ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Homework 5 - Automation testing

Course: Testing

Student:

Nguyễn Phúc An - 22127002

Lecturer

Dr. Lâm Quang Vũ

Dr. Hồ Tuấn Thanh

Dr Trương Phước Lộc

Sunday, 3th August, 2025

Member and task:

Here are the tasks handled by each team member

1.

MSSV	Thành viên	Screen/Features
22127002	Nguyễn Phúc An	Edit product (Admin) Edit user (Admin)
22127211	Phạm Đình Khôi	Admin favourite products Log in
22127164	Trần Nhật Huy	Sign up Order
22127406	Nguyễn Quốc Thuận	Contact Add new product type

1. Testing process explanation

- The automation testing was taken by **Katalon studio** - an application that enables data-driven testing.
- Data-driven testing is the branch of automation testing, a technique that allows a single test script to run with multiple data sets from sources like Excel or CSV files. This process returns a Pass/Fail output for each test case.
- Here are the process of applying data-driven testing for Edit product (Admin) feature on Katalon Studio.

1.1 Data preparation

- Following the constraints from HW2 - Domain testing, a 5-row dataset was created. Each row includes both **valid** and **invalid** values, adhering to the **Equivalence Partitioning (EP)** and **Boundary Value Analysis (BVA)** principles. For simplification, all inputs are formatted as strings.

firstname	lastname	address	postalcode	city	state	country	phone	email	dateofbirth
John	Doe	123 Main St	10001	New York	NY	Viet Nam	5551234	john.doe@example.com	01/02/2005
Jane	Smith	456 Oak Ave	90210	Los Angeles	CA	Italy	5555678	jane.smith@example.com	02/03/2005
Peter	Jones	789 Pine Ln	60601	Chicago	IL	China	555-90anc	peter.jones@example.com	02/03/2030
Mary	Brown	321 Elm Rd	75001	Dallas	TX	Uruguay	5553456	mary.brown@example.com	01/03/1999
David	Green	654 Cedar B	30303193012983200	Atlanta	GA	Canada	555-7890	david.green@example.com	01/05/2029

Dataset for Feature 1 - Edit user (Admin)

product name	description	stock	price
Milk	Milk for everyone	3.5	5
Iphone	This is a mobile	4	-5
Glass		4	12
Book	Hello, Harry potter	2	12
Gloves	Gloves for protecting	12	2
	This is a product with no name	-4	3.5

Dataset for Feature 2 - Edit product (Admin)

- Once the dataset completes, we will upload it on the **Data files section** on the **Tests Explorer** options - on the left side of the studio.

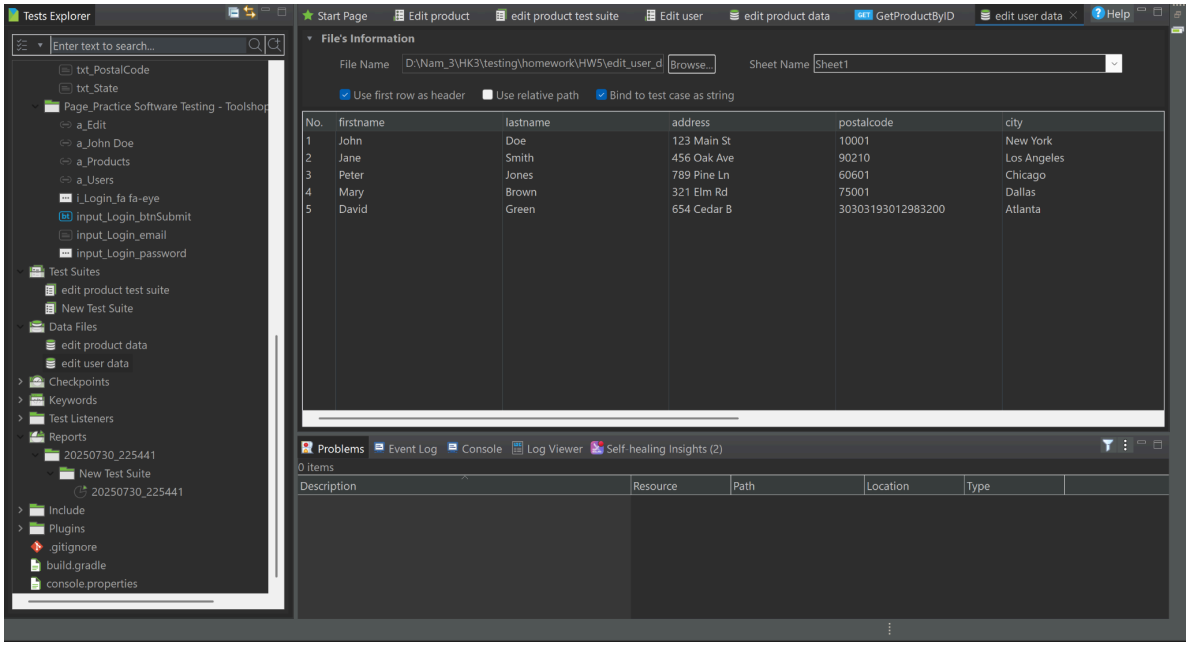
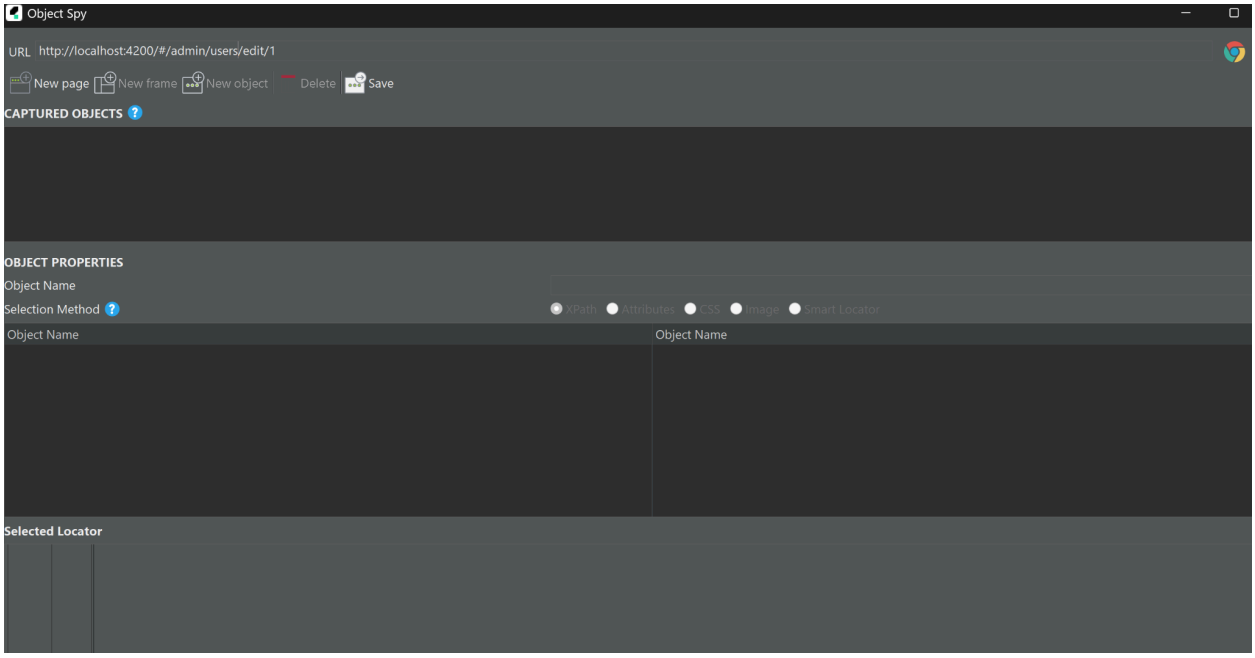


