BAN CƠ YẾU CHÍNH PHỦ
HỌC VIỆN KỸ THUẬT MẬT MÃ

ĐỀ CƯƠNG CHI TIẾT HỌC PHẦN KỸ THUẬT LẬP TRÌNH

Đề Tài: XÂY DỰNG CÔNG CỤ KIỂM THỬ API

Ngành: An toàn thông tin

Sinh viên thực hiện:

Trịnh Bá Quý	**Mã SV: AT180340**
Trịnh Văn Tráng	**Mã SV: AT180147**
Nguyễn Thiện Quý	**Mã SV: AT180440**
Đinh Thanh Quý	**Mã SV: AT180540**

Người hướng dẫn :
Giảng viên : Bùi Việt Thắng

Hà Nội, 2024

MỤC LỤC

GIỚI THIỆU ĐỀ TÀI

1. Cơ sở khoa học và tính cấp thiết của đề tài

- Ngày này hệ thống Internet ngày càng phát triển, phần mềm sử dụng hệ thống internet ngày càng nhiều. Các phần mềm đa dạng dẫn đến có rất nhiều yêu cầu cần được đáp ứng. Một số phần mềm đòi hỏi về lượng thông tin lớn, dữ liệu lớn… nhưng không thể lưu dữ liệu đó tại thiết bị sử dụng, một số loại yêu cầu được cập nhật realtime (theo thời gian thực) để đảm bảo sự đúng đắn của thông tin (chứng khoán, tiền tệ ...), một số phần mềm đòi hỏi xử lý nhanh và mạnh, mà các thiết bị lại không thể thực hiện được do cấu hình không đủ.
- Thông thường, để sử dụng các dịch vụ đó thì người dùng cần dùng trình duyệt, truy cập website và thực hiện. Nhưng người dùng chỉ có thể sử dụng các giao diện mà nhà cung cấp đã thiết kết sẵn tuy nhiên chúng không đáp ứng những mong muốn của người dùng. Để giải quyết vấn đề trên chúng ta cần xây dựng một ứng dụng có các tính năng như các dịch vụ đó nhưng giao diện thân thiện hơn. Vì vậy cần phải sử dụng những dịch vụ riêng biệt để tương tác với hệ thống cung cấp các dịch vụ nói trên. Một hệ thống như vậy được gọi là API

2. Mục tiêu và nhiệm vụ của đề tài

- Hiểu được các nguyên tắc của REST.
- Hiểu được loại dữ liệu được điều khiển bởi Tiến hành cài đặt API RESTful theo phương pháp trên các hệ thống dựa trên nền web.
- Đưa ra được phương pháp xây dựng cách thức truy cập dữ liệu sử dụng API REST.
- Tiến hành cài đặt API RESTful theo phương pháp trên.
- Cho thấy rằng API vừa cài đặt có thể dùng chung cho cả người và máy.

3. Phạm vi nghiên cứu

- Do có những hạn chế nhất định về cơ sở vật chất và điều kiện tiếp cận thực tế với lĩnh vực viễn thông nên việc cài đặt ứng dụng chủ yếu mang tính thử nghiệm.

CHƯƠNG 1: CÁC KHẢI NIỆM CƠ BẢN

1.1: API

- API (Application Programming Interface) là một phương tiện cho phép các ứng dụng khác nhau giao tiếp và trao đổi dữ liệu với nhau. API cung cấp một bộ các giao diện lập trình ứng dụng cho phép các nhà phát triển tạo ra các ứng dụng phần mềm, trang web và dịch vụ trực tuyến khác nhau.
- Các API thường được thiết kế để cung cấp các chức năng cụ thể cho ứng dụng. Ví dụ, một API của một dịch vụ trực tuyến có thể cung cấp các chức năng như đăng nhập, tải lên và tải xuống tệp tin, tìm kiếm và truy vấn dữ liệu từ cơ sở dữ liệu của dịch vụ đó.
- Các API thường được sử dụng để tạo ra các ứng dụng phức tạp bằng cách kết hợp nhiều dịch vụ và chức năng khác nhau. Ví dụ, một ứng dụng di động có thể sử dụng API của một dịch vụ định vị để hiển thị vị trí của người dùng, API của một dịch vụ xử lý thanh toán để thanh toán cho các sản phẩm và API của một dịch vụ mạng xã hội để chia sẻ nội dung trên mạng xã hội.
- Các API thường được sử dụng để cung cấp các dịch vụ và chức năng cho các ứng dụng khác nhau, giúp cho việc phát triển ứng dụng trở nên dễ dàng hơn. Thay vì phải thiết kế và xây dựng lại các tính năng từ đầu, các nhà phát triển có thể sử dụng các API có sẵn để nhanh chóng tích hợp các tính năng vào ứng dụng của mình.
- Tổng quan về API cho thấy rằng chúng là một công nghệ rất quan trọng và có ảnh hưởng đến nhiều lĩnh vực khác nhau, từ phát triển ứng dụng đến xử lý dữ liệu và tích hợp hệ thống.

1.2: REST

- REST (Representational State Transfer) là một kiến trúc dịch vụ web được sử dụng để thiết kế các API. REST được sử dụng rộng rãi để xây dựng các ứng dụng web và di động, đặc biệt là trong các ứng dụng phức tạp có nhiều tài nguyên khác nhau.
- REST được xây dựng trên các nguyên tắc cơ bản:
    - Sử dụng các phương thức HTTP (GET, POST, PUT, DELETE) để thao tác với các tài nguyên trên máy chủ: REST sử dụng các phương thức HTTP để thao tác với các tài nguyên trên máy chủ. Điều này có nghĩa là REST sử dụng các phương thức HTTP để truyền tải các yêu cầu từ máy khách đến máy chủ và trả về các phản hồi từ máy chủ đến máy khách. Các phương thức HTTP được sử dụng trong REST bao gồm GET, POST, PUT và DELETE.
    - Sử dụng các URI (Uniform Resource Identifier) để định danh các tài nguyên: REST sử dụng các URI để định danh các tài nguyên trên máy chủ. Các URI được sử dụng để chỉ định tài nguyên cụ thể mà máy khách muốn truy cập hoặc thao tác.
    - Sử dụng các định dạng dữ liệu chuẩn như JSON hoặc XML để truyền tải dữ liệu giữa máy khách và máy chủ: REST sử dụng các định dạng dữ liệu chuẩn như JSON hoặc XML để truyền tải dữ liệu giữa máy khách và máy chủ. Điều này giúp đảm bảo tính tương thích và dễ dàng tương tác giữa các ứng dụng khác nhau.
    - Không lưu trữ trạng thái trên máy chủ, mà sử dụng trạng thái của các yêu cầu để xác định trạng thái của tài nguyên: REST không lưu trữ trạng thái trên máy chủ. Thay vào đó, REST sử dụng trạng thái của các yêu cầu được gửi đến máy chủ để xác định trạng thái của tài nguyên. Điều này giúp đảm bảo tính mở rộng và dễ dàng quản lý các tài nguyên trên máy chủ.

- REST cho phép các ứng dụng web tương tác với các tài nguyên trên máy chủ một cách đơn giản và hiệu quả. Các ứng dụng có thể sử dụng các phương thức HTTP để thực hiện các yêu cầu như lấy dữ liệu, tạo mới hoặc cập nhật tài nguyên. REST cũng cho phép các ứng dụng truy cập vào các tài nguyên trên máy chủ từ bất kỳ địa điểm nào trên Internet.
- REST là một phương pháp thiết kế API được sử dụng rộng rãi trong các ứng dụng web và di động. Điều này đặc biệt hữu ích trong các ứng dụng phức tạp có nhiều tài nguyên khác nhau và các yêu cầu truy cập dữ liệu từ nhiều địa điểm khác nhau.

1.3: RESTful API

- RESTful API là một loại API được thiết kế để sử dụng kiến trúc REST, một kiến trúc phần mềm được xây dựng trên các nguyên tắc cơ bản như sử dụng các phương thức HTTP, sử dụng URI để định danh tài nguyên, sử dụng các định dạng dữ liệu chuẩn như JSON hoặc XML để truyền tải dữ liệu, và không lưu trữ trạng thái trên máy chủ.
- RESTful API thường được sử dụng để truyền tải dữ liệu giữa các ứng dụng web hoặc di động, và định nghĩa các tài nguyên và các phương thức để truy cập và thao tác với các tài nguyên đó. Các tài nguyên được định danh bằng URI, và các phương thức HTTP được sử dụng để thực hiện các hoạt động trên các tài nguyên đó. Ví dụ: GET để lấy thông tin tài nguyên, POST để thêm tài nguyên mới, PUT để cập nhật tài nguyên, và DELETE để xóa tài nguyên.
- RESTful API sử dụng các định dạng dữ liệu chuẩn như JSON hoặc XML để truyền tải dữ liệu giữa các ứng dụng khác nhau. Các định dạng này được sử dụng để đảm bảo tính tương thích và dễ dàng tương tác giữa các ứng dụng khác nhau.
- RESTful API không lưu trữ trạng thái trên máy chủ, mà sử dụng trạng thái của các yêu cầu để xác định trạng thái của tài nguyên. Điều này giúp đảm bảo tính mở rộng và dễ dàng quản lý các tài nguyên trên máy chủ.

