ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

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

Homework 7 - API Testing

Course: Testing

Student:

Nguyễn Phúc An - 22127002

Lecturer

Dr. Lâm Quang Vũ

Dr. Hồ Tuấn Thanh

Dr Trương Phước Lộc

Saturday, 23th August, 2025

ID	Name	Scenario
22127002	**Nguyễn Phúc An**	**/products/search? (GET)** **/products/{productId} (PUT)** **/products?page (GET)**
22127211	**Phạm Đình Khôi**	**/categories (GET)** **/categories (POST)** **/product/{productId} (GET)**
22127164	**Trần Nhật Huy**	**/users (GET)** **/users/{userId} (GET)** **/users/login (POST)**
22127406	**Nguyễn Quốc Thuần**	**/brands (GET)** **/brands (POST)** **/brands/{brandId} (GET)**

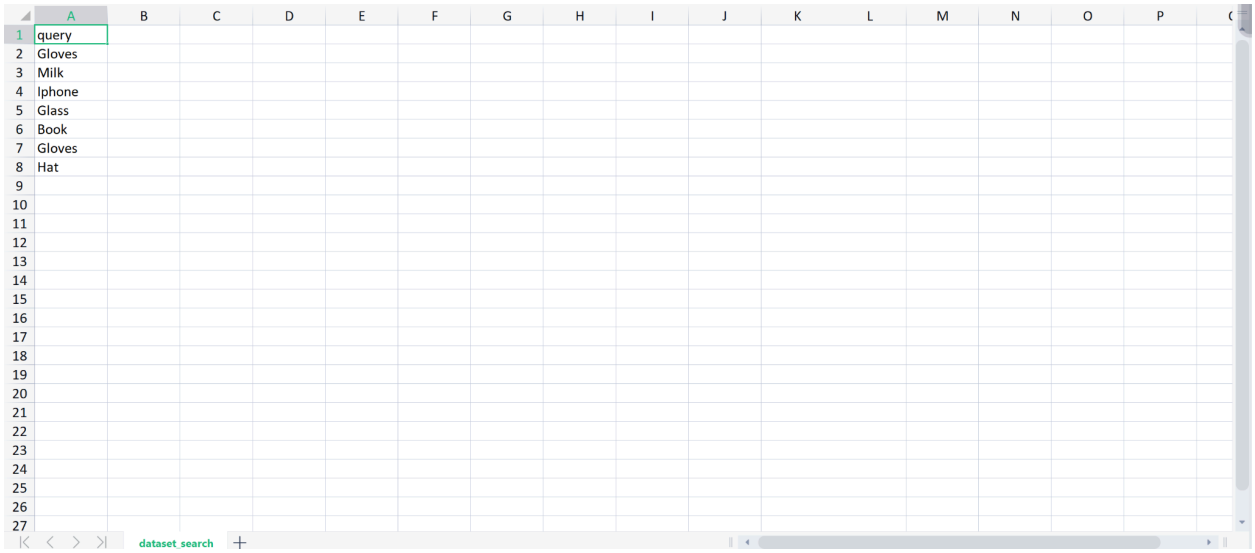
1. Tool Use

- This homework uses **Postman** with **Newman** to test the functional correctness of the Toolshop application (Search products, Update products, and List products). The tests are integrated into a **GitHub Actions CI/CD workflow**, which automatically runs on every push or pull request to Github Repository.
- We will use **Docker** to run the application locally via command line: **docker-compose up -d** or GUI desktop.
- On the other hand, **data-driven techniques** will be applied on four different types of performance testing: **stress test, spike test, load test and volume test.**

2. Step-by-step instructions

2.1 CSV datasets for data-driven testing

- CSV datasets will be provided for each selected API for this homework, including:
  + Search Products by Name.
  + Update Products.
  + View List products.
- Here are datasets prepared for each API.



Dataset for Search Products by Name

productId	name	description	stock	price	expectedStatus
1	Gloves	This is a pro	20	20	200
2	Milk	Milk for eve	3.5	5	200
3	Iphone	This is a mo	4	-5	400
4	Glass		4	12	400
5	Book	Hello, Harry	2	12	200
6	Gloves	Gloves for p	12	2	200
7	Hat	This is a pro	-4	3.5	400

Dataset for Updating Product

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	page															
2	1															
3	2															
4	3															
5	4															
6	5															
7	6															
8	7															
9																
10																
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																
21																
22																
23																
24																
25																
26																
27																

page_data

Dataset for Displaying List Products

2.2 Write a script file

To enable **Postman** to behave as expected during automated API testing, two configuration files need to be prepared:

2.2.1 Postman_collection.json

This file defines the **three APIs** used in our testing workflow:

- **Products – Search (GET)**
  - Executes `GET /products/search?q={{query}}` with data-driven queries from `dataset_search.csv`.
  - Test scripts validate status code 200, response body JSON structure, non-empty items, keyword matching in product names, and response time under 2 seconds.
- **Products – Update (PUT)**
  - Executes `PUT /products/{{productId}}` with data-driven inputs from `dataset_put.csv`.
  - Includes custom validation logic:
    - If request data is **valid** (non-negative price/stock, non-empty name/description) → must return **200** and response body fields must match the request.
    - If request data is **invalid** → must **not** return 200 (e.g., expected 400 or 422).
- **Products – List (GET)**
  - Executes `GET /products?page={{page}}` with pagination data from `page_data.csv`.
  - Validates status 200, JSON body existence, non-empty items, and expected product fields (`id`, `name`, `description`, `stock`, `price`).
  - Ensures response time under 2 seconds.

All requests are embedded with **JavaScript test scripts** inside Postman to check correctness automatically.

2.2.2 Postman_environment.json

This file defines the **environment variables** for different testing contexts.

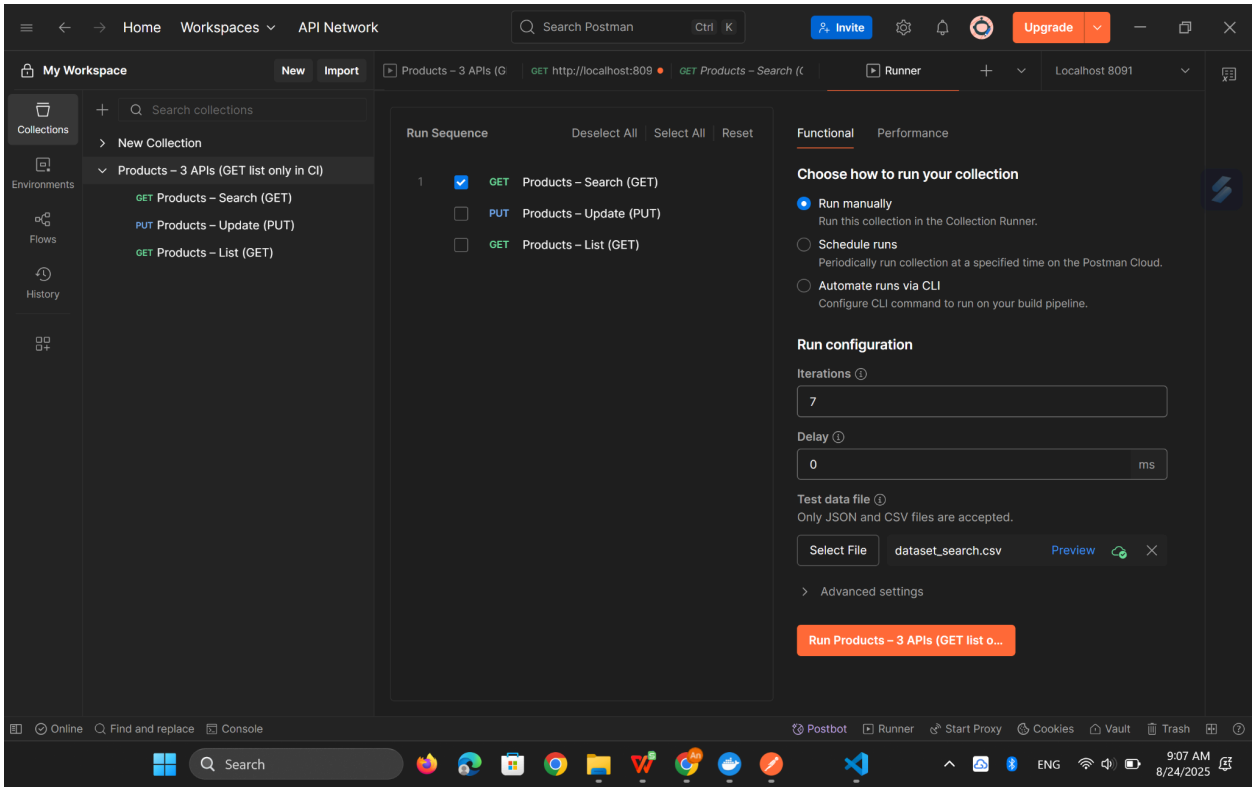
- **baseUrl**: Used as a placeholder in collection requests.
    - In CI pipelines:Using hosted domains for testing. `https://api-with-bugs.practicesoftwaretesting.com`
    - In local development: `http://localhost:8091`
- Developers can easily switch between environments without modifying the collection.