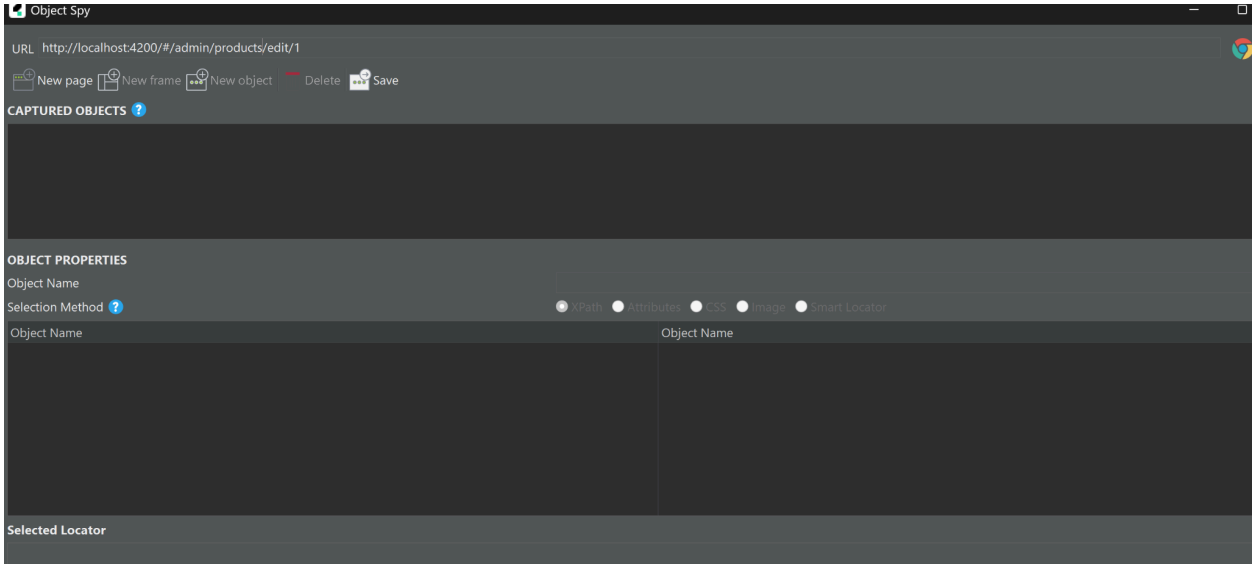
Fig2: Uploading dataset on Katalon Studio

1.2 User Interface (UI) binding

- To ensure the automation script can automatically populate the input fields, we need to bind the UI elements to our dataset. This is done by using the **Spy Web** or **Record Web** tools to capture the necessary input fields.