- Tóm lại, RESTful API là một loại API được thiết kế để hoạt động theo kiến trúc REST, cung cấp các phương thức và giao thức để truyền tải và truy cập dữ liệu giữa các ứng dụng khác nhau, thường là giữa máy khách và máy chủ.

1.4: HTTP Request

- HTTP request là yêu cầu được gửi từ một máy khách đến một máy chủ thông qua giao thức HTTP (Hypertext Transfer Protocol). Yêu cầu này chứa các thông tin mà máy chủ cần để thực hiện một hoạt động trên tài nguyên được yêu cầu, ví dụ như lấy thông tin, cập nhật hoặc xóa tài nguyên.
- HTTP request bao gồm ba phần chính: Body, URI (Uniform Resource Identifier) và headers (tiêu đề).
- URI: Là định danh duy nhất của tài nguyên được yêu cầu trên máy chủ. URI bao gồm hai phần: địa chỉ của máy chủ và đường dẫn đến tài nguyên được yêu cầu.
- Headers: Là các thông tin bổ sung được gửi cùng với yêu cầu, chẳng hạn như thông tin về phiên bản HTTP được sử dụng, định dạng của dữ liệu được gửi đi, các thông tin về quyền truy cập, các thông tin chứng thực và các thông tin bảo mật.
    - Method (phương thức): định nghĩa hành động mà client muốn thực hiện, bao gồm GET, POST, PUT, DELETE,..
    - Host: tên miền hoặc địa chỉ IP của server.
    - User-Agent: thông tin về client (trình duyệt) gửi yêu cầu.
    - Accept: định nghĩa các định dạng tài nguyên mà client sẵn sàng chấp nhận nhận được từ server.
    - Authorization: cung cấp thông tin xác thực nếu yêu cầu đòi hỏi xác thực.
    - Cookie: gửi thông tin cookie đến server.
- Body chứa nội dung của yêu cầu, bao gồm các tham số và giá trị mà client muốn gửi đến server

- HTTP request được gửi từ máy khách đến máy chủ thông qua một kết nối TCP/IP (Transmission Control Protocol/Internet Protocol) và sau đó máy chủ sẽ xử lý yêu cầu đó và trả về một HTTP response (phản hồi HTTP) cho máy khách. Việc gửi và nhận HTTP request và response là cơ chế cơ bản để truyền tải dữ liệu giữa các ứng dụng trên mạng sử dụng giao thức HTTP.

1.4.1: Các phương thức phổ biến trong HTTP request:

- GET: Yêu cầu máy chủ trả về một tài nguyên được chỉ định trong URI. Phương thức này thường được sử dụng để truy cập các trang web và lấy thông tin từ các tài nguyên trên mạng.
- POST: Yêu cầu máy chủ xử lý dữ liệu được gửi kèm theo yêu cầu và trả về phản hồi tương ứng. Phương thức này thường được sử dụng để gửi dữ liệu từ một biểu mẫu trên trang web đến máy chủ.
- PUT: Yêu cầu máy chủ cập nhật một tài nguyên được chỉ định trong URI với dữ liệu được gửi kèm theo yêu cầu. Phương thức này thường được sử dụng để cập nhật các tài nguyên trên mạng.
- DELETE: Yêu cầu máy chủ xóa một tài nguyên được chỉ định trong URI. Phương thức này thường được sử dụng để xóa các tài nguyên trên mạng.

1.5: HTTP Response

HTTP response của API bao gồm một số trường header và nội dung phản hồi. Các trường header trong phản hồi API bao gồm:

- HTTP status code: Mã trạng thái HTTP là một số ba chữ số chỉ ra kết quả của yêu cầu API. Các mã trạng thái thông thường bao gồm 200 OK để chỉ ra rằng yêu cầu đã được xử lý thành công, 400 Bad Request để chỉ ra rằng yêu cầu của client không hợp lệ hoặc 404 Not Found để chỉ ra rằng tài nguyên được yêu cầu không tìm thấy trên server.
- Content-Type: Đây là trường header cho biết loại nội dung của phản hồi API, ví dụ như JSON hoặc XML

- Content-Length: Đây là trường header cho biết độ dài của nội dung phản hồi được trả về từ server.
- Cache-Control: Đây là trường header cho biết liệu các trình duyệt hoặc các bộ nhớ cache có thể lưu trữ nội dung phản hồi API hoặc không.
- Nội dung của phản hồi API là thông tin trả về từ server, thường được trả về dưới dạng một định dạng chuẩn như JSON hoặc XML. Nội dung này thường chứa dữ liệu tương ứng với yêu cầu API được gửi từ client đến server, ví dụ như thông tin tài khoản người dùng hoặc kết quả của một truy vấn cơ sở dữ liệu.
- Ngoài các trường header và nội dung phản hồi cơ bản, các phản hồi API cũng có thể chứa các thông tin bổ sung như thời gian phản hồi, mã lỗi hoặc các thông tin liên quan đến bảo mật và xác thực. Các thông tin này giúp client xử lý phản hồi API và hiển thị các kết quả hoặc thông báo lỗi tương ứng cho người dùng.

1.5.1: HTTP Status Code

- 1xx (Thông tin): Thông báo về việc server đang xử lý yêu cầu của client.
- 2xx (Thành công): Thông báo về việc server đã xử lý yêu cầu của client thành công.
- 3xx (Chuyển hướng): Thông báo về việc client cần thực hiện thêm các hành động để hoàn tất yêu cầu.
- 4xx (Lỗi từ client): Thông báo về việc yêu cầu của client không hợp lệ hoặc không thể xử lý bởi server.
- 5xx (Lỗi từ server): Thông báo về việc server gặp sự cố trong khi xử lý yêu cầu của client.
- Một số mã trạng thái HTTP phổ biến trong HTTP response của API bao gồm:

- 200 OK: Yêu cầu của client đã được xử lý thành công và server trả về phản hồi dữ liệu theo yêu cầu.
- 400 Bad Request: Yêu cầu của client không hợp lệ hoặc thiếu thông tin bắt buộc để thực hiện yêu cầu.
- 401 Unauthorized: Client không được ủy quyền truy cập tài nguyên yêu cầu.
- 403 Forbidden: Server đã nhận được yêu cầu của client, nhưng không cho phép truy cập tài nguyên yêu cầu.

- 404 Not Found: Server không tìm thấy tài nguyên được yêu cầu.
- 500 Internal Server Error: Server gặp sự cố trong khi xử lý yêu cầu của client.
- Các mã trạng thái HTTP trong phản hồi API cung cấp cho client thông tin về kết quả của yêu cầu API và giúp client xử lý các trường hợp bất thường hoặc lỗi khi sử dụng API.

1.5.2: JSON

- JSON (JavaScript Object Notation) là một định dạng dữ liệu được sử dụng để truyền tải dữ liệu giữa các ứng dụng web. JSON được sử dụng để lưu trữ và truyền tải dữ liệu dưới dạng văn bản, và có thể được sử dụng bởi nhiều ngôn ngữ lập trình khác nhau.
- JSON được xây dựng dựa trên cú pháp của JavaScript, nhưng nó là một định dạng độc lập với ngôn ngữ. Định dạng JSON sử dụng các cặp key-value, tương tự như các đối tượng trong JavaScript. Mỗi cặp key-value được đặt trong dấu ngoặc nhọn "{}", và các cặp key-value được ngăn cách nhau bởi dấu phẩy. Ví dụ:

```
{
  "name": "John",
  "age": 30,
  "city": "New York"
}
```

  Trong đó các key là "name", "age", "city", và các giá trị tương ứng lần lượt là "John", 30, "New York".
- JSON có thể lưu trữ dữ liệu khái quát, bao gồm các kiểu dữ liệu như số, chuỗi, đối tượng và mảng, và nó được sử dụng để truyền tải dữ liệu giữa máy khách và máy chủ, hoặc giữa các ứng dụng khác nhau trên mạng. JSON được sử dụng rộng rãi trong các ứng dụng web và các API (Application Programming Interface) để trao đổi dữ liệu giữa các ứng dụng khác nhau.

1.5.3: XML

- XML (Extensible Markup Language) là một định dạng dữ liệu được sử dụng để truyền tải dữ liệu giữa các ứng dụng web. XML được thiết kế để lưu trữ và

truyền tải dữ liệu dưới dạng văn bản, và có thể được sử dụng bởi nhiều ngôn ngữ lập trình khác nhau.

- XML sử dụng các thẻ để định nghĩa các phần tử trong tài liệu. Mỗi phần tử được bao quanh bởi một cặp thẻ mở và thẻ đóng, và có thể có các thuộc tính được định nghĩa bên trong thẻ mở. Ví dụ:

```
<person>
  <name>John</name>
  <age>30</age>
  <city>New York</city>
</person>
```

  Trong đó, các phần tử là <person>, <name>, <age>, <city>, và các giá trị tương ứng lần lượt là "John", "30", "New York".

