

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
KHOA CÔNG NGHỆ THÔNG TIN**



BÁO CÁO BÀI TẬP

MÔN HỌC: KIỂM THỬ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

BÀI TẬP LỚN: TÌM HIỂU CÔNG CỤ KIỂM THỬ PYRESTTEST

Giảng viên: ThS. Nguyễn Thu Trang

Sinh viên: Trần Lương Minh Đức - 22028244
Đỗ Tiến Đạt - 22028240
Nguyễn Thị Vân Anh - 22028128
Đường Minh Hoàng - 22028186

Lớp: INT3117_1

HÀ NỘI - 4/2025

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin được gửi lời cảm ơn chân thành và sâu sắc nhất tới giảng viên môn học là cô Nguyễn Thu Trang. Cô không chỉ cung cấp những kiến thức bổ ích mà còn truyền đạt những kinh nghiệm quý báu, giúp chúng em rất nhiều trong quá trình viết bài báo cáo này.

Chủ đề của bài báo cáo này là tìm hiểu về Pyresttest, một công cụ kiểm thử API đơn giản, nhẹ và dễ sử dụng dành cho các dịch vụ RESTful API. Pyresttest cho phép thực hiện các bài kiểm thử tự động với cấu hình dưới dạng file YAML, giúp giảm thiểu việc lập trình phức tạp và tăng hiệu quả trong việc kiểm thử các API. Với khả năng tích hợp vào quy trình CI/CD và hỗ trợ kiểm thử nhiều yêu cầu HTTP khác nhau, Pyresttest là một lựa chọn phù hợp cho các nhóm phát triển phần mềm hiện nay. Báo cáo này cung cấp cái nhìn tổng quan từ cơ bản đến nâng cao về Pyresttest, bao gồm cách cài đặt, định nghĩa các ca kiểm thử, thực hiện kiểm thử cũng như phân tích kết quả. Ngoài ra, chúng em cũng tiến hành so sánh và đánh giá ưu nhược điểm của Pyresttest so với các công cụ kiểm thử API khác, từ đó đưa ra nhận định tổng quát.

Trong quá trình thực hiện báo cáo, chúng em đã cố gắng hết sức để vận dụng những kiến thức đã học, đồng thời tự nghiên cứu và khám phá thêm nhiều nội dung mới. Tuy nhiên, do trình độ và kinh nghiệm thực tiễn còn hạn chế, chúng em không thể tránh khỏi những thiếu sót và sai sót. Vì vậy, chúng em rất mong nhận được lời nhận xét và góp ý từ cô để có thể cải thiện những khuyết điểm này. Sự chỉ dẫn tận tình của cô không chỉ giúp chúng em nâng cao kiến thức và kỹ năng trong môn học hiện tại mà còn là động lực lớn giúp chúng em tiếp tục học hỏi và phát triển trong tương lai.

Một lần nữa, chúng em xin chân thành cảm ơn cô Nguyễn Thu Trang vì sự hỗ trợ và hướng dẫn tận tâm của cô.

Đóng góp của các thành viên

Họ và tên	MSSV	Đóng góp	Công việc
Đỗ Tiến Đạt	22028240	25%	<ul style="list-style-type: none">• Thực nghiệm trên hệ thống• Quay video demo
Nguyễn Thị Vân Anh	22028128	25%	<ul style="list-style-type: none">• Tìm hiểu và đánh giá công cụ• Tìm hiểu cách cài đặt và sử dụng công cụ• Hỗ trợ viết kịch bản kiểm thử
Trần Lương Minh Đức	22028244	25%	<ul style="list-style-type: none">• Tìm hiểu Tổng quan• Tìm hiểu các tính năng, đặc trưng của công cụ
Đường Minh Hoàng	22028186	25%	<ul style="list-style-type: none">• Tìm hiểu Tổng quan• Tìm hiểu cơ chế hoạt động, các thành phần và mối quan hệ của công cụ

Mục lục

1	Tổng quan	5
1.1	Đặt vấn đề	5
1.2	Functional Testing	6
1.2.1	Functional Testing là gì?	6
1.2.2	Tầm quan trọng của Functional Testing	6
1.2.3	Các loại Functional Testing	8
1.2.4	Quy trình thực hiện Functional Testing	8
1.3	Các thuật ngữ liên quan đến API	9
1.3.1	Khái niệm	9
1.3.2	Cách thức API hoạt động	10
1.3.3	REST API	11
2	Công cụ kiểm thử Pyresttest	12
2.1	Pyresttest là gì?	12
2.2	Tính năng của Pyresttest	13
2.2.1	Kiểm thử API REST	13
2.2.2	Đo lường hiệu suất API (Benchmarking)	14
2.2.3	Trích xuất và tái sử dụng dữ liệu (Extractors)	14
2.2.4	Tự động hóa và tích hợp dòng lệnh	14
2.3	Đặc trưng của Pyresttest	15
2.4	Cơ chế hoạt động	16
2.4.1	Đọc tệp cấu hình (YAML/JSON)	16
2.4.2	Tạo yêu cầu HTTP	16
2.4.3	Gửi yêu cầu đến API	17
2.4.4	Nhận và xử lý phản hồi	17
2.4.5	Xác thực phản hồi	17
2.4.6	Trích xuất dữ liệu	18
2.4.7	Báo cáo kết quả	18
2.5	Các thành phần và mối quan hệ	18
2.5.1	Tệp cấu hình (YAML/JSON)	19
2.5.2	Generators (Bộ tạo dữ liệu)	20
2.5.3	Validators (Bộ xác thực)	20

2.5.4	Extractors (Bộ trích xuất)	21
2.5.5	Benchmarking Tools (Công cụ đo lường hiệu suất) . .	21
3	Đánh giá công cụ	22
3.1	Ưu và Nhược điểm của Pyresttest	22
3.1.1	Ưu điểm	22
3.1.2	Nhược điểm	23
3.2	So sánh Pyresttest so với các công cụ kiểm thử khác	24
4	Thực nghiệm	27
4.1	Cài đặt và sử dụng	27
4.2	Thực nghiệm trên hệ thống	30
4.2.1	Kiểm thử HTTP Request	30
4.2.2	Kiểm thử hiệu năng (Performance Testing)	37
4.2.3	Kiểm thử tích hợp CI/CD	41
4.3	Demo	43
5	Kết luận	43

1 Tổng quan

1.1 Đặt vấn đề

Trong lĩnh vực phát triển phần mềm hiện đại, kiểm thử (testing) đóng vai trò then chốt, không chỉ là một bước trong quy trình mà còn là yếu tố quyết định đến chất lượng của sản phẩm cuối cùng. Quá trình kiểm thử kỹ lưỡng giúp phát hiện sớm các lỗi tiềm ẩn và đảm bảo hệ thống vận hành đúng theo yêu cầu đã đề ra, từ đó nâng cao độ tin cậy trước khi sản phẩm chính thức đến tay người dùng.

Các công cụ kiểm thử tự động đã và đang mang lại những thay đổi đột phá trong cách thức làm việc của các đội phát triển, nhờ khả năng tối ưu hóa nguồn lực, tiết kiệm đáng kể thời gian và chi phí, đồng thời đảm bảo tính nhất quán trong kết quả đánh giá bằng cách loại bỏ yếu tố sai sót do con người. Mặc dù chi phí đầu tư ban đầu cho việc triển khai các công cụ kiểm thử có thể cao, nhưng xét về lâu dài, lợi ích từ việc giảm thiểu rủi ro lỗi phần mềm và tối ưu hóa quá trình phát triển là vô cùng giá trị.

Trong bài báo cáo này, nhóm em sẽ tập trung vào **Pyresttest** – một công cụ kiểm thử API mã nguồn mở được viết bằng Python, giúp kiểm tra tính đúng đắn của các API REST để bảo đảm tính chính xác của hệ thống

1.2 Functional Testing

1.2.1 Functional Testing là gì?

Kiểm thử chức năng được định nghĩa là một loại kiểm thử xác minh rằng mỗi chức năng của ứng dụng phần mềm hoạt động theo yêu cầu và thông số kỹ thuật. Kiểm thử này không liên quan đến mã nguồn của ứng dụng. Mỗi chức năng của ứng dụng phần mềm được kiểm thử bằng cách cung cấp đầu vào kiểm thử phù hợp, mong đợi đầu ra và so sánh đầu ra thực tế với đầu ra mong đợi. Kiểm thử này tập trung vào việc kiểm tra giao diện người dùng, API, cơ sở dữ liệu, bảo mật, ứng dụng máy khách hoặc máy chủ và chức năng của Ứng dụng đang được kiểm thử. Kiểm thử chức năng có thể được thực hiện thủ công hoặc tự động.

Kiểm thử chức năng là một trong các quy trình đảm bảo chất lượng của lĩnh vực kiểm thử phần mềm. Đây là một loại kiểm thử black box tức là trường hợp nó cần xét đến sẽ dựa vào đặc tả của ứng dụng hoặc hệ thống đang thử nghiệm. Các chức năng sẽ được kiểm tra bằng cách nhập các giá trị đầu vào và sau đó kiểm tra, đánh giá kết quả đầu ra mà không cần quan tâm các cấu trúc hay cài đặt bên trong ứng dụng.

1.2.2 Tầm quan trọng của Functional Testing

Trong kiểm thử phần mềm có nhiều quy trình khác nhau, mỗi cái lại có 1 nhiệm vụ khác nhau:

- Kiểm thử đơn vị (Unit testing) sẽ kiểm tra sự khác biệt giữa đặc tả giao tiếp của đơn vị với thực tế đơn vị này cung cấp cho phần mềm.
- Kiểm thử hệ thống sẽ đánh giá độ phù hợp của phần mềm với mục tiêu đề ra
- Còn với kiểm thử chức năng, nó sẽ làm nốt phần còn lại, đánh giá độ phù hợp của phần mềm với các đặc tả bên ngoài của nó, về các hành vi của phần mềm mà người dùng thấy được.

Kiểm thử chức năng đem lại khá nhiều lợi ích, như:

- Kiểm tra từng chức năng của ứng dụng: Kiểm tra chức năng sẽ kiểm tra từng chức năng của ứng dụng bằng cách cung cấp đầu vào phù hợp và xác minh đầu ra so với các yêu cầu chức năng của ứng dụng.

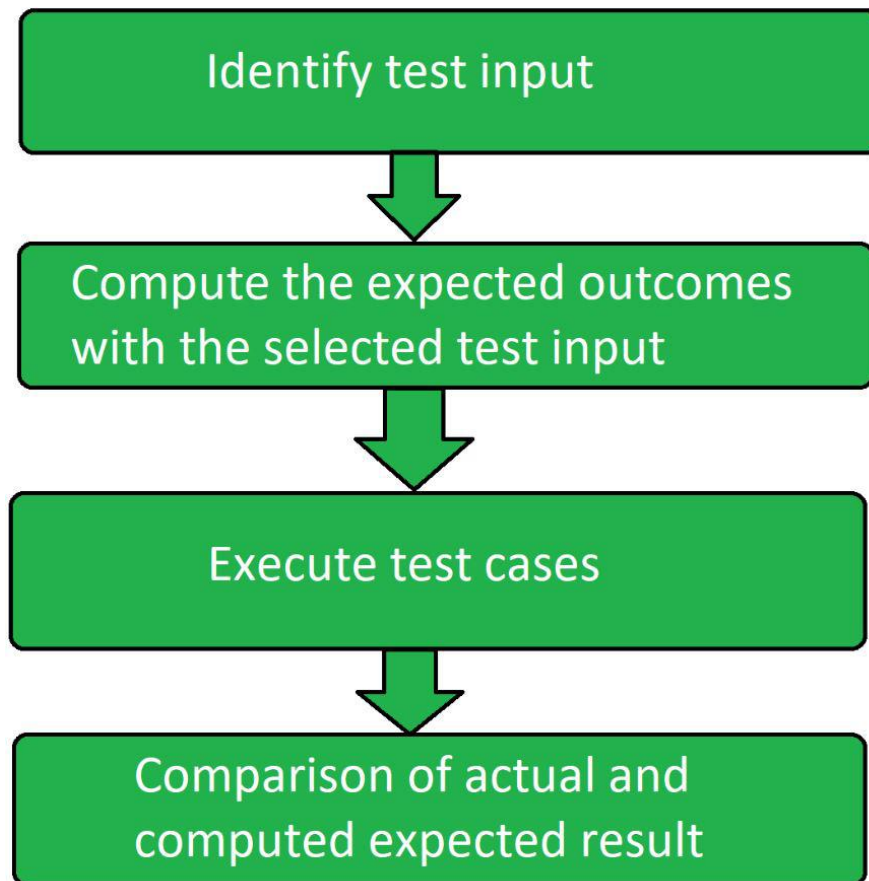
- Kiểm tra chức năng nhập chính: Trong thử nghiệm chức năng, người kiểm tra sẽ kiểm tra từng chức năng nhập của ứng dụng để kiểm tra tất cả các điểm nhập và thoát.
- Kiểm tra luồng của màn hình GUI: Trong thử nghiệm chức năng, luồng của màn hình GUI được kiểm tra để người dùng có thể điều hướng trong toàn bộ ứng dụng.