Object Spy for Edit users



Object Spy for Edit Products

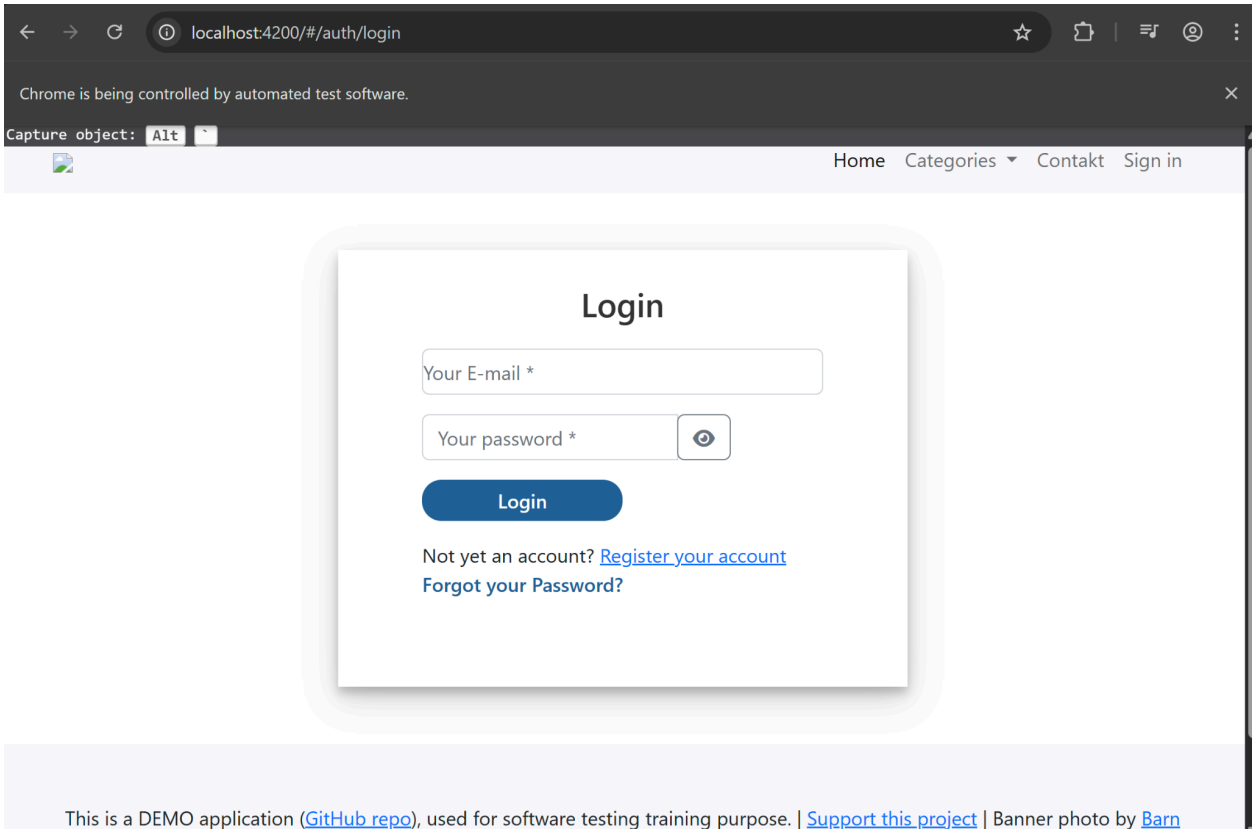


Fig3: Viewscreen of testing page

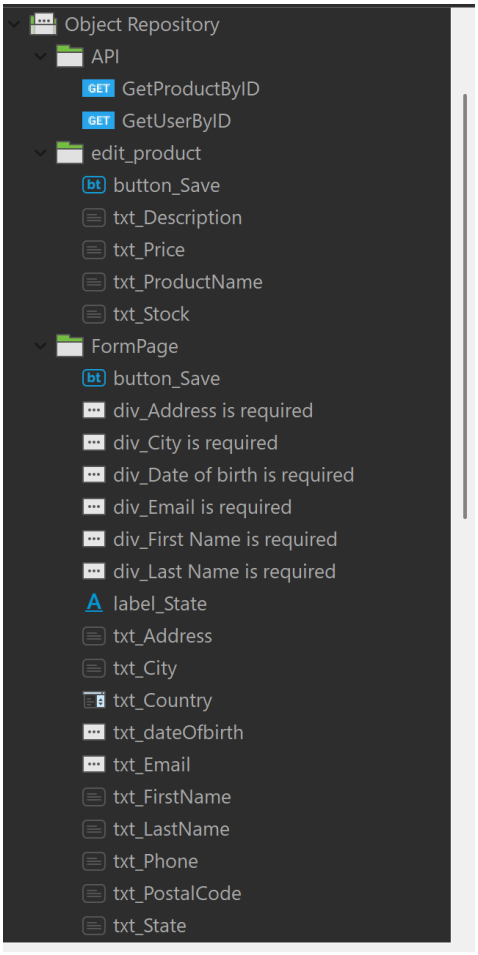


Fig4: Elements captured by Spy web or Record web

1.3 Test Case Implementation

With the UI elements and data file prepared, the next crucial step is to create the main test case that will execute the test logic. The script, written in a programming language (e.g., Groovy/Java) within Katalon Studio, encompasses the following key actions:

- **UI Interaction:** Using the captured UI objects, the script navigates to the test page, logs in as an administrator, and accesses the target user form. It then utilizes the data from the current row of the dataset to populate each input field (e.g., username, date of birth, email).
- **API Verification:** After submitting the form, the script calls a backend API (specifically `GET /users/{userID}`) to retrieve the updated user data. If the user data retrieved from the API matches the predefined requirements, the test case is marked as **PASS**; otherwise, it is marked as **FAIL**.
- **Data Binding:** This crucial step involves connecting the data from the dataset to the corresponding UI input fields, ensuring accurate and automatic population of all relevant fields during test execution. Moreover, variable declarations are also initialized for data binding.

