Contents

Introduction **2**

Test Environment **2**

Web Testing Summary (Swag Labs) **3**

API Testing Summary **5**

Automation Summary **14**

Bug Summary **17**

Conclusion **17**

QA Assessment Summary

I. Introduction

The purpose of this QA assessment was to evaluate the functionality, reliability, and performance of the Swag Labs application and the JSONPlaceholder API. The assessment covered three main areas: web testing, API testing, and automation testing. Web testing focused on user interactions, checkout processes, and validation of form fields. API testing verified endpoints for retrieving, creating, and updating resources while handling both valid and invalid scenarios. Automation testing was performed to streamline repetitive test flows and ensure consistent results.

II. Test Environment

The testing was conducted on both web and API platforms to ensure proper functionality, responsiveness, and reliability across different scenarios.

Web Application: https://www.saucedemo.com
API Under Test: https://jsonplaceholder.typicode.com

Browser: Chrome (Windows 11)
Tools Used:

- Postman - API testing
- Playwright - Automation script
- Excel - Test case and bug logging

Github Repo Used:

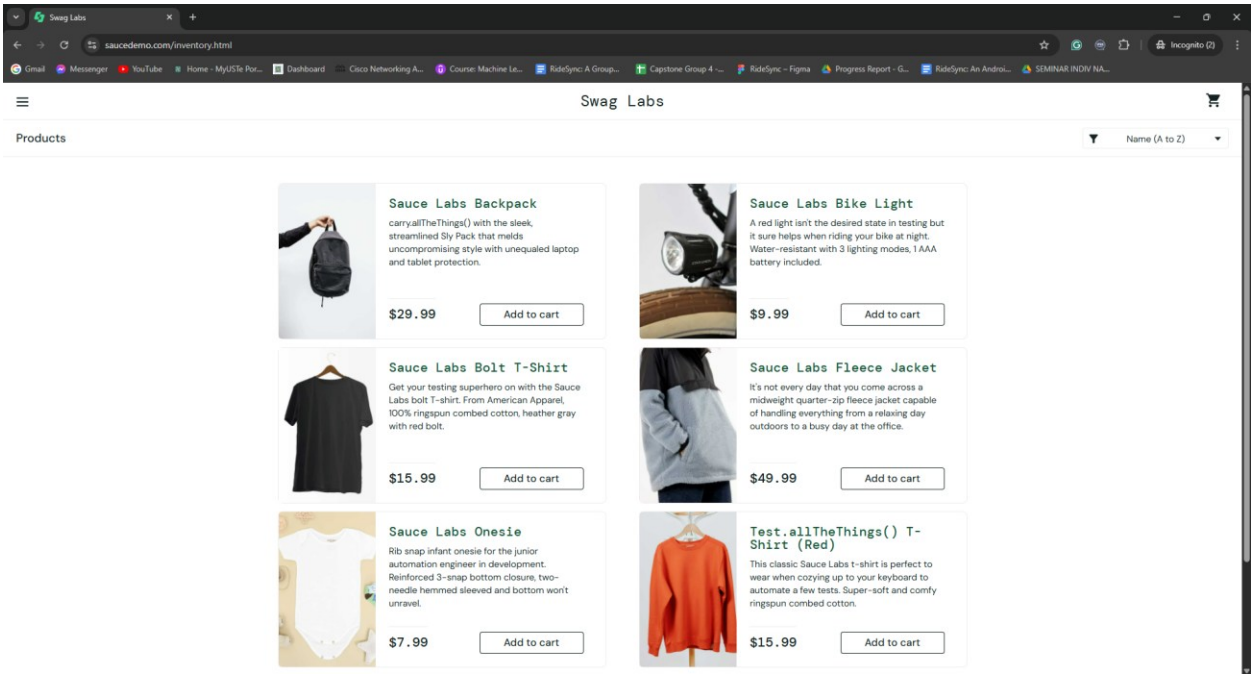
Note: All web and API test cases, along with the bug reports, are compiled in a single Excel file named QA_Assessment_TestCases_Bantucan.xlsx for easy reference and organization.

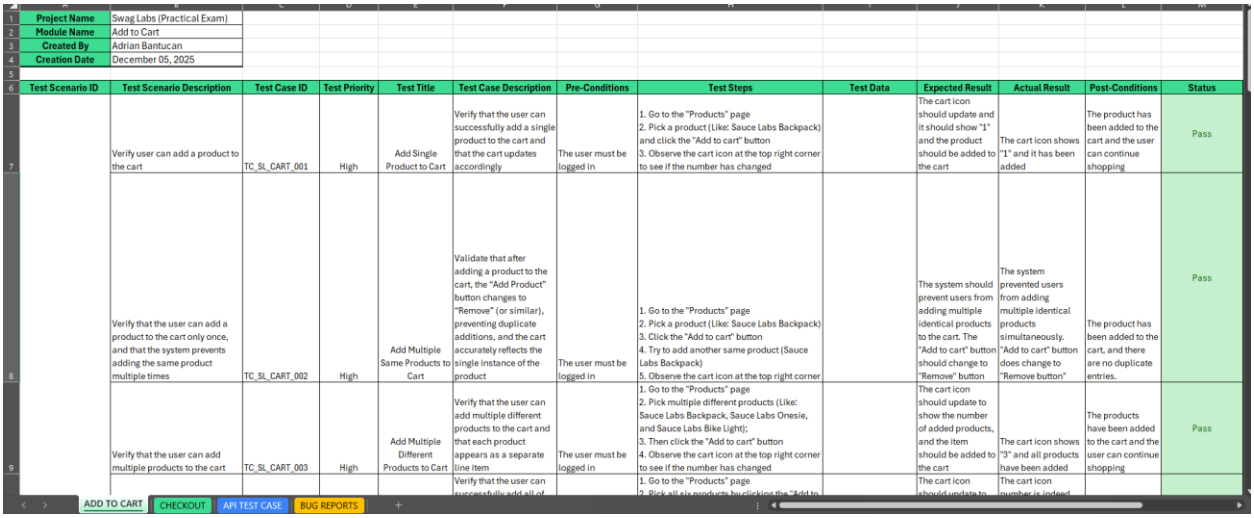
III. Web Testing Summary (Swag Labs)

Total Test Cases: 21
Pass: 11
Fail: 10

A total of 21 test cases were executed for the checkout module. Out of these, 11 passed while 10 failed. The tests included scenarios such as proceeding through the checkout steps, validating required fields, verifying order totals, and handling URL manipulation. Several major bugs were identified, including the ability to bypass checkout steps through URL changes, completing checkout with an empty cart, and resubmitting orders after completion. Minor UI issues, such as oversized clickable areas, were also noted. Screenshots captured during testing illustrate successful workflows and areas where problems occurred.

Screenshots:

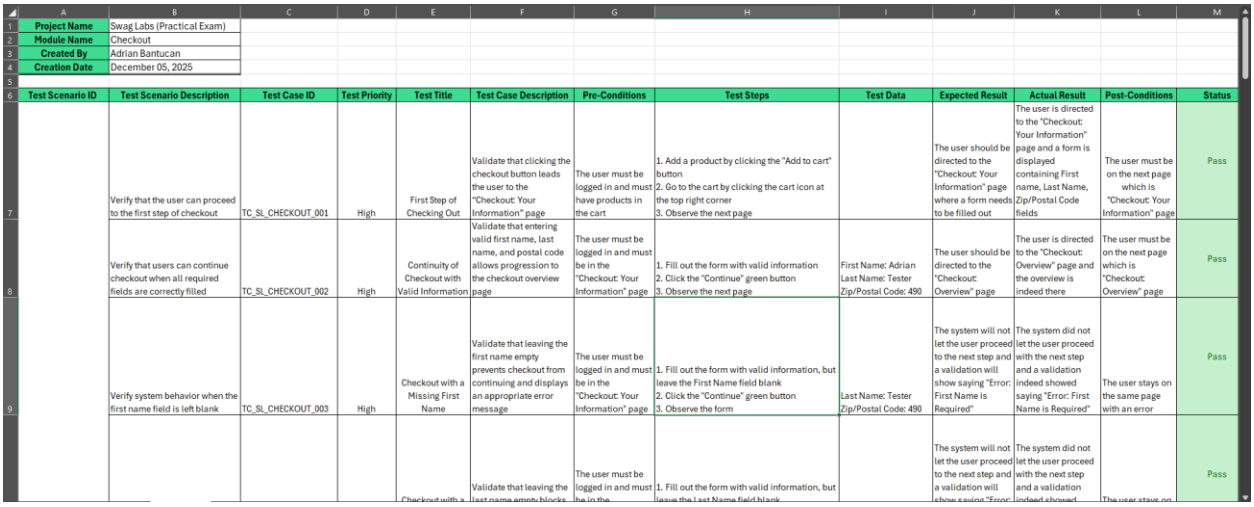




Project Name	Swag Labs (Practical Exam)
Module Name	Add to Cart
Created By	Adrian Bantucan
Creation Date	December 05, 2025