Example Workflow

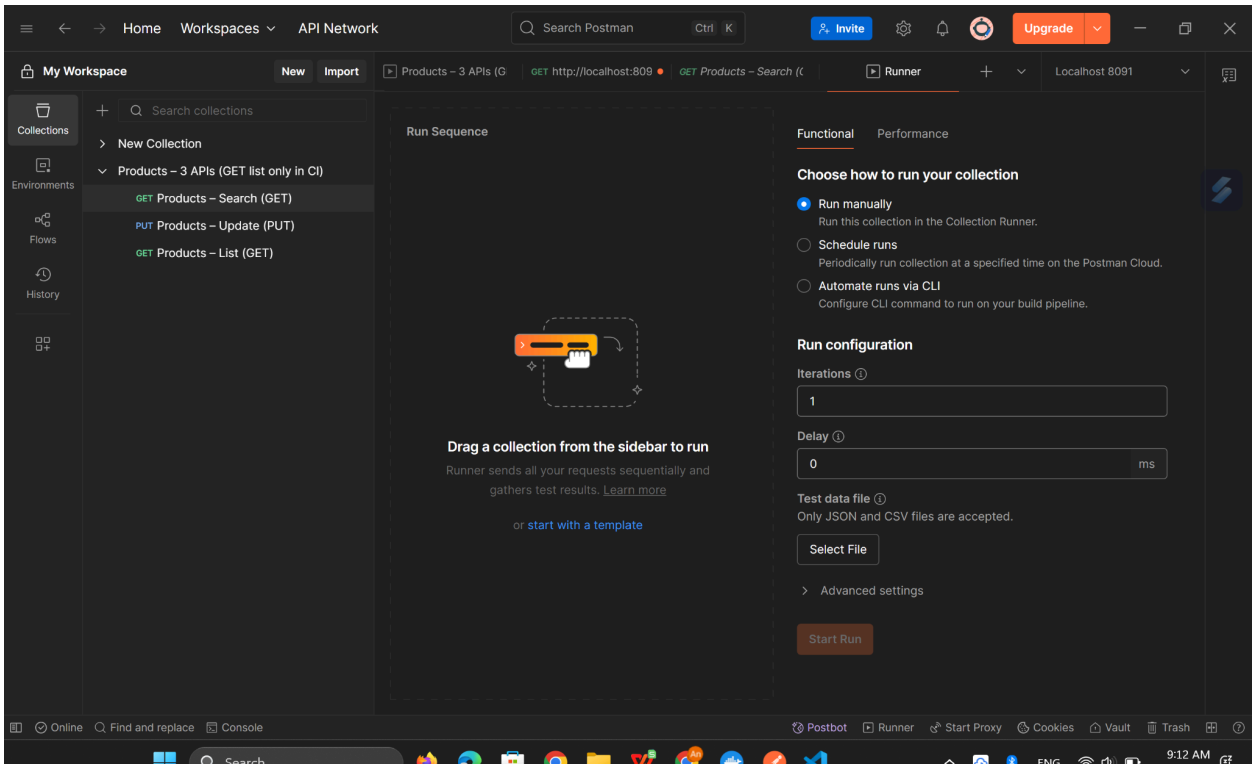
- When running locally, Postman/Newman uses `Localhost 8091` environment to point requests to the Dockerized application running on port 8091.

- When running in **CI/CD GitHub Actions**, the default environment (`api-with-bugs.practicesoftwaretesting.com`) is used, ensuring consistent behavior in automated pipelines.