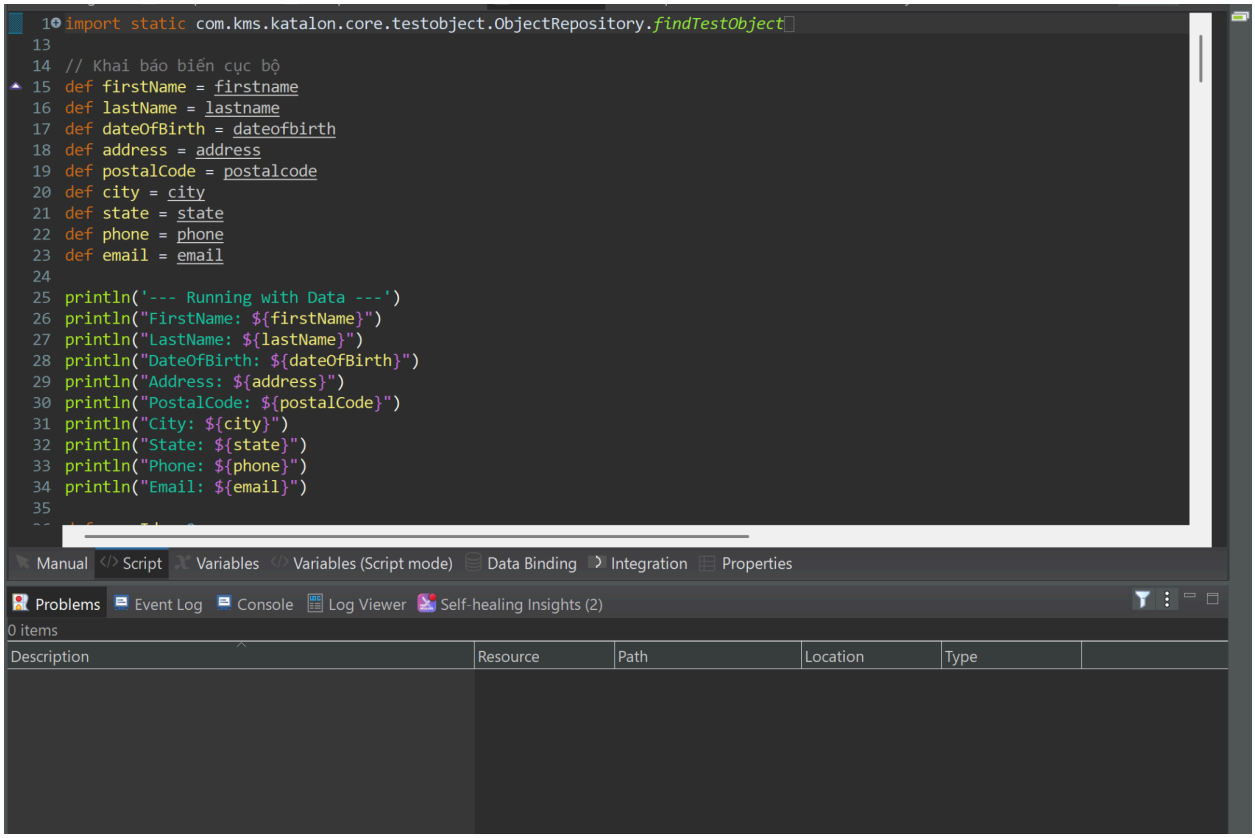
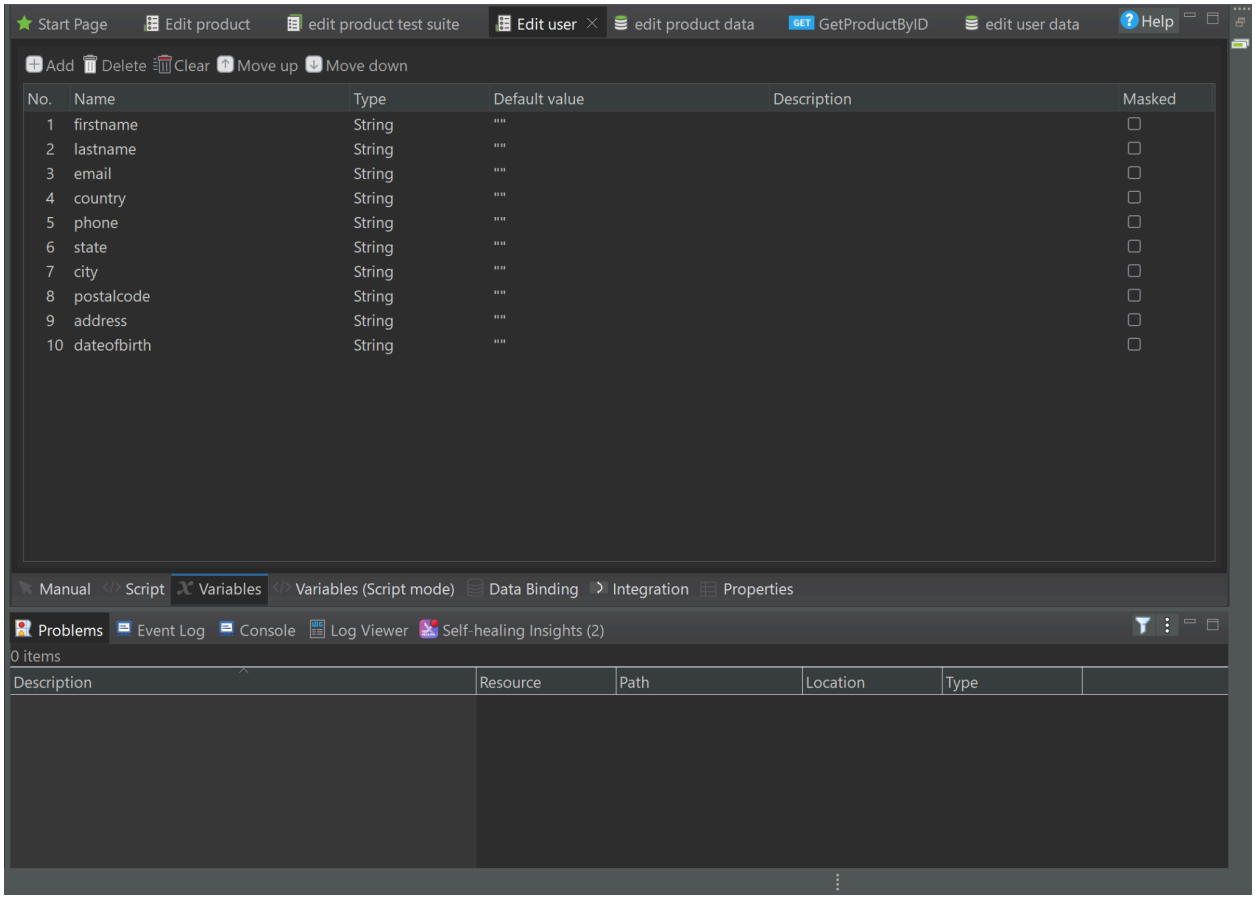