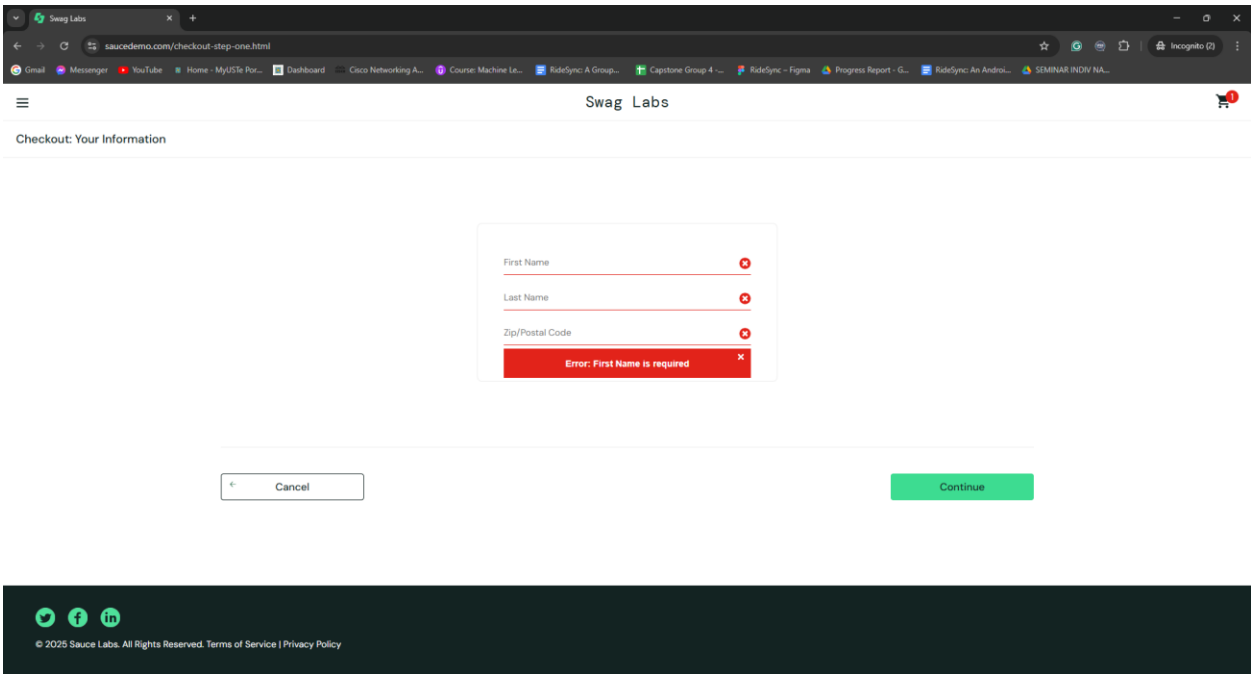
Test Scenario ID	Test Scenario Description	Test Case ID	Test Priority	Test Title	Test Case Description	Pre-Conditions	Test Steps	Test Data	Expected Result	Actual Result	Post-Conditions	Status
	Verify user can add a product to the cart	TC_SL_CART_001	High	Add Single Product to Cart	Verify that the user can successfully add a single product to the cart and that the cart updates accordingly	The user must be logged in	1. Go to the "Products" page 2. Pick a product (Like: Sauce Labs Backpack) and click the "Add to cart" button 3. Observe the cart icon at the top right corner to see if the number has changed		The cart icon should update and it should show "1" and the product should be added to the cart	The cart icon shows "1" and it has been added	The product has been added to the cart and the user can continue shopping	Pass
	Verify that the user can add a product to the cart only once, and that the system prevents adding the same product multiple times	TC_SL_CART_002	High	Add Multiple Same Products to Cart	Validate that after adding a product to the cart, the "Add Product" button changes to "Remove" (or similar), preventing duplicate additions, and the cart accurately reflects the single instance of the product	The user must be logged in	1. Go to the "Products" page 2. Pick a product (Like: Sauce Labs Backpack) 3. Click the "Add to cart" button 4. Try to add another same product (Sauce Labs Backpack) 5. Observe the cart icon at the top right corner		The system should prevent users from adding multiple identical products to the cart. The "Add to cart" button should change to "Remove" button	The system prevented users from adding multiple identical products simultaneously. "Add to cart" button does change to "Remove button"	The product has been added to the cart, and there are no duplicate entries.	Pass
	Verify that the user can add multiple products to the cart	TC_SL_CART_003	High	Add Multiple Different Products to Cart	Verify that the user can add multiple different products to the cart and that each product appears as a separate line item	The user must be logged in	1. Go to the "Products" page 2. Pick multiple different products (Like: Sauce Labs Backpack, Sauce Labs Onesie, and Sauce Labs Bike Light); 3. Then click the "Add to cart" button 4. Observe the cart icon at the top right corner to see if the number has changed		The cart icon should update to show the number of added products, and the item should be added to the cart	The cart icon shows "3" and all products have been added	The products have been added to the cart and the user can continue shopping	Pass
					Verify that the user can successfully add all of		1. Go to the "Products" page 2. Pick all six products by clicking the "Add to		The cart icon should update to	The cart icon number is indeed		

ADD TO CART | CHECKOUT | API TEST CASE | BUG REPORTS



Project Name	Swag Labs (Practical Exam)
Module Name	Checkout
Created By	Adrian Bantucan
Creation Date	December 05, 2025

Test Scenario ID	Test Scenario Description	Test Case ID	Test Priority	Test Title	Test Case Description	Pre-Conditions	Test Steps	Test Data	Expected Result	Actual Result	Post-Conditions	Status
	Verify that the user can proceed to the first step of checkout	TC_SL_CHECKOUT_001	High	First Step of Checking Out	Validate that clicking the checkout button leads the user to the "Checkout: Your Information" page	The user must be logged in and must have products in the cart	1. Add a product by clicking the "Add to cart" button 2. Go to the cart by clicking the cart icon at the top right corner 3. Observe the next page		The user should be directed to the "Checkout: Your Information" page where a form needs to be filled out	The user is directed to the "Checkout: Your Information" page and a form is displayed containing First name, Last Name, Zip/Postal Code fields	The user must be on the next page which is "Checkout: Your Information" page	Pass
	Verify that users can continue checkout when all required fields are correctly filled	TC_SL_CHECKOUT_002	High	Continuity of Checkout with Valid Information	Validate that entering valid first name, last name, and postal code allows progression to the checkout overview page	The user must be logged in and must be in the "Checkout: Your Information" page	1. Fill out the form with valid information 2. Click the "Continue" green button 3. Observe the next page	First Name: Adrian Last Name: Tester Zip/Postal Code: 490	The user should be directed to the "Checkout: Overview" page	The user is directed to the "Checkout: Overview" page and the overview is indeed there	The user must be on the next page which is "Checkout: Overview" page	Pass
	Verify system behavior when the first name field is left blank	TC_SL_CHECKOUT_003	High	Checkout with a Missing First Name	Validate that leaving the first name empty prevents checkout from continuing and displays an appropriate error message	The user must be logged in and must be in the "Checkout: Your Information" page	1. Fill out the form with valid information, but leave the First Name field blank 2. Click the "Continue" green button 3. Observe the form	Last Name: Tester Zip/Postal Code: 490	The system will not let the user proceed to the next step and a validation will show saying "Error: First Name is Required"	The system did not let the user proceed with the next step and a validation indeed showed saying "Error: First Name is Required"	The user stays on the same page with an error	Pass
					Validate that leaving the last name empty blocks	The user must be logged in and must be in the	1. Fill out the form with valid information, but leave the Last Name field blank		The system will not let the user proceed to the next step and a validation will show saying "Error:	The system did not let the user proceed with the next step and a validation indeed showed	The user stays on	Pass



IV. API Testing Summary

Endpoints Tested:
GET /users - retrieve all users
GET /users/1 - retrieve single user
GET /users/999 - non-existent user
POST /posts - create new post
PUT /posts/1 - update existing post

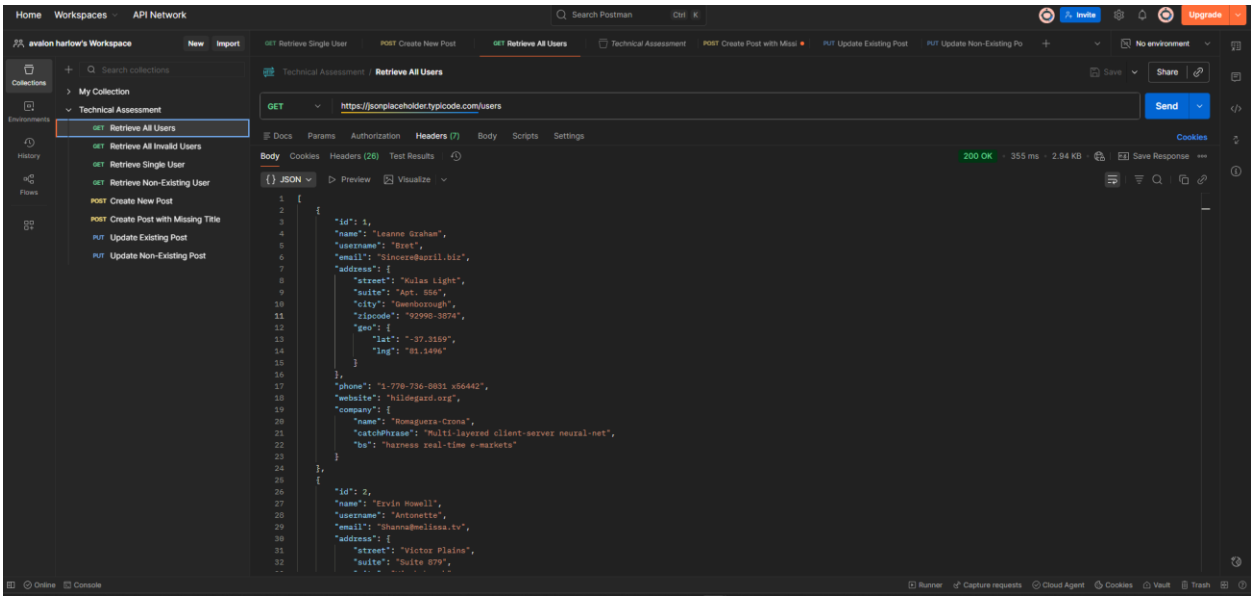
Status Codes Received:
200 OK - valid GET and PUT requests
201 Created - POST requests
404 Not Found - invalid endpoints or non-existent resources
500 Internal Server Error - updating a non-existing post