- XML có thể lưu trữ dữ liệu khái quát, bao gồm các kiểu dữ liệu như số, chuỗi, đối tượng và mảng, và nó được sử dụng để truyền tải dữ liệu giữa máy khách và máy chủ, hoặc giữa các ứng dụng khác nhau trên mạng. Tuy nhiên, vì XML sử dụng cú pháp phức tạp hơn so với JSON, nên JSON được sử dụng rộng rãi hơn trong các ứng dụng web hiện đại.

CHƯƠNG 2: GIỚI THIỆU KIỂM THỬ API

2.1: Giới thiệu API sử dụng

2.1.1: Ta sẽ sử dụng API được cung cấp sẵn được gọi là Reqres

- Reqres mô phỏng các kịch bản ứng dụng thực tế. Nếu ta muốn kiểm tra hệ thống xác thực người dùng, Reqres sẽ phản hồi yêu cầu đăng nhập/đăng ký thành công bằng mã thông báo để ta xác định người dùng mẫu hoặc Forbidden Response 403 đối với nỗ lực đăng nhập/đăng ký không thành công.

- Khi tạo nguyên mẫu một giao diện mới, ta chỉ muốn có một API ở đó, với nỗ lực thiết lập tối thiểu. Thông thường, Reqres sẽ chỉ cho những người không

quá quen thuộc với lập trình phụ trợ, Sailsjs có thể tự động tạo API REST cho bạn từ dòng lệnh.

2.1.2: Giới thiệu Model sử dụng trong API

Ta sẽ có 2 class Account, User

User gồm 5 trường bao gồm id, email, first_name, last_name và avatar

User

{

Id integer

Email string

first_name string

last_name string

avatar string

}

Account

{

email string

password string

}

2.2 Giới thiệu về kiểm thử API

Kiểm thử API là quá trình kiểm tra và xác định tính đúng đắn và chất lượng của các giao diện lập trình ứng dụng (APIs). Việc kiểm thử API giúp đảm bảo rằng API hoạt động chính xác và đáp ứng được các yêu cầu của người dùng cuối.

Các bước trong kiểm thử API bao gồm:

- Thiết lập môi trường kiểm thử: phát triển một môi trường kiểm thử độc lập với các hệ thống khác nhằm tránh ảnh hưởng đến các dịch vụ khác trong hệ thống.
- Xác định các ca kiểm thử: phân tích yêu cầu hệ thống, xác định các kịch bản kiểm thử và kiểm tra các tính năng của API.
- Chuẩn bị dữ liệu kiểm thử: chuẩn bị dữ liệu để thực hiện kiểm thử, đảm bảo rằng dữ liệu đó là chính xác và đủ để kiểm tra tất cả các tính năng của API.
- Thực hiện kiểm thử: sử dụng các công cụ kiểm thử như Postman, SoapUI hoặc JMeter để thực hiện kiểm thử API.
- Phân tích kết quả kiểm thử: phân tích và đánh giá kết quả kiểm thử để xác định tính đúng đắn và chất lượng của API.
- Ghi lại kết quả kiểm thử: ghi lại các kết quả kiểm thử và tạo báo cáo để phân tích và cải tiến các quy trình kiểm thử.

Có nhiều phương pháp và công cụ để thực hiện kiểm thử API, bao gồm kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử hệ thống. Mỗi phương pháp này sẽ có những kỹ thuật và công cụ riêng để thực hiện kiểm thử API.

Tuy nhiên, kiểm thử API cũng có một số thách thức nhất định. Ví dụ, một số API có thể bị giới hạn về số lượng yêu cầu truy cập hoặc có các ràng buộc về quyền truy cập, điều này có thể gây khó khăn cho quá trình kiểm thử. Ngoài ra, việc kiểm thử API cần phải đảm bảo tính bảo mật của dữ liệu, đặc biệt là trong trường hợp API cung cấp dữ liệu nhạy cảm.

2.2.1 Khái niệm kiểm thử response của các hàm HTTP Request

Các API thường sử dụng các hàm HTTP request để trao đổi dữ liệu giữa các ứng dụng. Khi ứng dụng gửi một yêu cầu HTTP request đến một API, API sẽ phản hồi bằng một HTTP response, chứa các thông tin được yêu cầu hoặc thể hiện trạng thái của yêu cầu.

Kiểm thử response của các hàm HTTP request trong API là một phần quan trọng của kiểm thử API. Nó đảm bảo rằng các yêu cầu và phản hồi được trao đổi giữa các ứng dụng qua API đáp ứng đúng đắn, chính xác và đáp ứng được các yêu cầu của người dùng cuối.

Khi kiểm thử response của các hàm HTTP request trong API, chúng ta cần chú ý đến các yếu tố sau:

- Kiểm tra trạng thái HTTP response: phải kiểm tra xem trạng thái của HTTP response có đúng như kỳ vọng hay không. Ví dụ, nếu chúng ta gửi một yêu cầu GET để lấy dữ liệu từ API, thì trạng thái HTTP response phải là 200 OK nếu dữ liệu được trả về thành công.
- Kiểm tra định dạng dữ liệu: phải kiểm tra định dạng của dữ liệu được trả về từ API, xem nó có đúng định dạng mà chúng ta mong đợi hay không.
- Kiểm tra dữ liệu trả về: phải kiểm tra dữ liệu trả về từ API, đảm bảo rằng nó chứa các thông tin mà chúng ta mong đợi và không có bất kỳ lỗi nào.
- Kiểm tra thời gian phản hồi: phải kiểm tra thời gian phản hồi của API, đảm bảo rằng nó không quá chậm và không gây ra sự cố giao tiếp.
- Để thực hiện kiểm thử response của các hàm HTTP request trong API, chúng ta có thể sử dụng các công cụ như Postman, SoapUI hoặc JMeter. Những công cụ này cung cấp một giao diện đồ họa cho phép chúng ta tạo và gửi các yêu cầu HTTP request đến API, và phân tích các phản hồi từ các yêu cầu đó. Chúng ta cũng có thể viết các kịch bản kiểm thử tự động để đảm bảo tính liên tục trong kiểm thử.

Ngoài ra, để đảm bảo tính toàn vẹn và độ chính xác của kiểm thử response của các hàm HTTP request trong API, chúng ta cần lưu ý các điểm sau:

- Đảm bảo rằng API đang hoạt động đúng và các endpoint đang trả về dữ liệu chính xác.
- Đảm bảo rằng các thông số đầu vào (request parameters) đang được gửi đúng và đủ để đáp ứng yêu cầu của người dùng.

- Đảm bảo rằng dữ liệu được truyền đi và trả về đúng định dạng và đúng cấu trúc.
- Kiểm tra các lỗi xảy ra trong quá trình giao tiếp, bao gồm các lỗi liên quan đến mạng và các lỗi phía server.
- Đảm bảo rằng các thông tin bảo mật như mã thông báo truy cập (access tokens) hoặc thông tin đăng nhập (login credentials) đang được truyền đi một cách an toàn và đảm bảo tính bảo mật.
- Cuối cùng, kiểm thử response của các hàm HTTP request trong API là một phần quan trọng trong việc đảm bảo chất lượng của ứng dụng phần mềm. Nó giúp đảm bảo rằng API đang hoạt động đúng và dữ liệu truyền đi và trả về đúng đắn, giúp cải thiện trải nghiệm người dùng và đảm bảo tính ổn định và đáng tin cậy của ứng dụng.

2.2.2: Các case để xác minh API endpoint có đúng và truy cập được

- Để xác minh rằng một API endpoint có đúng và có thể truy cập được hay không, chúng ta có thể sử dụng một số case sau:
- Kiểm tra endpoint có tồn tại hay không: Sử dụng phương thức GET để kiểm tra xem endpoint có tồn tại hay không bằng cách gửi một request đến endpoint đó. Nếu endpoint không tồn tại, API sẽ trả về một mã lỗi hoặc một thông báo lỗi.
- Kiểm tra response status code: Trong các phương thức HTTP, mã status code được sử dụng để xác định kết quả của một request. Một endpoint đúng sẽ trả về một mã status code hợp lệ như 200 OK hoặc 201 Created.
- Kiểm tra response data: Kiểm tra dữ liệu trả về từ endpoint có đúng và đầy đủ như mong đợi không. Các thông tin trả về có phù hợp với những gì được định nghĩa trong tài liệu API không.
- Kiểm tra authentication: Đối với những endpoint yêu cầu authentication, chúng ta cần kiểm tra rằng request đã được xác thực đúng cách hay không.

Các access token hoặc login credentials cần được cung cấp và xác thực trước khi truy cập vào endpoint.

- Kiểm tra permission: Đối với những endpoint có yêu cầu permission để truy cập, chúng ta cần kiểm tra rằng các permission đó được cấp phép đúng cho tài khoản người dùng đang được sử dụng để gửi request.
- Kiểm tra performance: Đối với những endpoint có yêu cầu xử lý dữ liệu lớn hoặc có tải lượng request lớn, chúng ta cần kiểm tra rằng API có thể xử lý được tải lượng đó và trả về response trong thời gian hợp lý.