1.2.3 Các loại Functional Testing

- **Unit Testing (Kiểm thử đơn vị):** Đây là loại kỹ thuật kiểm thử chức năng trong đó các mô đun riêng lẻ của ứng dụng được kiểm thử. Nó đảm bảo rằng mô đun được hoạt động chính xác. Hoạt động kiểm thử đơn vị có thể được thực hiện theo cách thủ công hoặc tự động.
- **Smoke Testing:** Đây là một loại kỹ thuật kiểm thử chức năng trong đó chức năng hoặc tính năng cơ bản của ứng dụng được kiểm tra để đảm bảo chức năng quan trọng nhất hoạt động bình thường.
- **Integration Testing:** Các đơn vị riêng lẻ kết hợp được kiểm thử theo nhóm và phát hiện lỗi trong tương tác giữa các đơn vị tích hợp.
- **System Testing:** Là loại kiểm thử phần mềm được thực hiện trên toàn bộ hệ thống để đánh giá mức độ tuân thủ của hệ thống với các yêu cầu tương ứng.
- **Regression Testing:** Đây là loại kiểm thử để đảm bảo các thay đổi về mã không ảnh hưởng đến chức năng hiện có và các tính năng của ứng dụng. Nó tập trung vào việc tất cả các phần có hoạt động hay không.
- **Acceptance Testing:** Là giai đoạn cuối cùng để phát hành ra thị trường. Loại thử nghiệm để kiểm tra sự hài lòng của người dùng.

1.2.4 Quy trình thực hiện Functional Testing

- **Bước 1:** Xác định đầu vào kiểm thử: Bao gồm việc xác định chức năng cần thử nghiệm. Điều này có thể thay đổi từ việc thử nghiệm các chức năng khả dụng và các chức năng chính đến các điều kiện lỗi.
- **Bước 2:** Tính toán kết quả mong đợi: Tạo ra dữ liệu đầu vào dựa trên các thông số kỹ thuật của hàm và xác định đầu ra dựa trên thông số kỹ thuật.
- **Bước 3:** Thực hiện các trường hợp kiểm thử: Thực hiện các trường hợp đã thử nghiệm và ghi lại kết quả.
- **Bước 4:** So sánh đầu ra thực tế với mong đợi: Bước này cho biết hệ thống có hoạt động như mong đợi hay không.

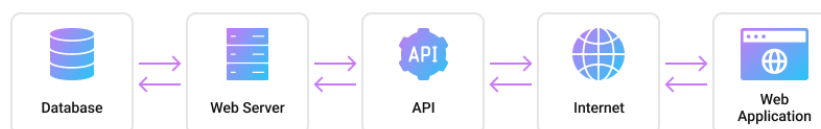


Hình 1: Quy trình thực hiện Functional Testing

1.3 Các thuật ngữ liên quan đến API

1.3.1 Khái niệm

What is an API?



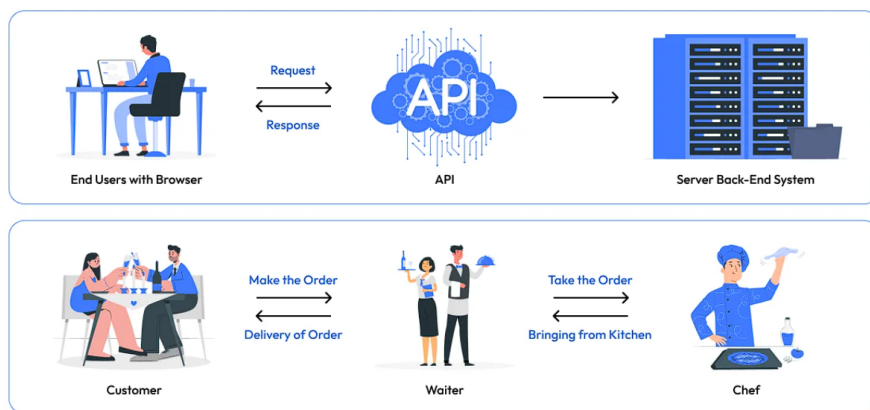
Hình 2: API là gì

API (Application Programming Interface) là tập hợp các giao thức truyền thông và chương trình con được nhiều chương trình khác nhau sử dụng để giao tiếp giữa chúng. Một lập trình viên có thể sử dụng nhiều công cụ API khác nhau để làm cho chương trình của họ dễ dàng và đơn

giản hơn. Ngoài ra, API tạo điều kiện cho các lập trình viên có cách hiệu quả để phát triển các chương trình phần mềm của họ.

API giúp hai chương trình hoặc ứng dụng giao tiếp với nhau bằng cách cung cấp cho chúng các công cụ và chức năng cần thiết. API tiếp nhận yêu cầu từ người dùng và gửi đến nhà cung cấp dịch vụ, sau đó lại gửi kết quả được tạo ra từ nhà cung cấp dịch vụ đến người dùng mong muốn. Nói đơn giản, API giống như một người trung gian giúp kết nối các hệ thống, ứng dụng mà không cần người dùng hay nhà phát triển phải hiểu hết cách hoạt động bên trong.

1.3.2 Cách thức API hoạt động



Hình 3: Cách thức API hoạt động

API hoạt động dựa trên mô hình yêu cầu và phản hồi (request-response). Khi một ứng dụng gửi yêu cầu (request) đến API, API sẽ xử lý và trả dữ liệu dữ liệu và trả dữ liệu hoặc thực hiện hành động theo yêu cầu đó. Các bước thực hiện trong hoạt động API :

- **Yêu cầu (Request):** Người dùng khởi tạo yêu cầu thông qua URI của API.
- **Xử lý:** API nhận yêu cầu, kiểm tra quyền truy cập và chuyển tiếp đến hệ thống liên quan.
- **Phản hồi (Response):** Hệ thống gửi phản hồi trở lại API với thông tin.
- API chuyển dữ liệu đến người dùng.

API an toàn về mặt tấn công vì bao gồm thông tin xác thực ủy quyền và cổng API để hạn chế quyền truy cập nhằm giảm thiểu các mối đe dọa bảo mật. Để cung cấp các lớp bảo mật bổ sung cho dữ liệu, tiêu đề HTTP, tham số chuỗi truy vấn hoặc cookie được sử dụng. Kiến trúc của API gồm:

- REST (chuyển giao trạng thái biểu diễn).
- SOAP (Giao thức truy cập đối tượng đơn giản).

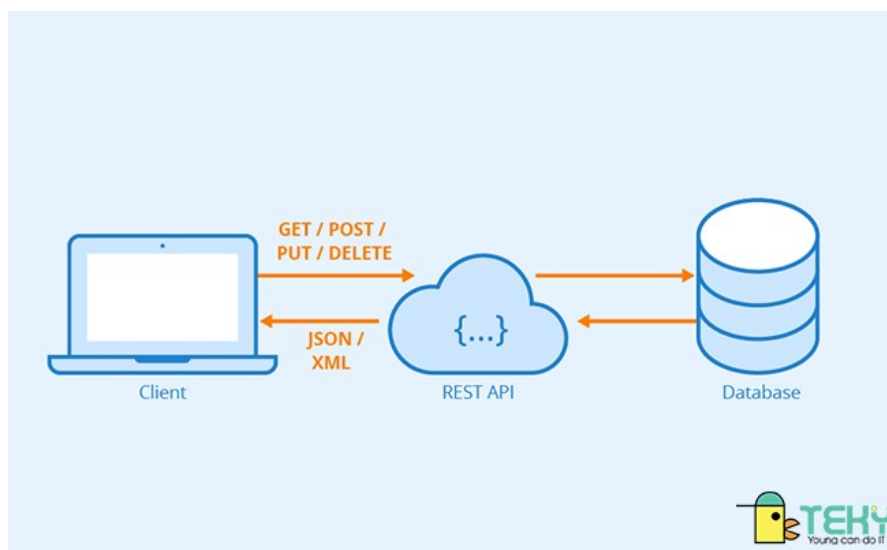
Cả hai đều định nghĩa một giao thức truyền thông chuẩn để trao đổi thông điệp ở định dạng XML.

Vì Pyresttest là một công cụ để test Restful API nên dưới đây sẽ là phần trình bày rõ ràng hơn về REST API.

1.3.3 REST API

REST API (Presentational State Transfer API) - Đây là một loại API cho phép giao tiếp giữa các hệ thống khác nhau thông qua Internet. REST API hoạt động bằng cách gửi yêu cầu và nhận phản hồi, thường ở định dạng JSON, giữa máy khách và máy chủ.

REST API sử dụng các phương thức HTTP như GET, POST, PUT, DELETE,... để xác định các hành động có thể thực hiện trên các tài nguyên. Các phương thức này thực hiện với các hành động CRUD (Create, Read, Update, Delete) được sử dụng để thao tác với tài nguyên trên trang web.



Hình 4: REST API

API RESTful là một tiêu chuẩn để sử dụng trong việc thiết kế các API cho web ứng dụng để quản lý tài nguyên. RESTful là một trong những API được sử dụng phổ biến nhất ngày nay để cho các ứng dụng khác (web, di động,...) giao tiếp với nhau.

RESTful không xác định ứng dụng mã logic và không bị giới hạn bởi ứng dụng cài đặt ngôn ngữ, bất kì ngôn ngữ hoặc khung nào cũng có thể sử dụng và thiết kế 1 API RESTful.

2 Công cụ kiểm thử Pyresttest

2.1 Pyresttest là gì?

Pyresttest là một công cụ mã nguồn mở dựa trên Python, được thiết kế để đơn giản hóa việc kiểm thử API RESTful và đo lường hiệu suất mạng. Sử dụng các tệp cấu hình YAML dễ đọc, Pyresttest cho phép người dùng định nghĩa các kịch bản kiểm thử mà không cần viết mã phức tạp, phù hợp cho cả người mới bắt đầu và nhà phát triển chuyên nghiệp. Công cụ này hỗ trợ gửi yêu cầu HTTP, xác thực phản hồi, tạo dữ liệu động và thu thập số liệu hiệu suất như độ trễ hoặc thông lượng. Với tính năng tích hợp vào quy trình CI/CD và khả năng mở rộng thông qua các tiện ích tùy chỉnh, Pyresttest là lựa chọn lý tưởng để đảm bảo chất lượng API một cách hiệu quả và linh hoạt.

Pyresttest có mục tiêu chính là:

- **Đơn giản hóa việc kiểm thử API REST:** Pyresttest cho phép người dùng định nghĩa các bài kiểm thử (test) bằng cách sử dụng các file cấu hình YAML, giúp việc kiểm thử trở nên dễ dàng, trực quan và không yêu cầu kỹ năng lập trình nâng cao.
- **Hỗ trợ kiểm thử tự động:** Công cụ này được xây dựng để tự động hóa các bài kiểm thử API, từ việc gửi yêu cầu (request) đến xác thực phản hồi (response), giúp tiết kiệm thời gian và giảm thiểu lỗi do kiểm thử thủ công.
- **Đo lường hiệu suất:** Ngoài kiểm thử chức năng, Pyresttest cung cấp khả năng đo lường hiệu suất (benchmarking) của API, giúp người dùng đánh giá tốc độ và độ ổn định của dịch vụ REST.

- **Tương thích đa nền tảng và dễ mở rộng:** Pyresttest được thiết kế để hoạt động trên nhiều hệ điều hành (Ubuntu, CentOS, Debian) và hỗ trợ cả Python 2 và Python 3. Công cụ cũng cho phép mở rộng thông qua các extension, giúp người dùng thêm các tính năng tùy chỉnh.
- **Hỗ trợ môi trường không chuyên Python:** Mặc dù được viết bằng Python, Pyresttest được thiết kế để sử dụng trong các môi trường mà Python không phải là ngôn ngữ chính, với mục tiêu dễ dàng tích hợp và sử dụng bởi các nhóm phát triển đa dạng.
- **Duy trì tính tương thích ngược:** Pyresttest cam kết duy trì tính tương thích ngược (back-compatibility) cho các định dạng YAML và giao diện dòng lệnh kể từ phiên bản 1.0, đảm bảo rằng các bài kiểm thử cũ vẫn hoạt động khi công cụ được cập nhật.

