.expect(order.order_ID).to.exist.and.to.be.a('string');
        pm.expect(order.status).to.exist.and.to.be.a('string');
        pm.expect(order.sort).to.exist.and.to.be.a('number');
        pm.expect(order.createdById).to.exist.and.to.be.a('number');
        pm.expect(order.updatedById).to.exist.and.to.be.a('number');
    });
});
```

Scenario Case 4 :
Get Order Scenario (400).
Method: GET

A GET validates retrieval of an existing order by its ID with No authorization, ensuring a 400 Bad Request response .



```
{
  "order_id": 01,
  "customer_id": 123,
  "status": "created",
  "items": [
    { "product_id": 101, "quantity": 2 },
    { "product_id": 102, "quantity": 1 }
  ]
}
```

Headers:

- **Authorization:** Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInJvbGVOYW1lIjoicm9vdCIsImlhdCI6MTczNTEzMjY0NCwiZXhwIjoxNzM3NzI0NjQ0fQ.k8WkeX1HpDEPBWXF-T8mjeV91YTm9j9LSFPx7pCCz8I
- **Content-Type:** application/json

Expected Response : 400 Bad Request
Actual Response :

```
1   Bad Request
```

Post-response Script :

```
pm.test("Response status code is 400", function () {
    pm.expect(pm.response.code).to.equal(400);
});


pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});


pm.test("Response Content-Type is text/plain", function () {
    pm.expect(pm.response.headers.get("Content-Type")).to.include("text/plain");
});


pm.test("Response body contains 'Bad Request'", function () {
    pm.expect(pm.response.text()).to.include("Bad Request");
});


pm.test("Schema validation for the response body", function () {
    pm.expect(pm.response.json()).to.be.empty;
});
```

Scenario Case 5 :
Update Order Scenario (200).
Method: PUT

A PUT Confirms the ability to update an existing order with valid data, returning a 200 OK response.

```
{
  "status": "2"
}
```

Headers:

- **Authorization:** Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInJvbGVOYW1lIjoicm9vdCIsImlhdCI6MTczNTEzMjY0NCwiZXhwIjoxNzM3NzI0NjQ0fQ.k8WkeX1HpDEPBWXF-T8mjeV91YTm9j9LSFPx7pCCz8I
- **Content-Type:** application/json

Expected Response : 200 OK

Actual Response :


```
"data": [
    {
        "createdAt": "2024-02-21T22:41:21.293Z",
        "updatedAt": "2024-04-04T02:18:16.113Z",
        "id": 1,
        "total": 12198,
        "company_ID": 3,
        "tax": 0,
        "contact_id": 49,
        "subtotal": 12198,
        "discount": 1,
        "adjustment": 0,
        "currency_ID": null,
        "order_ID": "202402210000",
        "status": "2",
        "sort": 1,
        "createdById": 1,
        "updatedById": 1
    },
```

Post-response Script :

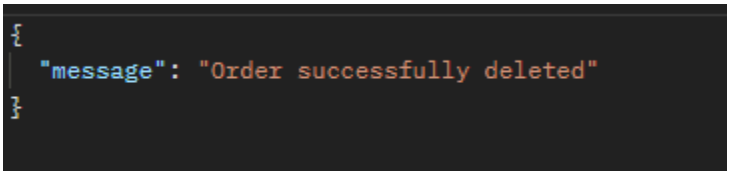
```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});


pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});


pm.test("Response has the required fields", function () {
  const responseData = pm.response.json();

  pm.expect(responseData).to.be.an('object');
  const requiredFields = ["createdAt", "updatedAt", "id", "total", "company_ID", "tax", "contact_id", "subtotal", "discount", "
    "currency_ID", "order_ID", "status", "sort", "createdById", "updatedById"];
  requiredFields.forEach(field => {
    pm.expect(responseData).to.have.property(field);
  });
});


pm.test("Total is a non-negative integer", function () {
  const responseData = pm.response.json();

  pm.expect(responseData.data).to.be.an('array').to.have.lengthOf.at.least(1);
  responseData.data.forEach(function(item) {
    pm.expect(item.total).to.be.a('number').and.to.satisfy((val) => val >= 0, "Total should be a non-negative integer");
  });
});
```

Scenario Case 6 :
Delete Order Scenario (200).
Method: DELETE.

URL: https://a_leibpaqkgx3.v7.demo.nocobase.com/api/order:list/1

Tested DELETE for order:list/1

A DELETE ensures successful deletion of a specified order with valid input and authorization.



```
{
  "message": "Order successfully deleted"
}
```

Post-response Script :

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});


pm.test("Response Content-Type header is application/json", function () {
    pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});


pm.test("Response time is less than 200ms", function () {
  pm.expect(pm.response.responseTime).to.be.below(200);
});


pm.test("Response contains required fields", function () {
    const responseData = pm.response.json();

    pm.expect(responseData.data).to.be.an('array').that.is.not.empty;

    responseData.data.forEach((order) => {
        pm.expect(order).to.include.all.keys(
            'createdAt', 'updatedAt', 'id', 'total', 'company_ID',
            'tax', 'contact_id', 'subtotal', 'discount', 'adjustment',
            'currency_ID', 'order_ID', 'status', 'sort', 'createdById', 'updatedById'
        );
    });
});
```

UI Testing Test cases:

Test Case 1: Successful Login

Purpose: To ensure that the Login page, when entered the right credentials on, Logs the user in successfully.