2.2.3: Các case để xác minh API endpoint có trả về đúng format

- Để xác minh rằng một API endpoint có trả về đúng định dạng (format) như mong đợi, chúng ta có thể sử dụng một số case sau:
- Kiểm tra response headers: Một trong những cách đơn giản nhất để xác định định dạng của response là kiểm tra response headers. Header "Content-Type" có thể cho chúng ta biết định dạng của response. Ví dụ, header "Content-Type: application/json" có nghĩa là response là định dạng JSON.
- Kiểm tra response body: Xác minh định dạng của response bằng cách kiểm tra body của response. Đối với định dạng JSON, chúng ta có thể kiểm tra định dạng của JSON bằng cách sử dụng một JSON parser và xác nhận rằng response có đúng định dạng JSON hay không.
- Kiểm tra response schema: Kiểm tra định dạng của response bằng cách so sánh response với một schema được định nghĩa trước. Ví dụ, nếu response trả về phải có định dạng JSON và theo một cấu trúc nhất định, chúng ta có thể sử dụng một JSON schema để kiểm tra response có đúng định dạng hay không.
- Kiểm tra response code: Một số định dạng response như JSON hoặc XML có thể có một mã lỗi hoặc thông báo lỗi được trả về trong trường hợp response không hợp lệ. Kiểm tra mã lỗi có phù hợp với định dạng của response không.

- Kiểm tra version: Đối với các API có nhiều phiên bản, chúng ta cần kiểm tra response có phù hợp với phiên bản của API mà chúng ta đang sử dụng hay không.

2.2.4: Các case để xác minh có tạo được user không

- Để xác minh rằng một API có tạo được user hay không, chúng ta có thể sử dụng một số case sau:
- Kiểm tra response status code: Khi tạo một user mới, nếu API trả về status code 201 (Created) hoặc 200 (OK) thì có nghĩa là user đã được tạo thành công. Nếu trả về status code 4xx hoặc 5xx, thì user chưa được tạo.
- Kiểm tra response body: Trong body của response, API có thể trả về thông tin về user đã được tạo, bao gồm ID của user hoặc các thông tin khác về user. Chúng ta có thể kiểm tra thông tin này để đảm bảo rằng user đã được tạo thành công.
- Kiểm tra cơ sở dữ liệu: Sau khi tạo user mới, chúng ta có thể kiểm tra cơ sở dữ liệu để đảm bảo rằng user đã được thêm vào cơ sở dữ liệu. Nếu user không tồn tại trong cơ sở dữ liệu, có thể xảy ra lỗi khi tạo user, hoặc API không thêm user vào cơ sở dữ liệu.
- Kiểm tra thông tin nhập vào: Nếu thông tin nhập vào để tạo user không hợp lệ, API sẽ trả về status code 4xx hoặc 5xx và không tạo user. Chúng ta có thể kiểm tra các thông tin như email, tên đăng nhập, mật khẩu,... để đảm bảo rằng chúng là hợp lệ.
- Kiểm tra tài khoản đã tồn tại: Nếu tài khoản đã tồn tại trước đó, API sẽ không tạo user mới và trả về status code 4xx hoặc 5xx. Chúng ta có thể kiểm tra tài khoản đã tồn tại trong cơ sở dữ liệu trước khi tạo user mới.

2.2.5: Các case để xác minh API endpoint có trả về đúng headers không

- Để xác minh rằng một API endpoint có trả về đúng headers hay không, chúng ta có thể sử dụng một số case sau:

- Kiểm tra response headers: Chúng ta có thể kiểm tra response headers trả về từ API endpoint để xác định xem chúng có đúng với những headers được mong đợi hay không. Các headers thường được sử dụng như Content-Type, Cache-Control, Authorization, Accept-Encoding, User-Agent, v.v.
- Kiểm tra giá trị của headers: Nếu API endpoint có trả về headers, chúng ta cần kiểm tra xem giá trị của các headers đó có đúng hay không. Ví dụ, nếu API endpoint trả về header Content-Type, chúng ta cần kiểm tra xem giá trị của header đó có đúng với định dạng dữ liệu trả về từ API hay không.
- Kiểm tra việc bổ sung headers: API endpoint cũng có thể yêu cầu việc bổ sung headers từ client để xác thực, mã hóa hoặc xác định kiểu dữ liệu truyền vào. Chúng ta có thể kiểm tra xem các headers yêu cầu này đã được bổ sung vào request hay không.
- Kiểm tra các thư viện HTTP client: Khi sử dụng các thư viện HTTP client như Axios, Fetch hoặc Request, chúng ta có thể kiểm tra xem chúng đã tự động thêm các headers cần thiết vào request hay không.

2.3: Các thư viện sử dụng

Để kiểm thử API, ta dùng python và các thư viện như: **request, pytest, pytest-html, sqlmap , tkinker**

- Requests: là một thư viện HTTP client được phát triển cho ngôn ngữ lập trình Python. Nó cho phép các lập trình viên gửi các yêu cầu HTTP như GET, POST, PUT, DELETE, v.v. và xử lý các phản hồi HTTP trả về. Thư viện này rất đơn giản và dễ sử dụng.
- Pytest: là một framework kiểm thử cho ngôn ngữ Python. Nó giúp đơn giản hóa quá trình kiểm thử, cho phép lập trình viên viết các ca kiểm thử bằng cách sử dụng các hàm assert, và tự động phát hiện và chạy các ca kiểm thử trong một thư mục chứa các tệp tin kiểm thử.

- Pytest-html: là một plugin của Pytest, nó cung cấp cho người dùng một giao diện web đơn giản để xem kết quả kiểm thử dưới dạng trang HTML. Nó cho phép các lập trình viên theo dõi các kết quả kiểm thử một cách trực quan và tiện lợi.
- Sqlmap: là một công cụ kiểm thử bảo mật cho các ứng dụng web. Nó được sử dụng để phát hiện và khai thác các lỗ hổng SQL Injection trong các ứng dụng web. Sqlmap cho phép người dùng thực hiện các cuộc tấn công kiểm thử SQL Injection trên các ứng dụng web một cách tự động và hiệu quả.

2.4: Kiểm thử về Bảo mật

- Ở đây ta sẽ kiểm thử SQL Injection và DOS

- SQL Injection: SQL Injection là một kỹ thuật tấn công bảo mật thường được sử dụng để tìm ra lỗ hổng trong hệ thống quản lý cơ sở dữ liệu của ứng dụng web. Kỹ thuật này bao gồm việc chèn các câu lệnh SQL độc hại vào các trường nhập liệu trong ứng dụng web, dẫn đến việc truy cập trái phép vào cơ sở dữ liệu và lấy thông tin nhạy cảm.
- Trong kiểm thử SQL Injection, các ca kiểm thử được thiết kế để tìm kiếm các lỗ hổng SQL Injection bằng cách nhập các giá trị đầu vào độc hại vào các trường nhập liệu và kiểm tra xem ứng dụng web có bị tấn công hay không. Các công cụ phát hiện lỗ hổng SQL Injection như SQLMap và OWASP ZAP có thể được sử dụng để tìm kiếm lỗ hổng trong các ứng dụng web.
- DOS (Denial-of-Service): DOS là một kỹ thuật tấn công bảo mật có thể gây ra sự cố cho ứng dụng web bằng cách tạo ra lưu lượng truy cập lớn đến máy chủ web, làm cho nó không thể xử lý các yêu cầu từ các người dùng thật sự. Tấn công DOS có thể gây ra sự cố tạm thời hoặc vĩnh viễn và dẫn đến sự gián đoạn dịch vụ cho người dùng.
- Trong kiểm thử DOS, các ca kiểm thử được thiết kế để tạo ra lưu lượng truy cập lớn đến máy chủ web, đánh giá khả năng chịu tải của ứng dụng web và

xác định điểm yếu của nó. Các công cụ kiểm thử DOS như LOIC (Low Orbit Ion Cannon) và HOIC (High Orbit Ion Cannon) có thể được sử dụng để tạo ra lưu lượng truy cập giả tạo và đánh giá khả năng chịu tải của máy chủ web. Tuy nhiên, việc sử dụng các công cụ này phải được thực hiện cẩn thận để tránh gây ra sự cố cho các ứng dụng web khác trên cùng máy chủ hoặc trên cùng mạng.

CHƯƠNG 3: XÂY DỰNG CHƯƠNG TRÌNH

3.1: Các thành phần sử dụng

Ngôn ngữ sử dụng : Python

Các thư viện sử dụng trong chương trình

- requests : Tương tác với API
- pytest : Để kiểm thử các test case
- pytest-html : Hỗ trợ xây dựng trang báo cáo
- sqlmap : kiểm tra lỗ hông sql injection
- tkinter : làm UI

API : https://reqres.in/api/users

3.2: Kiểm thử API

3.2.1: Kịch bản kiểm thử

- Xác định các yêu cầu chức năng của API cần được kiểm thử.
- Xác định các yêu cầu phi chức năng như độ tin cậy, hiệu suất, độ bảo mật và các yêu cầu khác.
- Thực hiện kiểm thử API bằng cách sử dụng các công cụ kiểm thử API và kiểm tra các kết quả.
- Phân tích kết quả và ghi nhận lại các lỗi được tìm thấy.