2.2 Tính năng của Pyresttest

Pyresttest cung cấp một loạt các tính năng mạnh mẽ để hỗ trợ kiểm thử API REST và đo lường hiệu suất. Dưới đây là các tính năng nổi bật:

2.2.1 Kiểm thử API REST

- **Kiểm thử API REST dựa trên YAML**
 - **Định nghĩa kiểm thử bằng YAML:** Pyresttest cho phép viết các bài kiểm thử trong file YAML, bao gồm URL, phương thức HTTP (GET, POST, PUT, DELETE), header, body, và các điều kiện xác thực (validators). Cú pháp YAML trực quan, phù hợp cho cả người không chuyên lập trình. Ví dụ:

```
- config:
  - testset: "Demo variable usage"
  - variable_binds: { var_str: 'string123', var_int: 123 }
- test:
  - name: "Variable in validator example"
  - url: "/my/path/"
  - headers: {"Content-Type": "application/json"}
  - method: "POST"
  - body: { template: '{ "action":"add", "mystring":"$var_str",
    "myinteger": $var_int }' }
  - expected_status: [200]
  - validators:
```

```
- compare: {jsonpath_mini: "my.returned.json.path.str", comparator: "eq", expected: {template: '$var_str'}}
```

- **Xác thực phản hồi mạnh mẽ:** Hỗ trợ các validator để kiểm tra phản hồi API, bao gồm:
 - * So sánh giá trị JSON bằng jsonpath_mini.
 - * Kiểm tra mã trạng thái HTTP.
 - * Bộ so sánh như eq (bằng), contains (chứa), exists (tồn tại).
- **Quản lý biến và template:** Hỗ trợ biến (variable binding) và template trong yêu cầu, cho phép tái sử dụng giá trị và tạo kiểm thử động.

2.2.2 Đo lường hiệu suất API (Benchmarking)

- Đo lường hiệu suất chi tiết: Pyresttest đo thời gian phản hồi, số lượng yêu cầu mỗi giây, và các chỉ số hiệu suất API. Ví dụ: File YAML có thể cấu hình để chạy benchmark với số lần lặp và đo thời gian phản hồi trung bình.
- Sử dụng PyCurl: Tận dụng PyCurl để thực hiện yêu cầu HTTP với hiệu suất cao, đảm bảo kết quả đo lường chính xác.

2.2.3 Trích xuất và tái sử dụng dữ liệu (Extractors)

- Trích xuất dữ liệu từ phản hồi: Cho phép trích xuất giá trị từ phản hồi JSON (bằng jsonpath_mini) để sử dụng trong các bài kiểm thử tiếp theo, hỗ trợ kiểm thử tuần tự. Ví dụ: Trích xuất token từ phản hồi để sử dụng trong yêu cầu tiếp theo.
- Tăng tính linh hoạt: Liên kết dữ liệu giữa các bài kiểm thử, phù hợp với kịch bản phức tạp như xác thực người dùng.

2.2.4 Tự động hóa và tích hợp dòng lệnh

- Giao diện dòng lệnh: Chạy kiểm thử bằng lệnh:

```
Pyresttest <base_url> <test_file.yaml> --log=debug --print-bodies
```

- Tùy chọn linh hoạt: Hỗ trợ in body, header, bỏ qua kiểm tra SSL (`-ssl_insecure`), hoặc chế độ tương tác (`-interactive`).
- Tự động hóa kiểm thử: Tích hợp vào pipeline CI/CD để tự động hóa quy trình kiểm thử API.

2.3 Đặc trưng của Pyresttest

- **Nhẹ, dễ cài đặt:** Pyresttest chỉ yêu cầu một số thư viện cơ bản, giúp việc cài đặt nhanh chóng. Điều này đảm bảo công cụ hoạt động ổn định trên nhiều nền tảng khác nhau.
- **Thân thiện, dễ sử dụng:** Công cụ sử dụng tệp YAML hoặc JSON đơn giản để cấu hình bài kiểm thử. Người dùng không cần kiến thức lập trình sâu vẫn có thể dễ dàng thao tác.
- **Kiểm thử đa năng:** Pyresttest hỗ trợ kiểm tra cả chức năng và hiệu suất của API. Nó giúp phát hiện sớm các lỗi liên quan đến tính đúng đắn và tốc độ xử lý.
- **Linh hoạt, dễ mở rộng:** Người dùng có thể trích xuất dữ liệu từ phản hồi API để tái sử dụng. Công cụ cũng hỗ trợ mở rộng chức năng thông qua lập trình Python.
- **Tích hợp tự động hóa:** Với giao diện dòng lệnh, Pyresttest dễ dàng tích hợp vào quy trình CI/CD. Điều này nâng cao hiệu quả tự động hóa trong quá trình phát triển phần mềm.

2.4 Cơ chế hoạt động

Cơ chế hoạt động của Pyresttest được thiết kế để đơn giản hóa việc kiểm thử API REST, từ việc định nghĩa các bài kiểm thử trong tệp YAML/JSON đến thực thi và xác thực kết quả. Với sự hỗ trợ của pycurl, các tính năng như generators, validators, và trích xuất dữ liệu, Pyresttest mang lại sự linh hoạt và hiệu quả cho cả kiểm thử thủ công lẫn tự động. Về cơ chế, Pyresttest hoạt động theo các bước sau:

2.4.1 Đọc tệp cấu hình (YAML/JSON)

Quá trình kiểm thử bắt đầu khi Pyresttest đọc các tệp cấu hình được định nghĩa bằng YAML hoặc JSON. Đây là nơi người dùng khai báo thông tin chi tiết về các bài kiểm thử, bao gồm URL của API, phương thức HTTP (như GET, POST, PUT, DELETE), headers, tham số truy vấn, body dữ liệu, và các tiêu chí xác thực phản hồi. Các tệp cấu hình này cho phép định nghĩa nhiều bài kiểm thử trong một tệp duy nhất, giúp tổ chức và quản lý dễ dàng. Ví dụ, một tệp YAML có thể chứa đoạn mã sau:

```
- test:
  - name: "Check if API is reachable"
  - url: "https://jsonplaceholder.typicode.com/posts"
  - method: "GET"
  - expected_status: 200
```

Pyresttest phân tích cú pháp tệp này để hiểu các yêu cầu kiểm thử và chuẩn bị cho các bước tiếp theo.

2.4.2 Tạo yêu cầu HTTP

Dựa trên thông tin từ tệp cấu hình, Pyresttest xây dựng các yêu cầu HTTP tương ứng. Công cụ hỗ trợ tất cả các phương thức HTTP chuẩn và cho phép người dùng tùy chỉnh headers, body dữ liệu (ví dụ: JSON hoặc form-data), và các tham số truy vấn. Một tính năng nổi bật là khả năng sử dụng generators để tạo dữ liệu động, chẳng hạn như ID ngẫu nhiên hoặc dữ liệu theo trình tự, giúp kiểm thử với nhiều kịch bản khác nhau mà không cần chỉnh sửa tệp cấu hình nhiều lần. Yêu

cầu HTTP được tạo ra ở bước này sẽ sẵn sàng để gửi đến API mục tiêu.

2.4.3 Gửi yêu cầu đến API

Sau khi yêu cầu HTTP được tạo, Pyresttest sử dụng thư viện pycurl để gửi chúng đến API đích. pycurl là một thư viện mạnh mẽ, hỗ trợ các tính năng nâng cao như xác thực, quản lý kết nối, và theo dõi chuyển hướng, đồng thời đảm bảo hiệu suất cao khi thực hiện các yêu cầu mạng. Pyresttest tận dụng pycurl để gửi yêu cầu một cách đáng tin cậy, đồng thời ghi lại các thông tin quan trọng như thời gian gửi và các tham số yêu cầu để phục vụ cho việc phân tích sau này.

2.4.4 Nhận và xử lý phản hồi

Khi API trả về phản hồi, Pyresttest thu thập và phân tích các thành phần của nó, bao gồm mã trạng thái (status code), headers, và body dữ liệu (thường ở định dạng JSON hoặc XML). Ngoài ra, công cụ có thể đo lường thời gian phản hồi để hỗ trợ đánh giá hiệu suất của API. Phản hồi được xử lý để chuẩn bị cho bước xác thực, đảm bảo rằng tất cả các thông tin cần thiết đều được trích xuất và sẵn sàng cho các validators.

2.4.5 Xác thực phản hồi

Pyresttest sử dụng các validators được định nghĩa trong tệp cấu hình để kiểm tra xem phản hồi có đáp ứng các tiêu chí mong đợi hay không. Các validators này có thể kiểm tra nhiều khía cạnh, chẳng hạn như mã trạng thái (ví dụ: phải là 200 OK), nội dung của body (so sánh giá trị JSON hoặc XML), hoặc các header cụ thể. Ví dụ:

```
- validators:
  - compare:
    jsonpath_mini: "title"
    expected: "partially updated title"
```

Trong trường hợp này, Pyresttest kiểm tra xem trường user.id trong JSON phản hồi có bằng 1 hay không. Nếu bất kỳ validator nào thất

bại, bài kiểm thử sẽ được ghi nhận là không thành công.

2.4.6 Trích xuất dữ liệu

Pyresttest cung cấp khả năng trích xuất dữ liệu từ phản hồi để tái sử dụng trong các yêu cầu tiếp theo, rất hữu ích cho các kịch bản kiểm thử đa bước. Ví dụ, sau khi gửi yêu cầu đăng nhập và nhận được token, người dùng có thể trích xuất token như sau:

```
extract_binds:
    new_post_id: {jsonpath_mini: "id"}
```

Token này sau đó có thể được tham chiếu trong các yêu cầu khác bằng biến token, chẳng hạn như thêm vào header Authorization. Tính năng này tăng tính linh hoạt cho các bài kiểm thử phức tạp.

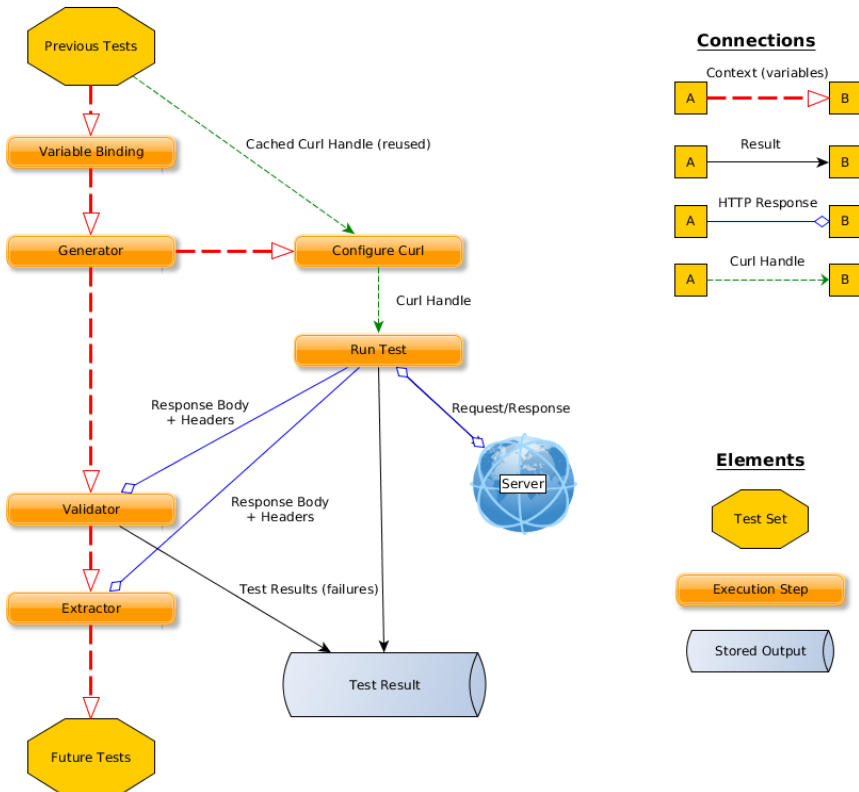
2.4.7 Báo cáo kết quả

Sau khi hoàn tất tất cả các bài kiểm thử, Pyresttest tạo báo cáo chi tiết về kết quả, bao gồm thông tin về bài kiểm thử thành công hay thất bại, thời gian thực thi, và chi tiết lỗi (nếu có). Báo cáo này được hiển thị trên giao diện dòng lệnh và có thể được lưu lại để phân tích sau. Ngoài ra, Pyresttest trả về mã thoát (exit code) – ví dụ, 0 cho thành công và khác 0 cho thất bại – để tích hợp dễ dàng vào các hệ thống tự động hóa như CI/CD pipelines, giúp các nhóm phát triển nhanh chóng phát hiện và xử lý vấn đề.

2.5 Các thành phần và mối quan hệ

Các thành phần của Pyresttest phối hợp theo một chu trình rõ ràng (Quy trình hoạt động tổng thể):

- Tập cấu hình định nghĩa bài kiểm thử và kết nối các thành phần.
- Generators tạo dữ liệu động cho yêu cầu HTTP.
- Yêu cầu được gửi đến API, và phản hồi được nhận về.
- Validators kiểm tra tính đúng đắn của phản hồi.



Hình 5: Các thành phần và mối quan hệ trong Pyresttest

- Extractors trích xuất dữ liệu cần thiết từ phản hồi để sử dụng trong các bước tiếp theo.
- Benchmarking Tools đo lường hiệu suất nếu được yêu cầu.
- Extensions hỗ trợ tùy chỉnh ở bất kỳ giai đoạn nào nếu cần.