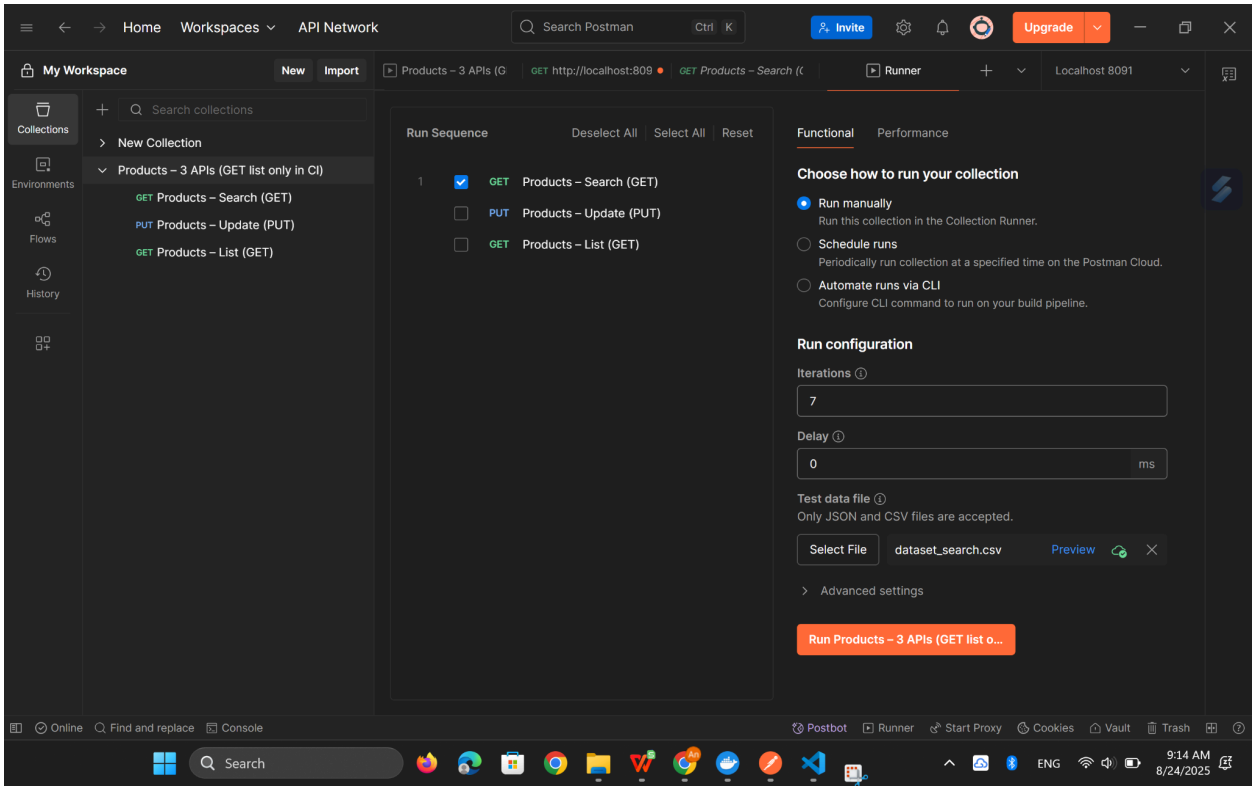
2.3 Running on Postman



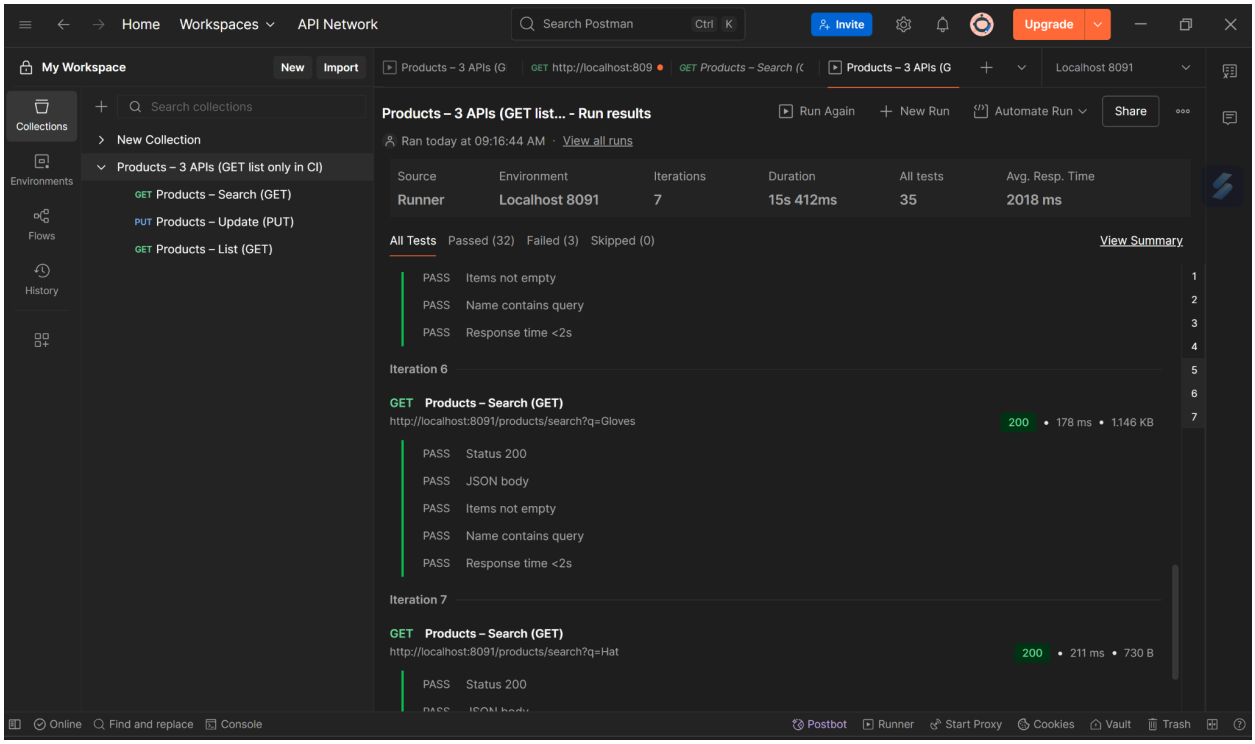
- From the **Collections** panel in the left sidebar, import the `postman_collection.json` file from the local desktop.
- The imported collection will appear in the list of available collections.