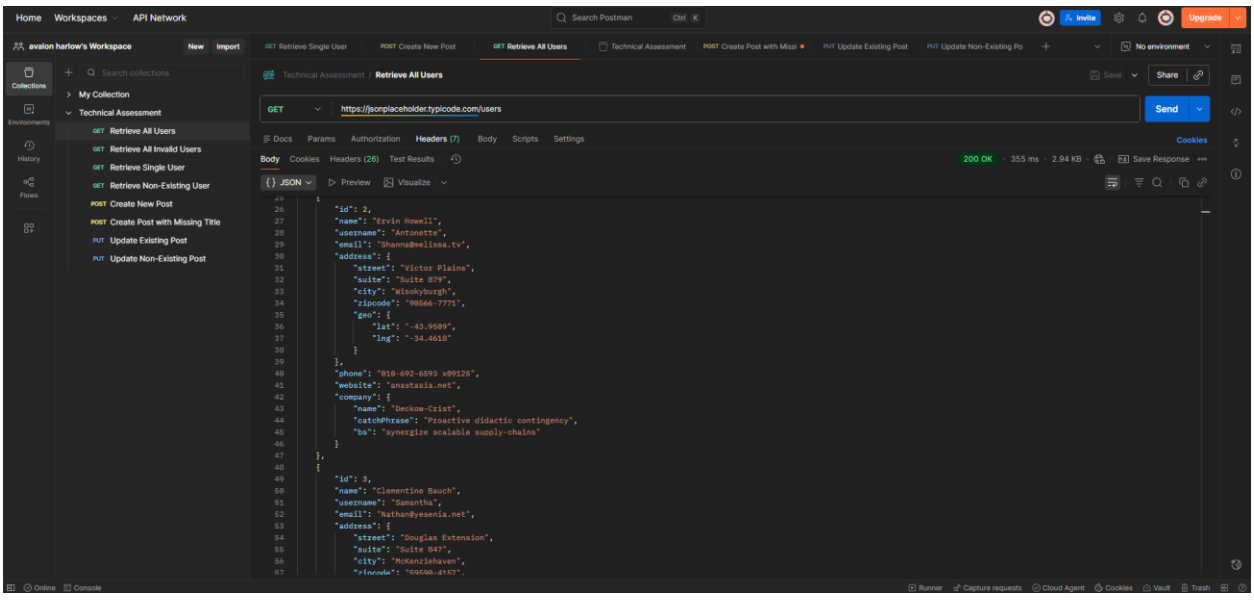
The API tests focused on the JSONPlaceholder endpoints for users and posts. Key endpoints tested included retrieving all users, retrieving a single user, creating new posts, and updating posts. Most tests returned the expected responses, including 200 OK and 201 Created status codes. Some edge cases were highlighted, such as the API allowing the creation of posts without required fields

and returning a 500 Internal Server Error when attempting to update non-existent posts. Sample responses were captured to validate data structure and content accuracy.

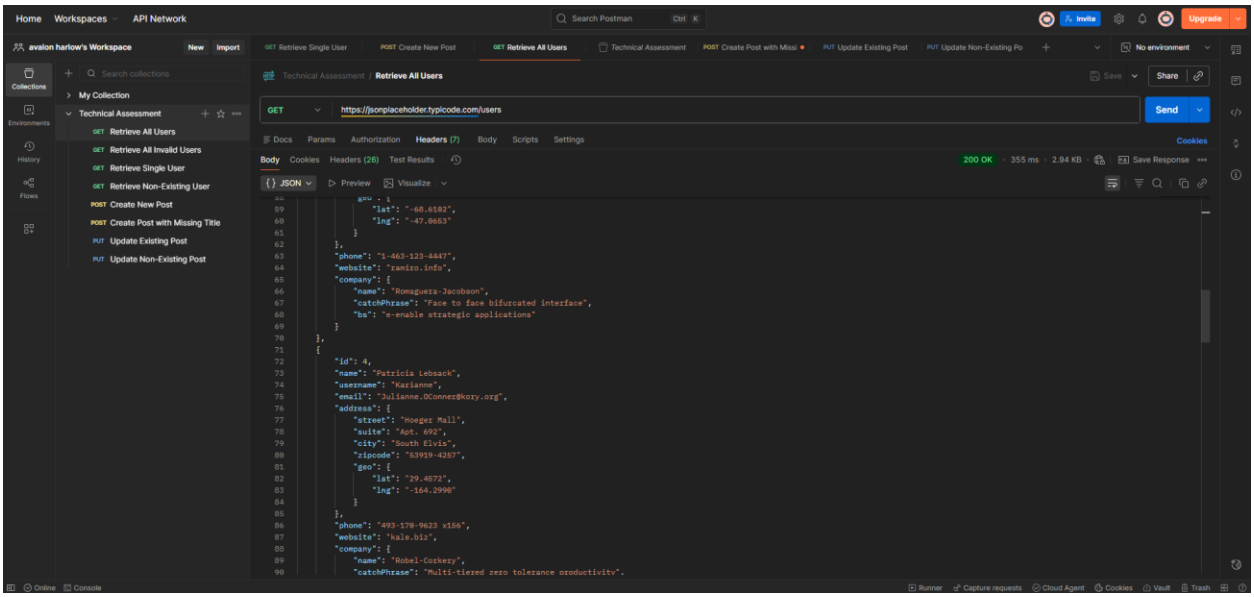
Screenshots:

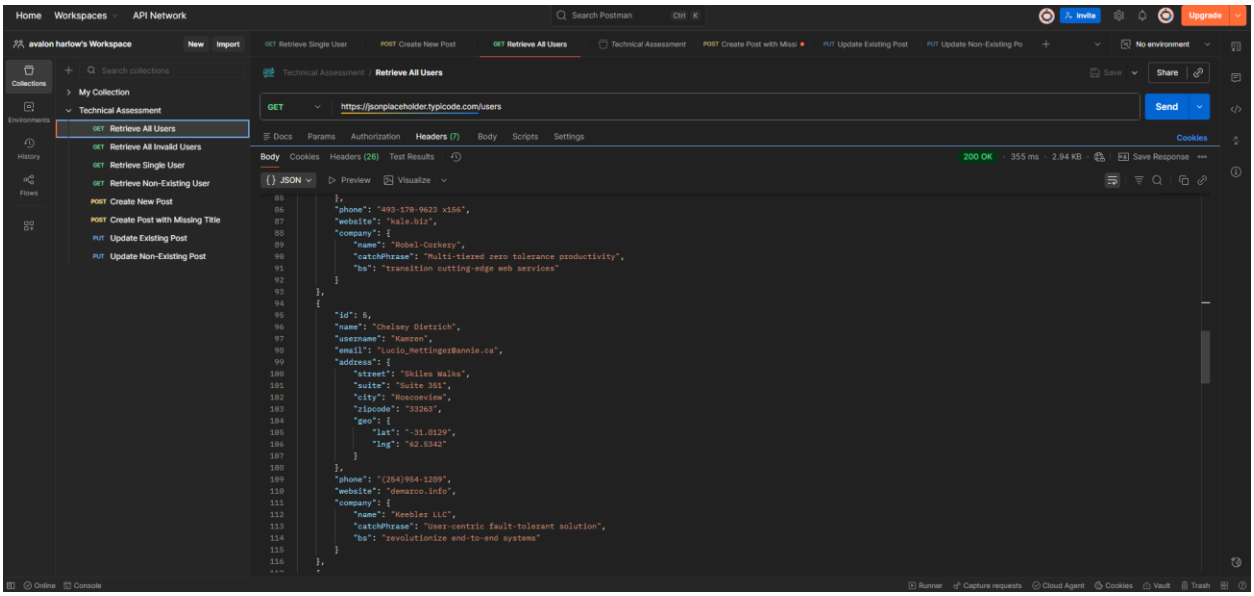
-GET Retrieve All Users



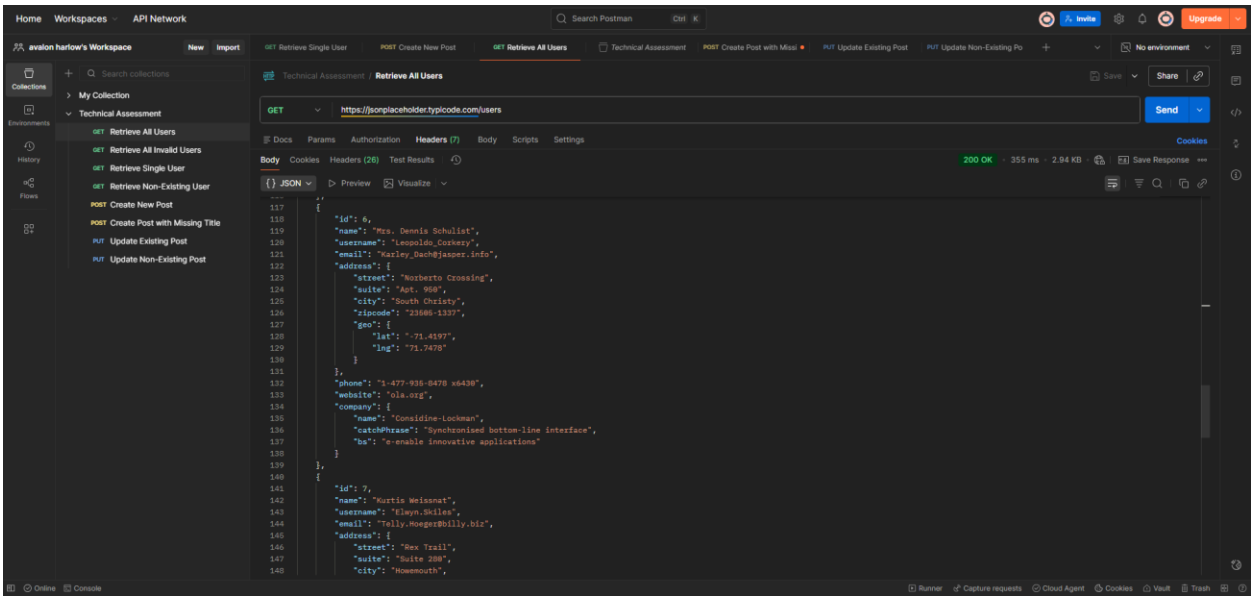


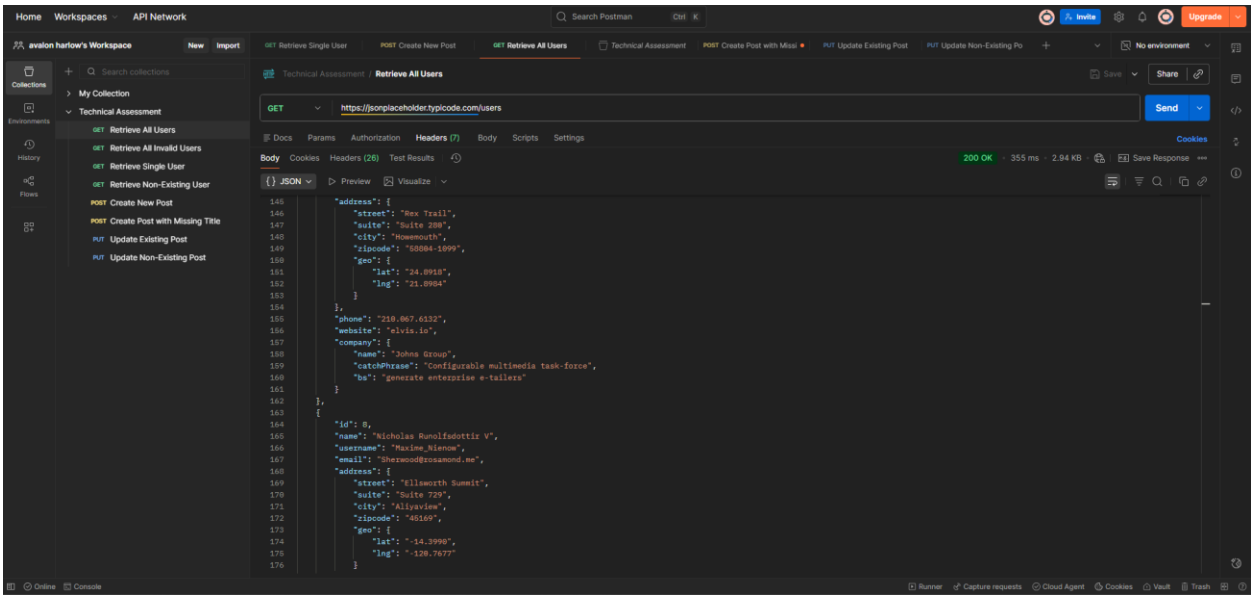
Bantucan, Adrian S.

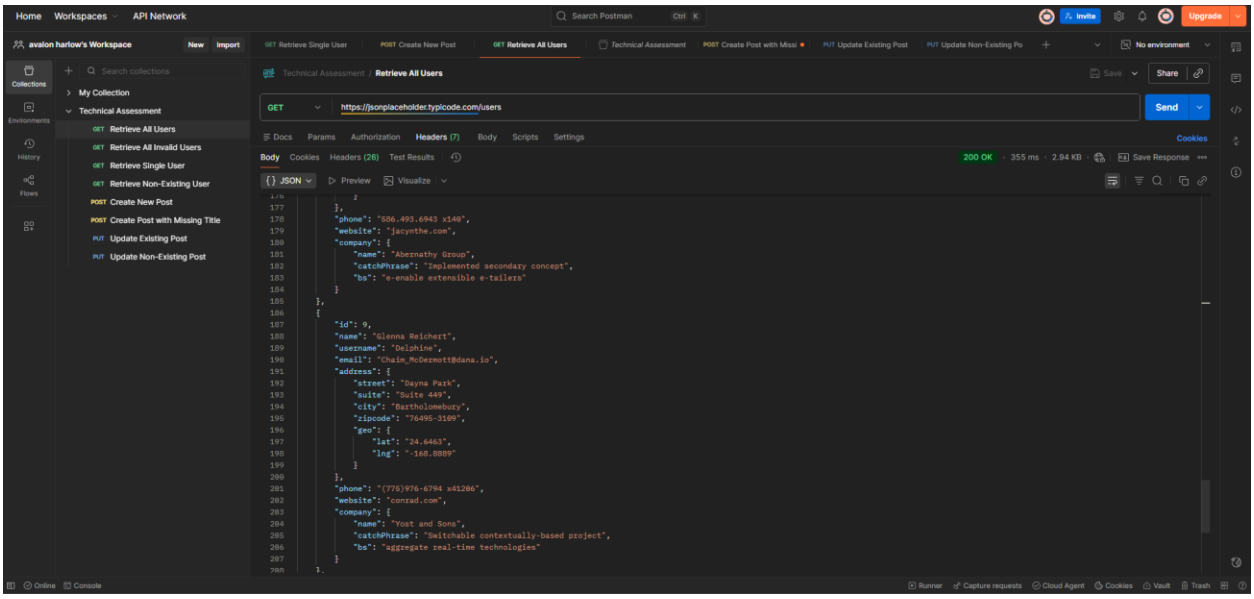


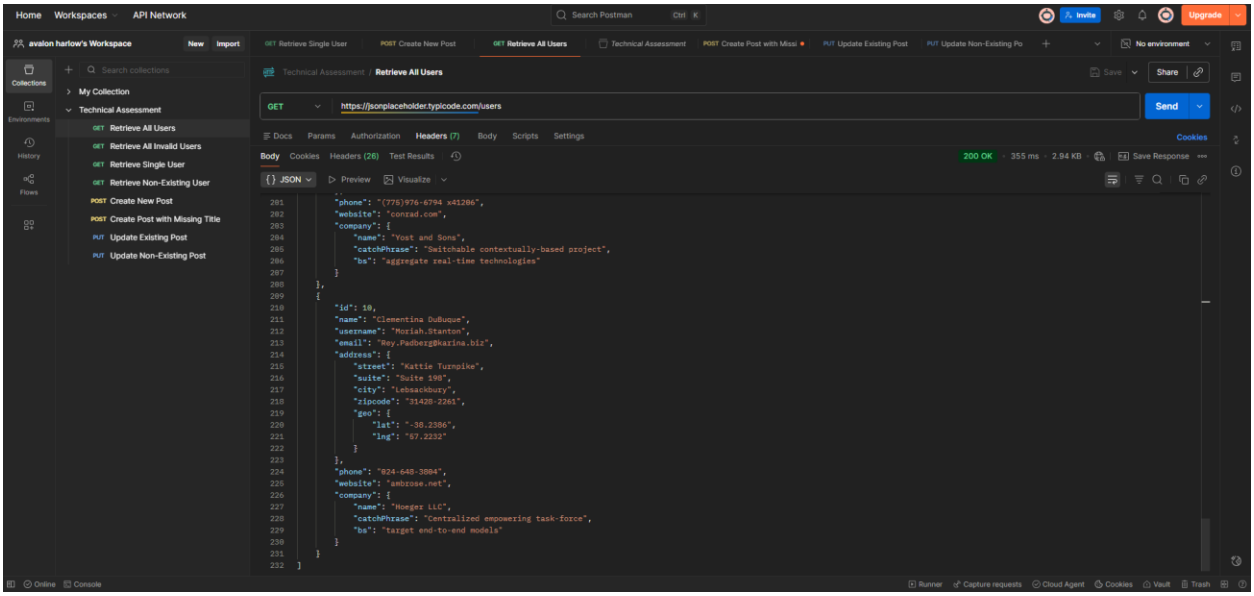


Bantucan, Adrian S.

