|  |
| --- |
| BAN CƠ YẾU CHÍNH PHỦ  **HỌC VIỆN KỸ THUẬT MẬT MÃ**  KHOA AN TOÀN THÔNG TIN    **BÁO CÁO BÀI TẬP LỚN**  **XÂY DỰNG CÔNG CỤ KIỂM THỬ API**  Ngành: An toàn thông tin  *Sinh viên thực hiện:*  **Nguyễn Bảo Ngọc** AT180436  **Cao Trung Du** AT180410  **Trần Minh Kiên** AT180426  **Khuất Hoàng Long** AT180429  *Giảng viên hướng dẫn:*  **TS. Nguyễn Mạnh Thắng**  Khoa An toàn thông tin – Học viện Kỹ thuật mật mã  Hà Nội, 2024 |

**MỤC LỤC**

|  |
| --- |
| **LỜI MỞ ĐẦU……………………………………………………………………..** |
| **CHƯƠNG I. TỔNG QUAN VỀ API…………………………………………...1** |
| **1.1 Giới thiệu chung về API…………………………………………………….1** |
| **1.2 Các loại API…………………………………………………………………1** |
| **1.2.1. Open APIs hoặc Public APIs (API mở)……………………………….1** |
| **1.2.2. Partner APIs (API đối tác)…………………………………………….1** |
| **1.2.3. Internal APIs (API nội bộ)…………………………………………….2** |
| **1.2.4. Composite APIs (Api tổng hợp)……………………………………….2** |
| **1.3 Cách thức hoạt động của API………………………………………………2** |
| **1.4 Các kiến trúc API…...………………………………………………………2** |
| **1.4.1. Kiến trúc REST……...………………………………………...………2** |
| **1.4.2. Kiến trúc SOAP…...…………………………………………………...4** |
| **1.4.3. Kiến trúc RPC…………….……………...……………………………4** |
| **1.5 Các ứng dụng của API………………………………………………………5** |
| **1.5.1. Web API………………………………………………………………..5** |
| **1.5.2. API trên hệ điều hành…………………………………………………5** |
| **1.5.3. API của thư viện phần mềm (Framework)…………………………...5** |
| **CHƯƠNG II. TỔNG QUAN VỀ KIỂM THỬ API……………………………6** |
| **2.1 Khái quát về kiểm thử API…………………………………………………6** |
| **2.1.1. Phương thức GET………………………..……………………………7** |
| **2.1.2. Phương thức POST…………………………………………...……….8** |
| **2.1.3. Phương thức PUT……………………………………………………...8** |
| **2.1.4. Phương thức DELETE……………………………………………...…9** |
| **2.2 Ưu điểm và lợi ích của kiểm thử API……………………………………….9** |
| **2.2.1. Ưu điểm của kiểm thử API...…………………………………………..9** |
| **2.2.2. Lợi ích của kiểm thử API…………………………………………….11** |
| **2.3 Giới thiệu về bộ công cụ……………………………………………………11** |
| **2.3.1. Phạm vi nghiên cứu và mục đích hướng đến………………………..11** |
| **2.3.2. Thành phần sử dụng trong bộ công cụ…………………………….12** |
| **a. Thư viện Requests……………………………………………………..12** |
| **b. Thư viện json…………………………………………………………..12** |
| **c. Thư viện Pytest………………………………………………………...12** |
| **d. Allure…………………………………….……………………………..12** |
| **e. Thư viện locust…………………………………………………………13** |
| **f. File endpoint.json…….………………………………………………...13** |
| **g. File create.json…….…………………………………………………...13** |
| **h. File update.json………………………………………………………..14** |
| **i. Module function\_base....………………………………………………14** |
| **k. Module api\_url………………………………………………………...15** |
| **l. Module common\_ultility………………………….……...……………15** |
| **m. Module file.py…………………………………….……………………16** |
| **n. File conftest.py………………………………….……………………...17** |
| **o. Module custom\_call.py………………………………………………..19** |
| **CHƯƠNG III. XÂY DỰNG CÔNG CỤ KIỂM THỬ API VÀ THỰC NGHIỆM.………………………………………………………………………21** |
| **3.1 Kiểm tra hoạt động của API với 4 phương thức (GET, POST, PUT, DELETE)..……………………………………………………………………...21** |
| **3.1.1. Với Authorization…………………………………………….………21** |
| **a. Trường hợp đúng Authorization……………………………………..21** |
| **b. Trường hợp sai và thiếu Authorization………………………………24** |
| **3.1.2. Với Param…………………………………………………………….27** |
| **3.1.3. Với Data…………….……....................................................................31** |
| **3.2 Kiểm tra hiệu suất…………………………………………………………32** |
| **KẾT LUẬN……………….…………………………………………………….39** |
| **TÀI LIỆU THAM KHẢO……………………………………………………..40** |
| **LỜI CẢM ƠN……………………………………………………………………..** |

**DANH MỤC MINH HỌA**

|  |
| --- |
| Hình 1. API testing………………………………………………………………..6 |
| Hình 2. Các phương thức API…………………………………………………….7 |
| Hình 3. Phương thức GET………………………………………………………...8 |
| Hình 4. Phương thức POST……………………………………………………….8 |
| Hình 5. Phương thức PUT………………………………………………………...9 |
| Hình 6. Phương thức DELETE…………………………………………………...9 |
| Hình 7. Kim tự tháp tự động hóa………………………………………………...10 |
| Hình 8. Module file.py…………….…………………………………………….16 |
| Hình 9a. Đoạn code chương trình kiểm tra hoạt động của API hoàn chỉnh…….21 |
| Hình 9b. Đoạn code chương trình kiểm tra hoạt động của API hoàn chỉnh….…22 |
| Hình 10. Kết quả chạy chương trình với các phương thức trên allure………….24 |
| Hình 11a. Đoạn code hoàn chỉnh chương trình kiểm tra với trường hợp sai và thiếu authorization………………………………………………………………25 |
| Hình 11b. Đoạn code hoàn chỉnh chương trình kiểm tra với trường hợp sai và thiếu authorization………………………………………………………………25 |
| Hình 11c. Đoạn code hoàn chỉnh chương trình kiểm tra với trường hợp sai và thiếu authorization………………………………………………………………25 |
| Hình 12. Kết quả chạy chương trình kiểm thử với trường hợp sai và thiếu authorization...…………………………………………………………………..27 |
| Hình 13a. Đoạn code hoàn chỉnh chương trình kiểm thử trường hợp thừa, thiếu và sai Param với POST và PUT.……….…………………………………..……28 |
| Hình 13b. Đoạn code hoàn chỉnh chương trình kiểm thử trường hợp thừa, thiếu và sai Param với POST và PUT.……….…………………………………..……29 |
| Hình 14. Kết quả chạy chương trình kiểm thử trường hợp thừa, thiếu và sai Param với POST và PUT.……….…………………………………..………..…30 |
| Hình 15. Đoạn code hoàn chỉnh chương trình kiểm thử trường hợp lỗi Data với phương thức POST..………….…………….……………………………………31 |
| Hình 16. Kết quả chạy chương trình kiểm thử trường hợp lỗi Data với phương thức POST…………….…………………………………………………………32 |
| Hình 17. Giao diện Web Locust…………….…………………………………...34 |
| Hình 18. Ví dụ trường hợp 1……....…………………………………………….35 |
| Hình 19. Thống kê về các phương thức được test ....……………………………35 |
| Hình 20. Biểu đồ thể hiện các thông số khi thực hiện Load Test khi giữ nguyên giá trị Ramp up………………………………………………….……………….36 |
| Hình 21. Chỉnh sửa số lượng User và giá trị Ramp up…….………..…………..37 |
| Hình 22a. Biểu đồ thể hiện các thông số khi thực hiện Load Test khi giá trị Ramp up thay đổi……………………………………………………………………....37  Hình 22b. Biểu đồ thể hiện các thông số khi thực hiện Load Test khi giá trị Ramp up thay đổi……………………………………………………………………....38 |

**BẢNG PHÂN CÔNG CÔNG VIỆC**

|  |  |  |
| --- | --- | --- |
| STT | Họ và tên | Nhiệm vụ |
| 1 | Nguyễn Bảo Ngọc | Trưởng nhóm; Viết báo cáo; Xây dựng công cụ kiểm thử API |
| 2 | Cao Trung Du | Tìm hiểu và nghiên cứu về tổng quan API; Xây dựng công cụ kiểm thử API |
| 3 | Trần Minh Kiên | Tìm hiểu và nghiên cứu về kiến trúc Restful API; Xây dựng công cụ kiểm thử API |
| 4 | Khuất Hoàng Long | Xây dựng slide; Xây dựng công cụ kiểm thử API |

**DANH MỤC TỪ VIẾT TẮT**

|  |  |
| --- | --- |
| **Từ viết tắt** | **Tiếng Anh** |
| API | Application Programming Interface |
| REST | Representational State Transfer |
| SOAP | Simple Object Access Protocol |
| RPC | Remote Procedure Call |
| GUI | Graphical User Interface |
| UI | User Interface |
| JSON | JavaScript Object Notation |
| XML | Extensible Markup Language |
| QA | Quality Assurance |
| HTTP | Hypertext Transfer Protocol |

**LỜI MỞ ĐẦU**

**1.** **Lý do chọn đề tài:**

Ngày nay công nghệ thông tin đang ngày càng phát triển nhanh chóng, kéo theo đó là hệ thống mạng, các phần mềm cũng gia tăng cả về số lượng theo quy mô rộng và cả về chất lượng phần mềm. Nhưng cũng từ đó đã nảy sinh ra nhiều vấn đề về lỗi hỏng hóc phần mềm không đáng có gây ra các ảnh hưởng nghiêm trọng đến xã hội, kinh tế, . . . Những lỗi này có thể do tự bản thân phần mềm bị hỏng do không được kiểm duyệt kỹ lưỡng trước khi đưa cho người dùng cuối hay cũng có thể do có người cố tình phá hoại nhằm đánh cắp thông tin cá nhân. Những vấn đề nan giải và cấp thiết này càng có xu hướng mở rộng trong các năm gần đây.