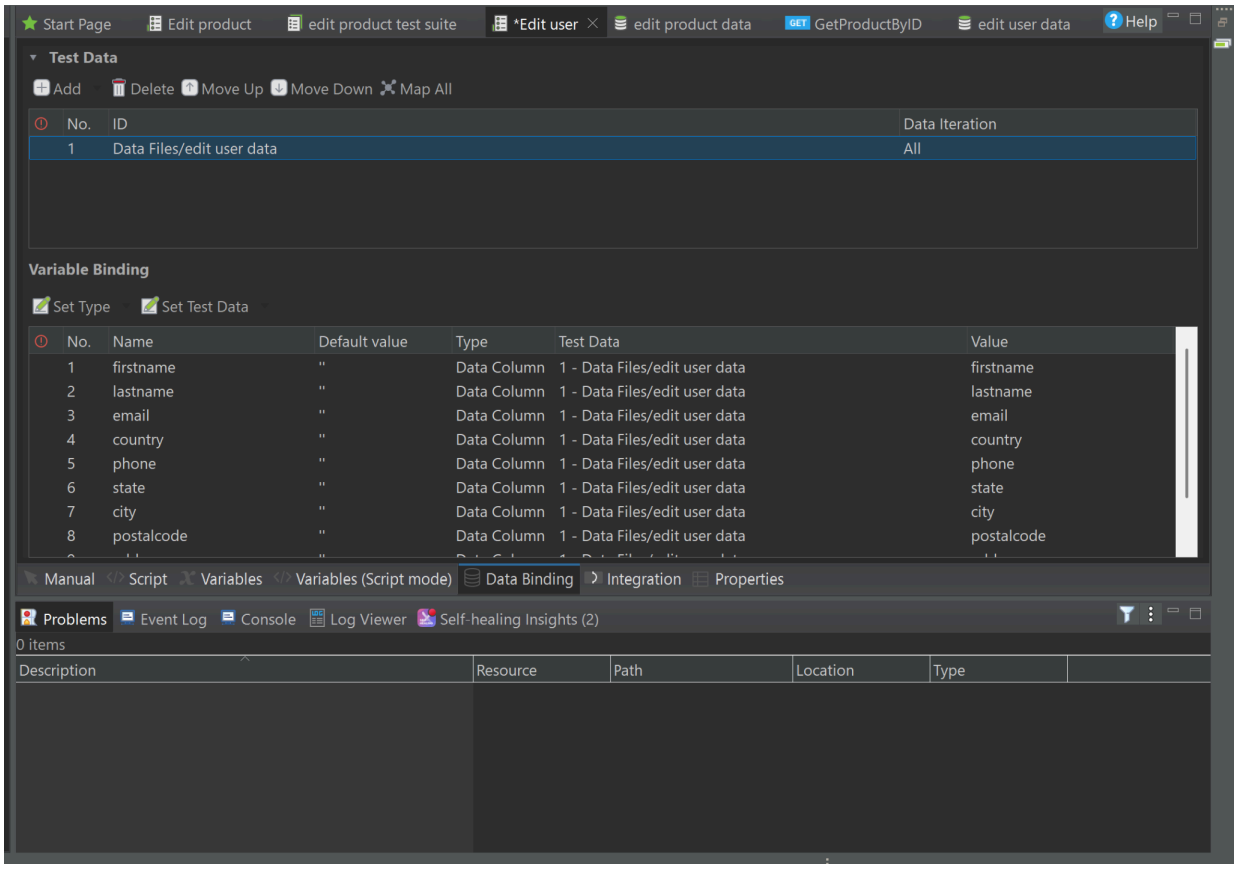