- Chạy kiểm thử lại để xác nhận các lỗi được sửa chữa.
- Xây dựng báo cáo

3.2.2: Thực hiện viết chương trình

3.2.2.1: Kiểm thử trên về các chức năng của API :

- Lấy thông tin user
- Tạo user
- Xóa User
- Cập nhập User
- Đăng Nhập
- Đăng kí

Kiểm thử bảo mật hiệu suất , bảo mật :

- Kiểm tra SQL_injection trên API
- Kiểm tra thời gian trả về
- Kiểm tra tấn công DOS
-

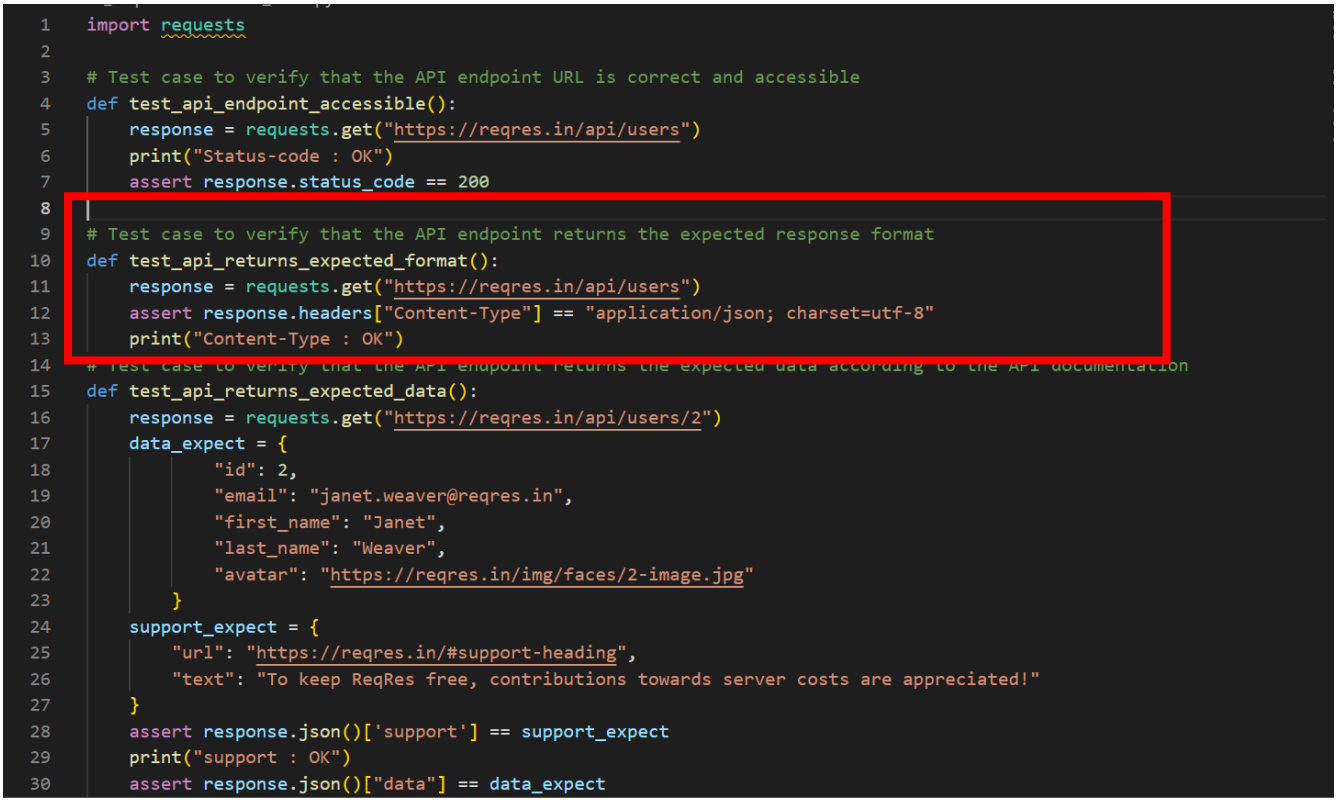
Với chức năng API thì ta có các case test :

- Xác thực kết nối đến url

```python
import requests

# Test case to verify that the API endpoint URL is correct and accessible
def test_api_endpoint_accessible():
    response = requests.get("https://reqres.in/api/users")
    print("Status-code : OK")
    assert response.status_code == 200

# Test case to verify that the API endpoint returns the expected response format
def test_api_returns_expected_format():
    response = requests.get("https://reqres.in/api/users")
    assert response.headers["Content-Type"] == "application/json; charset=utf-8"
    print("Content-Type : OK")
# Test case to verify that the API endpoint returns the expected data according to the API documentation
def test_api_returns_expected_data():
    response = requests.get("https://reqres.in/api/users/2")
    data_expect = {
            "id": 2,
            "email": "janet.weaver@reqres.in",
            "first_name": "Janet",
            "last_name": "Weaver",
            "avatar": "https://reqres.in/img/faces/2-image.jpg"
        }
    support_expect = {
        "url": "https://reqres.in/#support-heading",
        "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
    }
    assert response.json()['support'] == support_expect
    print("support : OK")
    assert response.json()["data"] == data_expect
```

Hàm test_api_end_poind() sẽ gửi request GET đến API sau đó nhận về response trong trường hợp status không bằng 200 thì sẽ bị phát hiện đánh dấu là failed

- Định dữ liệu trả về



```python
import requests

# Test case to verify that the API endpoint URL is correct and accessible
def test_api_endpoint_accessible():
    response = requests.get("https://reqres.in/api/users")
    print("Status-code : OK")
    assert response.status_code == 200

# Test case to verify that the API endpoint returns the expected response format
def test_api_returns_expected_format():
    response = requests.get("https://reqres.in/api/users")
    assert response.headers["Content-Type"] == "application/json; charset=utf-8"
    print("Content-Type : OK")
# Test case to verify that the API endpoint returns the expected data according to the API documentation
def test_api_returns_expected_data():
    response = requests.get("https://reqres.in/api/users/2")
    data_expect = {
            "id": 2,
            "email": "janet.weaver@reqres.in",
            "first_name": "Janet",
            "last_name": "Weaver",
            "avatar": "https://reqres.in/img/faces/2-image.jpg"
        }
    support_expect = {
        "url": "https://reqres.in/#support-heading",
        "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
    }
    assert response.json()['support'] == support_expect
    print("support : OK")
    assert response.json()["data"] == data_expect
```

Có nhiều định dạng 2 loại định dạng trả về phổ biến xml , json trong API đã chọn tì định dạng trả về là json và dạng mã hoá là chuẩn utf-8

- Dữ liệu trả về mong đợi

```python
# Test case to verify that the API endpoint returns the expected data according to the API documentation
def test_api_returns_expected_data():
    response = requests.get("https://reqres.in/api/users/2")
    data_expect = {
            "id": 2,
            "email": "janet.weaver@reqres.in",
            "first_name": "Janet",
            "last_name": "Weaver",
            "avatar": "https://reqres.in/img/faces/2-image.jpg"
        }
    support_expect = {
        "url": "https://reqres.in/#support-heading",
        "text": "To keep ReqRes free, contributions towards server costs are appreciated!"
    }
    assert response.json()['support'] == support_expect
    print("support : OK")
    assert response.json()["data"] == data_expect
    print("data : OK")

# Test case to verify that the API endpoint returns the correct HTTP status code
def test_api_returns_correct_status_code_when_user_single_not_found():
    response = requests.get("https://reqres.in/api/users/23")
    assert response.status_code == 404
    print("Error-status : OK")

# Test case to verify that the API endpoint returns the correct headers
def test_api_returns_correct_headers():
    response = requests.get("https://reqres.in/api/users")
    assert response.headers["Cache-Control"] == "max-age=14400"
```

Ở hàm test_api_returns_expected_data():

Ta sẽ định nghĩa trc data của response sau đó sẽ gửi một request có response ứng với data đó theo dữ liệu trên database . Nếu trả về đúng thì sẽ pass qua case này

- Xử lí lỗi

```
52      assert response.headers["Connection"] == "keep-alive"
53      print("Connection : OK")
54  # Test case to verify that the API endpoint handles errors and returns appropriate error responses for invalid reques
55  def test_api_handles_errors():
56      response = requests.get("https://reqres.in/api/users?page=99")
57      assert response.status_code == 200
58      assert response.json()["data"] == []
59      print("Status-code : OK")
60
61  # Test case to verify that the API endpoint returns the correct data for different query parameters
62  def test_api_returns_correct_data_for_query_params_page():
63      response = requests.get("https://reqres.in/api/users?page=2")
64      assert response.json()["page"] == 2
65      print("page : OK")
66      assert len(response.json()["data"]) == response.json()["per_page"]
67      print("per_page : OK")
68
69
70  # Test case to verify that the API endpoint returns data in a reasonable amount of time and does not time out
71  def test_api_returns_data_in_reasonable_time():
72      response = requests.get("https://reqres.in/api/users?delay=3")
73      assert response.elapsed.total_seconds() < 5  # Verify that the request took less than 5 seconds
74      print("total_time : OK")
75
```

Hàm xử lí lỗi khi gửi những param sai trong đoạn code trên là page=99 thì sẽ không bị lỗi 404 mà vẫn sẽ trả về response nhưng không có data

- Kiểm tra header

```
66  def test_api_returns_correct_headers():
67      data = {
68          "email": "eve.holt@reqres.in",
69          "password": "pistol"
70      }
71      response = requests.post("https://reqres.in/api/login", data=data)
72      assert response.headers.get("Server") == "cloudflare"
73      print("server : OK")
74      assert response.headers.get("X-Powered-By") == "Express"
75      print("X-Powered-By : OK")
76      assert response.headers.get("Access-Control-Allow-Origin") == "*"
77      print("Access-Control-Allow-Origin : :")
78      assert response.headers.get("Connection") == "keep-alive"
79      print("Connect : OK")
80      assert response.headers.get("CF-Cache-Status") == "DYNAMIC"
81      print("CF-Cache-Status : OK")
82
83
84
```

Server: Thuộc tính này chỉ ra tên của máy chủ web (server) đang xử lý yêu cầu của client. Nó được sử dụng để định danh và xác định server để các client có thể giao tiếp với nó.