Sự tương tác chặt chẽ giữa các thành phần này đảm bảo Pyresttest có thể xử lý các bài kiểm thử từ đơn giản đến phức tạp, đồng thời cung cấp khả năng tùy biến cao. Cụ thể:

2.5.1 Tập cấu hình (YAML/JSON)

- **Vai trò:** Tập cấu hình là nền tảng của Pyresttest, nơi người dùng định nghĩa toàn bộ bài kiểm thử. Được viết bằng định dạng YAML hoặc JSON, tập này chứa thông tin về URL của API, phương thức HTTP (GET, POST, v.v.), headers, body dữ liệu, và các tiêu chí xác thực. Đây là nơi tích hợp các thành phần khác như generators, validators, và extractors thông qua các tham số cấu hình.

- **Ví dụ:** Một tệp cấu hình có thể chỉ định gửi yêu cầu POST đến `https://api.example.com/login` với body JSON chứa dữ liệu động từ generator, sau đó xác thực mã trạng thái 200 bằng validator.
- **Mối quan hệ:** Tệp cấu hình đóng vai trò trung tâm, cung cấp hướng dẫn cho tất cả các thành phần khác. Nó chỉ định dữ liệu nào cần tạo (qua generators), cách xác thực phản hồi (qua validators), dữ liệu nào cần trích xuất (qua extractors), và liệu có đo lường hiệu suất hay không (qua benchmarking tools).

2.5.2 Generators (Bộ tạo dữ liệu)

- **Vai trò:** Generators là các công cụ tạo dữ liệu động cho các yêu cầu HTTP, giúp kiểm thử với nhiều bộ dữ liệu khác nhau mà không cần chỉnh sửa thủ công tệp cấu hình. Ví dụ, chúng có thể tạo ID ngẫu nhiên, chuỗi ký tự, hoặc dữ liệu tuần tự (sequence).
- **Ví dụ:** Generator có thể tạo một chuỗi email ngẫu nhiên như `user_{random}@example.com` để kiểm tra đăng ký người dùng.
- **Mối quan hệ:** Generators được nhúng trong tệp cấu hình và cung cấp dữ liệu cho các yêu cầu HTTP. Chúng tương tác với phần gửi yêu cầu để đảm bảo mỗi lần chạy kiểm thử đều có thể sử dụng dữ liệu mới, tăng tính linh hoạt của bài kiểm thử.

2.5.3 Validators (Bộ xác thực)

- **Vai trò:** Validators chịu trách nhiệm kiểm tra xem phản hồi từ API có đáp ứng các tiêu chí mong đợi hay không. Chúng có thể xác thực mã trạng thái HTTP (như 200 OK), nội dung body (JSON, XML), hoặc các header cụ thể.
- **Ví dụ:** Validator có thể kiểm tra xem phản hồi JSON có chứa trường `"status": "success"` hay không.
- **Mối quan hệ:** Validators được định nghĩa trong tệp cấu hình và hoạt động sau khi nhận được phản hồi từ API. Chúng tương tác trực tiếp với dữ liệu phản hồi, so sánh với các điều kiện đã đặt ra để xác định bài kiểm thử thành công hay thất bại.

2.5.4 Extractors (Bộ trích xuất)

- **Vai trò:** Extractors cho phép trích xuất dữ liệu từ phản hồi của một yêu cầu để sử dụng trong các yêu cầu tiếp theo. Điều này đặc biệt hữu ích trong các kịch bản kiểm thử phức tạp, chẳng hạn như lấy token xác thực từ phản hồi đăng nhập để sử dụng trong các yêu cầu sau.
- **Ví dụ:** Extractor có thể lấy giá trị của "token" từ phản hồi JSON và lưu vào biến để sử dụng trong header của yêu cầu tiếp theo.
- **Mối quan hệ:** Extractors hoạt động sau khi nhận phản hồi và được định nghĩa trong tệp cấu hình. Dữ liệu trích xuất được lưu trữ dưới dạng biến, tạo mối liên kết giữa các bước kiểm thử trong chuỗi.

2.5.5 Benchmarking Tools (Công cụ đo lường hiệu suất)

- **Vai trò:** Pyresttest cung cấp các công cụ benchmarking nhẹ để đo lường hiệu suất API, bao gồm thời gian phản hồi, thông lượng (throughput), và số lượng yêu cầu thành công trên giây. Đây là tính năng quan trọng khi cần đánh giá khả năng tải của API.
- **Ví dụ:** Công cụ có thể đo thời gian trung bình để API xử lý 100 yêu cầu đồng thời.
- **Mối quan hệ:** Benchmarking tools được kích hoạt thông qua tệp cấu hình và áp dụng cho các bài kiểm thử cụ thể. Chúng thu thập dữ liệu từ quá trình gửi yêu cầu và nhận phản hồi, sau đó cung cấp báo cáo hiệu suất.

3 Đánh giá công cụ

3.1 Ưu và Nhược điểm của Pyresttest

3.1.1 Ưu điểm

- **Đơn giản và dễ sử dụng:** Pyresttest không yêu cầu người dùng phải có kiến thức lập trình sâu để sử dụng. Kịch bản kiểm thử được định nghĩa trong các tệp cấu hình dạng JSON hoặc YAML, giúp dễ dàng tạo các kiểm thử cho API mà không cần mã hóa phức tạp. Có thể dùng ngay cả cho người mới bắt đầu với kiểm thử API, không cần phải làm quen với nhiều công cụ phức tạp.
- **Hỗ trợ kiểm thử RESTful API cơ bản:** Pyresttest hỗ trợ các phương thức HTTP phổ biến như GET, POST, PUT, DELETE, PATCH, và HEAD để có thể dễ dàng kiểm tra các API RESTful với các phương thức này, đồng thời kiểm tra các mã trạng thái HTTP (2xx, 4xx, 5xx) trả về cùng với phản hồi của API.
- **Hỗ trợ kiểm thử hiệu năng cơ bản (Benchmarking):** Pyresttest có thể thực hiện kiểm thử tải với số lượng yêu cầu đồng thời. Nó cho phép người dùng gửi một lượng lớn yêu cầu để kiểm tra hiệu suất của API. Mặc dù không mạnh mẽ như các công cụ chuyên biệt như JMeter hoặc Locust, nhưng đối với các yêu cầu kiểm thử hiệu suất đơn giản, Pyresttest vẫn là một lựa chọn tốt.
- **Tích hợp dễ dàng với CI/CD:** Vì Pyresttest được sử dụng qua command-line, nó có thể dễ dàng tích hợp vào các quy trình CI/CD (Continuous Integration/Continuous Deployment) để tự động hóa kiểm thử API khi có thay đổi hoặc triển khai mới. Điều này giúp tiết kiệm thời gian và đảm bảo tính ổn định của hệ thống.
- **Khả năng mở rộng với kịch bản kiểm thử phức tạp:** Mặc dù không phải là công cụ mạnh mẽ nhất về tính linh hoạt, Pyresttest vẫn cho phép xây dựng các kịch bản kiểm thử phức tạp hơn bằng cách định nghĩa nhiều yêu cầu và các kiểm tra liên quan đến dữ liệu phản hồi (như kiểm tra mã trạng thái, giá trị trong dữ liệu JSON, v.v.).
- **Không yêu cầu cài đặt phức tạp:** Pyresttest không yêu cầu cài đặt

phức tạp và các bước cấu hình đơn giản thông qua command line. Người dùng chỉ cần một môi trường Python và có thể sử dụng công cụ ngay lập tức mà không gặp phải vấn đề về các phụ thuộc phức tạp như nhiều công cụ kiểm thử khác.

- **Miễn phí và mã nguồn mở:** Pyresttest là một công cụ mã nguồn mở, hoàn toàn miễn phí. Điều này rất hữu ích cho các tổ chức hoặc cá nhân có ngân sách hạn chế và muốn sử dụng công cụ kiểm thử API mà không phải trả phí cho các phần mềm thương mại.

3.1.2 Nhược điểm

- **Ngừng phát triển và bảo trì:** Pyresttest đã ngừng phát triển từ năm 2016, nghĩa là không có các bản cập nhật, vá lỗi hay tính năng mới. Việc thiếu bảo trì có thể gây khó khăn khi gặp các vấn đề hoặc cần hỗ trợ các tính năng API hiện đại.
- **Hạn chế về khả năng kiểm thử phức tạp:** Pyresttest phù hợp cho các kiểm thử đơn giản, nhưng không thể thực hiện các kiểm thử phức tạp như kiểm thử các API có xác thực OAuth2.0, JWT do Pyresttest không có thêm các trường header, token, v.v. hoặc các API yêu cầu phê duyệt hoặc các yếu tố bảo mật phức tạp khác.
- **Không hỗ trợ kiểm thử API có giao diện đồ họa (UI):** Khác với Postman, Pyresttest chỉ hoạt động qua dòng lệnh và không có giao diện người dùng đồ họa. Điều này có thể là một bất lợi đối với những người không quen với việc làm việc qua CLI hoặc những người muốn một giao diện đồ họa trực quan để tạo và quản lý các kịch bản kiểm thử.
- **Không có tính năng ghi lại hoặc mô phỏng lưu lượng (Traffic Recording/Playback):** Một số công cụ kiểm thử API như Postman hoặc SoapUI cung cấp tính năng ghi lại và mô phỏng lại các yêu cầu HTTP, nhưng Pyresttest không hỗ trợ tính năng này, điều này có thể gây khó khăn trong việc tạo lại các tình huống kiểm thử từ các phiên làm việc thực tế và gây mất thời gian, dễ nhầm lẫn trong việc gõ lại hoặc copy paste các kịch bản kiểm thử khác.
- **Không hỗ trợ kiểm thử API SOAP hoặc các API không phải RESTful:** Pyresttest chỉ hỗ trợ kiểm thử RESTful API, vậy nên nếu

cần kiểm thử API SOAP hoặc các loại không phải RESTful API thì Pyresttest sẽ không phù hợp.

- **Khả năng báo cáo hạn chế:** Mặc dù Pyresttest có thể đưa ra kết quả kiểm thử dưới dạng các thông báo dòng lệnh, nhưng nó thiếu các báo cáo chi tiết và giao diện đồ họa như Postman hay JMeter. Điều này có thể gây khó khăn trong việc phân tích kết quả kiểm thử, đặc biệt đối với các nhóm lớn hoặc khi kiểm thử nhiều API.
- **Không hỗ trợ kiểm thử tải tích hợp sẵn:** Mặc dù Pyresttest có thể thực hiện một số bài kiểm thử hiệu suất cơ bản (như đo lường thời gian phản hồi), nhưng không có tính năng kiểm thử tải nâng cao (mô phỏng hàng nghìn người dùng đồng thời). Các công cụ như JMeter hoặc Locust được thiết kế đặc biệt để kiểm thử tải, với khả năng tạo ra các kịch bản tải nặng và đo lường hiệu suất của API dưới các điều kiện căng thẳng.
- **Hỗ trợ cộng đồng hạn chế:** Vì Pyresttest đã ngừng phát triển từ lâu, cộng đồng người dùng và các tài liệu hỗ trợ không còn phong phú như các công cụ kiểm thử API phổ biến khác nên đây có thể là một bất lợi lớn khi gặp phải lỗi hoặc cần sự giúp đỡ từ cộng đồng.

3.2 So sánh Pyresttest so với các công cụ kiểm thử khác

Để so sánh Pyresttest với các công cụ kiểm thử API tốt, có tính tương đồng, nhóm em sẽ chọn Postman và Karate. Đây là những công cụ cũng tập trung vào kiểm thử API, và phù hợp với các mục đích sử dụng trong môi trường phát triển hiện đại. Cả ba công cụ đều hỗ trợ kiểm thử API, nhưng mỗi công cụ có những đặc điểm và khả năng riêng biệt.