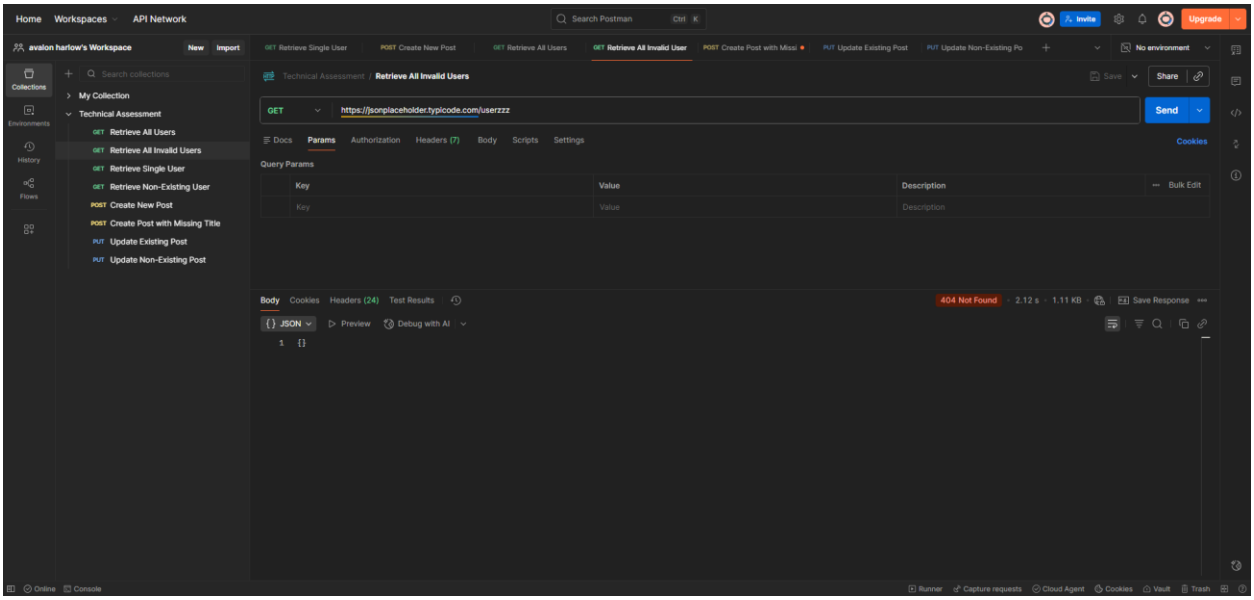




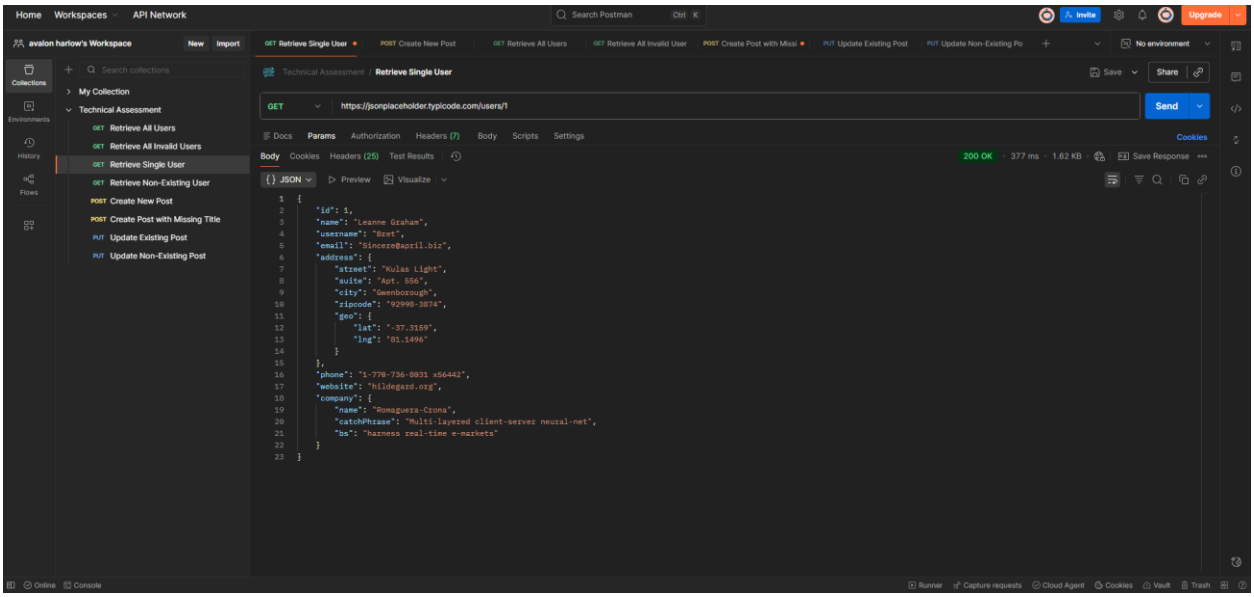


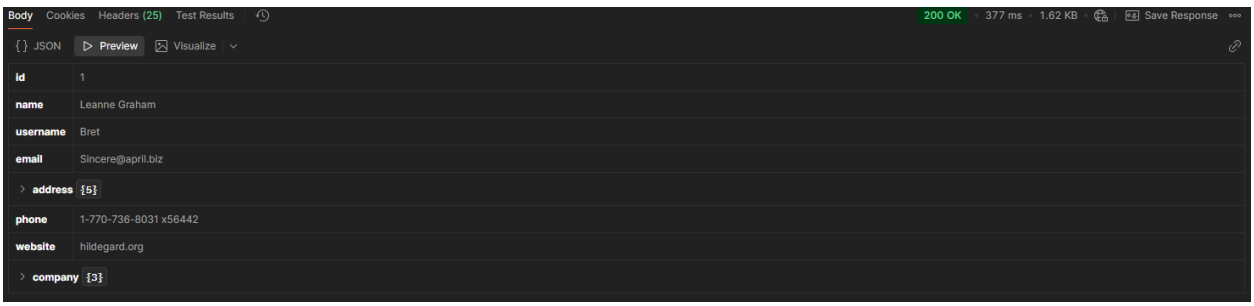


-GET Retrieve All Invalid Users



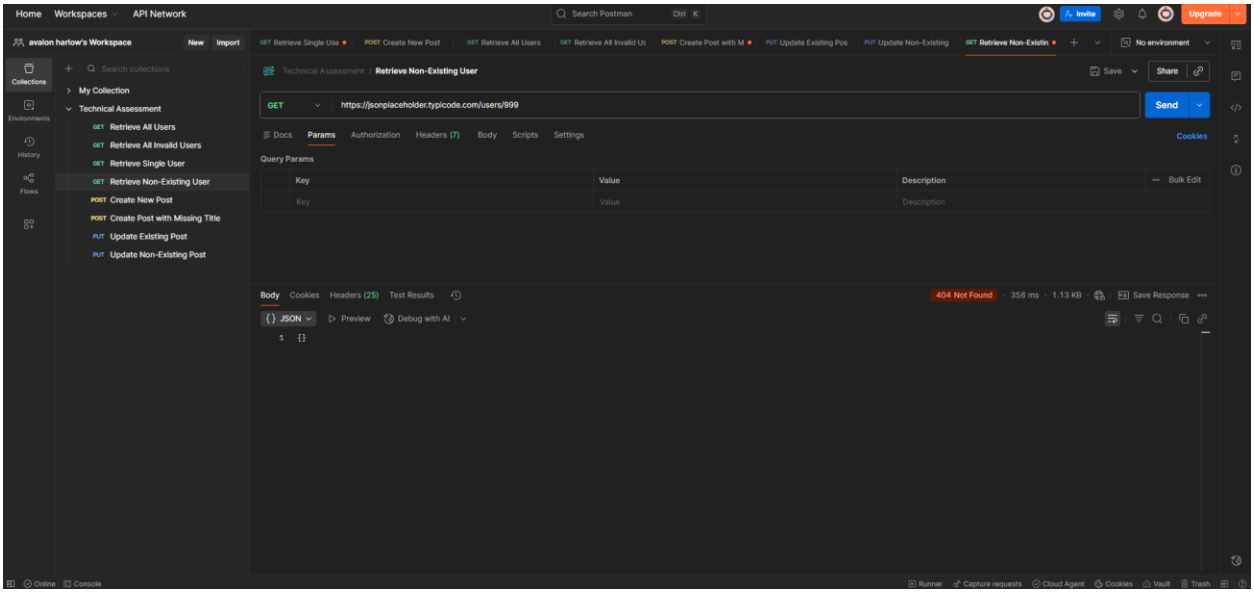
-GET Retrieve Single User



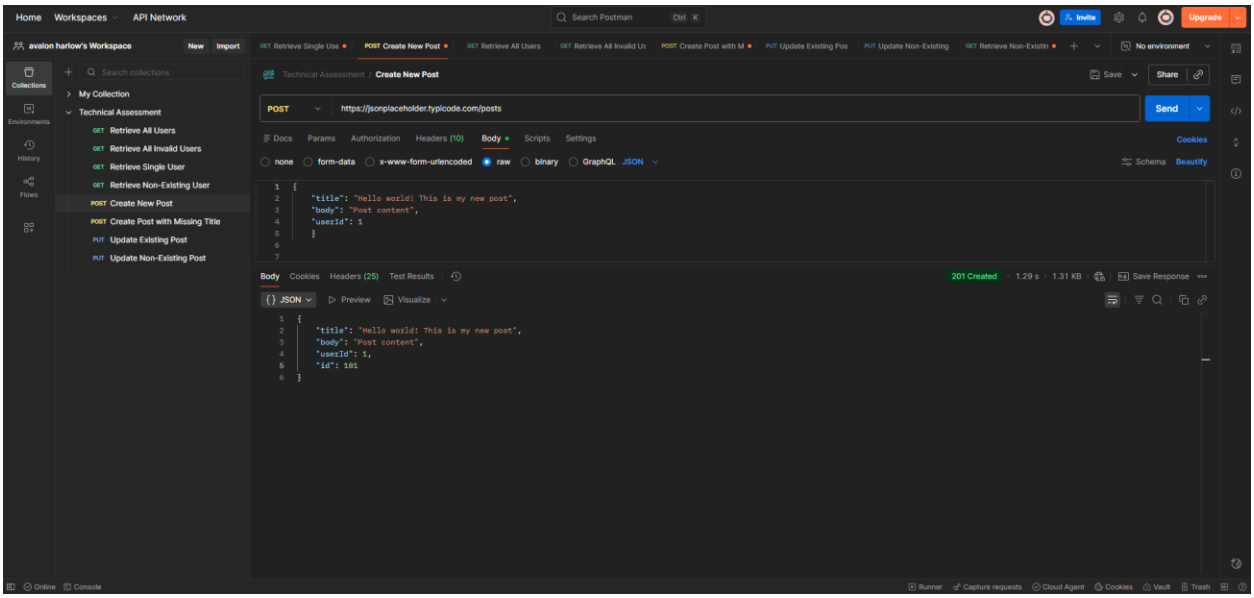


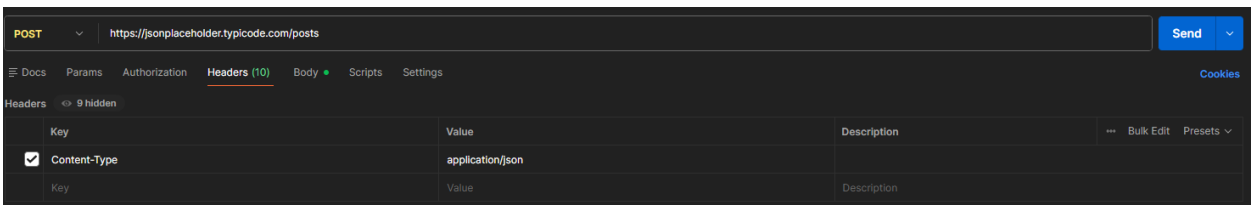
Bantucan, Adrian S.