Do đó yêu cầu đặt ra là cần có công tác kiểm thử thật kỹ lưỡng. Tuy nhiên vì phần mềm ngày càng lớn, hàng nghìn module, có thể do cả một công ty hàng nghìn người phát triển vì vậy để kiểm thử được một phần mềm lớn như vậy sẽ tốn rất nhiều công sức và thời gian nếu làm thủ công, chưa kể đến chất lượng kiểm thử sẽ không cao và chính xác. Theo nhiều tính toán thì công việc kiểm thử đóng vai trò hết sức quan trọng trong quy trình phát triển. Vì vậy, cần có các hệ thống kiểm thử phần mềm một cách tự động cho phép người dùng thực hiện được các công việc một cách nhanh chóng và độ an toàn, chính xác cao nhất. Và đó là lý do chúng em quyết định thực hiện đề tài: “**Xây Dựng Công Cụ Kiểm Thử API**”

**2. Mục tiêu**

Mục tiêu mà nhóm mong muốn đạt được sau khi hoàn thành báo cáo đó là:

* Tăng thêm hiểu biết về API, kiểm thử API
* Biết được các phương pháp về kiểm thử API
* Xây dựng công cụ kiểm thử API
* Cuối cùng là thực nghiệm cụ thể để hiểu hơn về bộ công cụ này.

**CHƯƠNG I. TỔNG QUAN VỀ API**

**1.1 Giới thiệu tổng quan về API**

API là từ viết tắt của cụm từ Application Programming Interface - Giao diện lập trình ứng dụng. (Trong ngữ cảnh API, từ “Ứng dụng” đề cập đến mọi phần mềm có chức năng riêng biệt. Giao diện có thể được xem là một hợp đồng dịch vụ giữa 2 ứng dụng. Hợp đồng này xác định cách thức hai ứng dụng giao tiếp với nhau thông qua các yêu cầu và phản hồi. Tài liệu API của các ứng dụng này chứa thông tin về cách nhà phát triển xây dựng cấu trúc cho những yêu cầu và phản hồi đó. API là cơ chế cho phép 2 thành phần phần mềm giao tiếp với nhau bằng một tập hợp các định nghĩa và giao thức.)

**1.2 Các loại API**

API được phân loại theo cả kiến trúc và phạm vi sử dụng. Có 4 loại API:

***1.2.1. Open APIs hoặc Public APIs (API mở)***

Còn có tên gọi khác là API công khai, có sẵn nên có thể được sử dụng bởi bất kỳ nhà phát triển nào. Đổi lại, các Open APIs thông thường sẽ yêu cầu các biện pháp xác thực hoặc ủy quyền thấp và bị hạn chế chức năng khi chia sẻ công khai. Một số Open APIs sẽ được chia sẻ miễn phí, một số khác sẽ yêu cầu tính phí khi sử dụng. Chi phí này thường được tính dựa trên số lượng “lệnh gọi” (calls) đến API được sử dụng.

***1.2.2. Partner APIs (API đối tác)***

API này cần có quyền hoặc giấy phép cụ thể mới truy cập được. Thường dành cho các nhà phát triển bên ngoài ủy quyền để hỗ trợ đầu mối hợp tác giữa doanh nghiệp với doanh nghiệp. Một số doanh nghiệp lựa chọn Partner APIs vì muốn kiểm soát tốt hơn người dùng có thể truy cập vào tài nguyên của họ và chỉ rõ cách thức sử dụng các tài nguyên đó.

***1.2.3. Internal APIs (API nội bộ)***

Không giống như API mở hay API đối tác, API nội bộ không dành cho các bên thứ ba sử dụng, thường dùng trong phạm vi công ty. Công ty sử dụng API này để để kết nối các hệ thống cũng như dữ liệu nội bộ của công ty/tổ chức.

***1.2.4. Composite APIs (API tổng hợp)***

Kết hợp hai hay nhiều API khác nhau để giải quyết các yêu cầu phức tạp của hệ thống. Nếu cần dữ liệu từ các ứng dụng hoặc từ nhiều nguồn dữ liệu khác nhau, người dùng nên sử dụng API tổng hợp. Ngoài ra, người dùng có thể sử dụng API tổng hợp để thiết lập một chuỗi các “lệnh gọi” (calls) và phản hồi tự động mà không cần chủ động can thiệp vào.

**1.3 Cách thức hoạt động của API**

API giao tiếp thông qua một tập hợp các quy tắc để xác định phương thức mà các máy tính, ứng dụng hoặc máy móc có thể tương tác với nhau. API hoạt động như một người trung gian giữa hai thiết bị bất kỳ muốn kết nối với nhau phục vụ cho một tác vụ được chỉ định. Kiến trúc API thường được giải thích dưới dạng máy chủ và máy khách. Ứng dụng gửi yêu cầu được gọi là máy khách, còn ứng dụng gửi phản hồi được gọi là máy chủ.

Ví dụ đơn giản: Khi muốn đăng nhập Facebook thông qua ứng dụng trên điện thoại bằng tài khoản của người dùng. Lúc này, ứng dụng Facebook sẽ thực hiện một lệnh tới API để truy xuất tài khoản và thông tin đăng nhập của người dùng. Sau đó, Facebook sẽ truy cập thông tin này từ một trong các máy chủ của mình và trả dữ liệu về ứng dụng di động.

**1.4 Các kiến trúc API**

***1.4.1. Kiến trúc REST***

REST viết tắt của Representational State Transfer, là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. REST thường được dùng cho các ứng dụng web, hoặc có thể làm việc với dữ liệu phần mềm.

API REST (hoặc API “RESTful”) là một API tuân theo các nguyên tắc REST và được sử dụng để truyền dữ liệu từ máy chủ đến máy khách yêu cầu.

Các API REST dựa trên URL, giao thức HTTP và dựa trên 6 ràng buộc kiến trúc sau:

* *Dựa trên máy khách - máy chủ (Client-server based):*

Sự ràng buộc này hoạt động dựa vào ý tưởng máy khách và máy chủ phải hoàn toàn tách biệt và được phép phát triển riêng lẻ, độc lập. Máy khách xử lý quá trình giao diện người dùng trong khi máy chủ xử lý phần phụ trợ. Phương thức hoạt động chính của REST là tách biệt giao diện người dùng ra khỏi dữ liệu lưu trữ.

Với cách thức này, người dùng có thể thực hiện thay đổi với các ứng dụng di động của mình một cách độc lập. Việc này không làm ảnh hưởng đến cấu trúc dữ liệu hoặc thiết kế cơ sở dữ liệu của máy chủ. Ngược lại, việc điều chỉnh cơ sở dữ liệu hoặc thay đổi ứng dụng của máy chủ cũng không ảnh hưởng đến ứng dụng của máy khách.

* *Giao diện thống nhất (Uniform interface):*

Xác định giao diện giữa máy khách và máy chủ, giúp cho tổng thể kiến trúc hệ thống trở nên đơn giản hóa. Khả năng hiển thị của các tương tác cũng được cải thiện đáng kể.

* *Không trạng thái (Stateless):*

Bất kỳ một RESTful API nào cũng ở dạng không trạng thái. Nghĩa là mỗi yêu cầu từ máy khách đến máy chủ phải độc lập và chứa tất cả các thông tin cần thiết để máy chủ có thể hiểu và xử lý cho phù hợp. Ngoài ra, yêu cầu của máy khách không thể lạm dụng bất kỳ thông tin nào trên máy chủ. Điều này sẽ giúp tăng độ tin cậy cho API, hạn chế lỗi và giảm tài nguyên sử dụng.

* *Lưu vào bộ nhớ cache (Cacheable):*

API không trạng thái có thể tăng số lượng yêu cầu (request), nhất là khi có nhiều “lệnh gọi” đến và đi. Vì thế, RESTful API được thiết kế để lưu trữ dữ liệu vào cache để tăng tính tái sử dụng.

Cụ thể, các ràng buộc này sẽ yêu cầu mỗi phản hồi phải đánh dấu dữ liệu bên trong là được lưu hay không lưu vào cache. Nếu được lưu vào cache, máy khách có thể sử dụng lại dữ liệu phản hồi đó cho các yêu cầu tương tự sau này.

* *Hệ thống phân lớp (Layered system):*

Các lớp được sắp xếp theo thứ bậc để mỗi lớp chỉ có thể "nhìn thấy" lớp tương ứng mà chúng đang tương tác. Kiểu hệ thống phân lớp cho phép một kiến trúc chứa nhiều lớp phân cấp. Mỗi lớp sẽ có một chức năng và trách nhiệm cụ thể. Cách thức của REST là hạn chế hành vi của các thành phần trong một lớp.

* *Mã theo yêu cầu (Code on demand):*

Ràng buộc này cho phép người dùng mở rộng chức năng của máy khách bằng cách tải xuống và thực thi mã dưới dạng các applet và script. Điều này đơn giản hóa cho máy khách, bằng cách giảm số lượng các tính năng bắt buộc phải triển khai trước.

***1.4.2. Kiến trúc SOAP***

SOAP là viết tắt của cụm từ Simple Object Access Protocol, được tạm dịch là giao thức truy cập đối tượng đơn giản. Đây là một giao thức để truyền dữ liệu qua các mạng và có thể được sử dụng để xây dựng các API. SOAP dựa trên tiêu chuẩn hóa bởi World Wide Web Consortium (W3C) và sử dụng XML để mã hóa thông tin. SOAP có thể được thực hiện trên nhiều giao thức tiêu chuẩn khác nhau, bao gồm giao thức HTTP.