Bảng 1: So sánh Pyresttest, Postman và Karate

Tiêu chí	Pyresttest	Postman	Karate
Chi phí	Miễn phí, mã nguồn mở	Miễn phí (phiên bản giới hạn) và trả phí (đầy đủ)	Miễn phí, mã nguồn mở
Cài đặt	Cài qua Python	Không cần lập trình, có GUI	Cần Java và cấu hình cho dự án Java
Xây dựng test	YAML/JSON, dễ chỉnh sửa	GUI trực quan, hỗ trợ scripting	Gherkin + mã Java
Giao thức hỗ trợ	REST API	REST, SOAP, WebSocket	REST, GraphQL, JSON, XML
Cơ chế recording	Không hỗ trợ	Có “Recorder” hỗ trợ ghi lại	Không hỗ trợ
Luồng	Đơn luồng	Đơn luồng	Đa luồng
Hiệu suất	Tốt với số lượng vừa phải	Chậm với test lớn	Tốt cho dự án lớn
Báo cáo phân tích	Log đơn giản	Báo cáo chi tiết	Không có báo cáo chi tiết
Tiêu thụ tài nguyên	Thấp, dễ tích hợp CI/CD	Trung bình	Cao hơn do đa luồng
Loại test hỗ trợ	Functional + Stress cơ bản	Functional, ít stress/load	Functional, stress và load testing

Tiếp ở trang sau

Bảng tiếp theo từ trang trước

Tiêu chí	Pyresttest	Postman	Karate
Cộng đồng và tài liệu	Nhỏ, hạn chế tài liệu	Lớn, tài liệu phong phú	Nhỏ, hỗ trợ tốt cho Java

Nhìn chung qua những ưu, nhược điểm trên ta có thể rút ra kết luận về tình huống có thể sử dụng công cụ cho tối ưu nhất phù hợp với dự án của mình như sau:

- **Pyresttest** nếu cần một công cụ nhẹ, miễn phí, dễ tích hợp vào CI/CD và có nhu cầu kiểm thử RESTful API với các yêu cầu đơn giản.
- **Postman** nếu bạn cần một công cụ kiểm thử API mạnh mẽ với GUI, hỗ trợ nhiều giao thức và tích hợp mạnh mẽ với các công cụ khác, phù hợp với cả người mới và chuyên gia.
- **Karate** nếu bạn làm việc trong môi trường Java và cần một công cụ linh hoạt, hỗ trợ kiểm thử REST API, GraphQL và tích hợp tốt vào các dự án Java.

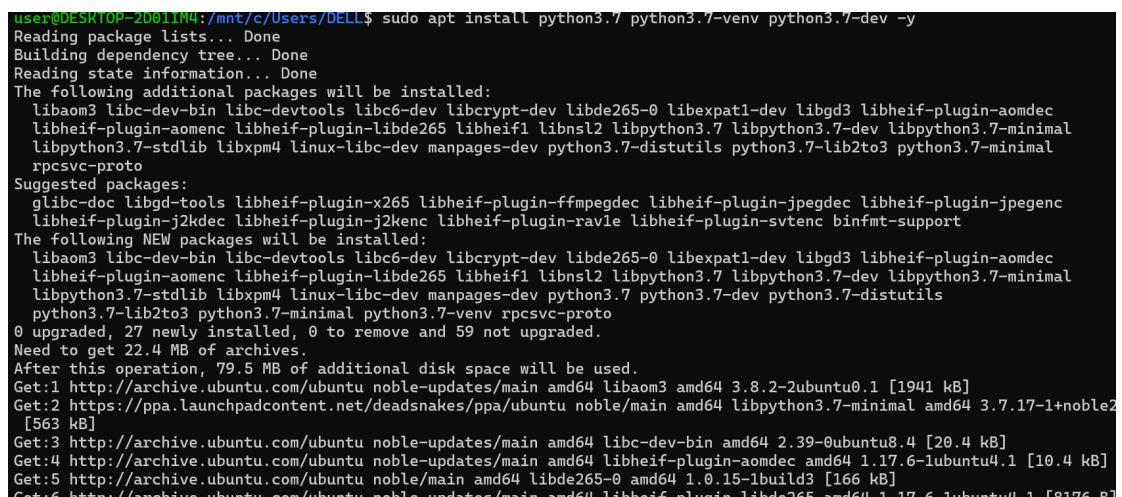
4 Thực nghiệm

4.1 Cài đặt và sử dụng

Do Pyresttest chỉ hỗ trợ trên Linux hoặc Mac với Python phiên bản 2.6, 2.7 hoặc 3.3+, để sử dụng trên Windows, ta sẽ cài đặt và chạy trên nền tảng WSL (Windows Subsystem for Linux) vì phần mềm này nhẹ và dễ cài đặt hơn so với việc sử dụng máy ảo và hệ điều hành riêng. Quy trình cài đặt được thực hiện trên Ubuntu trong WSL, gồm các bước sau:

- **Bước 1 - Cài đặt Python 3.7:** Thêm kho lưu trữ deadsnakes để tải Python 3.7, sau đó cài đặt Python cùng các gói bổ trợ.

```
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt install python3.7 python3.7-venv python3.7-dev -y
```

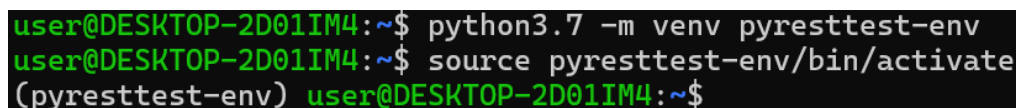


```
user@DESKTOP-2D01IM4:/mnt/c/Users/DELL$ sudo apt install python3.7 python3.7-venv python3.7-dev -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libaom3 libc-dev-bin libc-devtools libc6-dev libcrypt-dev libde265-0 libexpat1-dev libgd3 libheif-plugin-aomdec
  libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libns12 libpython3.7 libpython3.7-dev libpython3.7-minimal
  libpython3.7-stdlib libxpm4 linux-libc-dev manpages-dev python3.7-distutils python3.7-lib2to3 python3.7-minimal
  rpcsvc-proto
Suggested packages:
  glibc-doc libgd-tools libheif-plugin-x265 libheif-plugin-ffmpegdec libheif-plugin-jpegdec libheif-plugin-jpegenc
  libheif-plugin-j2kdec libheif-plugin-j2kenc libheif-plugin-rav1e libheif-plugin-svtenc binfmt-support
The following NEW packages will be installed:
  libaom3 libc-dev-bin libc-devtools libc6-dev libcrypt-dev libde265-0 libexpat1-dev libgd3 libheif-plugin-aomdec
  libheif-plugin-aomenc libheif-plugin-libde265 libheif1 libns12 libpython3.7 libpython3.7-dev libpython3.7-minimal
  libpython3.7-stdlib libxpm4 linux-libc-dev manpages-dev python3.7 python3.7-dev python3.7-distutils
  python3.7-lib2to3 python3.7-minimal python3.7-venv rpcsvc-proto
0 upgraded, 27 newly installed, 0 to remove and 59 not upgraded.
Need to get 22.4 MB of archives.
After this operation, 79.5 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libaom3 amd64 3.8.2-2ubuntu0.1 [1941 kB]
Get:2 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble/main amd64 libpython3.7-minimal amd64 3.7.17-1+noble2
[563 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libc-dev-bin amd64 2.39-0ubuntu8.4 [20.4 kB]
Get:4 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libheif-plugin-aomdec amd64 1.17.6-1ubuntu4.1 [10.4 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble/main amd64 libde265-0 amd64 1.0.15-1build3 [166 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libheif-plugin-libde265 amd64 1.17.6-1ubuntu4.1 [8176 B]
```

Hình 6: Bước 1 - Cài đặt Python 3.7

- **Bước 2 - Tạo và kích hoạt môi trường ảo:** Tạo môi trường ảo để cô lập thư viện, sau đó kích hoạt môi trường.

```
python3.7 -m venv Pyresttest-env
source Pyresttest-env/bin/activate
```



```
user@DESKTOP-2D01IM4:~$ python3.7 -m venv pyresttest-env
user@DESKTOP-2D01IM4:~$ source pyresttest-env/bin/activate
(pyresttest-env) user@DESKTOP-2D01IM4:~$
```

Hình 7: Bước 2 - Tạo và kích hoạt môi trường ảo

- **Bước 3 - Chuẩn bị hệ thống:** Cập nhật hệ thống và cài đặt các gói phụ trợ cần thiết.

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y build-essential libssl-dev libcurl4-openssl-dev
python3-dev python3-pip
```

```
(pyresttest-env) user@DESKTOP-2D01IM4:~$ sudo apt install -y build-essential libssl-dev libcurl4-openssl-dev python3-dev
python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libssl-dev is already the newest version (3.0.13-0ubuntu3.5).
libcurl4-openssl-dev is already the newest version (8.5.0-2ubuntu10.6).
The following additional packages will be installed:
  bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++ g++-13 g++-13-x86-64-linux-gnu
  g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu javascript-common
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan8 libatomic1 libcc1-0 libdpkg-perl
  libfakeroot libfile-fcntllock-perl libgcc-13-dev libgomp1 libhwasan0 libisl23 libitm1 libjs-jquery libjs-sphinxdoc
  libjs-underscore liblsan0 libmpc3 libpython3-dev libpython3.12-dev libquadmath0 libstdc++-13-dev libtsan2 libubsan1
  lto-disabled-list make python3-wheel python3.12-dev zlib1g-dev
Suggested packages:
  bzip2-doc cpp-doc gcc-13-locales cpp-13-doc debian-keyring g++-multilib g++-13-multilib gcc-13-doc gcc-multilib
  autoconf automake libtool flex bison gdb gcc-doc gcc-13-multilib gdb-x86-64-linux-gnu apache2 | lighttpd | httpd bzip
  libstdc++-13-doc make-doc
The following NEW packages will be installed:
  build-essential bzip2 cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu dpkg-dev fakeroot g++ g++-13
  g++-13-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-13 gcc-13-base gcc-13-x86-64-linux-gnu gcc-x86-64-linux-gnu
  javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan8 libatomic1
  libcc1-0 libdpkg-perl libfakeroot libfile-fcntllock-perl libgcc-13-dev libgomp1 libhwasan0 libisl23 libitm1
  libjs-jquery libjs-sphinxdoc libjs-underscore liblsan0 libmpc3 libpython3-dev libpython3.12-dev libquadmath0
  libstdc++-13-dev libtsan2 libubsan1 lto-disabled-list make python3-dev python3-pip python3-wheel python3.12-dev
  zlib1g-dev
```

Hình 8: Bước 3 - Chuẩn bị hệ thống

- **Bước 4 - Cài đặt thư viện phụ thuộc:** Cài đặt thư viện future để hỗ trợ tương thích mã Python 2 trên Python 3, và pycurl với các biến môi trường phù hợp.

```
pip install future
CFLAGS="-I/usr/include" LDFLAGS="-L/usr/lib" pip install --no-cache-dir --compile pycurl
```

```
(pyresttest-env) user@DESKTOP-2D01IM4:~$ pip install future
Collecting future
  Downloading future-1.0.0-py3-none-any.whl (491 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 491.3/491.3 kB 3.2 MB/s eta 0:00:00
Installing collected packages: future
Successfully installed future-1.0.0

[notice] A new release of pip is available: 23.0.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
```

Hình 9: Bước 4 - Cài đặt thư viện phụ thuộc

- **Bước 5 - Cài đặt Pyresttest**

```
pip install Pyresttest
```

- **Bước 6 - Kiểm tra cài đặt:** Kiểm tra thông tin Pyresttest và đảm bảo công cụ hoạt động

```
pip show Pyresttest
/home/user/Pyresttest-env/bin/Pyresttest help
```

```
(pyresttest-env) user@DESKTOP-2D01IM4:~$ CFLAGS="-I/usr/include" LDFLAGS="-L/usr/lib" pip install --no-cache-dir --compile pycurl
Collecting pycurl
  Downloading pycurl-7.45.6.tar.gz (239 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 239.5/239.5 kB 2.5 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: pycurl
  Building wheel for pycurl (pyproject.toml) ... done
  Created wheel for pycurl: filename=pycurl-7.45.6-cp37-cp37m-linux_x86_64.whl size=303869 sha256=7d7a3ccb6d7cfad0e9d0592c654c3ea736a2922e30e556e0dde5ce355e17cdaba
  Stored in directory: /tmp/pip-ephem-wheel-cache-ue2jes1e/wheels/e1/65/6c/8dc208d026808b9c47e8f4570930383e0c5f117e5717cde610
Successfully built pycurl
Installing collected packages: pycurl
Successfully installed pycurl-7.45.6

[notice] A new release of pip is available: 23.0.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
```

Hình 10: Bước 4 - Cài đặt thư viện phụ thuộc

```
(pyresttest-env) user@DESKTOP-2D01IM4:~$ pip install pyresttest
Collecting pyresttest
  Downloading pyresttest-1.7.1.tar.gz (43 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.9/43.9 kB 570.3 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting pyyaml
  Downloading PyYAML-6.0.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (670 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 670.1/670.1 kB 4.8 MB/s eta 0:00:00
Requirement already satisfied: pycurl in ./pyresttest-env/lib/python3.7/site-packages (from pyresttest) (7.45.6)
Requirement already satisfied: future in ./pyresttest-env/lib/python3.7/site-packages (from pyresttest) (1.0.0)
Installing collected packages: pyyaml, pyresttest
  DEPRECATION: pyresttest is being installed using the legacy 'setup.py install' method, because it does not have a 'pyproject.toml' and the 'wheel' package is not installed. pip 23.1 will enforce this behaviour change. A possible replacement is to enable the '--use-pep517' option. Discussion can be found at https://github.com/pypa/pip/issues/8559
  Running setup.py install for pyresttest ... done
Successfully installed pyresttest-1.7.1 pyyaml-6.0.1