- Drag the collection imported recently to the **Run Sequence** area.
- The three APIs defined in the collection (`Products – Search`, `Products – Update`, and `Products – List`) will be displayed in the execution list.



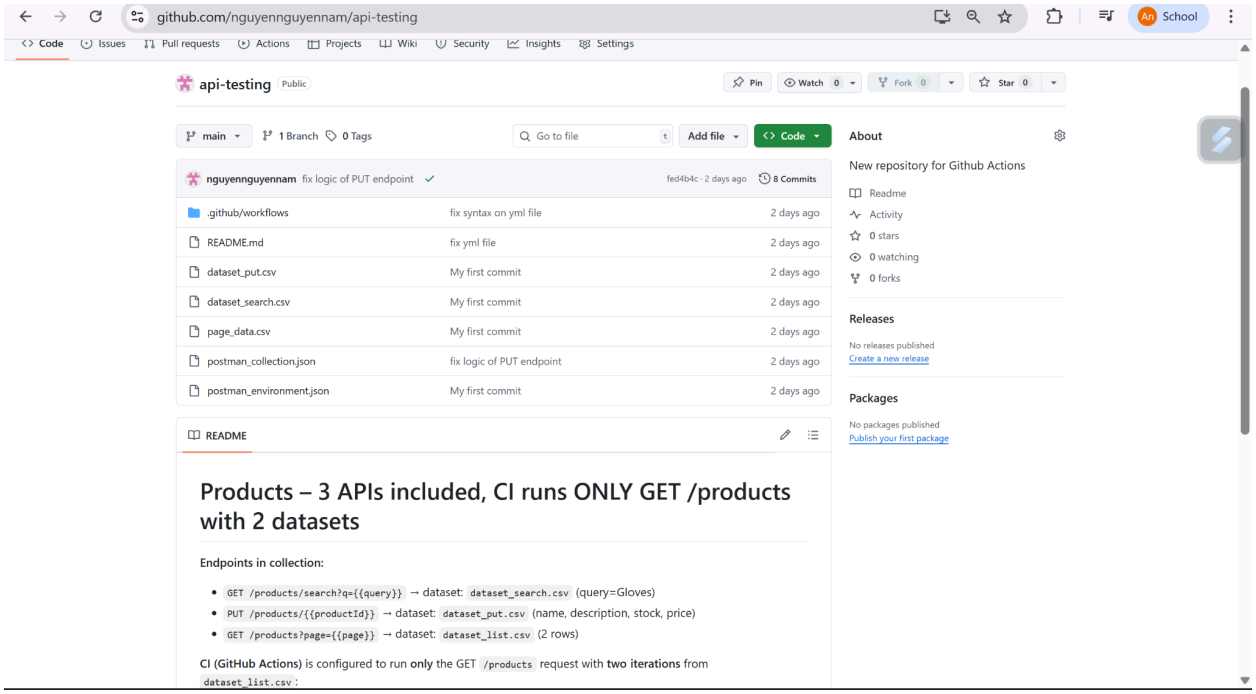
- Select the API you want to test and attach the corresponding **CSV dataset**:
  - `dataset_search.csv` for the **Search API**
  - `dataset_put.csv` for the **Update API**
  - `page_data.csv` for the **List API**
- Click the Run Products button at the bottom-right corner of the Runner window.
- Postman will automatically iterate through the dataset rows and execute the defined test scripts.