X-Powered-By: Thuộc tính này cho biết công nghệ, ngôn ngữ lập trình hoặc các thành phần cụ thể được sử dụng để xây dựng server. Nó giúp cho các hacker có thể dễ dàng xác định các lỗ hổng bảo mật của server.

Access-Control-Allow-Origin: Thuộc tính này cho phép các tài nguyên (như font, script, hoặc API) trên server có thể được truy cập từ các domain khác. Nếu thuộc tính này được đặt là "*", thì bất kỳ domain nào đều có thể truy cập vào tài nguyên đó. Điều này có thể gây ra một số vấn đề bảo mật nếu không được cấu hình đúng cách.

CF-Cache-Status: Thuộc tính này được sử dụng trong dịch vụ CDN (Content Delivery Network) của Cloudflare để xác định trạng thái của bộ nhớ cache khi phản

hồi được trả về từ server. Nó có thể có các giá trị "hit" (được lấy từ cache), "miss" (không có trong cache và phải lấy từ server), "expired" (bản sao trong cache đã hết hạn) hoặc "revalidated" (bản sao trong cache đã được kiểm tra và xác nhận là còn hợp lệ).

- Dữ liệu trả về trong trường hợp sai

```
print("data : OK")

# Test case to verify that the API endpoint returns the correct HTTP status code
def test_api_returns_correct_status_code_when_user_single_not_found():
    response = requests.get("https://reqres.in/api/users/23")
    assert response.status_code == 404
    print("Error-status : OK")
```

Khi request gửi đi mà không phù hợp với dữ liệu thực tế trong db thì response trả về sẽ có status_code=404

- Dữ liệu có đúng query param

```
52      assert response.headers["Connection"] == "keep-alive"
53      print("Connection : OK")
54  # Test case to verify that the API endpoint handles errors and returns appropriate error responses for invalid reques
55  def test_api_handles_errors():
56      response = requests.get("https://reqres.in/api/users?page=99")
57      assert response.status_code == 200
58      assert response.json()["data"] == []
59      print("Status-code : OK")
60
61  # Test case to verify that the API endpoint returns the correct data for different query parameters
62  def test_api_returns_correct_data_for_query_params_page():
63      response = requests.get("https://reqres.in/api/users?page=2")
64      assert response.json()["page"] == 2
65      print("page : OK")
66      assert len(response.json()["data"]) == response.json()["per_page"]
67      print("per_page : OK")
68
69
70  # Test case to verify that the API endpoint returns data in a reasonable amount of time and does not time out
71  def test_api_returns_data_in_reasonable_time():
72      response = requests.get("https://reqres.in/api/users?delay=3")
73      assert response.elapsed.total_seconds() < 5  # Verify that the request took less than 5 seconds
74      print("total_time : OK")
75
```

Kiểm tra xem phần param có được xử lí đúng ta gửi request với param=2 sau đó check lại trong response trả về xem có đúng là 2 nếu đúng thì sẽ pass

- Thời gian trả về

```
52      assert response.headers["Connection"] == "keep-alive"
53      print("Connection : OK")
54  # Test case to verify that the API endpoint handles errors and returns appropriate error responses for invalid reques
55  def test_api_handles_errors():
56      response = requests.get("https://reqres.in/api/users?page=99")
57      assert response.status_code == 200
58      assert response.json()["data"] == []
59      print("Status-code : OK")
60
61  # Test case to verify that the API endpoint returns the correct data for different query parameters
62  def test_api_returns_correct_data_for_query_params_page():
63      response = requests.get("https://reqres.in/api/users?page=2")
64      assert response.json()["page"] == 2
65      print("page : OK")
66      assert len(response.json()["data"]) == response.json()["per_page"]
67      print("per_page : OK")
68
69
70  # Test case to verify that the API endpoint returns data in a reasonable amount of time and does not time out
71  def test_api_returns_data_in_reasonable_time():
72      response = requests.get("https://reqres.in/api/users?delay=3")
73      assert response.elapsed.total_seconds() < 5  # Verify that the request took less than 5 seconds
74      print("total_time : OK")
75
```

Ta lấy được tổng thời gian trả về ít hơn 5s thì sẽ đảm bảo được hiệu suất api

Các hàm POST,PUT,DELETE ta cũng sẽ viết các test case để khớp với response mong muốn.

3.2.2.2: Phần phần bảo mật

- Có 2 dạng tấn công dễ xảy ra với API
    - SQL_Injection
    - Dos

Dưới đây là 2 testcase kiểm tra về vấn đề an toàn của API

- Sql_Injection

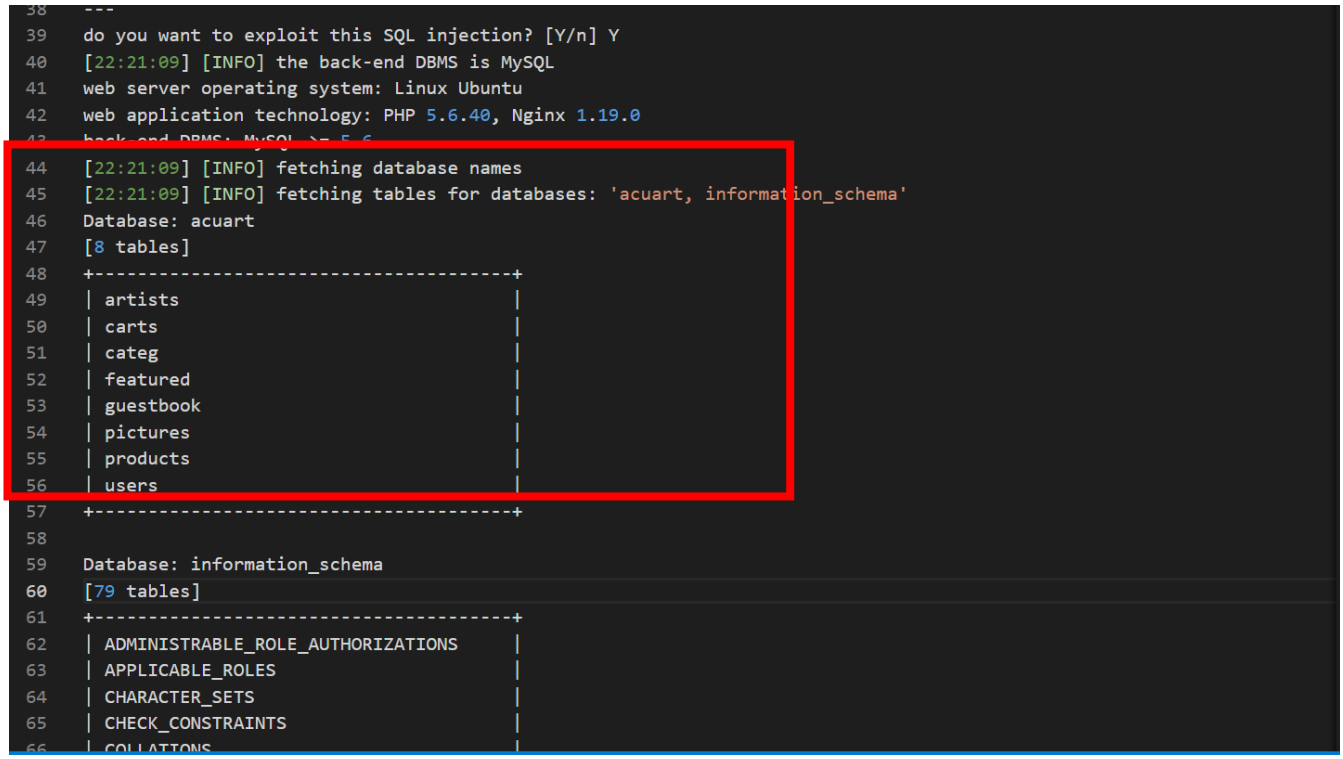
```python
import os
import subprocess

# Mở file log để ghi kết quả vào
def test_sql_injection_db():
    # Xác định đường dẫn đến thư mục logs
    log_folder = "/home/nv/Documents/study/KTLT_final/logs"
    # Tạo thư mục logs nếu chưa tồn tại
    os.makedirs(log_folder, exist_ok=True)
    log_file_path = os.path.join(log_folder, f'sqlmap.log')
    with open(log_file_path, 'w') as log_file:
        # Thực hiện kiểm thử SQL Injection bằng sqlmap và ghi log
        result = subprocess.run(
            ['sqlmap', '-u', 'https://reqres.in/api/users?id=2', '--dbs', "--batch"],
            capture_output=True, text=True)
        log_file.write(result.stdout)
        assert "the back-end DBMS is" not in result.stdout