***1.4.3. Kiến trúc RPC***

RPC là viết tắt của Remote Procedure Call, là một mô hình kỹ thuật mạng hay còn được biết đến là cơ chế giao tiếp giữa hai tiến trình. Không giống như REST và SOAP tạo điều kiện cho việc truyền dữ liệu, các API RPC gọi các quy trình. Nói cách khác, chúng thực thi các tập lệnh trên một máy chủ.

**1.5 Các ứng dụng của API**

***1.5.1. Web API***

Hệ thống API thường được sử dụng trong các hệ thống website. Việc ứng dụng Web API ở hầu hết website để cho phép kết nối, lấy dữ liệu hoặc cập nhật cơ sở dữ liệu một cách hiệu quả. Các website sẽ được thiết kế theo tiêu chuẩn RESTful. Ví dụ: Thiết kế tính năng login thông qua Google, Facebook, Twitter...

***1.5.2. API trên hệ điều hành***

Hệ điều hành phổ biến như Windows hay Linux có rất nhiều API, cung cấp các tài liệu API đặc tả các hàm, phương thức, giao thức kết nối. Nhờ API, các lập trình viên có thể dễ dàng tạo ra các phần mềm ứng dụng cần thiết, tương tác với hệ điều hành.

***1.5.3. API của thư viện phần mềm (Framework)***

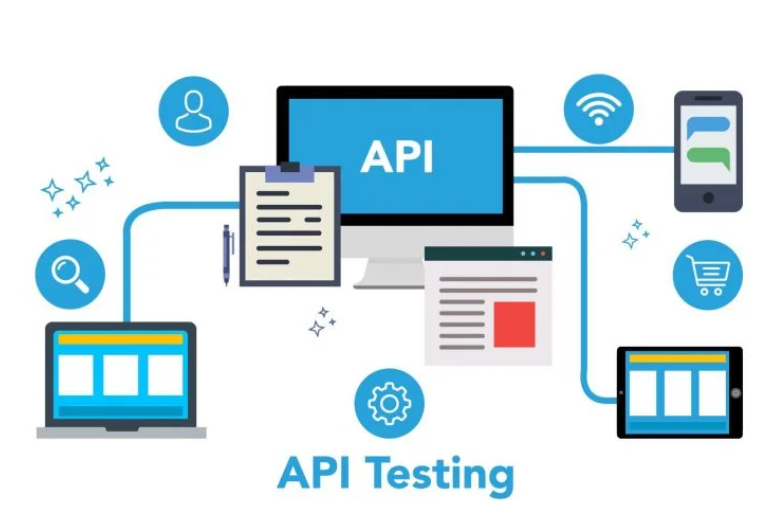
API mô tả, quy định các hành động mong muốn mà các thư viện cung cấp. Một API có thể có nhiều cách triển khai khác nhau và API giúp cho một chương trình viết bằng ngôn ngữ này có thể sử dụng được thư viện của ngôn ngữ khác. Ví dụ: Có thể dùng ngôn ngữ PHP để yêu cầu một thư viện tạo file PDF được viết bằng ngôn ngữ C++.

**CHƯƠNG II. TỔNG QUAN VỀ KIỂM THỬ API**

* 1. **Khái quát kiểm thử API**

Kiểm thử API là một loại kiểm thử phần mềm bao gồm kiểm tra trực tiếp các giao diện lập trình ứng dụng và là một phần của kiểm thử tích hợp để xem phần mềm có đáp ứng được những mong đợi về chức năng, hiệu suất, độ tin cậy bảo mật hay không. Hay hiểu một cách đơn giản hơn nó là phần mềm trung gian giữa Client và Server để gọi tới API, nhận kết quả đầu ra và ghi lại phản hồi của hệ thống.

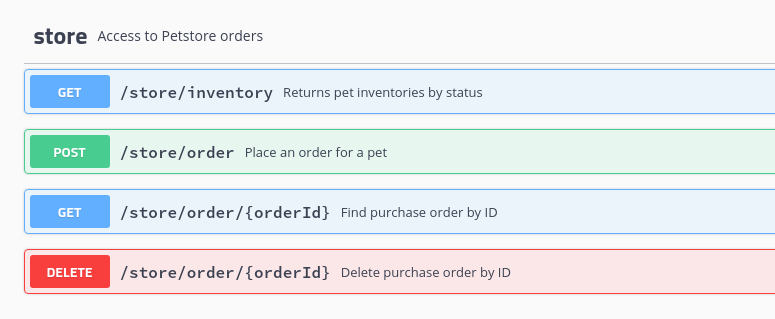
Trong API, thường sử dụng giao thức để Client và server giao tiếp với nhau. Trong đó giao thức chính để server và Client giao tiếp với nhau là HTTP. Và API được xây dựng trên 2 thành phần chính là: Yêu cầu (request) và phản hồi (response).



*Hình 1: API testing*

API gồm nhiều phương thức, chủ yếu có 4 phương thức phổ biến sau:

* GET: Truy xuất một tài nguyên, nhận dữ liệu từ server và hiển thị
* POST: Tạo một tài nguyên trên server
* PUT: Thay đổi trạng thái một tài nguyên hoặc cập nhật nó
* DELETE: Huỷ bỏ hoặc xoá một tài nguyên

*Hình 2: Các phương thức API*

***2.1.1. Phương thức GET***

GET được sử dụng để request dữ liệu từ một tài nguyên chỉ định với query string là một cặp ***name/value*** được gắn vào URL của request.

**Một vài điểm chú ý về request GET:**

* request GET có thể được lưu vào bộ nhớ đệm (cached)
* request GET có thể được giữ lại trong lịch sử trình duyệt
* request GET có thể được lưu lại trong bookmarked
* request GET không nên sử dụng khi người dùng gửi đi những dữ liệu nhạy cảm ví dụ như mật khẩu, số tài khoản …
* request GET hạn chế về độ dài
* request GET chỉ được sử dụng để lấy dữ liệu (không sửa đổi)



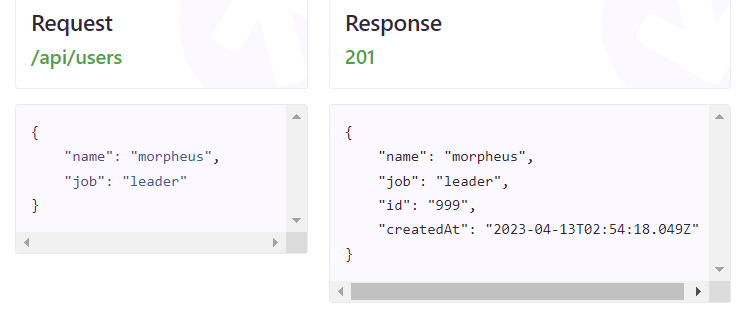
*Hình 3: Phương thức GET*

***2.1.2. Phương thức POST***

POST được sử dụng để gửi dữ liệu tới server để thêm mới/cập nhật (create/update) một tài nguyên. Dữ liệu gửi tới server với phương thức POST được lưu trữ trong thân của request HTTP.

**Một vài chú ý với request POST:**

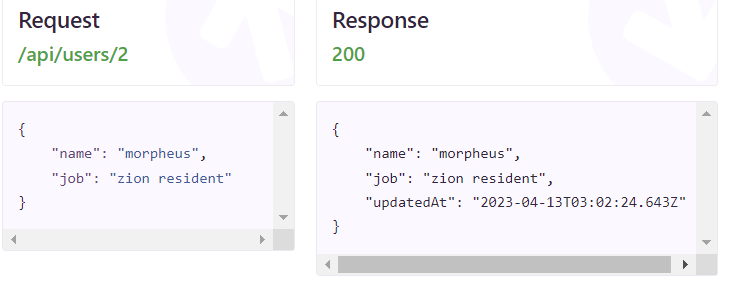
* request POST không bao giờ được lưu vào bộ nhớ đệm (cached)
* request POST không giữ lại trong lịch sử trình duyệt
* request POST không thể lưu lại trong bookmarked
* request POST không bị hạn chế về độ dài dữ liệu



*Hình 4: Phương thức POST*

***2.1.3. Phương thức PUT***

PUT được sử dụng để thay đổi trạng thái một tài nguyên hoặc cập nhật.



*Hình 5: Phương thức PUT*

***2.1.4. Phương thức DELETE***

DELETE được sử dụng để hủy bỏ hoặc xóa tài nguyên.