-GET Retrieve Non-Existing User

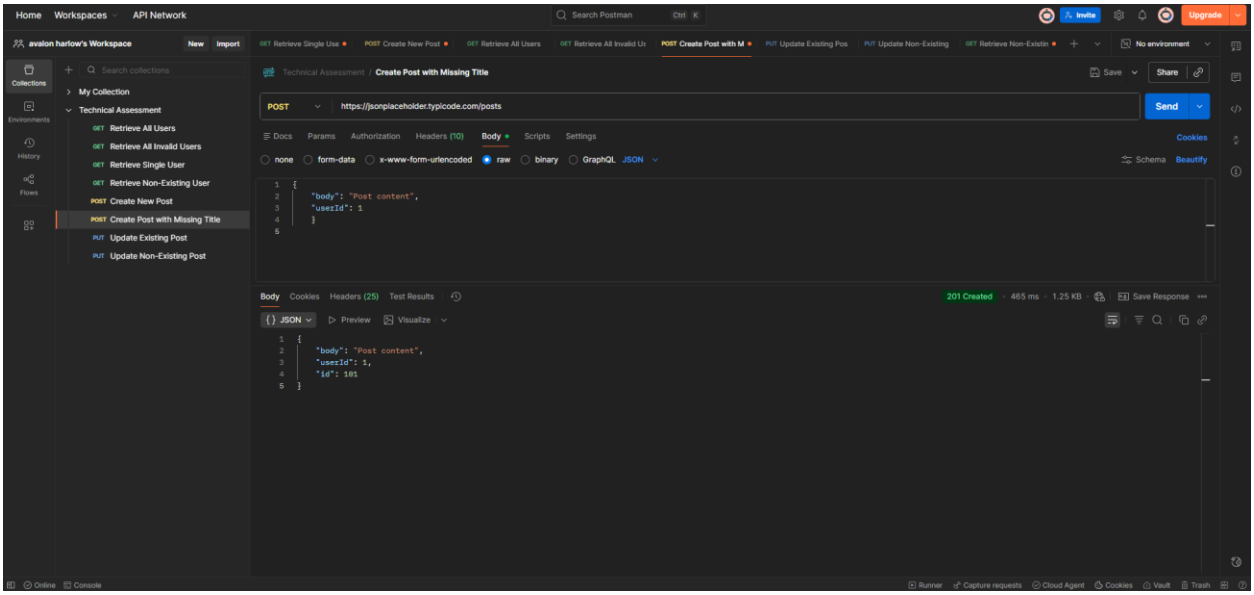


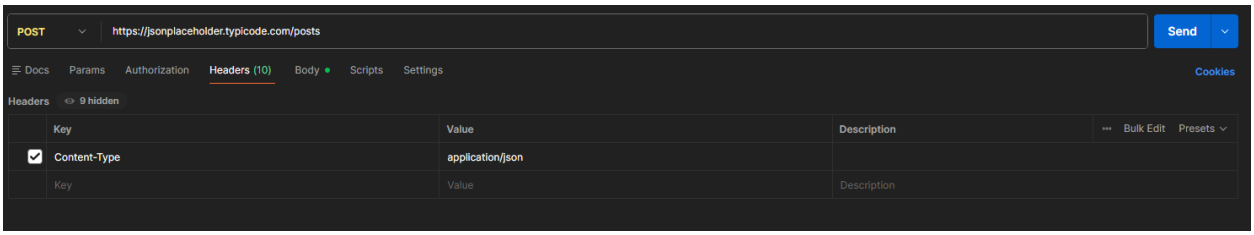
-POST Create New Post



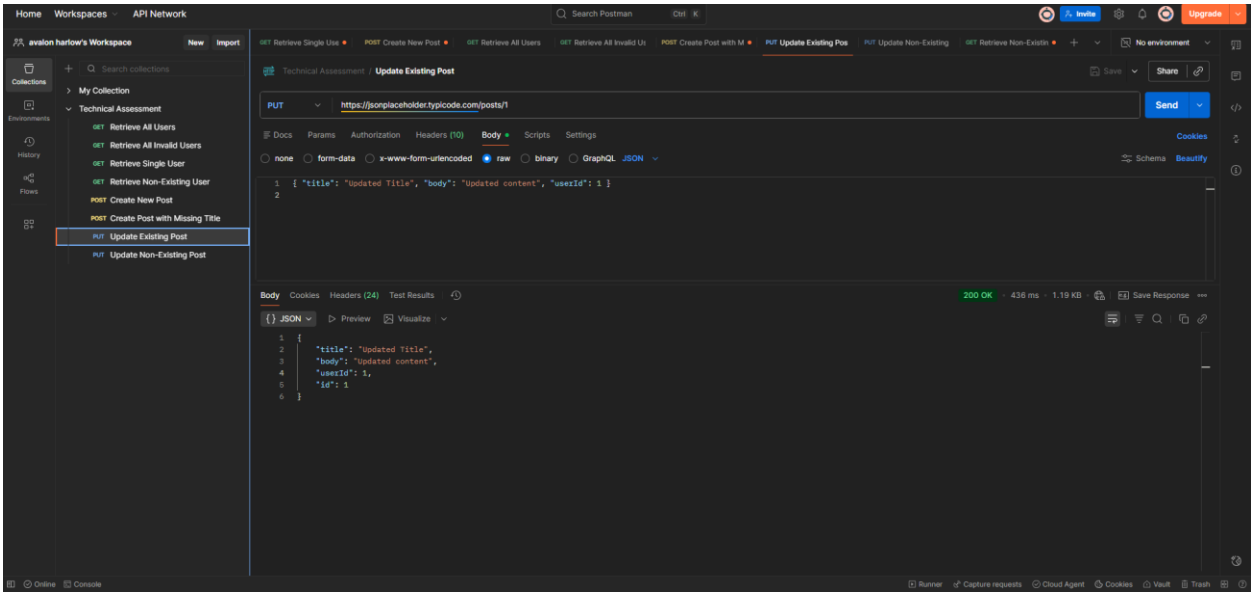


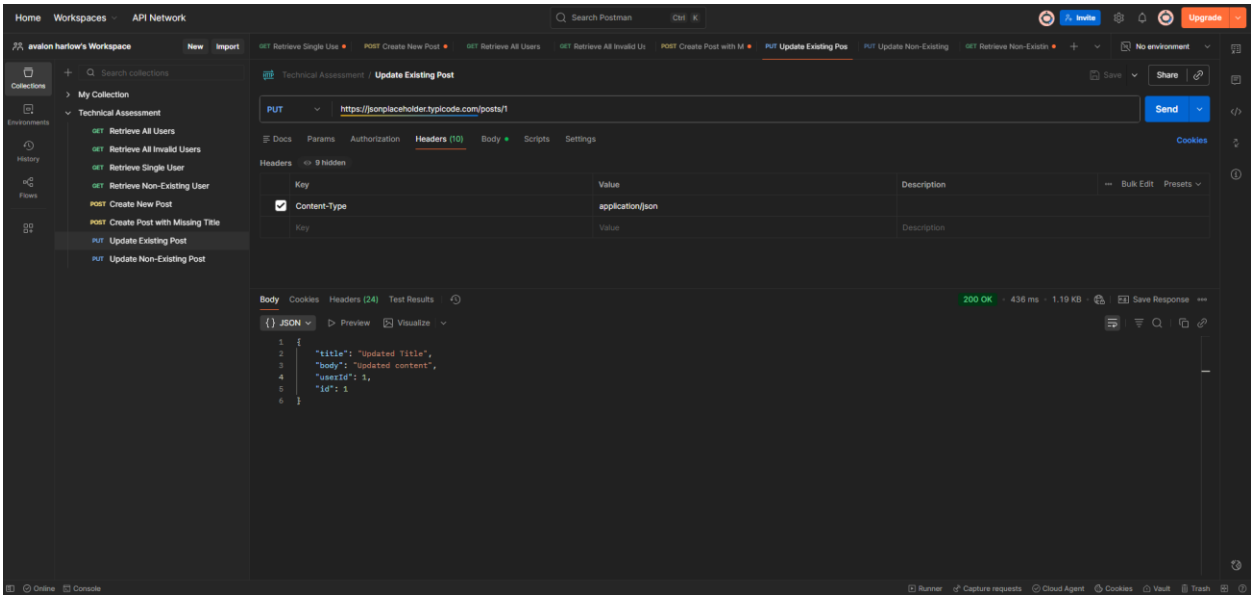
-POST Create Post with Missing Link



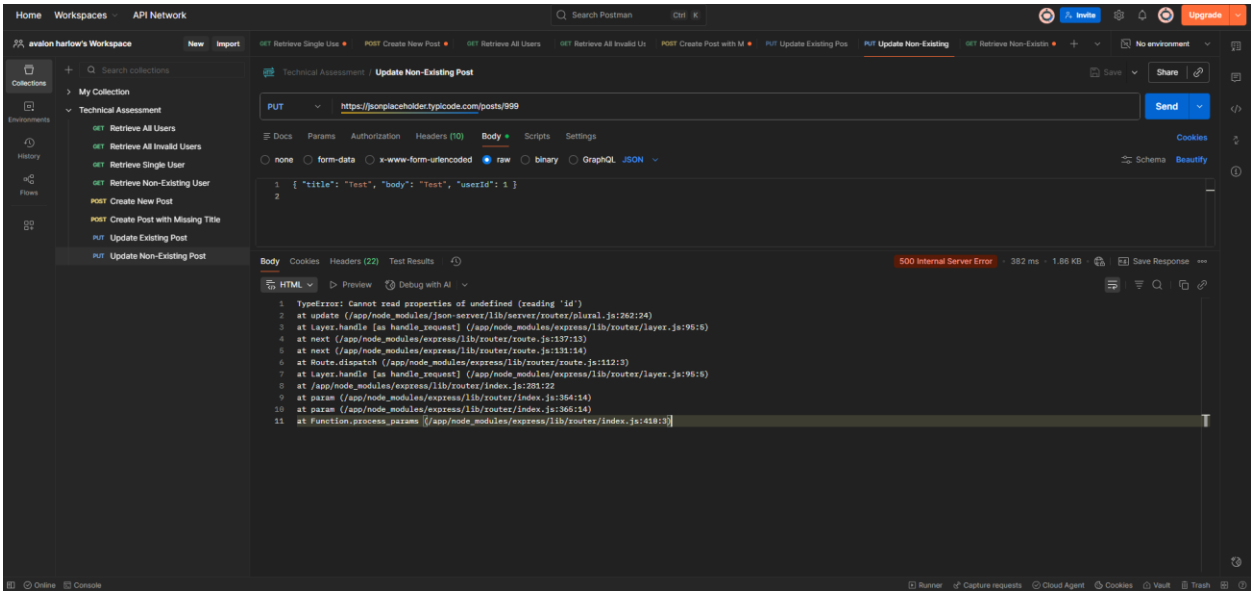


-PUT Update Existing Post





-PUT Update Non-Existing Post

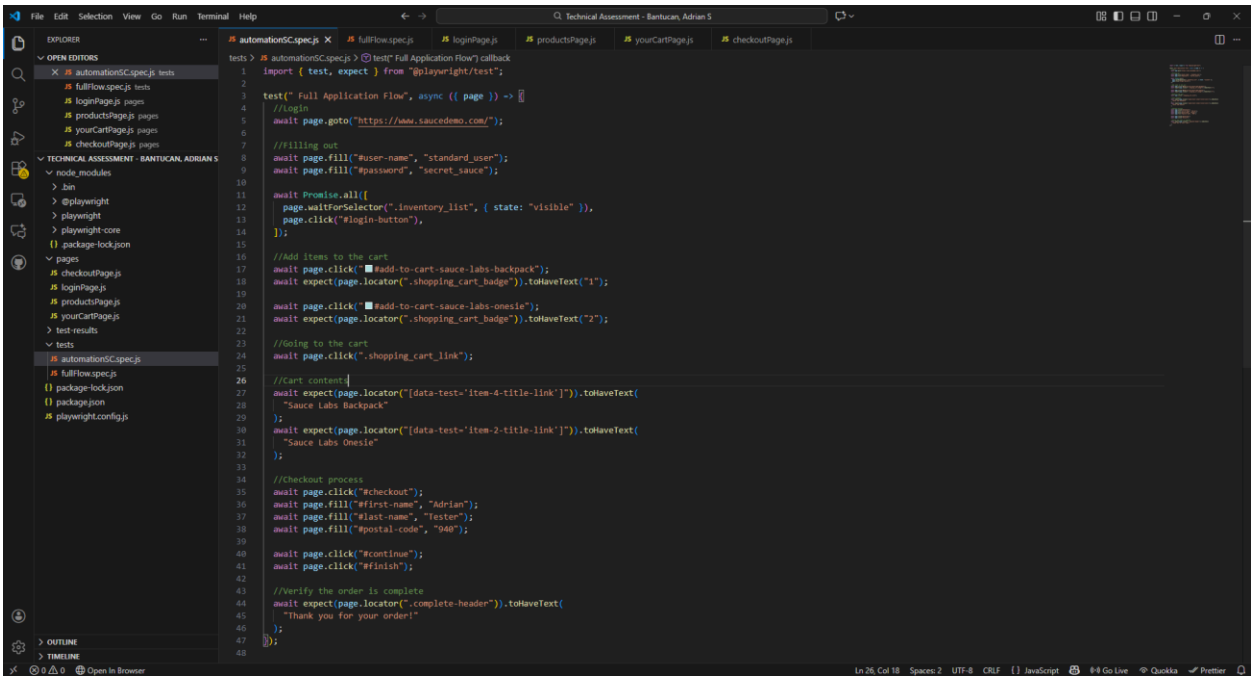


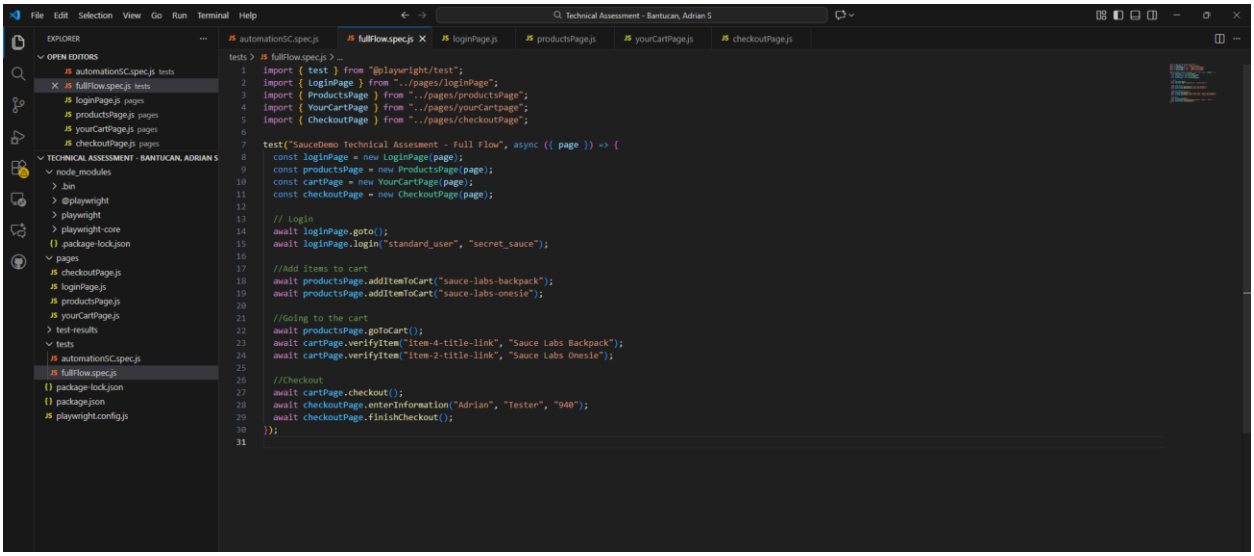
V. Automation Summary

Two approaches were used for automation. The straightforward approach consolidated all test modules into a