Python Code Snippet:

```python
import selenium
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def test_successful_login():
    # Set up WebDriver
    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    # Navigate to the URL
    driver.get("https://a_leibpaqkgx3.v7.demo.nocobase.com/signin?redirect=/admin")
    print("Page opened successfully")

    # Locate the username/email field and enter text
    try:
        username_field = WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//input[@placeholder='Username/Email']
        )
        username_field.send_keys("admin@nocobase.com")
        print("Entered username/email")
    except Exception as e:
        print(f"Error locating username/email field: {e}")
        driver.quit()
        return
```

```python
# Locate the password field and enter text
try:
    password_field = WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.XPATH, "//input[@placeholder='Password']"))
    )
    password_field.send_keys("admin123")
    print("Entered password")
except Exception as e:
    print(f"Error locating password field: {e}")
    driver.quit()
    return

# Locate the sign-in button and click it
try:
    sign_in_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//button[@type='submit']"))
    )
    sign_in_button.click()
    print("Clicked sign-in button")
except Exception as e:
    print(f"Error locating sign-in button: {e}")
    driver.quit()
    return

# Wait for the admin dashboard to load
try:
    WebDriverWait(driver, 20).until(EC.title_contains("Admin Dashboard"))
```

```python
        print("Admin Dashboard loaded")
    except Exception as e:
        print(f"Error loading Admin Dashboard: {e}")
        driver.quit()
        return

    # Verify the result (example: check the page title)
    expected_title = "Admin Dashboard"
    actual_title = driver.title
    assert expected_title == actual_title, f"Test Failed: {actual_title} != {expected_title}"
    print("Login successful")

    # Close the browser
    driver.quit()

# Run the test case
test_successful_login()
```

Output In Ideal Case: Successfully logged into the portal.

Test Case 2: Unsuccessful Login

Purpose: To ensure that if wrong credentials are entered, the website does not let the user log in.

Python code snippet:

```python
import selenium
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

def test_invalid_login():
    # Set up WebDriver
    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    # Navigate to the URL
    driver.get("https://a_leibpaqkgx3.v7.demo.nocobase.com/signin?redirect=/admin")
    print("Page opened successfully")

    # Locate the username/email field and enter text
    try:
        username_field = WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//input[@placeholder='Username/Email']"))
        )
        username_field.send_keys("invalid_user")
        print("Entered invalid username/email")
    except Exception as e:
        print(f"Error locating username/email field: {e}")
        driver.quit()
        return
```

```
    # Locate the password field and enter text
    try:
        password_field = WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//input[@placeholder='Password']"))
        )
        password_field.send_keys("invalid_password")
        print("Entered invalid password")
    except Exception as e:
        print(f"Error locating password field: {e}")
        driver.quit()
        return

    # Locate the sign-in button and click it
    try:
        sign_in_button = WebDriverWait(driver, 20).until(
            EC.element_to_be_clickable((By.XPATH, "//button[@type='submit']"))
        )
        sign_in_button.click()
        print("Clicked sign-in button")
    except Exception as e:
        print(f"Error locating sign-in button: {e}")
        driver.quit()
        return

    # Verify the result (example: check for error message)
    try:
        error_message = WebDriverWait(driver, 20).until(
```

```
                EC.presence_of_element_located((By.XPATH, "//div[@class='error-message']"))
            )
            assert error_message.is_displayed(), "Test Failed: Error message not displayed"
            print("Error message displayed")
        except Exception as e:
            print(f"Error locating error message: {e}")
            driver.quit()
            return

        # Close the browser
        driver.quit()

    # Run the test case
    test_invalid_login()
```

Output in ideal cases: The website does not let the user log in.

Test Case 3: Password masking

Purpose: To test if the password being typed is masked to ensure privacy.

Python code snippet:

```
def test_password_masking():
    # Set up WebDriver
    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    # Navigate to the URL
    driver.get("https://a_leibpaqkgx3.v7.demo.nocobase.com/signin?redirect=/admin")
    print("Page opened successfully")

    # Locate the password field and enter text
    try:
        password_field = WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//input[@placeholder='Password']"))
        )
        password_field.send_keys("password")
        print("Entered password")
    except Exception as e:
        print(f"Error locating password field: {e}")
        driver.quit()
        return

    # Verify that the password field masks the entered characters
    assert password_field.get_attribute("type") == "password", "Test Failed: Password field is not masked"
    print("Password field is masked")

    # Close the browser
    driver.quit()

# Run the test case
test_password_masking()
```

Output in ideal case: The password being typed is masked/hidden.

Test Case 4: Verify navigating to “Users” page

Purpose: To ensure that the user is successfully able to navigate to the “Users” page of the website.