Fig5: Test script - Edit user



Variables declaration - Edit user

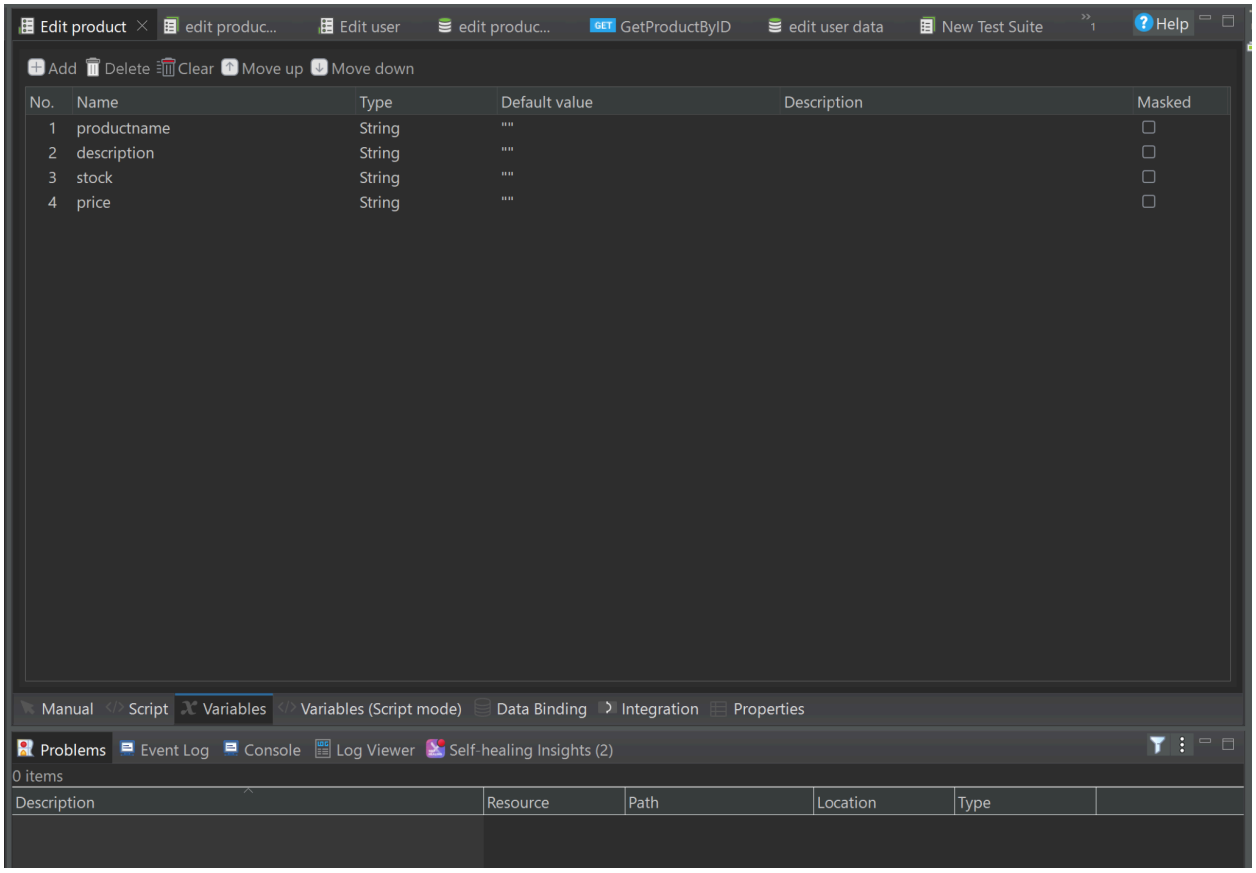


Data binding - Edit user

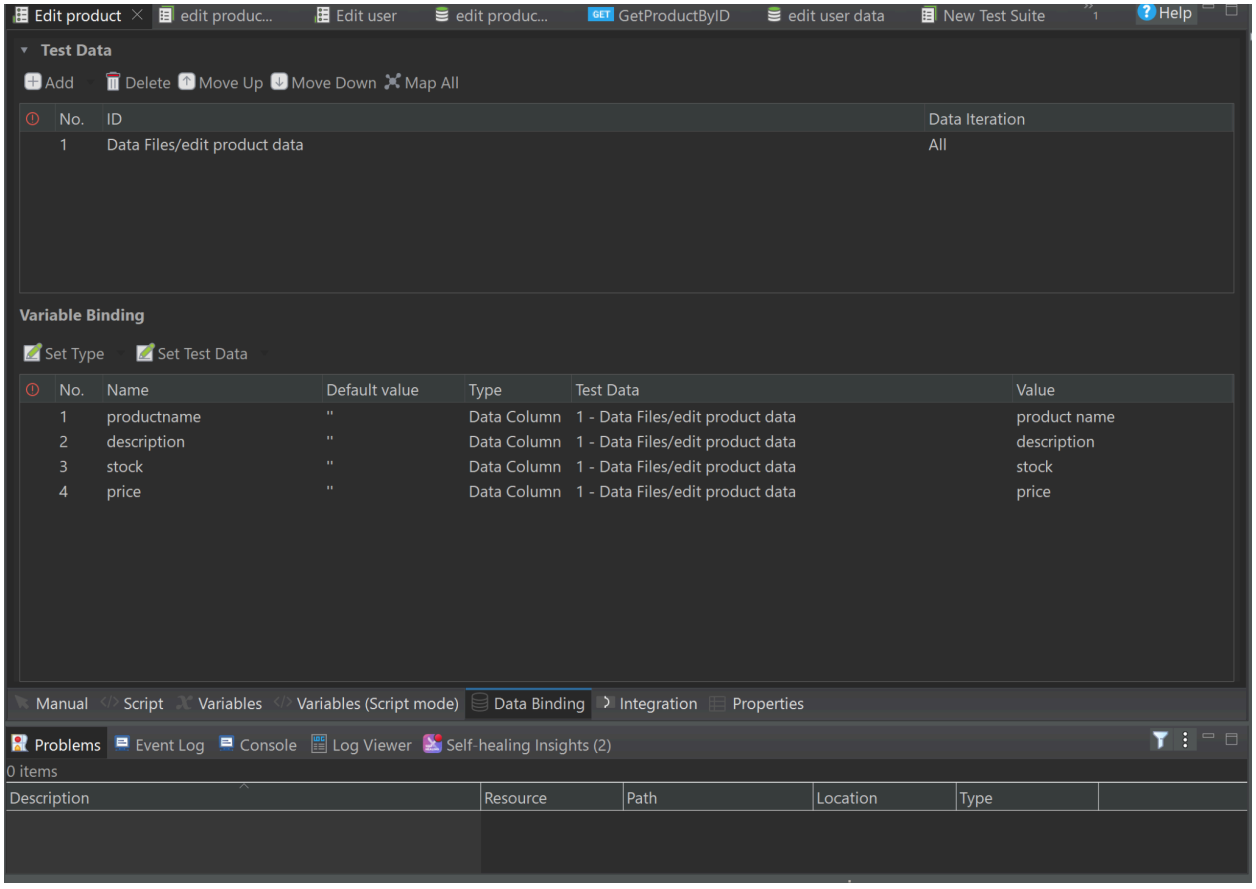
With **Edit products, we will follow the same process with edit users testcase:**

```groovy
import static com.kms.katalon.core.testobject.ObjectRepository.findTestObject

// === 1. KHAI BÁO VÀ CHUẨN BỊ DỮ LIỆU ===
def productName = productname
def description = description
def stock = stock as String
def price = price as String

println('--- Running with Data ---')
println("ProductName: ${productName}")
println("Description: ${description}")
println("Stock: ${stock}")
println("Price: ${price}")

def productId = 1

// Lấy dữ liệu gốc từ API trước khi chỉnh sửa
def originalData = getOriginalProductData(productId)
println("Đã lấy dữ liệu gốc từ API thành công: " + originalData)

// === 2. TƯƠNG TÁC VỚI UI ĐỂ GỬI FORM ===
try {
    WebUI.openBrowser('')
    WebUI.navigateToUrl('http://localhost:4200/#/auth/login')
    WebUI.waitForPageLoad(10)

    WebUI.setText(findTestObject('Object Repository/Page_Practice Software Testing - Toolshop -_8413b3/input_Login_email
    WebUI.setEncryptedText(findTestObject('Object Repository/Page_Practice Software Testing - Toolshop -_8413b3/input_Log
    WebUI.click(findTestObject('Object Repository/Page_Practice Software Testing - Toolshop -_8413b3/input_Login_btnSubm
    WebUI.waitForPageLoad(10)
```

Test script - Edit product



Variable declaration - Edit product



Data binding - Edit product

- **UI Interaction:** Using the captured UI objects, the script navigates to the test page, logs in as an administrator, and accesses the target user form. It then utilizes the data from the current row of the dataset to populate each input field (e.g.,product name, stock,...).