single file (automationSC.spec.js), making it simple to run end-to-end tests quickly. The Page Object Model (POM) approach (fullFlow.spec.js) organized tests into separate pages and modules, following the structure I practiced during my internship. This approach improves maintainability, readability, reduces code duplication, and allows easier updates when the application changes.

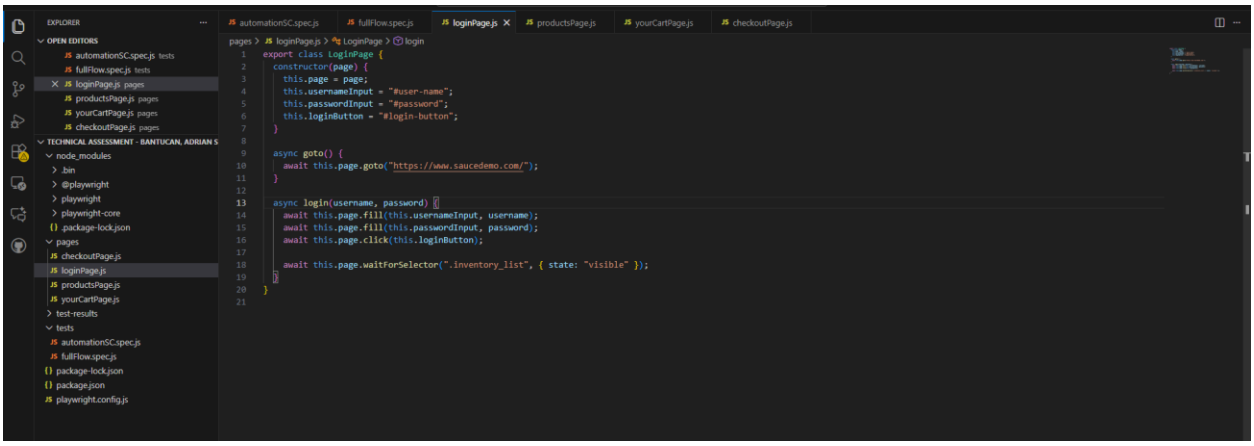
The automation scripts cover login, product selection, and checkout workflows. To run the tests, use the following commands from the project root:

Screenshots:

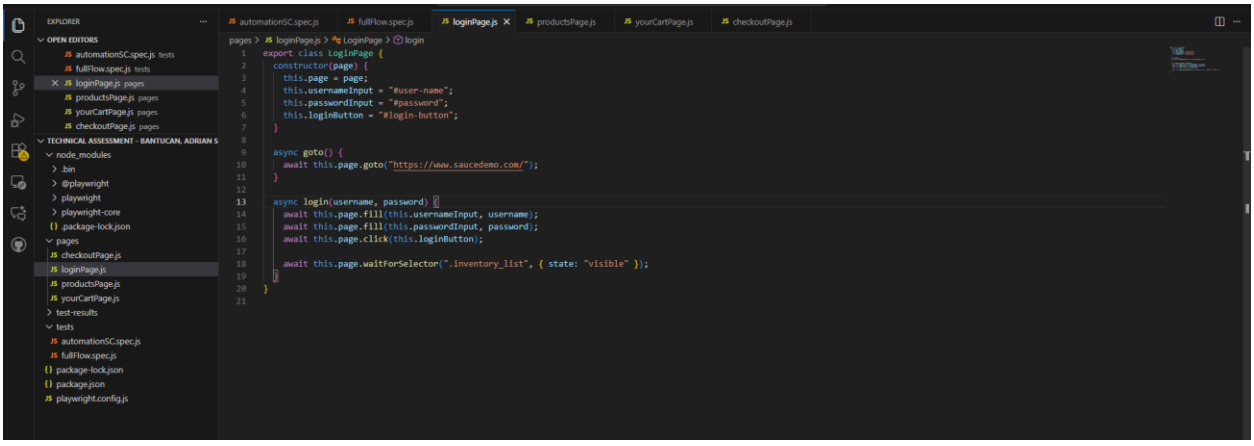




```
import { test } from "@playwright/test";
import { LoginPage } from "../pages/loginPage";
import { ProductsPage } from "../pages/productsPage";
import { YourCartPage } from "../pages/yourCartpage";
import { CheckoutPage } from "../pages/checkoutPage";

test("SauceDemo Technical Assesment - Full Flow", async ({ page }) => {
  const loginPage = new LoginPage(page);
  const productsPage = new ProductsPage(page);
  const cartPage = new YourCartPage(page);
  const checkoutPage = new CheckoutPage(page);

  // Login
  await loginPage.goto();
  await loginPage.login("standard_user", "secret_sauce");

  //Add items to cart
  await productsPage.addItemToCart("sauce-labs-backpack");
  await productsPage.addItemToCart("sauce-labs-onesie");

  //Going to the cart
  await productsPage.goToCart();
  await cartPage.verifyItem("item-4-title-link", "Sauce Labs Backpack");
  await cartPage.verifyItem("item-2-title-link", "Sauce Labs Onesie");

  //Checkout
  await cartPage.checkout();
  await checkoutPage.enterInformation("Adrian", "Tester", "940");
  await checkoutPage.finishCheckout();
});
```

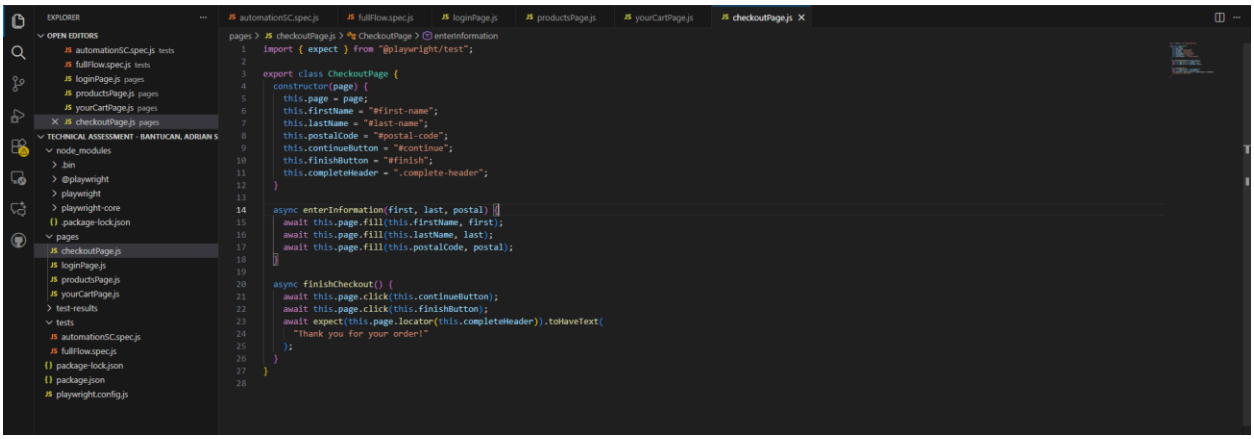


```
export class LoginPage {
  constructor(page) {
    this.page = page;
    this.usernameInput = "#user-name";
    this.passwordInput = "#password";
    this.loginButton = "#login-button";
  }

  async goto() {
    await this.page.goto("https://www.saucedemo.com/");
  }

  async login(username, password) {
    await this.page.fill(this.usernameInput, username);
    await this.page.fill(this.passwordInput, password);
    await this.page.click(this.loginButton);

    await this.page.waitForSelector(".inventory_list", { state: "visible" });
  }
}
```

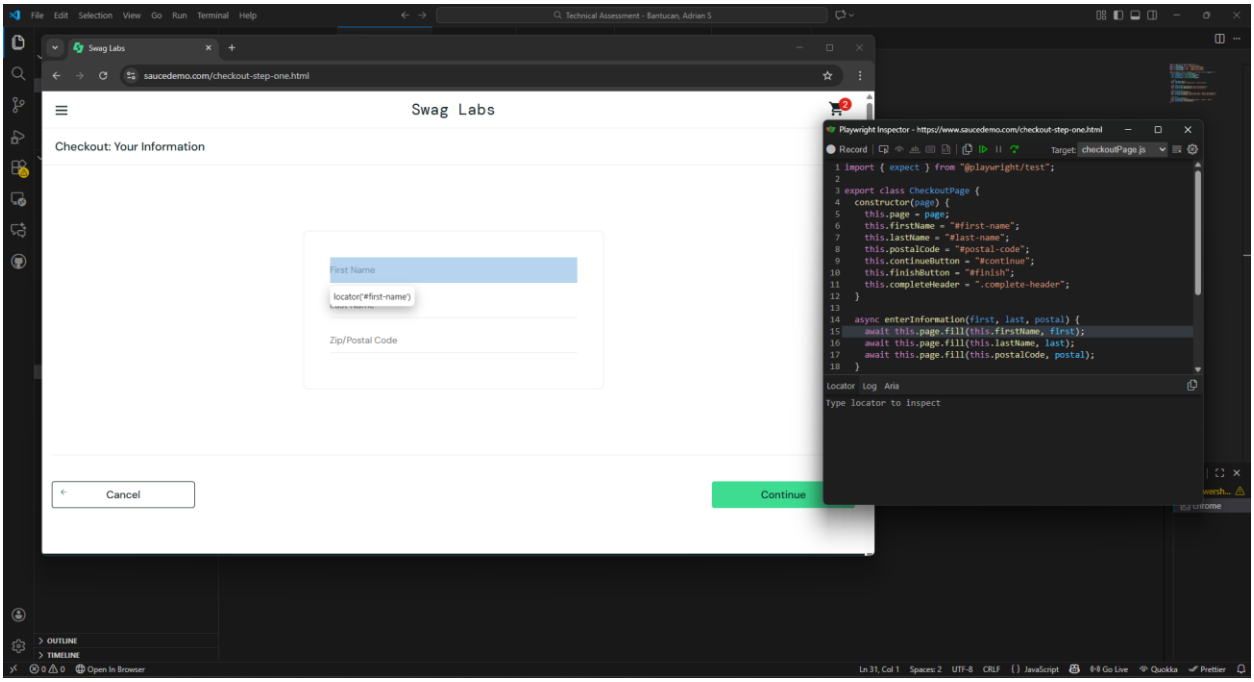


```
export class LoginPage {
  constructor(page) {
    this.page = page;
    this.usernameInput = "#user-name";
    this.passwordInput = "#password";
    this.loginButton = "#login-button";
  }

  async goto() {
    await this.page.goto("https://www.saucedemo.com/");
  }

  async login(username, password) {
    await this.page.fill(this.usernameInput, username);
    await this.page.fill(this.passwordInput, password);
    await this.page.click(this.loginButton);

    await this.page.waitForSelector(".inventory_list", { state: "visible" });
  }
}
```

```
import { expect } from "@playwright/test";

export class YourCartPage {
  constructor(page) {
    this.page = page;
  }

  async verifyItem(itemDataTest, expectedName) {
    await expect(this.page.locator(`[data-test='${itemDataTest}']`)).toHaveText(
      expectedName
    );
  }

  async checkout() {
    await this.page.click("#checkout");
  }
}
```



```
import { expect } from "@playwright/test";

export class CheckoutPage {
  constructor(page) {
    this.page = page;
    this.firstName = "#first-name";
    this.lastName = "#last-name";
    this.postalCode = "#postal-code";
    this.continueButton = "#continue";
    this.finishButton = "#finish";
    this.completeHeader = ".complete-header";
  }

  async enterInformation(first, last, postal) {
    await this.page.fill(this.firstName, first);
    await this.page.fill(this.lastName, last);
    await this.page.fill(this.postalCode, postal);
  }

  async finishCheckout() {
    await this.page.click(this.continueButton);
    await this.page.click(this.finishButton);
    await expect(this.page.locator(this.completeHeader)).toHaveText(
      "Thank you for your order!"
    );
  }
}
```




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\A\Documents\GitHub\Technical Assessment - Bantucan, Adrian S> npx playwright test tests/fullFlow.spec.js --debug

Running 1 test using 1 worker

  ✓  1 tests\fullFlow.spec.js:7:5 › SauceDemo Technical Assesment - Full Flow (37.4s)

  1 passed (41.3s)
PS C:\Users\A\Documents\GitHub\Technical Assessment - Bantucan, Adrian S>
```

VI. Bug Summary

Several bugs were identified during testing. Major bugs included checkout bypass through URL manipulation, completing checkout with an empty cart, session inconsistencies across browser tabs, and the ability to resubmit completed orders. Minor bugs primarily involved UI issues, such as oversized clickable areas. Each bug was documented with steps to reproduce, expected behavior, actual results, and suggested improvements. The severity and priority levels provide guidance for development teams to address critical issues first.

VII. Conclusion

Overall, the QA assessment confirmed that the application's key functionalities, including login, product selection, and checkout, are mostly functioning as intended. The web and API tests validated standard workflows and identified areas for improvement. Automation scripts successfully replicated manual tests, offering a reliable framework for future regression testing. The documented bugs highlight opportunities to enhance both functionality and user experience. This testing effort provides confidence in the application's stability and delivers actionable insights for the development team.

Note: If the video proof in the bug report is not working, please contact me.