[notice] A new release of pip is available: 23.0.1 -> 24.0
[notice] To update, run: pip install --upgrade pip
```

Hình 11: Bước 5 - Cài đặt Pyresttest

```
(pyresttest-env) user@DESKTOP-2D01IM4:~$ /home/user/pyresttest-env/bin/pyresttest --help
Usage: pyresttest base_url test_filename.yaml [options]

Options:
  -h, --help            show this help message and exit
  --print-bodies=PRINT_BODIES
                        Print all response bodies
  --print-headers=PRINT_HEADERS
                        Print all response headers
  --log=LOG              Logging level
  --interactive=INTERACTIVE
                        Interactive mode
  --url=URL              Base URL to run tests against
  --test=TEST            Test file to use
  --import_extensions=IMPORT_EXTENSIONS
                        Extensions to import, separated by semicolons
  --vars=VARS            Variables to set, as a YAML dictionary
  --verbose              Put cURL into verbose mode for extra debugging power
  --ssl-insecure         Disable cURL host and peer cert verification
  --absolute-urls        Enable absolute URLs in tests instead of relative paths
  --skip_term_colors     Turn off the output term colors
```

Hình 12: Bước 6 - Kiểm tra cài đặt

Nếu câu lệnh trên chạy thành công và hiển thị thông tin trợ giúp mà không báo lỗi, Pyresttest đã được cài đặt thành công.

Lưu ý: Cách cài đặt Pyresttest trên MacOS và hệ điều hành Ubuntu/Linux tương tự như ta làm trên giao diện WSL.

4.2 Thực nghiệm trên hệ thống

4.2.1 Kiểm thử HTTP Request

Để thực hiện kiểm thử HTTP, ta sử dụng lệnh sau:

```
/home/user/Pyresttest-env/bin/Pyresttest https://jsonplaceholder.
typicode.com test_httpreq.yaml --verbose
```

- `/home/user/Pyresttest-env/bin/Pyresttest`: Đường dẫn tuyệt đối đến công cụ Pyresttest.
- `https://jsonplaceholder.typicode.com`: API cần kiểm thử (có thể là API local hoặc API công khai).
- `test_httpreq.yaml`: File chứa kịch bản kiểm thử, định dạng YAML hoặc JSON.

Cấu trúc file YAML:

- **Cấu hình (config)**: Xác định các thiết lập chung như thời gian chờ, mô tả bộ kiểm thử.
- **Bài kiểm thử (test)**: Danh sách các yêu cầu HTTP và kết quả mong đợi.
- **Bộ kiểm tra (validators)**: Xác minh phản hồi từ API.

Dưới đây là file `test_httpreq.yaml` với 7 test case:

```
- config:
  - testset: "JSONPlaceholder API Test" # Tên của TestGroup
  - timeout: 5 # Timeout cho mọi test trong TestGroup

# Kiểm tra xem API có truy cập được hay không
# Truy cập vào đường dẫn API trong trường url để lấy danh sách bài viết
  với phương thức GET
# Nếu mã trạng thái HTTP trả về là 200 như expected_status thì testcase
  thành công
- test:
  - name: "Check if API is reachable"
  - url: "https://jsonplaceholder.typicode.com/posts"
  - method: "GET"
  - expected_status: 200

# Kiểm tra khả năng lấy thông tin của một bài viết cụ thể
# Truy cập vào đường dẫn API trong trường url để lấy bài viết id=1 với
  phương thức GET
```

```
# Neu ma trang thai HTTP tra ve la 200 nhu expected_status va truong id
    co ton tai trong phan hoi JSON
# Thoa man cac dieu kien tren thi Testcase thanh cong
- test:
    - name: "Retrieve a specific post"
    - url: "https://jsonplaceholder.typicode.com/posts/1"
    - method: "GET"
    - expected_status: 200
    # Kiem tra xem truong id co ton tai trong phan hoi JSON hay khong
    # Su dung jsonpath_mini de trich xuất va kiem tra du lieu
    - validators:
        - compare: {jsonpath_mini: "id", comparator: "ne", expected: null
            }

# Kiem tra kha nang tao mot bai viet moi thong qua API.
# Truy cap vao duong dan API trong truong url de tao bai viet voi phuong
    thuc POST
# Trich xuất truong id tu phan hoi va luu vao bien userId de su dung sau.
# Neu ma trang thai HTTP tra ve la 201 nhu expected_status
# Neu truong id co ton tai trong phan hoi JSON va gia tri truong title
    bang foo
# Thoa man cac dieu kien tren thi Testcase thanh cong
- test:
    - name: "Create a new post"
    - url: "https://jsonplaceholder.typicode.com/posts"
    - method: "POST"
    - headers: # Dinh dang du lieu gui di la JSON
        Content-Type: "application/json"
    - body: '{"title": "foo", "body": "bar", "userId": 1}' # Noi dung bai
        viet moi
    - expected_status: 201
    - validators:
        # Kiem tra truong id ton tai
        - compare: {jsonpath_mini: "id", comparator: "ne", expected: null
            }

        # Kiem tra gia tri truong title bang foo
        - compare:
            jsonpath_mini: "title"
            expected: "foo"
    - extract_binds: # Luu gia tri userId tu phan hoi vao bien userId
        userId: {jsonpath_mini: "userId"}

# Kiem tra kha nang lay thong tin cua mot bai viet cu the tu viec su dung
    bien userId da luu
# Truy cap vao duong dan API trong truong url de lay bai viet co id=
    userId voi phuong thuc GET
# Neu ma trang thai HTTP tra ve la 200 nhu expected_status va truong id
    co ton tai trong phan hoi JSON
# Thoa man cac dieu kien tren thi Testcase thanh cong
```



```
- test:
  - name: "Retrieve a specific post 2"
  - url: {template: "https://jsonplaceholder.typicode.com/posts?userId=
    $userId"}
  - method: "GET"
  - expected_status: 200

# Kiểm tra khả năng cập nhật toàn bộ thông tin của một bài viết
# Truy cập vào đường dẫn API trong trường url để cập nhật bài viết với
  phương thức PUT
# Nếu mã trạng thái HTTP trả về là 200 như expected_status
# Nếu giá trị trường title bằng updated title
# Thỏa mãn các điều kiện trên thì Testcase thành công
- test:
  - name: "Update an existing post"
  - url: "https://jsonplaceholder.typicode.com/posts/1"
  - method: "PUT"
  - headers: # Định dạng dữ liệu gửi đi là JSON
    Content-Type: "application/json"
  # Nội dung cập nhật toàn bộ bài viết.
  - body: '{"id": 1, "title": "updated title", "body": "updated body",
    "userId": 1}'
  - expected_status: 200
  - validators:
    # Kiểm tra giá trị trường title bằng updated title
    - compare:
      jsonpath_mini: "title"
      expected: "updated title"

# Kiểm tra khả năng cập nhật một phần thông tin của bài viết
# Truy cập vào đường dẫn API trong trường url để cập nhật bài viết với
  phương thức PATCH
# Nếu mã trạng thái HTTP trả về là 200 như expected_status
# Nếu giá trị trường title bằng updated title
# Thỏa mãn các điều kiện trên thì Testcase thành công
- test:
  - name: "Partially update a post"
  - url: "https://jsonplaceholder.typicode.com/posts/1"
  - method: "PATCH"
  - headers: # Định dạng dữ liệu gửi đi là JSON
    Content-Type: "application/json"
  # Chỉ cập nhật trường title
  - body: '{"title": "partially updated title"}'
  - expected_status: 200
  - validators:
    # Kiểm tra giá trị trường title bằng updated title
    - compare:
      jsonpath_mini: "title"
      expected: "partially updated title"
```

```
# Kiểm tra khả năng xóa một bài viết
# Truy cập vào đường dẫn API trong trường url để xóa bài viết với phương
  thực DELETE
# Nếu mã trạng thái HTTP trả về là 200 như expected_status
# Thỏa mãn điều kiện trên thì Testcase thành công
- test:
  - name: "Delete a post"
  - url: "https://jsonplaceholder.typicode.com/posts/1"
  - method: "DELETE"
  - expected_status: 200
```

- Phần config:

- test: Tên cấu hình (Test Group Name)
- timeout: Thời gian chờ (timeout cho mọi yêu cầu trong nhóm này, đơn vị giây)
- numberOfThreads: Số lượng thread chạy song song (số thread chạy cùng lúc trong nhóm này)

- Phần test: Mọi bài kiểm thử được định nghĩa trong 1 cấu trúc có thể là "test"

- name: tên bài test
- url: URL để API mà test trích dẫn
- method: Phương thức HTTP được sử dụng (GET, POST, ...)
- headers: Phần định nghĩa các tiêu đề HTTP được gửi
- body: Nội dung dữ liệu gửi đi
- expected: status: Mã trạng thái HTTP mong đợi từ phản hồi
- extract: Binds: Phần định nghĩa cách lấy thông tin từ phản hồi để sử dụng trong các bước tiếp theo
- VD extract: binds:
 - * new post: {jsonpath: min "id"}
 - * Thì lấy new post là giá trị của trường "id" trong phản hồi API trả về

- Phần validators: Phần định nghĩa các bước kiểm tra trong API

- compare: Bắt buộc tra cứu trường danh sách kiểm tra
- jsonpath: Truy vấn node bộ lọc trả về sẽ được kiểm tra

- **expect**: Dạng điều tra
 - * **not**, **exists**, **equals**, **not equals**, **greater than**, **less than**, **matches**, v.v.
- **expected**: Giá trị mong đợi của trường được kiểm tra
- **jsonpath**: min
- Nếu ta muốn kiểm tra trường này với các bước so sánh khác nhau

Kết quả thực thi: Đây là ví dụ về mẫu phản hồi cho 1 Test trong bộ kiểm thử, các test còn lại cũng trả về các trường kiểm tra tương tự.

```
(pyresttest-env) user@DESKTOP-2D01IM4:~$ /home/user/pyresttest-env/bin/pyresttest https://jsonplaceholder.typicode.com/posts test_httpreq.yaml --verbose
* Host jsonplaceholder.typicode.com:443 was resolved.
* IPv6: 2606:4700:3030::6815:4001, 2606:4700:3030::6815:5001, 2606:4700:3030::6815:6001, 2606:4700:3030::6815:1001, 2606:4700:3030::6815:7001, 2606:4700:3030::6815:3001, 2606:4700:3030::6815:2001
* IPv4: 104.21.96.1, 104.21.80.1, 104.21.48.1, 104.21.16.1, 104.21.32.1, 104.21.64.1, 104.21.112.1
* Trying 104.21.96.1:443...
* Connected to jsonplaceholder.typicode.com (104.21.96.1) port 443
* ALPN: curl offers h2,http/1.1
* CAfile: /etc/ssl/certs/ca-certificates.crt
* CApath: /etc/ssl/certs
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384 / X25519 / id-ecPublicKey
* ALPN: server accepted h2
* Server certificate:
* subject: CN=typicode.com
* start date: Feb 13 08:49:36 2025 GMT
* expire date: May 14 09:47:35 2025 GMT
* subjectAltName: host "jsonplaceholder.typicode.com" matched cert's "*.typicode.com"
* issuer: C=US; O=Google Trust Services; CN=WE1
* SSL certificate verify ok.
* Certificate level 0: Public key type EC/prime256v1 (256/128 Bits/secBits), signed using ecdsa-with-SHA256
* Certificate level 1: Public key type EC/prime256v1 (256/128 Bits/secBits), signed using ecdsa-with-SHA384
* Certificate level 2: Public key type EC/secp384r1 (384/192 Bits/secBits), signed using ecdsa-with-SHA384
* using HTTP/2
* [HTTP/2] [1] OPENED stream for https://jsonplaceholder.typicode.com/posts
* [HTTP/2] [1] [:method: GET]
* [HTTP/2] [1] [:scheme: https]
* [HTTP/2] [1] [:authority: jsonplaceholder.typicode.com]
* [HTTP/2] [1] [:path: /posts]
* [HTTP/2] [1] [user-agent: PycURL/7.45.6 libcurl/8.5.0 OpenSSL/3.0.13 zlib/1.3 brotli/1.1.0 zstd/1.5.5 libidn2/2.3.7 li
```

Hình 13: Ví dụ về mẫu phản hồi cho 1 Test trong bộ kiểm thử

• Kết nối đến máy chủ

- Pyresttest được dùng để kiểm tra kết nối IPv6 và IPv4, chọn mô tả chi tiết kết nối, sau đó thiết lập phiên bảo mật SSL/TLS (giáo thức TLSv1.3) với xác thực chứng chỉ SSL.
- Sau đó lấy thông tin HTTP từ API với các thông tin sau trong request:
 - * **method**: Phương thức gửi
 - * **scheme**: Giao thức (HTTP)
 - * **authority**: Tên miền của máy chủ
 - * **path**: Đường dẫn
 - * **User-Agent**: Thông tin về client
 - * **accepts**: Loại dữ liệu trả về

* **Connection:** Yêu cầu đóng kết nối sau khi hoàn thành hoặc tiếp tục