*Hình 6: Phương thức DELETE*

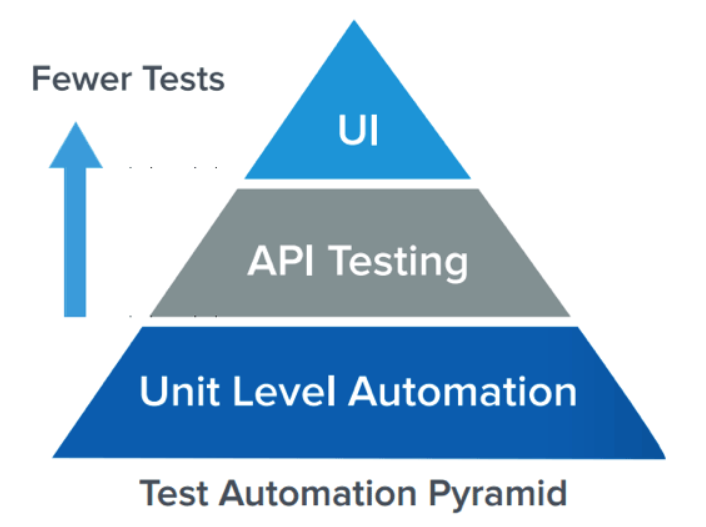
* 1. **Ưu điểm và lợi ích của kiểm thử API**

***2.2.1. Ưu điểm của kiểm thử API***

API không cần giao diện người dùng mà vẫn kiểm thử ứng dụng sớm:

Nếu người dùng tìm thấy lỗi càng muộn thì họ càng mất nhiều thời gian và công sức để sửa nó. API Testing sẽ giúp người kiểm thử tham gia sớm vào vòng đời phát triển của sản phẩm. Với API Testing, người dùng hoàn toàn có thể bắt đầu kiểm thử ứng dụng sớm mà không cần đến giao diện người dùng. Điều này sẽ giúp người dùng sớm khắc phục được các vấn đề trong vòng đời phát triển, nếu không thì sẽ mất nhiều chi phí để khắc phục khi lỗi được xác định ở quá trình kiểm thử GUI. Ưu điểm của API Testing là có thể kiểm tra rất nhiều logic mà không bị phụ thuộc vào GUI.

* + Tiết kiệm chi phí và xây dựng được chiến lược kiểm thử tự động hoàn hảo.
  + Nếu hiểu được “Kim tự tháp tự động hóa (Automation pyramid), người dùng có thể xây dựng một chiến lược tự động hiệu quả.



Hình 7: Kim tự Tháp tự động hóa

Đi từ tầng dưới của kim tự tháp, các chi phí liên quan đến việc tạo ra và duy trì các phương pháp, thời gian thực hiện, phạm vi kiểm thử sẽ dần tăng lên. Kim tự tháp chỉ ra rằng người dùng cần làm nhiều kiểm thử tự động thông qua Uni Test và API Testing hơn là thực hiện kiểm thử dựa trên GUI.

Trên thực tế, việc liên tục tích hợp, thời gian để kiểm thử hồi quy GUI mất quá nhiều thời gian để nhận lại phản hồi. Các chi phí liên quan đến việc thực hiện và duy trì các phương pháp kiểm thử sẽ dần tăng lên.

* + Hạn chế kiểm thử hồi quy bằng tay và phát triển phần mềm theo phương pháp Agile. Điều giúp duy trì tính nhanh chóng do sự cần thiết của các đội Agile. Tăng mức độ kiểm thử API và giảm sự phụ thuộc của họ vào kiểm tra GUI.
  + Bằng cách tích hợp API Testing sẽ làm giảm áp lực của kiểm thử hồi quy của nhóm QA. Nhóm QA có thể phản hồi nhanh về chất lượng ứng dụng ngay khi dự án được triển khai (deploy), hệ thống được đánh giá một cách nhanh chóng trước khi kiểm thử GUI. API testing yêu cầu code ít hơn, phạm vi kiểm thử rộng hơn và cung cấp kết quả nhanh hơn.

***2.2.2. Lợi ích của kiểm thử API***

Kiểm thử API mang lại nhiều lợi ích quan trọng cho quá trình phát triển phần mềm và triển khai ứng dụng. Việc kiểm thử mang lại nhiều lợi ích:

* + Đảm bảo tính đúng đắn: Kiểm thử API giúp đảm bảo rằng API hoạt động chính xác và đáp ứng các yêu cầu và ràng buộc. Nó đảm bảo rằng API trả về dữ liệu chính xác và thực hiện các chức năng như mong đợi.
  + Kiểm tra tính bảo mật: Kiểm thử API cho phép xác minh tính bảo mật của API. Nó giúp đảm bảo rằng API không có lỗ hổng bảo mật và không mở ra các lỗ hổng tiềm năng trong hệ thống.
  + Đảm bảo tính ổn định: Kiểm thử API giúp xác định và giải quyết các lỗi và sự cố trong API. Nó giúp đảm bảo rằng API hoạt động ổn định và không gây ra các vấn đề không mong muốn trong ứng dụng sử dụng API.
  + Tăng cường khả năng tái sử dụng: Kiểm thử API giúp xác định các lỗi và cải thiện tính đúng đắn của API. Điều này giúp tăng cường khả năng tái sử dụng của API, cho phép nhiều ứng dụng khác nhau sử dụng API một cách tin cậy và hiệu quả.
  + Tăng độ tin cậy và chất lượng: Kiểm thử API giúp cải thiện độ tin cậy và chất lượng của ứng dụng. Điều này đồng nghĩa với việc cung cấp trải nghiệm tốt hơn cho người dùng và giảm thiểu các lỗi và sự cố không mong muốn.
  + Giảm rủi ro và chi phí: Kiểm thử API sớm trong quá trình phát triển giúp phát hiện và khắc phục các vấn đề trước khi chúng trở thành các vấn đề lớn. Điều này giúp giảm rủi ro và chi phí phát sinh sau này trong quá trình triển khai và vận hành ứng dụng.
  1. **Giới thiệu về bộ công cụ**

***2.3.1. Phạm vi nghiên cứu và mục đích hướng đến***

* *Phạm vi nghiên cứu:*
  + - Nghiên cứu RESTful API
    - Xây dựng công cụ kiểm thử bằng Python
* *Mục đích hướng đến:*

- Mục đích hướng đến khi thực hiện kiểm thử API bao gồm một số yếu tố quan trọng để đảm bảo tính an toàn, bảo mật và độ tin cậy của hệ thống.

- Đầu tiên, cần đảm bảo rằng API hoạt động đúng theo thiết kế và mô tả đã được xác định trước. Điều này bao gồm việc kiểm tra tính chính xác của dữ liệu được truyền vào và trả về từ API.

- Ngoài ra, việc đảm bảo API trả về các mã lỗi và thông báo hợp lý trong các tình huống không mong muốn hoặc lỗi là một phần quan trọng. Điều này giúp người dùng và hệ thống khác có thể xử lý các tình huống bất ngờ một cách hiệu quả và không gây ra sự cố lớn.

- Một mục tiêu khác của kiểm thử API là đánh giá hiệu suất của API. Điều này bao gồm việc đo lường thời gian phản hồi của các yêu cầu, tải trọng mà hệ thống có thể chịu đựng, và các chỉ số hiệu suất khác để đảm bảo rằng API hoạt động một cách hiệu quả và ổn định trong các tình huống khác nhau.

***2.3.2. Thành phần sử dụng trong bộ công cụ***

* + - 1. **Thư viện requests**: đóng vai trò quan trọng trong việc tương tác với API bằng cách cung cấp các công cụ để yêu cầu HTTP và xử lý phản hồi từ API trong các ứng dụng và dịch vụ của bạn.
      2. **Thư viện json**: là thư viện tiêu chuẩn cho phép người dùng làm việc với định dạng dữ liệu JSON.
      3. **Thư viện pytest**: là một framework kiểm thử cho python, được sử dụng để viết và chạy các bài kiểm thử cho các dự án python.
      4. **Allure**: là một framework báo cáo và tạo báo cáo kiểm thử được sử dụng trong nhiều dự án tập chung giúp dễ đọc và tương tác, giúp người dùng hiểu rõ hơn về kết quả của các bài kiểm thử.
      5. **Thư viện locust**: là một user load testing tool được viết bằng python, thường được dùng để load testing cho website, các hệ thống api, ... và để tìm ra số lượng người dùng đồng thời mà hệ thống có thể xử lí.
      6. **File endpoint.json:**

{

"uat": {

"base\_url": "https://gorest-uat.co.in/"

},

"prod": {

"base\_url": "https://gorest.co.in/"

},

"environment": {

"env": "None"

}

}

File chứa thông tin cho các môi trường, trong đó:

* “uat” và “prod” là các khối định danh cho các môi trường khác nhau, mỗi khối này chứa thông tin cấu hình cho một môi trường cụ thể.
* “base\_url” là những url đặc trưng cho những môi trường khác nhau.

- “environment” là khối định danh chứa thông tin về môi trường hiện tại đang được sử dụng.

**g. File create.json:**

{

"name": "Pankaj G",

"email": "pankaj1112131@gmail.com",

"gender": "male",

"status": "active"

}

File chứa các thông tin được sử dụng cho phương thức POST.

**h. File update.json:**

{

"name": "Pan",

"status": "active"

}

File chứa các thông tin được sử dụng cho phương thức PUT.

**i. Module function\_base:**

import json

def get\_base\_end\_point():

with open("C:\api-testing\config\endpoint.json","r") as json\_file:

properties = json.load(json\_file)

env = properties["environment"]["env"]

return properties[env]["base\_url"]

* Hàm get\_base\_end\_point() được sử dụng để đọc và trả về URL cơ sở của một môi trường cụ thể từ file JSON endpoint.json.
* with open("C:\api-testing\config\endpoint.json","r") as json\_file: Sử dụng câu lệnh with để mở tệp JSON có đường dẫn "C:\api-testing\config\endpoint.json" trong chế độ đọc ("r") và gán nó vào biến json\_file. Việc sử dụng with đảm bảo rằng tệp sẽ được đóng một cách tự động sau khi kết thúc việc đọc.
* properties = json.load(json\_file): Sử dụng hàm json.load() để đọc nội dung của tệp JSON từ json\_file và chuyển đổi nó thành một cấu trúc dữ liệu Python. Kết quả được gán vào biến properties.
* env = properties["environment"]["env"]: Truy cập vào giá trị của khóa "environment" trong từ điển properties, sau đó truy cập vào giá trị của khóa con "env". "env" là khóa con chứa tên của môi trường hiện tại.
* return properties[env]["base\_url"]: Trả về giá trị của khóa "base\_url" trong từ điển con tương ứng với môi trường được chỉ định bởi env.