```

```python
def test_sql_injection_tables():
    # Xác định đường dẫn đến thư mục logs
    log_folder = "/home/nv/Documents/study/KTLT_final/logs"
    # Tạo thư mục logs nếu chưa tồn tại
    os.makedirs(log_folder, exist_ok=True)
    log_file_path = os.path.join(log_folder, f'sqlmap.log')
    with open(log_file_path, 'w') as log_file:
        # Thực hiện kiểm thử SQL Injection bằng sqlmap và ghi log
        result = subprocess.run(
            ['sqlmap', '-u', 'http://testphp.vulnweb.com/listproducts.php?cat=1', '--tables', "--batch"],
            capture_output=True, text=True)
        log_file.write(result.stdout)
        assert "fetching tables for databases: " not in result.stdout
```

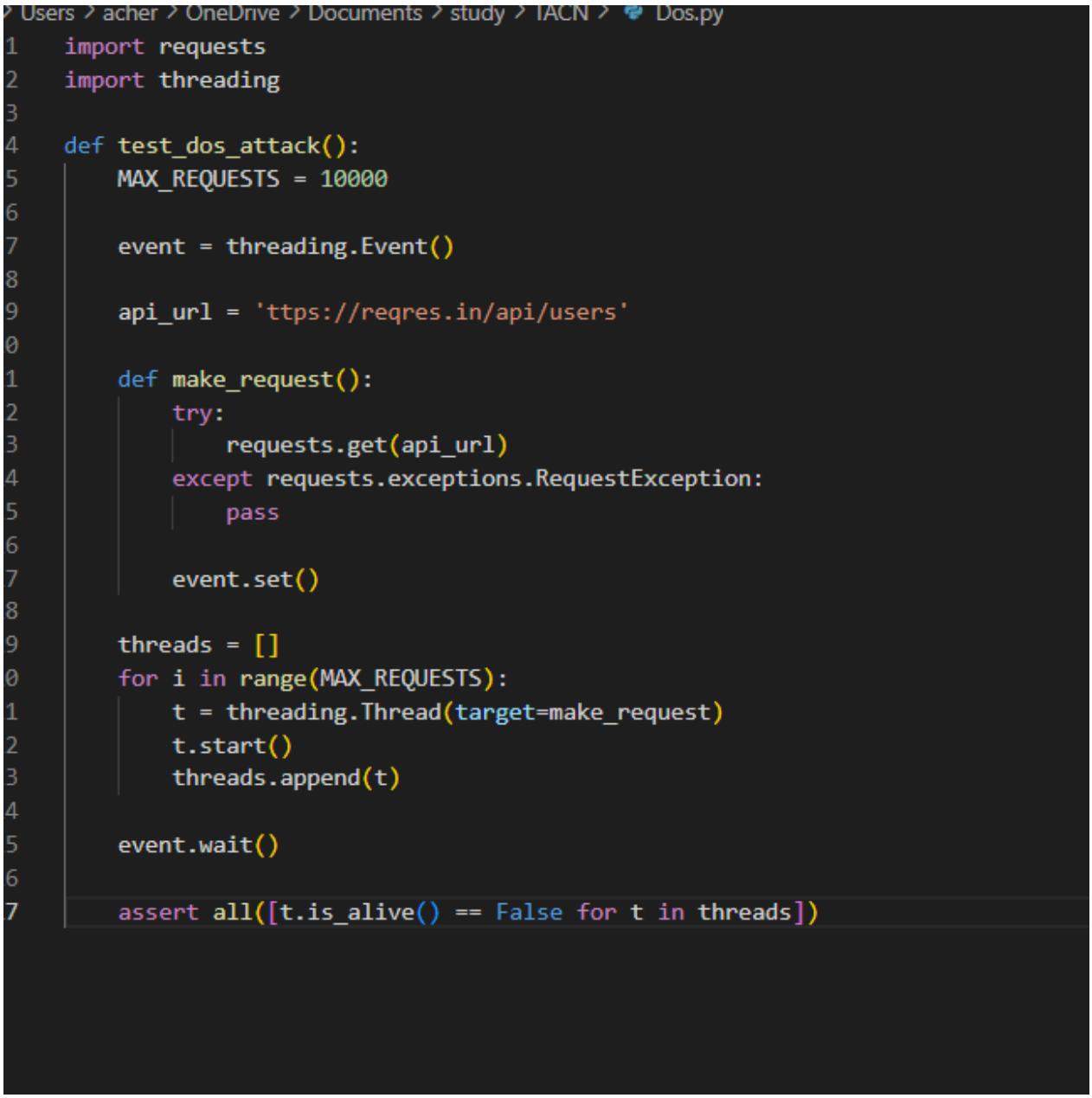
Đoạn code sử dụng sqlmap chạy câu lệnh để quét lấy tên các database

từ đấy chúng ta có thể thu được thông tin db sau đó là tên bảng -> dữ liệu trong các bảng

Nếu có thể tấn công vào bằng sqlmap thì log sẽ đc thu ở trong folder logs



- **DOS :**



Chúng ta đã sử dụng một danh sách các luồng để gửi yêu cầu tới API, và sử dụng một đối tượng Event để theo dõi kết thúc của tất cả các luồng. Sau khi tất cả các yêu cầu đã được gửi đi, chúng ta đã chờ đợi tất cả các luồng kết thúc và kiểm

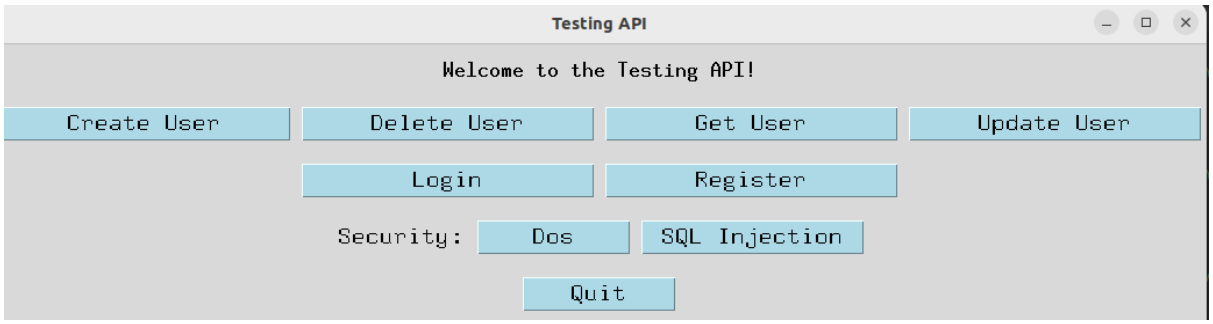
tra xem tất cả các yêu cầu đã được xử lý thành công hay không. Nếu bất kỳ một yêu cầu nào còn đang chạy sau khi tất cả các luồng đã kết thúc, thì điều đó có thể cho thấy API của không ổn định và có thể bị DOS.

3.2.2.3: Xây dựng UI trực quan

Về phần UI xây dựng bằng thư viện tkinter (Một thư viện UI trong python)