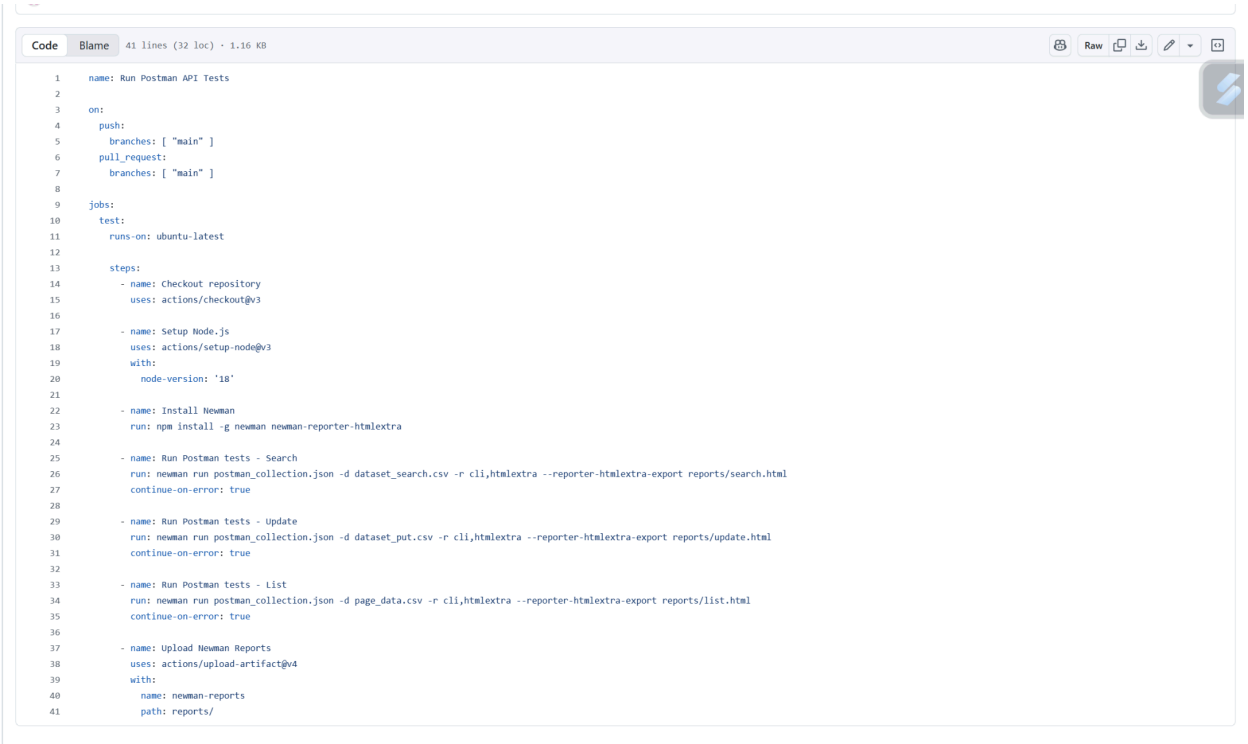
- Once execution is complete, Postman displays a detailed report with:
    - Pass/Fail results for each request and assertion.
    - Response codes and execution times.
    - Any assertion errors if the API does not return the expected response.