Python Code Snippet:

```
    # Wait for the Users page to load
    try:
        users_button = WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//div[@role='button' and @aria-label='Users']"))
        )
        users_button.click()
        print("Clicked Users button")

        WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//div[@role='button' and @aria-label='Users']"))
        )
        print("Users page loaded")
    except Exception as e:
        print(f"Error loading Users page: {e}")
        driver.quit()
        return

    # Verify the result (example: check the page title)
    expected_title = "Users"
    actual_title = driver.title
    assert expected_title == actual_title, f"Test Failed: {actual_title} != {expected_title}"
    print("Navigation to Users page successful")

    # Close the browser
    driver.quit()

# Run the test case
test_navigation_to_users_page()
```

Output in ideal case: The website successfully navigates to the “Users” page.

Test Case 5: Navigating to the leads page and searching for a user that exists in the database.

Python Code Snippet:

```
# Navigate to the Leads page
try:
    leads_button = WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.XPATH, "//div[@role='button' and @aria-label='Leads']"))
    )
    leads_button.click()
    print("Clicked Leads button")

    WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.XPATH, "//div[@role='button' and @aria-label='Leads']"))
    )
    print("Leads page loaded")
except Exception as e:
    print(f"Error loading Leads page: {e}")
    driver.quit()
    return

# Search for the lead
try:
    search_field = WebDriverWait(driver, 20).until(
        EC.presence_of_element_located((By.XPATH, "//input[@class='ant-input css-9akr02' and @type='text']"))
    )
    search_field.send_keys("Billy Bennett")
    print("Entered lead name")

    filter_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//button[@role='button' and @aria-label='action-Action-Filter-submit-lead-filter-form']"))
    )
    filter_button.click()
    print("Clicked Filter button")
except Exception as e:
    print(f"Error searching for lead: {e}")
    driver.quit()
    return
```

```
    # Verify the lead exists
    try:
        lead_list = WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//div[@class='ant-select-selection-overflow']"))
        )
        assert "Billy Bennett" in lead_list.text, "Test Failed: Lead not found"
        print("Lead found successfully")
    except Exception as e:
        print(f"Error verifying lead: {e}")
        driver.quit()
        return

    # Close the browser
    driver.quit()

# Run the test case
test_search_lead()
```

Output in ideal case: The search system sucessfully finds Billy Bennett, And existing user.

Test Case 6: Deactivate the existing Lead

Python Code Snippet:

```python
# Deactivate the lead
try:
    deactivate_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//a[@role='button' and @aria-label='action-Action.Link-Deactivate-customize:update-lead-table-Billy Bennett']"))
    )
    deactivate_button.click()
    print("Clicked Deactivate button")
except Exception as e:
    print(f"Error deactivating lead: {e}")
    driver.quit()
    return

# Press the first confirmation button
try:
    confirmation_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//button[@type='button' and @class='ant-btn css-52f0p9 ant-btn-primary']"))
    )
    confirmation_button.click()
    print("Clicked first OK button")
except Exception as e:
    print(f"Error clicking first OK button: {e}")
    driver.quit()
    return

# Press the final OK button
try:
    final_ok_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//button[@type='button' and @class='ant-btn css-52f0p9 ant-btn-primary']"))
    )
    final_ok_button.click()
    print("Clicked final OK button")
except Exception as e:
    print(f"Error clicking final OK button: {e}")
    driver.quit()
    return
```

```python
    # Verify the lead is deactivated
    try:
        inactive_leads = WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, "//div[@class='ant-select-selection-overflow']"))
        )
        assert "Billy Bennett" in inactive_leads.text, "Test Failed: Lead not deactivated"
        print("Lead deactivated successfully")
    except Exception as e:
        print(f"Error verifying deactivation: {e}")
        driver.quit()
        return

# Run the test case
test_deactivate_user()
```

Output in ideal Case: The website successfully deactivates the found existing lead.

Test Case 7: Converting an existing lead to a User.

Python Code Snippet:

```
# Convert the lead to a user
try:
    convert_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//a[@role='button' and @aria-label='action-Action.Link-Convert to user-customize:popup-lead-table-Dr. Darryl Daniel']"))
    )
    convert_button.click()
    print("Clicked Convert to user button")
except Exception as e:
    print(f"Error converting lead to user: {e}")
    driver.quit()
    return

# Press the confirmation button
try:
    confirmation_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//button[@role='button' and @aria-label='action-Action-Convert to user-customize:triggerWorkflows-lead-form-Dr. Darryl Daniel']"))
    )
    confirmation_button.click()
    print("Clicked Convert to user confirmation button")
except Exception as e:
    print(f"Error clicking Convert to user confirmation button: {e}")
    driver.quit()
    return

# Press the final OK button
try:
    final_ok_button = WebDriverWait(driver, 20).until(
        EC.element_to_be_clickable((By.XPATH, "//button[@type='button' and @class='ant-btn css-52f0p9 ant-btn-primary']"))
    )
    final_ok_button.click()
    print("Clicked final OK button")
except Exception as e:
    print(f"Error clicking final OK button: {e}")
    driver.quit()
    return
```

```
# Run the test case
test_convert_lead_to_user()
```

Output in ideal case: The searched and filtered lead is successfully converted to a user.