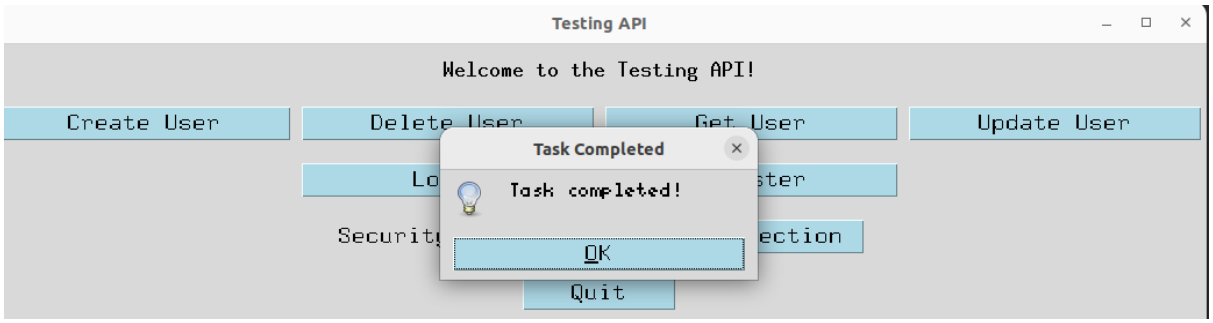
```python
from tkinter import *
from tkinter import messagebox
import os
from tkinter import ttk
def create_user():
    os.system("pytest src/check_response/Create_user.py --html=report/$(date +%d.%m.%Y)/Create_user.html")
    messagebox.showinfo("Task Completed", "Task completed!")

def delete_user():
    os.system("pytest src/check_response/Delete_user.py --html=report/$(date +%d.%m.%Y)/Delete_user.html")
    messagebox.showinfo("Task Completed", "Task completed!")

def get_user():
    os.system("pytest src/check_response/Get_users.py --html=report/$(date +%d.%m.%Y)/Get_user.html")
    messagebox.showinfo("Task Completed", "Task completed!")
def update_user():
    os.system("pytest src/check_response/Update_user.py --html=report/$(date +%d.%m.%Y)/Update_user.html")
    messagebox.showinfo("Task Completed", "Task completed!")

def login():
    os.system("pytest src/check_response/Login.py --html=report/$(date +%d.%m.%Y)/Login.html")
    messagebox.showinfo("Task Completed", "Task completed!")

def register():
    os.system("pytest src/check_response/Register.py --html=report/$(date +%d.%m.%Y)/Register.html")
    messagebox.showinfo("Task Completed", "Task completed!")

def dos():
    os.system("pytest src/security/Dos.py --html=report/$(date +%d.%m.%Y)/DOS.html")
    messagebox.showinfo("Task Completed", "Task completed!")

def sql_injection():
    os.system("pytest src/security/SQL_Injection.py --html=report/$(date +%d.%m.%Y)/SQL_Injection.html")
    messagebox.showinfo("Task Completed", "Task completed!")

def all():
    os.system("pytest src/check_response/Create_user.py --html=report/$(date +%d.%m.%Y)/Create_user.html")
    os.system("pytest src/check_response/Delete_user.py --html=report/$(date +%d.%m.%Y)/Delete_user.html")
    os.system("pytest src/check_response/Get_users.py --html=report/$(date +%d.%m.%Y)/Get_user.html")
    os.system("pytest src/check_response/Update_user.py --html=report/$(date +%d.%m.%Y)/Update_user.html")
    os.system("pytest src/check_response/Login.py --html=report/$(date +%d.%m.%Y)/Login.html")
    os.system("pytest src/check_response/Register.py --html=report/$(date +%d.%m.%Y)/Register.html")
    os.system("pytest src/security/Dos.py --html=report/$(date +%d.%m.%Y)/DOS.html")
```

```
    messagebox.showinfo("All Task Completed", "All Task completed!")
root = Tk()
root.title("Testing API")

root.option_add("*Font", "Arial")

style = ttk.Style()
style.configure('TButton', font=('Arial', 14), foreground="black", background="light blue", width=20)

label = Label(root, text="Welcome to the Testing API!", font=("Arial", 16))
label.pack(pady=10)

frame1 = Frame(root)
frame1.pack(pady=10)

button1 = ttk.Button(frame1, text="Create User", width=20,command=create_user)
button1.pack(side=LEFT, padx=5)

button2 = ttk.Button(frame1, text="Delete User", width=20,command=delete_user)
button2.pack(side=LEFT, padx=5)

button3 = ttk.Button(frame1, text="Get User", width=20,command=get_user)
button3.pack(side=LEFT, padx=5)

button4 = ttk.Button(frame1, text="Update User", width=20,command=update_user)
button4.pack(side=LEFT, padx=5)

frame2 = Frame(root)
frame2.pack(pady=10)

button5 = ttk.Button(frame2, text="Login", width=20,command=login)
button5.pack(side=LEFT, padx=5)

button6 = ttk.Button(frame2, text="Register", width=20,command=register)
button6.pack(side=LEFT, padx=5)

frame3 = Frame(root)
frame3.pack(pady=10)

label2 = Label(frame3, text="Security:", font=("Arial", 14))
label2.pack(side=LEFT, padx=5)
```



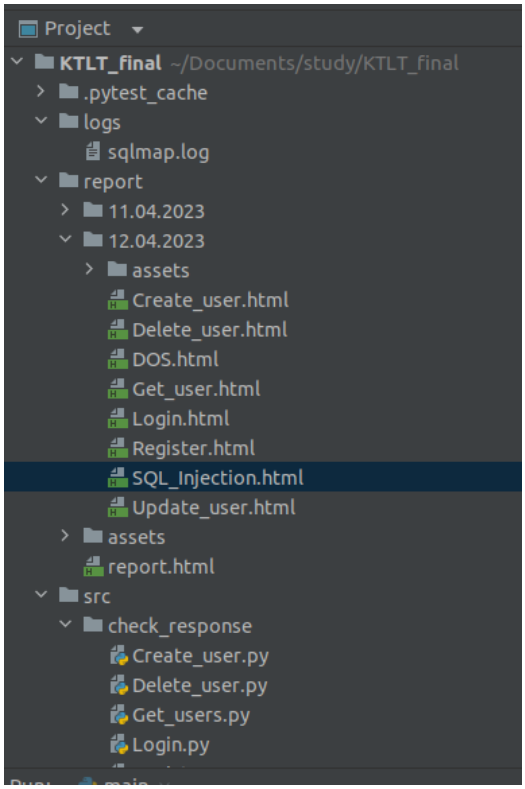
Mỗi button tương ứng với chức năng ta muốn kiểm thử

Sau khi chạy xong sẽ có thông báo



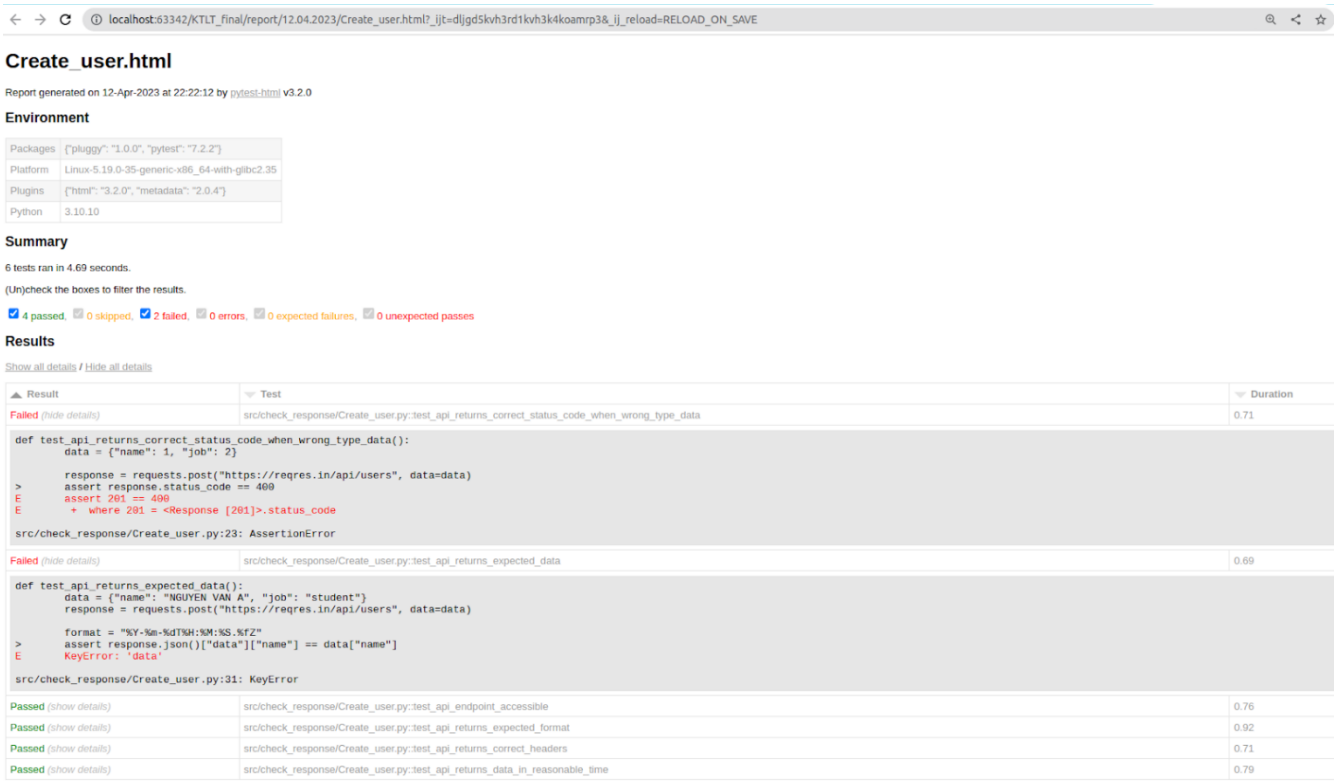
3.2.2.4 : Chức năng báo cáo lưu trữ

Sau khi ấn kiểm thử chức năng sẽ trong folder report



Output của các phần sẽ được lưu trữ theo ngày và dưới định dạng .html ứng với tên của chức năng như hình vẽ

Dưới đây là báo cáo sau khi kiểm thử chức năng create_user



Với mỗi case test ta có 4 trạng thái chính

- pass : thành công vượt qua được test case
- skipped : bị bỏ qua
- failed : thất bại
- error : code bị lỗi

Ngay tại mỗi case chúng ta sẽ có thể nhìn được output của hàm