2.4 Integrating with CI/CD Pipelines by Github Actions

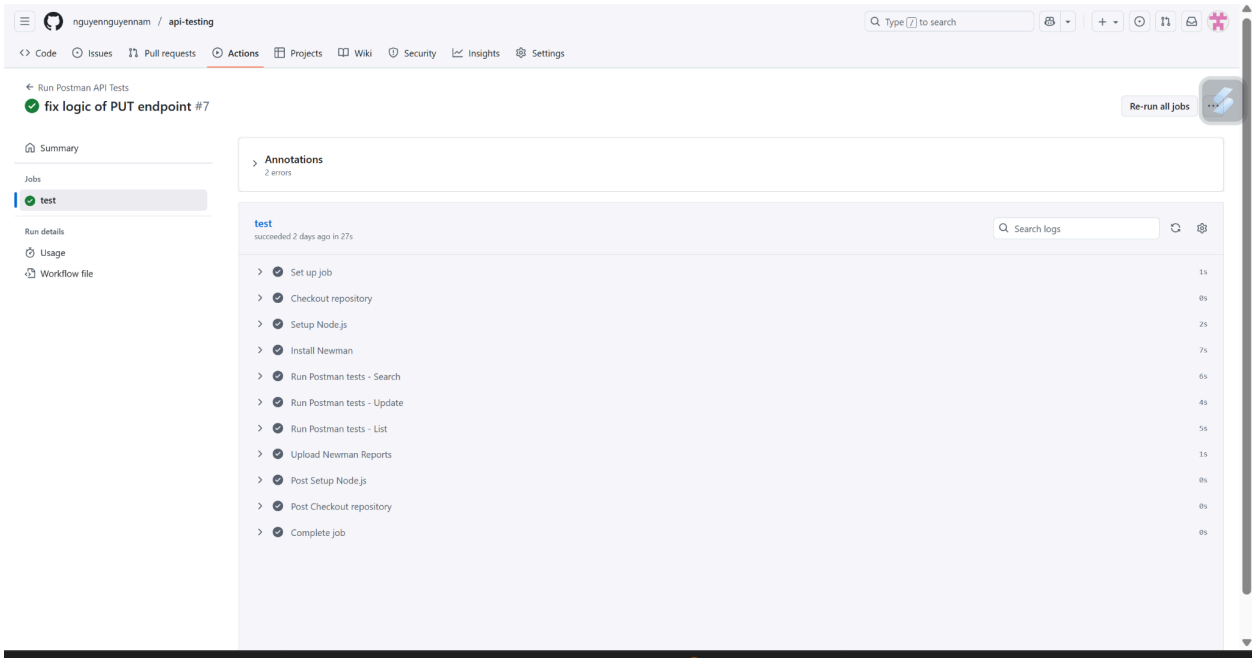
- After validating the API tests locally using Postman and Newman, the next step is to automate their execution within a **Continuous Integration / Continuous Deployment (CI/CD) pipeline**. This ensures that API correctness is continuously validated whenever new code is pushed to the repository.
- The chosen platform for automation is **GitHub Actions**, which provides a simple way to define workflows as YAML configuration files.
- A workfile named `.github/workflows/api-tests.yml`, was created in the repository.
- The workflow runs automatically on a push or pull request event. Once triggered, GitHub Actions provisions a runner, installs dependencies, executes the Postman tests using Newman, and generates HTML reports.

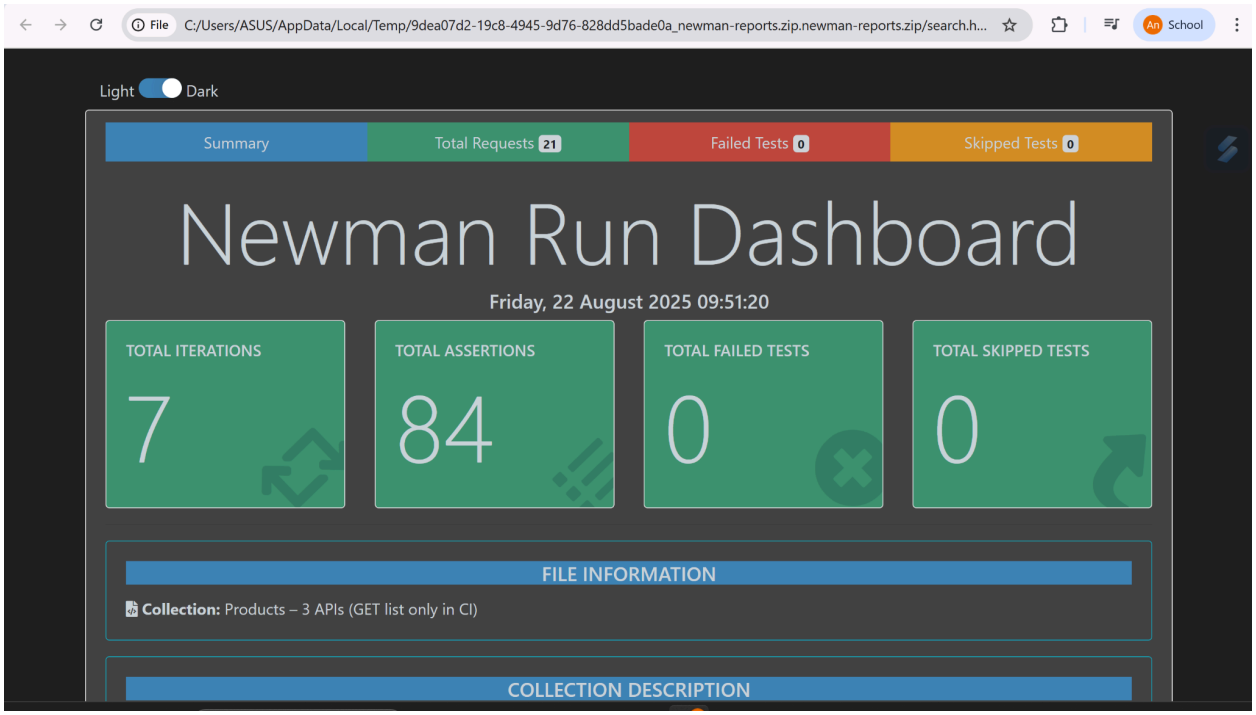


- Github repository for storing materials executing testing and integrating CI/CD pipeline.