- **API Verification:** After submitting the form, the script calls a backend API (specifically `GET /product/{productID}`) to retrieve the updated product data. If the product data retrieved from the API matches the predefined requirements, the test case is marked as **PASS**; otherwise, it is marked as **FAIL**.
- **Data Binding:** This crucial step involves connecting the data from the dataset to the corresponding UI input fields, ensuring accurate and automatic population of all relevant fields during test execution. Moreover, variable declarations are also initialized for data binding.

1.4 Verification

Based on the comparison results, the test case's final outcome is determined. A GET method will be called with the users to fetch the latest data saved recently.

- **PASS:** If the actual value from the API matches the expected value for all fields, the test case is marked as **PASS**.
- **FAIL:** If even one field fails to match the expected value, the test case is marked as **FAIL**. A detailed log is recorded, specifying which fields had mismatched data, the expected value, and the actual value retrieved from the API. This granular reporting is essential for quickly identifying the root cause of the failure.

2. Implementation fo data-driven techniques

This code snippet effectively demonstrates data-driven testing by separating test logic from test data.

- **Dynamic Variable Utilization:** Variables like firstName, lastName, email, etc., are declared at the beginning of the script. These variables are dynamically populated with values from an external dataset (linked via a Katalon Test Suite). This allows the same script to execute multiple times with different user data without modification.
- **Dynamic UI and API Interaction:**

  UI commands such as WebUI.setText() use these variables to dynamically fill form fields (e.g., WebUI.setText(..., firstName)).

