UI and API Testing Report

1. Introduction

This document outlines the architectural choices of the task. The solution was developed using the Playwright framework and SpecFlow, following a basic clean architecture approach to ensure scalability and maintainability.

---

2. Architecture Overview

Solution Structure

The solution is divided into two projects:

1. **Task**:
    - Contains dependencies and feature implementations.
    - Includes:
        - **Dependencies**: Initialization of Playwright, JSON configurations,Logger and API client initialization.
        - **Features**: Includes the Specflow scenarios, organized per feature.
        - **Pages (Page Object Model - POM)**: Defines UI elements and actions.
        - **Step Definitions**: Contains test execution logic implemented with SpecFlow.

2. **Task.Contracts**:
    - Contains shared components for reusability.
    - Includes:
        - **Enums**: Stores locators for UI elements.
        - **Interfaces**: Initialization of API operations, maps test data from settings as well as defines reusable interfaces for Page Object Models (POM).
        - **Models**: Defines request and response for API testing.

- **Dependency Management**
- Playwright setup is dynamically configured:
    - GUI-related tests run only when tagged with `@gui`.
- Logging is handled using **Serilog**, providing structured and detailed test execution logs.
- **Fluent Assertions** is used for readable and expressive assertions.
- **RestClient** facilitates API interaction. In the same context, API tests run only when they are tagged with `@api`.

---

3. Test Implementation

Test Plan

GUI Testing

Features Tested:

1. **Login Functionality**:
    - Verify login with valid credentials.
    - Ensure error messages appear for invalid credentials.
    - Verify login and logout.
2. **Cart Functionality**:
    - Add items to the cart and complete a purchase.
    - Validate the prices of items in the cart that match the expected values.
    - Attempt to complete an order with missing personal information.

API Testing

APIs Tested:

1. **User Management**:
    - `GET`: Fetch user details and validate response schema and status codes.
    - `POST`: Create a new user and ensure data integrity in the response.
    - `PUT`: Update user details and verify changes.
    - `DELETE`: Delete a user and confirm the resource is removed.

Test Data

All test data is stored in a JSON file and accessed via an interface for consistency. This approach captures:

- Test data management.
- Ease of updates and reuse across tests.

Page Object Model (POM)

- UI locators are stored in an **Enum**, ensuring:
    - Reusability across multiple tests.
    - Simplified maintenance when locators change.
- Actions on UI elements are encapsulated in Page classes.

---

4. Challenges and Resolutions

Challenge 1: Using Playwright

I have only worked with Selenium up until now and for this project, I used Playwright to try something new, as it was preferred for the task.

Challenge 2: Basic Clean Architecture

I did some basic reading on clean architecture and tried to apply it to the project

Challenge 3: Locator Maintenance

- **Solution**: Centralized locators in an Enum, making better updates for locators.

Challenge 4: Data Management

- **Solution**: Centralized test data in a JSON file to ensure reusability reducing the possibility of errors.

- ## 5. Packages and Plugins

FluentAssertions
- Microsoft.Playwright
- Microsoft.Playwright.NUnit
- NUnit
- NUnit.Analyzers
- NUnit3TestAdapter
- RestSharp
- Serilog
- Specflow
- Specflow.NUnit
- Specflow.Tools.MsBuild
- Microsoft.Extensions.Configuration
- Microsoft.Extensions.Configuration.JSON

Plugins:

- Gherkin
- Reqnroll for Rider