- The structure of a yml file.





- Once a push or pull request is done, Github Action will start running workflows, and return the overall result (PASS/FAIL) along with downloadable HTML file reports.

2.5 Results analysis

Search API

- Always returned `200 OK` with valid JSON.

Update API

- Valid inputs worked correctly (`200 OK`, fields updated).
- Invalid inputs (negative price/stock, empty description) often **still returned `200` instead of `400/422`**.
- Reveals a **backend validation bug** — the server accepts invalid data.

List API

- Pagination generally worked.
- Assertions failed when expected products were missing.

3. AI Promoting

- AI models: ChatGPT Plus
- Prompt:

> Students are asked to work on appropriate API testing tools. Each student must test at least **three (3) APIs**. All test scripts must use **data driven** techniques.
>
> The **test cases** and any identified **bugs** must be reported in the tracking tools.
>
> Finally, **integrate** the designed API test cases into a CI/CD (continuous integration) workflow (e.g. Github Actions, Jenkin…).
>
> Here are the requirements for my homework about API testing. Please give me a **step-by-step plan** to complete this homework, including:
>
> 1. How to set up the Toolshop application for testing.
> 2. How to design data-driven test scripts in Postman/Newman.
> 3. How to organize and report bugs clearly.
> 4. How to write and run a GitHub Actions workflow for automated API testing.

5. Example commands, YAML config, and dataset structure.

- Results:

Perfect 👍 Thanks for clarifying. Since your homework is based on the **Toolshop application**, I'll give you a **step-by-step plan** you can follow, with practical details, examples, and ready-to-adapt snippets.

Step 1. Set up the Toolshop application for testing

1. **Clone the repository**:

```bash
git clone https://github.com/testsmith-io/practice-software-testing.git
cd practice-software-testing
```

2. **Run locally with Docker** (easiest way):

```bash
docker-compose up -d
```

   - Backend API runs at: `http://localhost:8091`
   - Web app runs at: `http://localhost:3000`

3. **Verify APIs are accessible**:
   Test endpoints in browser or Postman:

Step 2. Design Data-Driven Test Scripts in Postman/Newman

1. **Create a Postman Collection** with at least **3 APIs**:
   - `GET /products/search?q={{query}}` (search product)
   - `PUT /products/{{productId}}` (update product)
   - `GET /products?page={{page}}` (list products)
2. **Define assertions** in Postman Test tab (JavaScript):

```js
pm.test("Status is 200", () => pm.response.to.have.status(200));
let body = pm.response.json();
pm.test("Items not empty", () => pm.expect(body.length).to.be.above(0));
```

3. **Use CSV datasets** for data-driven testing:

   Example `dataset_search.csv`:

```csv
query
Gloves
Milk
Book
```

   Example `dataset_put.csv`:

4. **Run with Newman locally**:

```bash
newman run postman_collection.json -d dataset_search.csv
newman run postman_collection.json -d dataset_put.csv
```

Step 3. Organize and Report Bugs Clearly

1. **Tracking Tool**: You can use **GitHub Issues**, **Jira**, or even a spreadsheet.
2. **Bug Report Template**:
   - **ID**: BUG-001
   - **Title**: PUT /products accepts negative price
   - **Precondition**: Product exists
   - **Steps to Reproduce**:
     1. Send PUT request with `"price": -5`
   - **Expected Result**: API returns `400 Bad Request`.
   - **Actual Result**: API returns `200 OK`.
   - **Severity**: High
3. Attach **screenshots from Postman/Newman report** for evidence.

---

4. Youtube link

The Youtube link demonstrating how to apply API testing:
https://youtu.be/ZKaKxKhhWNs

5. Self-assessment

Criteria	Description	Max Points	Self Assessment

API1	Missing any of the following “report”, “test cases”, “bug report”, “data”, or “video” results in 0 points Report: 1.0 Test cases: 0.5 Bug report: 0.5 Video: 1.0	3.0	3.0
API2	Missing any of the following “report”, “test cases”, “bug report”, “data”, or “video” results in 0 points Report: 1.0 Test cases: 0.5 Bug report: 0.5 Video: 1.0	3.0	3.0
API3	Missing any of the following “report”, “test cases”, “bug report”, “data”, or “video” results in 0 points Report: 1.0 Test cases: 0.5 Bug report: 0.5 Video: 1.0	3.0	3.0
Use of AI Tools	Prompt transparency, critical validation, added value	1.0	1.0

Total		**10.0**	**10.0**