  The API endpoint (/users/${userId}) is also dynamic, utilizing the userId variable to fetch data for the specific user being tested in the current iteration.

Data Transformation: The handling of dateOfBirth (parsing from API's yyyy-MM-dd to UI's M/d/yyyy) showcases a common data-driven need to transform data formats to ensure compatibility between different system layers.

3. Verification

The script employs robust checkpoints and assertions to validate the expected behavior of the application.

- API Status Code Verification: WS.verifyResponseStatusCode(apiResponse, 200) is a fundamental assertion ensuring that the API call itself was successful.
- Detailed Field-Level Data Verification: This is the core of the validation.
    - The script iterates through each user field (e.g., firstName, email) retrieved from the post-submission API response.
    - Intelligent Expected Value Logic: A key assertion is the calculation of expectedValue. If the input formValue was isValid (based on helper validation functions), the expectedValue is the new formValue. If the input was invalid, the expectedValue reverts to the originalValue (fetched before UI interaction). This verifies the application's business logic for both valid updates and invalid input handling.
    - Consistent Comparison: Helper functions like compareValues ensure accurate comparisons across different data types (e.g., numbers, strings, dates).
- Granular Failure Reporting: If a mismatch is detected, the newFailedFields list records detailed information (field name, expected value, actual value).
- Overall Test Case Status: Finally, KeywordUtil.markFailedAndStop() is used to explicitly mark the test case as FAIL if any assertion fails, providing a clear and actionable test result.

4. Test results

Feature 1: Edit user

Test case 1	Pass	All inputs valid
Test case 2	Pass	All inputs valid
Test case 3	Fail	Date of birth in the future, phone has non-digit characters
Test case 4	Pass	All inputs valid
Test case 5	Fail	Postal code >10, dob in the future
Test case 6	Fail	Empty form

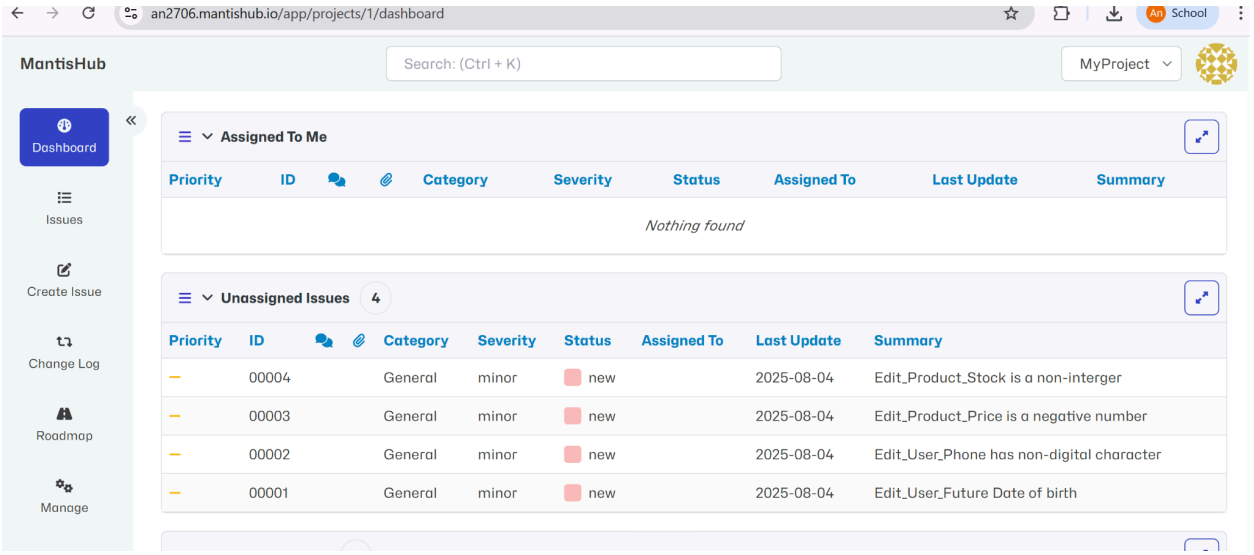
Feature 2: Edit product

Test case 1	Fail	Stock = 3.5
Test case 2	Fail	Price <0
Test case 3	Fail	Empty description
Test case 4	Pass	All inputs valid
Test case 5	Pass	All inputs valid
Test case 6	Fail	Empty product name

5. Youtube link

Edit user: https://youtu.be/t46KBooIXTo
Edit product: https://youtu.be/3g90NLcZIYs

6. Bug Report



7. Evaluation

No.	Criteria	Grade	Self-Assessed Grade
1	**Name of Feature 1 (missing any of the following: "report", "script" or "video" results in 0 points)**	**50**	**48**
	1.1 Report + Bug	15	15
	1.2 Test cases	5	5
	1.3 Script files	10	10
	1.4 Data files	10	10
	1.5 Videos	10	8
2	**Name of Feature 2 (missing any of the following: "report", "script" or "video" results in 0 points)**	**30**	**28**
	2.1 Report + Bug	15	15
	2.2 Test cases	5	5
	2.3 Script files	10	10
	2.4 Data files	10	10
	2.5 Videos	10	8
	Total	**100**	**96**