**k. Module** **api\_url**:

import function\_base

GET\_URL = function\_base.get\_base\_end\_point() + "/public/v2/users"

Kết hợp các base\_url từ module function\_base với các endpoint để tạo thành url hoàn chỉnh.

**l. Module common\_ultility:**

def get\_header():

headers = {

"Content-Type": "application/json",

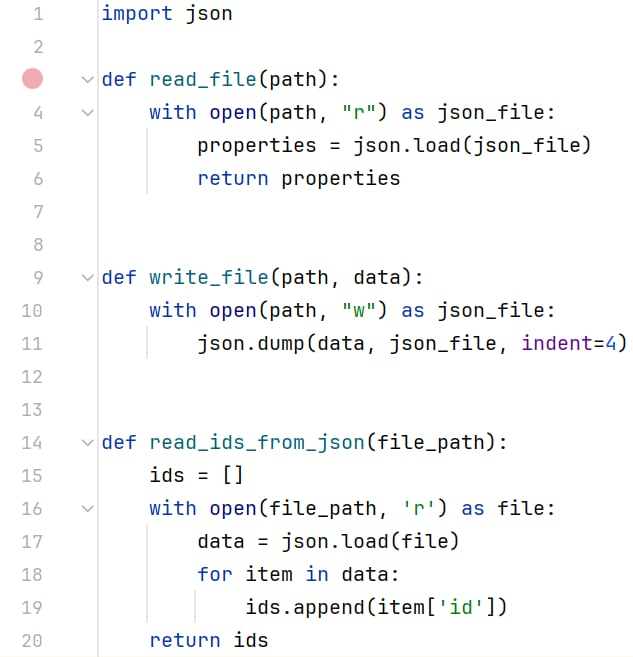
"Authorization": "Bearer 18661386073d8ec7e241a5395233b5ce5457a4a7acca76f0a37f0ff14e057a7b"

}

return headers

Hàm get\_header() chứa các thông tin header được cung cấp được sử dụng để gửi các requests.

**m. Module file.py:**



*Hình 8. Module file.py*

- read\_file(path): Đọc nội dung của một tệp JSON và trả về dữ liệu dưới dạng Python. Tham số path là đường dẫn của tệp JSON cần đọc. Hàm này mở tệp JSON, đọc nội dung và sử dụng json.load() để phân tích nội dung JSON thành dữ liệu Python, sau đó trả về dữ liệu đó.

- write\_file(path, data): Ghi dữ liệu vào một tệp JSON. Tham số path là đường dẫn của tệp JSON cần ghi và data là dữ liệu Python mà bạn muốn ghi vào tệp. Hàm này mở tệp JSON để ghi và sử dụng json.dump() để viết dữ liệu Python vào tệp dưới định dạng JSON.

- read\_ids\_from\_json(file\_path): Đọc các ID từ một tệp JSON và trả về một danh sách các ID. Tham số file\_path là đường dẫn của tệp JSON chứa danh sách các ID. Hàm này mở tệp JSON, sử dụng json.load() để phân tích nội dung JSON thành dữ liệu Python, sau đó lặp qua mỗi mục trong dữ liệu và trích xuất các ID vào một danh sách, cuối cùng trả về danh sách các ID này.

**n. File conftest.py**

def pytest\_addoption(parser):

parser.addoption(

"--env", action="store", default="prod"

)

- Hàm pytest\_addoption(parser) được sử dụng để thêm một command line option (một tham số được truyền vào khi chạy một chương trình từ dòng lệnh) vào pytest.

- “--env” là command line option.

- action = “store” Xác định cách parser xử lý tùy chọn. Trong trường hợp này, "store" đơn giản là lưu giá trị của tùy chọn dưới dạng một chuỗi.

- default="prod": Xác định giá trị mặc định cho tùy chọn nếu không có giá trị nào được cung cấp. Trong trường hợp này, nếu không có giá trị --env được chỉ định trên dòng lệnh, mặc định sẽ là "prod".

def update\_env(config):

with open("C:\api-testing\config\endpoint.json","r+") as json\_file:

data = json.load(json\_file)

json\_file.truncate(0)

json\_file.seek(0)

data["environment"]["env"] = config.getoption("--env").lower()

json.dump(data, json\_file, indent=4)

- Hàm update\_env(config) trong đoạn mã trên có chức năng là cập nhật giá trị của môi trường (environment) trong tệp JSON cấu hình, dựa trên giá trị được cung cấp qua tham số config.

+ with open("C:\api-testing\config\endpoint.json","r+") as json\_file: Sử dụng câu lệnh with để mở tệp JSON "C:\api-testing\config\endpoint.json" trong chế độ đọc và ghi ("r+"). Tùy chọn này cho phép đọc từ tệp và ghi vào tệp.

+ data = json.load(json\_file): Sử dụng hàm json.load() để đọc nội dung của tệp JSON từ json\_file và chuyển đổi nó thành một cấu trúc dữ liệu Python, được lưu trong biến data.

+ json\_file.truncate(0): Xóa nội dung của tệp JSON bằng cách sử dụng truncate(0), để chuẩn bị cho việc ghi lại dữ liệu sau khi cập nhật.

+ json\_file.seek(0): Đặt con trỏ đọc/ghi của tệp về vị trí đầu tiên để bắt đầu ghi lại dữ liệu.

+ data["environment"]["env"]=config.getoption("--env").lower(): Cập nhật giá trị của môi trường trong cấu trúc dữ liệu data bằng cách sử dụng giá trị của tùy chọn dòng lệnh --env được cung cấp thông qua tham số config, và chuyển đổi thành chữ thường bằng phương thức lower().

+ json.dump(data, json\_file, indent=4): Sử dụng hàm json.dump() để ghi lại cấu trúc dữ liệu data vào tệp JSON json\_file, với định dạng được lùi cách (indent) là 4 khoảng trắng.

@pytest.hookimpl

def pytest\_configure(config):

update\_env(config)

- @pytest.hookimpl là một decorator. Khi người dùng đánh dấu một hàm với @pytest.hookimpl, pytest sẽ nhận ra rằng đó là một hook và sẽ tự động gọi nó tại các điểm cụ thể trong quá trình chạy. Các hook là các điểm mà người dùng có thể gắn các hàm để tùy chỉnh hoặc mở rộng hành vi của pytest.

- Hàm pytest\_configure(config) là một hook cho phép người dùng thực hiện các thao tác cấu hình hoặc chuẩn bị môi trường trước khi chạy các bài kiểm.

**o. Module custom\_call.py**

import requests

import pytest

def call\_api\_custom(method, url, r\_json = None, headers = None, status = None):

response = requests.request(method, url,json = r\_json,headers= headers)

try:

assert response.status\_code == status, f"API ERROR, status code is {response.status\_code}"

return response

except:

pytest.fail("API ERROR")

- Hàm call\_api\_custom() có chức năng thực hiện một yêu cầu API tùy chỉnh thông qua phương thức và URL được chỉ định, cùng với dữ liệu JSON và tiêu đề (headers) nếu có. Sau đó, nó kiểm tra mã trạng thái của phản hồi và so sánh với mã trạng thái được mong đợi. Nếu mã trạng thái không khớp với mã trạng thái mong đợi, nó sẽ ghi nhận một testcase thất bại.

- Hàm call\_api\_custom() được định nghĩa với các tham số sau:

+ method: Phương thức HTTP (GET, POST, PUT, DELETE) được sử dụng cho yêu cầu.

+ url: URL của API được gọi.

+ r\_json: Dữ liệu JSON được gửi với yêu cầu, mặc định là None.

+ headers: Các tiêu đề (headers) của yêu cầu, mặc định là None.

+ status: Mã trạng thái HTTP mong đợi của phản hồi, mặc định là None.

- requests.request(): Dòng này gửi một yêu cầu HTTP đến URL được chỉ định bằng phương thức được chỉ định, với dữ liệu JSON và tiêu đề được cung cấp. Kết quả phản hồi được gán cho biến response.

- assert response.status\_code == status: Dòng này kiểm tra xem mã trạng thái của phản hồi có bằng với mã trạng thái mong đợi hay không. Nếu không, nó sẽ ghi nhận một testcase thất bại với thông điệp "API ERROR, status code is {response.status\_code}".

- except: pytest.fail("API ERROR"): Nếu có bất kỳ ngoại lệ nào xảy ra trong quá trình kiểm tra, nó cũng sẽ ghi nhận một testcase thất bại với thông điệp "API ERROR".

**CHƯƠNG III. XÂY DỰNG CÔNG CỤ KIỂM THỬ API VÀ THỰC NGHIỆM**

**3.1 Kiểm tra hoạt động của API với 4 phương thức (GET, POST, PUT, DELETE)**

**3.1.1. Với Authorization**

***a) Trường hợp đúng Authorization***

**- Nhập các mô đun cần thiết:** import pytest

import api\_url

import custom\_call

import common\_utility

import file

id = None

****

Hình 9a. Đoạn code chương trình kiểm tra hoạt động của API hoàn chỉnh



Hình 9b. Đoạn code chương trình kiểm tra hoạt động của API hoàn chỉnh

- “id” là biến cục bộ dùng để lưu trữ id khi thực hiện phương thức POST trong quá trình kiểm thử.

**- Test Method GET (test\_get()):**

+ Hàm test sẽ gửi một yêu cầu GET tới URL được xác định.

+ Header được thiết lập bằng cách gọi hàm common\_utility.get

\_header().

+ API được gọi thông qua hàm custom\_call.call\_api\_custom() với phương thức GET, URL và header đã được thiết lập.

+ Phản hồi từ API được kiểm tra để đảm bảo mã trạng thái là 200.