```
* [HTTP/2] [1] [authority: jsonplaceholder.typicode.com]
* [HTTP/2] [1] [path: /posts]
* [HTTP/2] [1] [user-agent: PycURL/7.45.6 libcurl/8.5.0 OpenSSL/3.0.13 zlib/1.3 brotli/1.1.0 zstd/1.5.5 libidn2/2.3.7 libpsl/0.21.2 (+libidn2/2.3.7) libssh/0.10.6/openssl/zlib nghttp2/1.59.0 librtmp/2.3 OpenLDAP/2.6.7]
* [HTTP/2] [1] [accept: /*]
> GET /posts HTTP/2
Host: jsonplaceholder.typicode.com
User-Agent: PycURL/7.45.6 libcurl/8.5.0 OpenSSL/3.0.13 zlib/1.3 brotli/1.1.0 zstd/1.5.5 libidn2/2.3.7 libpsl/0.21.2 (+libidn2/2.3.7) libssh/0.10.6/openssl/zlib nghttp2/1.59.0 librtmp/2.3 OpenLDAP/2.6.7
Accept: /*
Connection: close

* old SSL session ID is stale, removing
< HTTP/2 200
< date: Fri, 28 Mar 2025 17:31:08 GMT
< content-type: application/json; charset=utf-8
< report-to: {"group":"heroku-nel","max_age":3600,"endpoints":[{"url":"https://nel.heroku.com/reports?ts=1742886074&sid=e11707d5-02a7-43ef-b45e-2cf4d2036f7d&s=3wPt6JJU3J7tDncYq0ixquqcYlJpL29e5fQJ2husklw%3D"}]}
< reporting-endpoints: heroku-nel=https://nel.heroku.com/reports?ts=1742886074&sid=e11707d5-02a7-43ef-b45e-2cf4d2036f7d&s=3wPt6JJU3J7tDncYq0ixquqcYlJpL29e5fQJ2husklw%3D
< nel: {"report_to":"heroku-nel","max_age":3600,"success_fraction":0.005,"failure_fraction":0.05,"response_headers":["Via"]}
< x-powered-by: Express
< x-ratelimit-limit: 1000
< x-ratelimit-remaining: 999
< x-ratelimit-reset: 1742886114
< vary: Origin, Accept-Encoding
< access-control-allow-credentials: true
< cache-control: max-age=43200
< pragma: no-cache
```

Hình 14: Ví dụ về mẫu phản hồi cho 1 Test trong bộ kiểm thử

- **Nhận phản hồi từ máy chủ:** Máy chủ phản hồi bằng HTTP/2 với mã trạng thái sau đó

- * **date:** Ngày giờ máy chủ gửi phản hồi.
- * **content-type:** Dữ liệu trả về có định dạng JSON với mã hóa UTF-8.
- * **report-to:** Máy chủ chỉ định endpoint để thu thập báo cáo lỗi từ trình duyệt (NEL - Network Error Logging).
- * **reporting-endpoints:** Endpoint để thu thập báo cáo về lỗi mạng.
- * **nel:** Câu nhận chỉnh sách báo cáo lỗi mạng (Network Error Logging).
- * **x-powered-by:** Máy chủ backend sử dụng framework Express.
- * **x-ratelimit-limit:** Số lượng yêu cầu tối đa mà client có thể gửi trong một khoảng thời gian (1000 yêu cầu).
- * **x-ratelimit-remaining:** Số yêu cầu còn lại (999 yêu cầu).
- * **x-ratelimit-reset:** Thời gian (epoch timestamp) khi giới hạn ngày được đặt lại.
- * **vary:** Máy chủ có thể phản hồi khác nhau dựa trên giá trị Origin hóa Accept-Encoding.

- * **access-control-allow-credentials**: Cho phép gửi yêu cầu có chứa thông tin xác thực (cookies, tokens) trong trình duyệt.
- * **cache-control**: Không dùng ổ thể dự trữ lâu và bỏ không cache trong 43200 giây (12 giờ).
- * **pragma: no-cache & expires: -1**: Trình duyệt không nên dùng bản cache cũ.

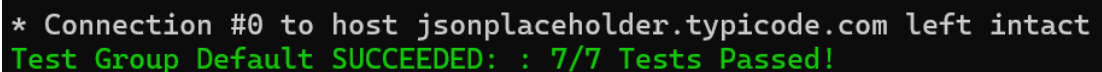
```
< cache-control: max-age=43200
< pragma: no-cache
< expires: -1
< x-content-type-options: nosniff
< etag: W/"6b80-Ybsq/K6GwwqrYkAsFxqDXGC7DoM"
< via: 1.1 vegur
< cf-cache-status: HIT
< age: 7079
< server: cloudflare
< cf-ray: 9278dbffdb42cb25-HKG
< alt-svc: h3=":443"; ma=86400
< server-timing: cFL4;desc="?proto=TCP&rtt=25973&min_rtt=25622&rtt_var=10310&sent=6&recv=7&lost=0&retrans=0&sent_bytes=3411&recv_bytes=891&delivery_rate=100457&wnd=110&unsent_bytes=0&cid=549961325b081dc2&ts=60&x=0"
<
* Connection #0 to host jsonplaceholder.typicode.com left intact
* Found bundle for host: 0x7945310 [can multiplex]
* Re-using existing connection with host jsonplaceholder.typicode.com
* [HTTP/2] [3] OPENED stream for https://jsonplaceholder.typicode.com/posts/1
* [HTTP/2] [3] [:method: GET]
* [HTTP/2] [3] [:scheme: https]
* [HTTP/2] [3] [:authority: jsonplaceholder.typicode.com]
* [HTTP/2] [3] [:path: /posts/1]
```

Hình 15: Ví dụ về mẫu phản hồi cho 1 Test trong bộ kiểm thử

- * **x-content-type-options: nosniff**: Ngăn trình duyệt đoán định dạng nội dung khác với Content-Type đã khai báo.
- * **etag**: Một mã định danh duy nhất cho phiên bản hiện tại của tài nguyên, giúp tối ưu hóa cache.
- * **via**: Máy chủ proxy (vegur) trung gian đã xử lý yêu cầu.
- * **cf-cache-status: HIT**: Cloudflare đã tìm thấy phiên bản được lưu trong cache, giúp tăng tốc độ phản hồi.
- * **age**: Dữ liệu đã được lưu trong cache 7079 giây.
- * **server**: Máy chủ sử dụng dịch vụ Cloudflare
- * **cf-ray**: Mã định danh của yêu cầu khi đi qua Cloudflare (HKG = Hong Kong).
- * **alt-svc**: Chỉ định HTTP/3 (h3) có thể được sử dụng trên cổng 443 trong 86400 giây (1 ngày).
- * **server-timing**: Thông tin về thời gian phản hồi của máy chủ, độ trễ mạng, và hiệu suất truyền dữ liệu.
 - **proto**: Giao thức mạng
 - **rtt**: Round Trip Time (RTT): Thời gian cần thiết để một gói tin đi từ máy khách đến máy chủ và quay trở lại, tính bằng

- microsecond (us).
- min_rtt: Minimum RTT: Giá trị thấp nhất của RTT được ghi nhận trong quá trình kết nối.
- rtt_var: RTT Variation: Độ biến động của RTT, giúp đánh giá sự ổn định của kết nối.
- sent: Số lượng gói tin đã được gửi từ máy khách.
- recv: Số lượng gói tin đã được nhận từ máy chủ.
- lost: Số lượng gói tin bị mất trong quá trình truyền.
- retrans: Số lượng gói tin bị gửi lại do lỗi truyền tải.
- sent_bytes: Tổng số byte đã gửi từ máy khách đến máy chủ (3.411 bytes).
- recv_bytes: Tổng số byte đã nhận từ máy chủ về máy khách (891 bytes).
- delivery_rate: Tốc độ truyền dữ liệu (Delivery Rate) tính bằng bytes/giây.
- cwnd: Congestion Window (CWND): Số lượng gói tin tối đa mà TCP có thể gửi mà không cần nhận ACK từ máy chủ. CWND lớn giúp cải thiện hiệu suất mạng.
- unsent_bytes: Số byte chưa được gửi.
- cid: Connection ID: Mã định danh của kết nối TCP này.
- ts: Timestamp: Thời điểm (có thể tính bằng giây hoặc độ vi khác) liên quan đến yêu cầu này.

Tổng kết kết quả: Cuối cùng sau khi chạy hết tất cả bộ test, ta sẽ có kết quả 7/7 Tests passed do mã trạng thái và validators đều khớp với kỳ vọng.



```
* Connection #0 to host jsonplaceholder.typicode.com left intact
Test Group Default SUCCEEDED: : 7/7 Tests Passed!
```

Hình 16: Ví dụ về mẫu phản hồi cho 1 Test trong bộ kiểm thử

4.2.2 Kiểm thử hiệu năng (Performance Testing)

Phần này sử dụng tính năng benchmark của Pyresttest để đo lường hiệu năng của API <https://jsonplaceholder.typicode.com/posts> dưới các mức tải

khác nhau (50, 500, 5000 request). Các thông số được đo bao gồm thời gian phản hồi tổng (total_time) và tốc độ tải (speed_download).

Để đi sâu vào vấn đề đo lường hiệu năng của API, ta sẽ phải tạo 1 Entity benchmark cùng cấp với Entity test trong kịch bản kiểm thử .yaml. Trong trường hợp này để ngắn gọn, nhóm em sẽ chỉ test hiệu năng 3 kịch bản benchmark, mỗi kịch bản sử dụng phương thức GET để truy vấn danh sách bài viết từ API:

```
- config:
  - testset: "JSONPlaceholder API Test"
  - timeout: 5

- benchmark:
  - name: "Basic get"
  - url: "https://jsonplaceholder.typicode.com/posts"
  - warmup_runs: 5
  - benchmark_runs: "50"
  - metrics:
    - total_time: mean
    - total_time: median
    - speed_download: median

- benchmark:
  - name: "Basic get 500"
  - url: "https://jsonplaceholder.typicode.com/posts"
  - warmup_runs: 5
  - benchmark_runs: "500"
  - metrics:
    - total_time: mean
    - total_time: median
    - speed_download: median

- benchmark:
  - name: "Basic get 5000"
  - url: "https://jsonplaceholder.typicode.com/posts"
  - warmup_runs: 5
  - benchmark_runs: "5000"
  - metrics:
    - total_time: mean
    - total_time: median
    - speed_download: median
```

Ý nghĩa của các trường trong kịch bản testcase:

- **warmup_runs**: Số lần chạy thử (5 lần) để làm nóng hệ thống, đảm bảo kết quả trên đủ chính xác khi đo hiệu năng.

- **benchmark_runs**: Số lần chạy chính thức (50, 500, 5000) để đo hiệu năng dựa trên các mẫu thử khác nhau.
- **metrics**:
 - **total_time: mean**: Thời gian trung bình (tính bằng giây) để hoàn thành một request, bao gồm cả thời gian kết nối và truyền dữ liệu.
 - **total_time: median**: Thời gian trung vị, phản ánh giá trị điển hình, ít bị ảnh hưởng bởi các giá trị ngoại lai.
 - **speed_download: median**: Tốc độ tải trung vị (tính bằng bytes/giây), do tốc độ nhấn dữ liệu từ máy chủ.

Lý do nhóm em chọn các chỉ số này: `total_time` (mean và median) giúp đánh giá thời gian phản hồi tổng thể, trong khi `speed_download` đánh giá hiệu suất truyền dữ liệu. Các chỉ số khác như `connect_time` không được chọn vì API này sử dụng cache của Cloudflare (như đã thấy ở phần kiểm thử HTTP), nên thời gian kết nối không phải là yếu tố chính.

Kết quả thực thi: Sau khi chạy lệnh

```
home/user/Pyresttest-env/bin/Pyresttest https://jsonplaceholder.typicode.com
```

kết quả benchmark được trả về như hình dưới.