+ Dữ liệu JSON từ phản hồi được in ra và ghi vào tệp "start.json".

+ Một thông báo "API success" được in ra.

**- Test Method POST (test\_post()):**

+ Hàm test sẽ gửi một yêu cầu POST tới URL được xác định.

+ Header được thiết lập bằng cách gọi hàm common\_utility.get

\_header().

+ Dữ liệu từ tệp JSON "create.json" được đọc và cập nhật với một địa chỉ email mới bằng cách gọi common\_utility.get\_email().

+ API được gọi thông qua hàm custom\_call.call\_api\_custom() với phương thức POST, URL, header và dữ liệu đã được thiết lập.

+ Phản hồi từ API được kiểm tra để đảm bảo mã trạng thái là 201.

+ id từ phản hồi được lưu và in ra, cùng với một thông báo "API success".

**- Test Method PUT (test\_put()):**

+ Hàm test sẽ gửi một yêu cầu PUT tới URL được xác định với id đã được lưu từ phản hồi của test method POST.

+ Header được thiết lập bằng cách gọi hàm common\_utility.get

\_header().

+ Dữ liệu từ tệp JSON "update.json" được đọc.

+ API được gọi thông qua hàm custom\_call.call\_api\_custom() với phương thức PUT, URL, header và dữ liệu đã được thiết lập.

+ Phản hồi từ API được kiểm tra để đảm bảo mã trạng thái là 200.

+ Dữ liệu JSON từ phản hồi được in ra, cùng với một thông báo "API success".

**- Test Method DELETE (test\_delete()):**

+ Hàm test sẽ gửi một yêu cầu DELETE tới URL được xác định với id đã được lưu từ phản hồi của test method POST.

+ Header được thiết lập bằng cách gọi hàm common\_utility.get

\_header().

+ API được gọi thông qua hàm custom\_call.call\_api\_custom() với phương thức DELETE, URL và header đã được thiết lập.

+ Phản hồi từ API được kiểm tra để đảm bảo mã trạng thái là 204.

+ Một thông báo "API success" được in ra.

* ***Chạy chương trình:***

**- Nhập lệnh:**

pytest -s -v .\test\_script1.py --env=prod -–alluredir=

"C:\api-testing\api\_test\script1"

allure serve "C:\api-testing\api\_test\script1"

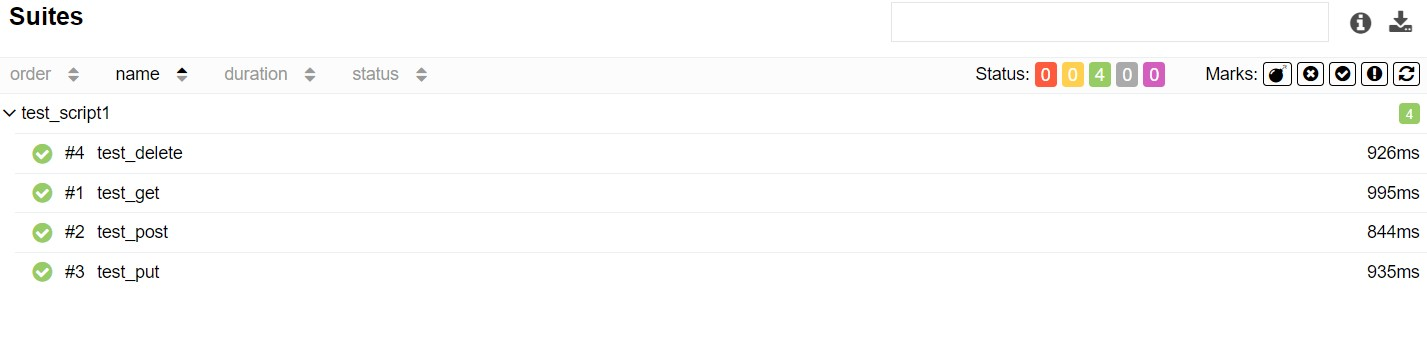
+ pytest -s -v .\test\_script1.py thực hiện kiểm thử file test\_script1.py trong đó “-s” là hiển thị đầu ra của các hàm test, “-v” là hiển thị chi tiết kết quả khi thực hiện test.

+ --env=prod truyền thông tin môi trường từ bàn phím.

+ -–alluredir="C:\api-testing\api\_test\script1" trích xuất dữ liệu báo cáo đến file test\_script1.py.

+ allure serve "C:\api-testing\api\_test\script1" tạo một localhost và hiển thị báo cáo trong trình duyệt web.

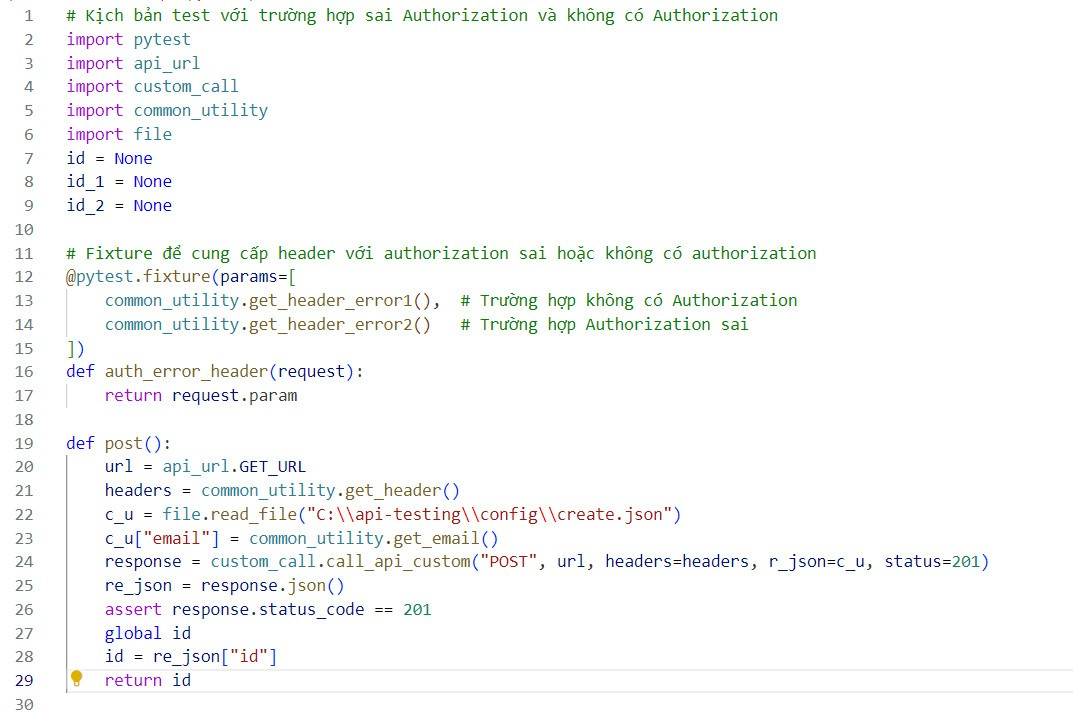
* ***Kết quả:***

******

*Hình 10. Kết quả chạy chương trình với phương thức các phương thức trên allure*

* Kết quả trên cho thấy 4 phương thức đều thành công với trường hợp đúng authorization.

**b) Trường hợp sai và thiếu Authorization**

****

*Hình 11a. Đoạn code hoàn chỉnh chương trình kiểm tra với trường hợp sai và thiếu Authorization*

****

*Hình 11b. Đoạn code hoàn chỉnh chương trình kiểm tra với trường hợp sai và thiếu Authorization*

****

*Hình 11c. Đoạn code hoàn chỉnh chương trình kiểm tra với trường hợp sai và thiếu Authorization*

- fixture auth\_error\_header: Trả về các header có chứa authorization và không chứa authorization, sử dụng để kiểm thử các trường hợp liên quan đến lỗi authorization.

- Hàm post(): Gọi API POST để tạo một bản ghi mới và trả về id của bản ghi đó.

- test\_get\_auth(), test\_post\_auth(), test\_put\_auth(), test\_delete\_auth(): Các test case này đều nhận vào fixture auth\_error\_header để kiểm tra các trường hợp lỗi authorization. Mỗi test case gọi một loạt các API tương ứng (GET, POST, PUT, DELETE) với các header được cung cấp bởi fixture auth\_error\_header.

- Trong các test case PUT và DELETE, sử dụng các biến global id, id\_1, id\_2 để xác định ID của bản ghi cần cập nhật hoặc xóa.

* ***Chạy chương trình:***

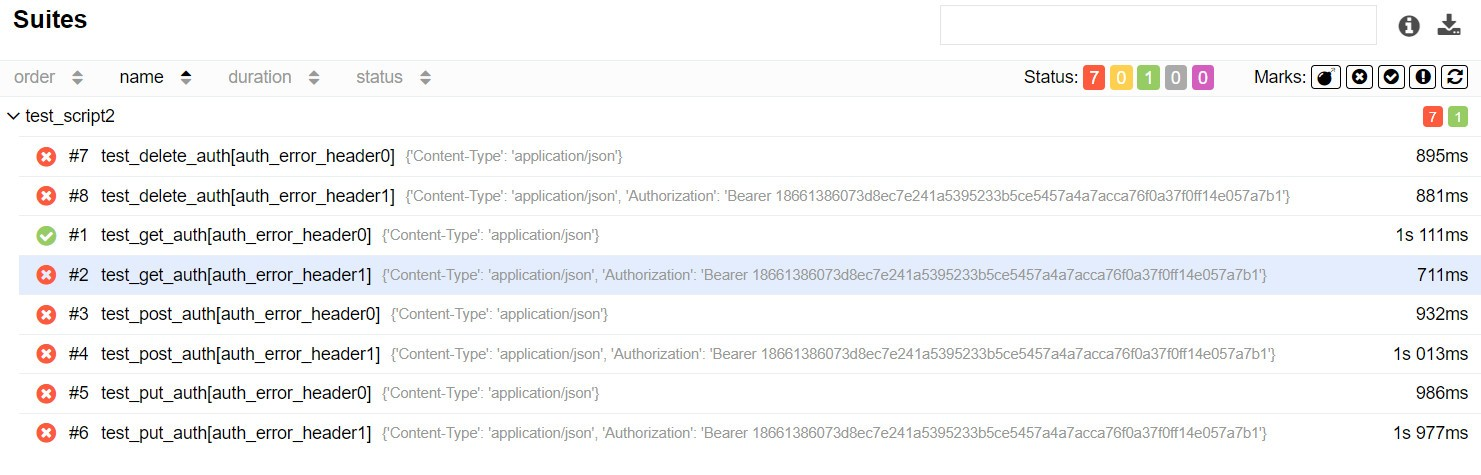
**- Nhập lệnh:**

pytest -s -v .\test\_script2.py --env=prod -–alluredir=

"C:\api-testing\api\_test\script2"

allure serve "C:\api-testing\api\_test\script2"

* ***Kết quả:***

****

*Hình 12. Kết quả của chương trình kiểm thử với trường hợp sai và thiếu authorization*

* Có thể thấy ngoại trừ trường hợp không có authorization ở phương thức GET là thành công với những dữ liệu không cần ủy quyền, các trường hợp còn lại đều xảy ra lỗi 401 – Lỗi không được ủy quyền.

**3.1.2. Với Param**

*****Hình 13a. Đoạn code hoàn chỉnh chương trình kiểm thử trường hợp thừa, thiếu và sai Param với POST và PUT*

**

*Hình 13b. Đoạn code hoàn chỉnh chương trình kiểm thử trường hợp thừa, thiếu và sai Param với POST và PUT*

- Biến c, u với giá trị khởi tạo là None để lưu trữ dữ liệu được sử dụng trong các test case.

- Test POST với các trường hợp thừa thiếu và sai param:

+ Sử dụng fixture param\_error1 với các tham số được đọc từ các tệp JSON chứa các trường hợp thừa thiếu và sai param trong yêu cầu POST.

+ Trong hàm test\_post\_param(param\_error1), tiến hành gọi API POST với các tham số từ fixture và kiểm tra xem API trả về mã trạng thái 201 Created hay không.

- Test PUT với các trường hợp sai và thừa tham số:

+ Sử dụng fixture param\_error2 với các dữ liệu được đọc từ các tệp JSON chứa các trường hợp sai và thừa param trong yêu cầu PUT.

+ Trong hàm test\_put\_param(param\_error2), trước tiên gọi hàm post() để tạo một bản ghi mới và lưu id của nó vào biến toàn cục id.

+ Tiếp theo, thực hiện gọi PUT với các tham số từ fixture và kiểm tra xem API trả về mã trạng thái 200 OK hay không.

* ***Chạy chương trình***

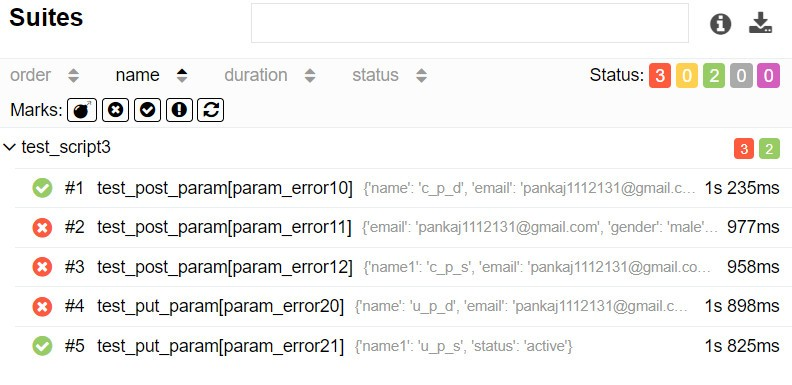
**- Nhập lệnh:**

pytest -s -v .\test\_script3.py --env=prod -–alluredir=

"C:\api-testing\api\_test\script3"

allure serve "C:\api-testing\api\_test\script3"

* **Kết quả**

****

*Hình 14. Kết quả chạy chương trình kiểm thử trường hợp thừa, thiếu và sai Param với*

*POST và PUT*

* Kết quả trên cho thấy thành công với những trường hợp POST thừa Param và PUT sai Param (với dữ liệu sai Param sẽ không được sửa đổi), còn lại đều xảy ra lỗi 422 – Lỗi xác thực dữ liệu không thành công.

**3.1.3. Với Data**

****

*Hình 15. Đoạn code hoàn chỉnh chương trình kiểm thử trường hợp lỗi Data với phương thức POST*

- Biến c\_u với giá trị khởi tạo là None để lưu trữ dữ liệu được sử dụng trong các test case.

- fixture data\_error: chứa dữ liệu gửi kèm theo yêu cầu POST, bao gồm các trường hợp dữ liệu không hợp lệ như trùng ID, trùng email, hoặc email sai định dạng. Mỗi lần chạy test case, fixture sẽ trả về một biến dữ liệu khác nhau từ các tập tin JSON tương ứng.

- test\_post\_data: Test case này sử dụng dữ liệu được trả về từ fixture data\_error để thực hiện yêu cầu POST đến API. Nếu dữ liệu là trùng ID, test case sẽ tự động tạo một email mới và gửi yêu cầu POST. Sau đó, yêu cầu POST được gửi đi và phản hồi được kiểm tra để đảm bảo rằng trạng thái phản hồi là 201 (Created).

* ***Chạy chương trình***

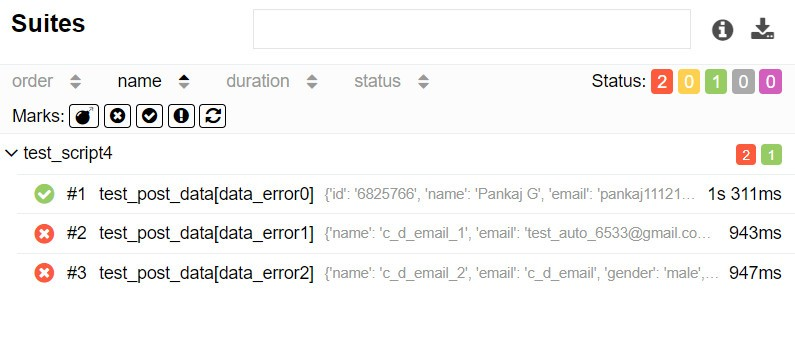
**- Nhập lệnh:**

pytest -s -v .\test\_script4.py --env=prod -–alluredir=

"C:\api-testing\api\_test\script4"

allure serve "C:\api-testing\api\_test\script4"

* **Kết quả**

****

*Hình 16. Kết quả chạy chương trình kiểm thử trường hợp lỗi Data với phương thức POST*

* Có thể thấy thành công với những trường hợp POST trùng id (Ghi đè một id mới), còn lại đều xảy ra lỗi 422 – Lỗi xác thực dữ liệu không thành công.

**3.2 Kiểm tra hiệu suất của API**

* Bước 1: Tạo một file locustfile.py và khai báo module Locust và import các thành phần cần thiết .

From locust import HttpUser, task

**- HttpUser:**

+ Module “HttpUser” là một class trong thư viện Locust, được sử dụng để định nghĩa hành vi của một user trong kịch bản kiểm thử tải.

+ User được tạo bằng cách kế thừa từ class này và mô phỏng hành vi của một user thực sự khi truy cập vào các URL và thực hiện các thao tác trên các ứng dụng web.

**- Task:**

+ Module task là một decorator trong thư viện Locust, được sử dụng để đánh dấu một method trong class HttpUser là một task mà user có thể thực hiện.

+ Các task được định nghĩa để mô phỏng các hành động cụ thể mà một user có thể thực hiện khi sử dụng ứng dụng web, chẳng hạn như nhấp chuột vào các liên kết, điền vào mẫu đăng nhập, hoặc gửi yêu cầu HTTP đến server.

* Bước 2:  Khởi tạo một class UserBehavior chứa các phương thức hay là các hành vi mà người dùng sẽ tương tác với API.