```
{
  "aggregates": [
    [
      "total_time",
      "mean",
      0.054918279999999998,
      [
        "total_time",
        "median",
        0.0528535,
        [
          "speed_download",
          "median",
          520684.0
        ]
      ],
      "results": {},
      "name": "Basic get 50",
      "group": "Default",
      "failures": 0
    ],
    [
      "total_time",
      "mean",
      0.061505844000000004,
      [
        "total_time",
        "median",
        0.057019,
        [
          "speed_download",
          "median",
          482648.0
        ]
      ],
      "results": {},
      "name": "Basic get 500",
      "group": "Default",
      "failures": 0
    ],
    [
      "total_time",
      "mean",
      0.058017764600000002,
      [
        "total_time",
        "median",
        0.0532645,
        [
          "speed_download",
          "median",
          516666.0
        ]
      ],
      "results": {},
      "name": "Basic get 5000",
      "group": "Default",
      "failures": 0
    ]
  ],
  "max_seconds": 5
}
```

Hình 17: Ví dụ về kết quả benchmark

So sánh và nhận xét:

- Thời gian phản hồi (`total_time`):
 - Thí nghiệm với 50 lần 500 request, thời gian trung bình trả về 55 mili giây với độ trễ nhỏ (thấp hơn 12%). Điều này cho thấy hệ thống xử lý tốt trong trường hợp có tải cao, đặc biệt là khi sử dụng cơ chế cache Cloudflare hoạt động hiệu quả.
 - Với 500 lần 500 request, thời gian trung bình phản hồi là 65 mili giây (giảm khoảng 7% so với không dùng cache). Điều này cho thấy API xử lý tốt trong trường hợp tải cao, đặc biệt khi có cơ chế cache Cloudflare hoạt động hiệu quả.

- Thời gian trung bình phản hồi với 50 request là 55 mili giây. Với 500 request, thời gian trung bình là 65 mili giây (giảm khoảng 4-5 mili giây) cho thấy hệ thống phản hồi tốt trong mọi trường hợp tải.
- Tốc độ tải (speed_download):
 - Tốc độ tải trung bình là 521 KB/s (50 request), tương ứng 482 KB/s (500 request), giảm khoảng 7%. Điều này cho thấy hệ thống hoạt động hiệu quả trong điều kiện tải cao.
 - Với 500 lần 500 request, tốc độ tải trung bình là 516 KB/s (tăng khoảng 7% so với 50 request), điều này cho thấy hệ thống xử lý tốt trong điều kiện tải cao.
- Tính ổn định:
 - Không có request nào bị thất bại (failures: 0 trong cả 3 lần kiểm tra), chứng minh API hoạt động ổn định.
 - Đã kiểm tra tổng thời gian: mean và total_time, median phản hồi (khoảng 2-4 mili giây), cho thấy API hoạt động ổn định trên mọi tình huống.

Đánh giá tổng quát:

- **Hiệu năng API:** API jsonplaceholder.typicode.com thể hiện hiệu năng ổn định dưới các mức tải khác nhau. Thời gian phản hồi trung bình dao động từ 55 mili giây (50 request) đến 61.5 mili giây (500 request), sau đó giảm nhẹ xuống 58 mili giây (5000 request). Sự thay đổi này không quá lớn (tăng tối đa 12% từ 50 lên 500 request), cho thấy API xử lý tốt dưới tải cao, đặc biệt nhờ cơ chế cache của Cloudflare (như đã thấy ở phần kiểm thử HTTP với cf-cache-status: HIT). Tốc độ tải dao động từ 482 KB/s đến 521 KB/s, cho thấy hiệu suất truyền dữ liệu khá ổn định, không bị ảnh hưởng nhiều bởi tải. Tuy nhiên, thời gian phản hồi trung bình vẫn ở mức 58-61 mili giây, có thể chưa đáp ứng được các hệ thống yêu cầu độ trễ cực thấp (dưới 20 mili giây), đặc biệt nếu API được sử dụng trong các ứng dụng thời gian thực.
- **Khả năng của Pyresttest:** Pyresttest thực hiện tốt kiểm thử hiệu năng, với khả năng chạy 5000 request liên tiếp mà không gặp lỗi (failures: 0). Tính năng benchmark cung cấp các chỉ số chi tiết (total_time, speed_download), giúp đánh giá hiệu năng API một cách hiệu quả.

Tuy nhiên, công cụ này còn hạn chế ở chỗ không hỗ trợ chỉ số nâng cao như `time_to_first_byte`, vốn hữu ích để phân tích chi tiết các giai đoạn của request (kết nối, gửi, nhận). Ngoài ra, Pyresttest không cung cấp tính năng xuất biểu đồ trực quan hoặc phân tích độ trễ theo thời gian thực, điều mà các công cụ như JMeter hoặc Locust có thể làm tốt hơn.

4.2.3 Kiểm thử tích hợp CI/CD

Link Github: [Kiểm thử tích hợp CI/CD](#)

Nhờ tác dụng của Non-Gui mà Pyresttest có thể tích hợp với CI/CD. Nhóm sử dụng Github Actions để demo quá trình này. Các bước tích hợp như sau:

- **Bước 1:** Tải file `test_httpreq.yaml` và thêm 1 entity benchmark để đảm bảo đầy đủ chức năng của Pyresttest lên repo github
- **Bước 2:** Cấu hình job thực thi Pyresttest trong build file ở thư mục `.github/workflows` `api_test.yaml` để test chức năng của API

```
name: API Testing with Pyresttest

on:
  push:
    branches:
      - main # Chạy khi có push lên nhánh main
  pull_request:
    branches:
      - main # Chạy khi có PR vào nhánh main

jobs:
  api_test:
    runs-on: ubuntu-22.04 # Chạy trên mọi trường Ubuntu
    steps:
      - name: Checkout code
        uses: actions/checkout@v3 # Lay ma nguon

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.7' # Dung Python 3.7 (ho tro Pyresttest)

      - name: Install dependencies
        run: |
```

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y build-essential libssl-dev libcurl4-
    openssl-dev python3-dev python3-pip
CFLAGS="-I/usr/include" LDFLAGS="-L/usr/lib" pip install --
    no-cache-dir --compile pycurl
pip install future
pip install Pyresttest

- name: Run API Tests
  run: Pyresttest https://jsonplaceholder.typicode.com
    test_httpreq.yaml
```

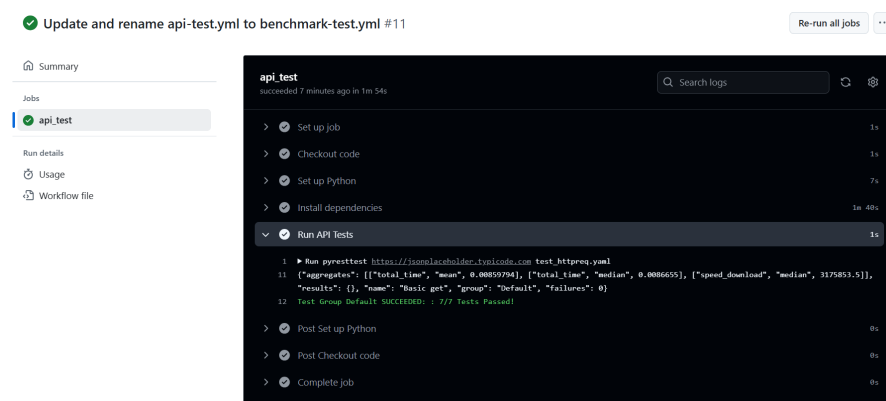
Các luồng thực hiện của actions trên là: cài đặt python3.7 để làm tiên đề cài đặt Pyresttest. Sau đó cài các thư viện phụ thuộc như pycurl, future rồi thực thi file kiểm thử và tạo báo cáo. Sau khi cấu hình file như trên, ta sẽ có mỗi lần muốn push code hoặc merge code vào nhánh main, ta cần thực thi đoạn code và sẽ trả về kết quả passed hoặc failed.

- **Bước 3:** Cấu hình job thực thi Pyresttest trong build file ở thư mục `.github/workflows benchmark_test.yaml` để test hiệu năng của API Tương tự như file `api_test.yaml` nhưng khác ở dòng cuối là:

```
run: Pyresttest https://jsonplaceholder.typicode.com
    test_benchmark.yaml
```

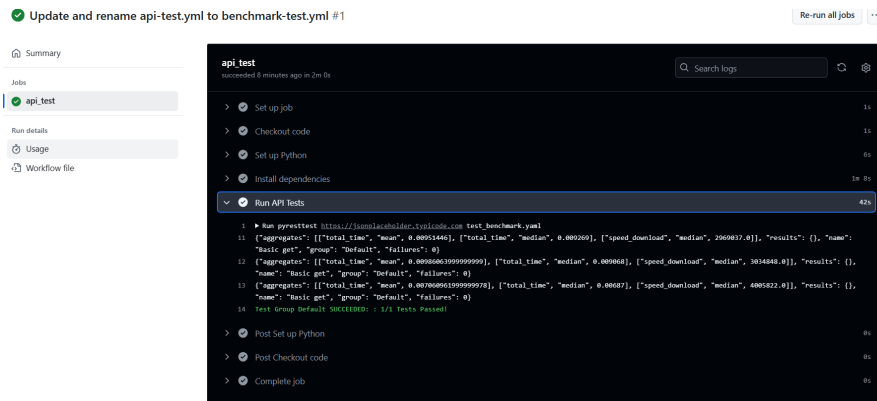
- **Bước 4:** Xem kết quả ở phần Actions

– Test chức năng



Hình 18: Test chức năng - Kiểm thử tích hợp CI/CD

– Test hiệu năng



Hình 19: Test hiệu năng - Kiểm thử tích hợp CI/CD

4.3 Demo

Link Video demo: [Demo công cụ kiểm thử Pyresttest](#)

5 Kết luận

Qua quá trình tìm hiểu và thực nghiệm với Pyresttest, nhóm chúng em nhận thấy đây là một công cụ kiểm thử API RESTful đơn giản và dễ sử dụng, phù hợp cho các dự án nhỏ đến vừa hoặc các nhóm phát triển cần một giải pháp kiểm thử cơ bản mà không đòi hỏi khả năng lập trình quá cao. Pyresttest cho phép định nghĩa các kịch bản kiểm thử bằng tệp YAML hoặc JSON, hỗ trợ các phương thức HTTP phổ biến, kiểm tra phản hồi và trích xuất dữ liệu khá hiệu quả. Việc tích hợp vào quy trình CI/CD cũng giúp tự động hóa kiểm thử, tiết kiệm thời gian và nâng cao hiệu quả làm việc cho nhóm.

Pyresttest cũng giải quyết được các khó khăn trong kiểm thử API như:

- **Kiểm thử tự động:** Kiểm thử API thủ công là công việc tốn rất nhiều thời gian, nhất là khi API có nhiều endpoint cần kiểm tra thường xuyên. Chẳng hạn, với một hệ thống thương mại điện tử, việc phải tự tay gửi yêu cầu đăng nhập, thêm sản phẩm vào giỏ hàng rồi thanh toán, sau đó kiểm tra từng phản hồi thật sự rất mất thời gian và dễ sai sót. Nếu API cập nhật, như thêm tham số mới, lại phải làm lại từ đầu. Pyresttest giúp tự động hóa quy trình này bằng cách dùng tệp YAML hoặc JSON để định nghĩa các bước kiểm thử. Ví dụ, chỉ cần viết một tệp cấu hình

là có thể kiểm tra toàn bộ quy trình mà không cần làm thủ công, vừa nhanh vừa chính xác.

- **Xử lý trạng thái và luồng dữ liệu phức tạp:** Nhiều API cần thực hiện một chuỗi yêu cầu liên kết, như trong ứng dụng ngân hàng, phải đăng nhập để lấy mã thông báo rồi mới truy cập tài khoản hay chuyển tiền được. Làm thủ công thì khó mà quản lý trạng thái cho đúng, dễ bị lỗi như quên gắn mã thông báo vào yêu cầu tiếp theo. Pyresttest hỗ trợ bằng tính năng "variable binding", cho phép lấy dữ liệu từ phản hồi (như mã thông báo) và dùng lại ở bước sau. Nhờ vậy, ta có thể kiểm tra được toàn bộ luồng dữ liệu một cách trơn tru và không phải lo sót lỗi.
- **Kiểm thử hiệu suất cơ bản:** Đảm bảo API chạy tốt khi có nhiều yêu cầu cùng lúc là một thách thức lớn. Ví dụ, một API có thể nhanh với 10 người dùng nhưng chậm hoặc lỗi khi có 1000 người dùng. Pyresttest không mạnh như các công cụ chuyên dụng, nhưng nó vẫn cho phép đo thời gian phản hồi của từng yêu cầu, giúp người dùng phát hiện sớm các vấn đề hiệu suất cơ bản.

Tuy nhiên, Pyresttest cũng có hạn chế. Công cụ ngừng phát triển từ 2016 nên không còn tương thích tốt với các tiêu chuẩn mới như OAuth2.0 hay JWT. Nó cũng không có giao diện đồ họa, gây khó cho người dùng không quen sử dụng lệnh, và thiếu các tính năng nâng cao như phân tích hiệu suất chi tiết – những thứ mà Postman hay JMeter làm tốt hơn. Vì vậy, với các dự án phức tạp, Pyresttest có thể không đáp ứng đủ.

Tóm lại, Pyresttest là một công cụ hữu ích cho kiểm thử API cơ bản, đặc biệt khi người dùng cần sự đơn giản và dễ sử dụng. Dù không phải giải pháp hoàn hảo, nó vẫn hỗ trợ tốt trong các dự án phù hợp, giúp đảm bảo chất lượng phần mềm một cách hiệu quả.