- API được sử dụng ở đây là một API với dữ liệu ảo (fake data) có địa chỉ URL là **http:\\gorest.co.in.**

- Truy cập vào URL, người dùng sẽ thấy các endpoint chứa các data có sẵn có thể sử dụng ("endpoint" thường được hiểu là một điểm cuối của một dịch vụ web hay một ứng dụng. Đây là nơi mà các yêu cầu HTTP được gửi tới để tương tác với dịch vụ hoặc ứng dụng đó. Endpoint thường được định nghĩa bằng một URL duy nhất.).

class UserBehavior(HttpUser):

@task

def get\_tests(self):

self.client.get("/public/v2/users")

@task

def post\_tests(self):

data = {

"id": "5555649",

"name": "load testing",

"email": "api@gmail.com",

"gender": "male"

}

self.client.post("/public/v2/users", json=data)

- Như đã giới thiệu ở trên, trước mỗi phương thức mà user dùng để tương tác với API người dùng chèn thêm decorater “@task” để đánh dấu bắt đầu các hành động. Tiếp theo, bắt đầu định nghĩa các phương thức cơ bản và ở ví dụ trên là 2 phương thức GET và POST. API sử dụng ở đây là một API với dữ liệu ảo (fake data) [**http://gorest.co.in**](http://gorest.co.in) **.**

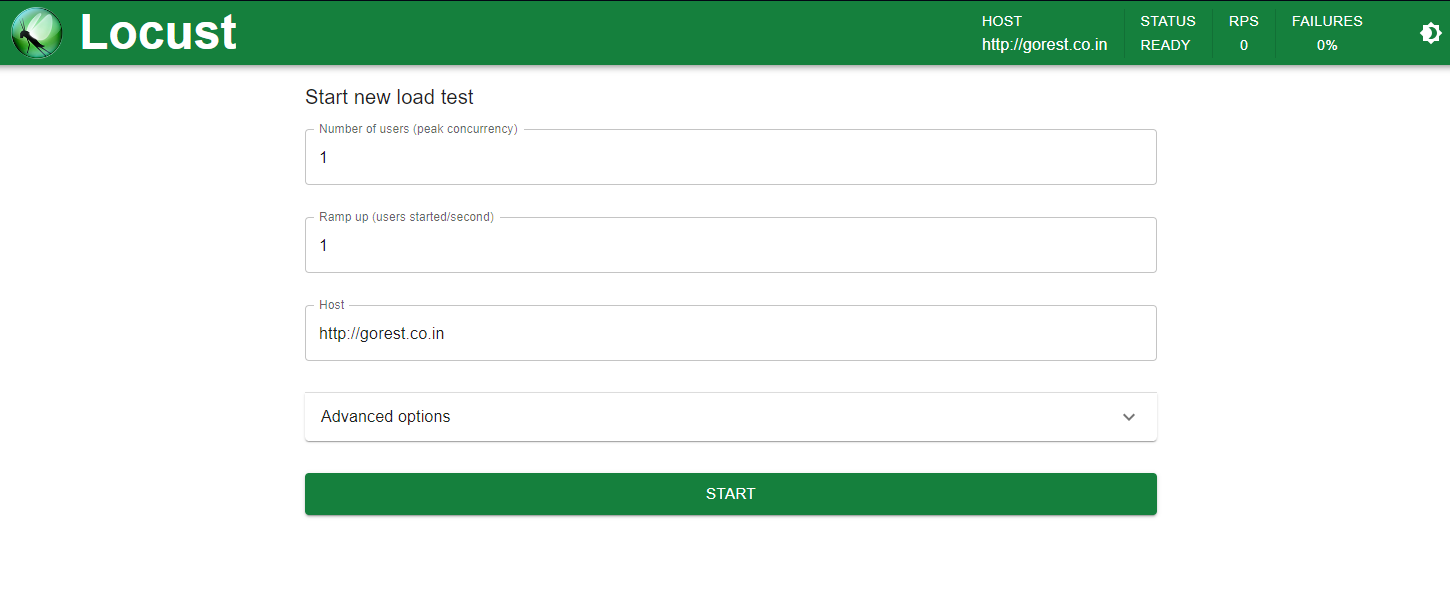
+ Hàm **get\_tests** có đối số truyền vào là “self”. Giá trị này đại diện cho một HttpUser hoặc nói cách khác là một người dùng ảo khi thực hiện một task. Trong phương thức này có một thuộc tính là client (là một object dạng HttpSession) để người dùng có thể gửi những request với định dạng khác nhau, và tùy chỉnh request theo đó.

+ Tương tự như hàm **get\_tests**, hàm **post\_tests** người dùng cũng có đối số truyền vào là “self” và chứa thuộc tính client. Vì phương thức post dùng để yêu cầu tạo mới một dữ liệu trong database nên trong hàm **post\_tests** sẽ khởi tạo một biến data chứa các thông tin cơ bản tùy người dùng nhập vào và biến data này sẽ được ép kiểu sang kiểu dữ liệu JSON ( kiểu dữ liệu phổ biến dùng cho web ).

* Bước 3: Tiến hành chạy lệnh sau

locust --host http://gorest.co.in/ -f locustfile.py

Nhấp chọn vào liên kết [**http://localhost:8089**](http://localhost:8089) để chuyển đến giao diện Web Locust.



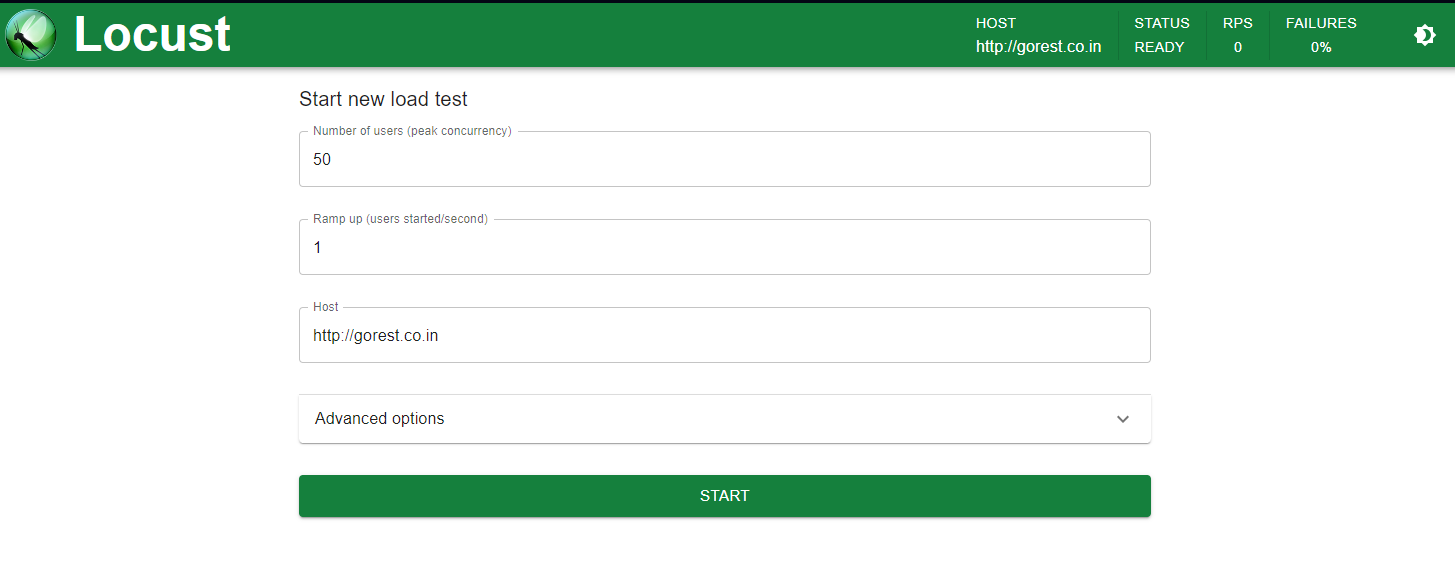
*Hình 17. Giao diện Web Locust*

Tại đây người dùng có thể thấy các thông số hiển thị trên màn hình như số lượng user truy cập tối đa, số lượng user truy cập đồng thời mỗi giây và URL mục tiêu. Dựa vào các thông số trên người dùng có thể chia ra các trường kiểm thử để từ đó đưa ra so sánh và nhận xét về mức độ chịu tải của API.

* Trường hợp 1: Set giá trị ô **Number of user** tức là số lượng người dùng ví dụ như 10, 100, 200, 1000… và giữ nguyên giá trị ô **Ramp up** (số lượng người dùng truy cập vào mỗi giây) sau đó quan sát số lượng request được gửi.
* Trường hợp 2: Set giá trị ô **Ramp up** tức là số lượng người dùng truy cập vào mỗi giây ví dụ như 10, 20, 30 người cùng lúc truy cập.

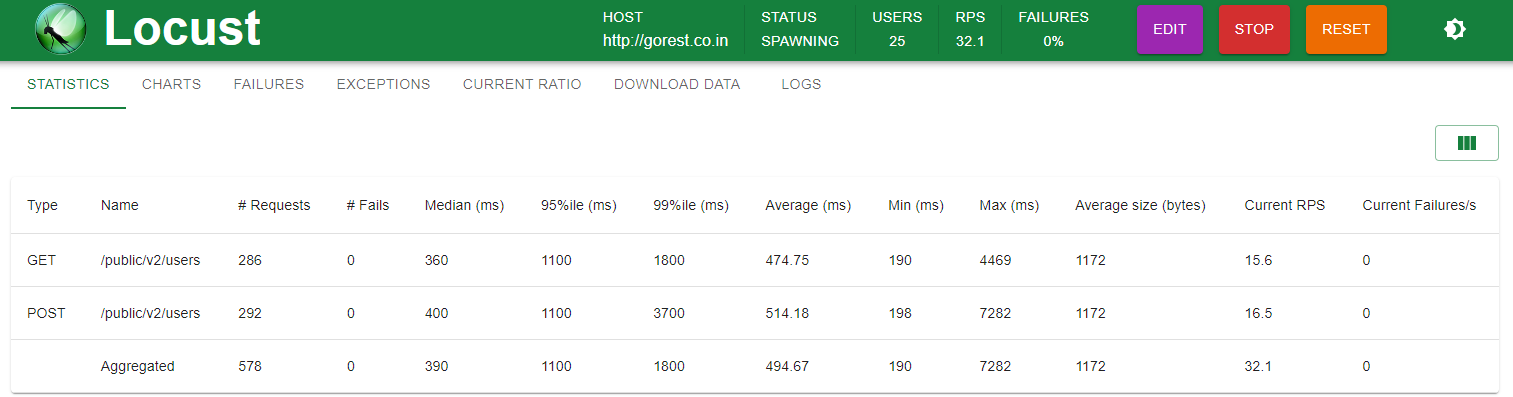
Hãy xét từng trường hợp:

- *Trường hợp 1*: Set giá trị ô **Number of user** bằng 50 và giữ nguyên giá trị ô **Ramp up.**



*Hình 18. Ví dụ trường hợp 1*

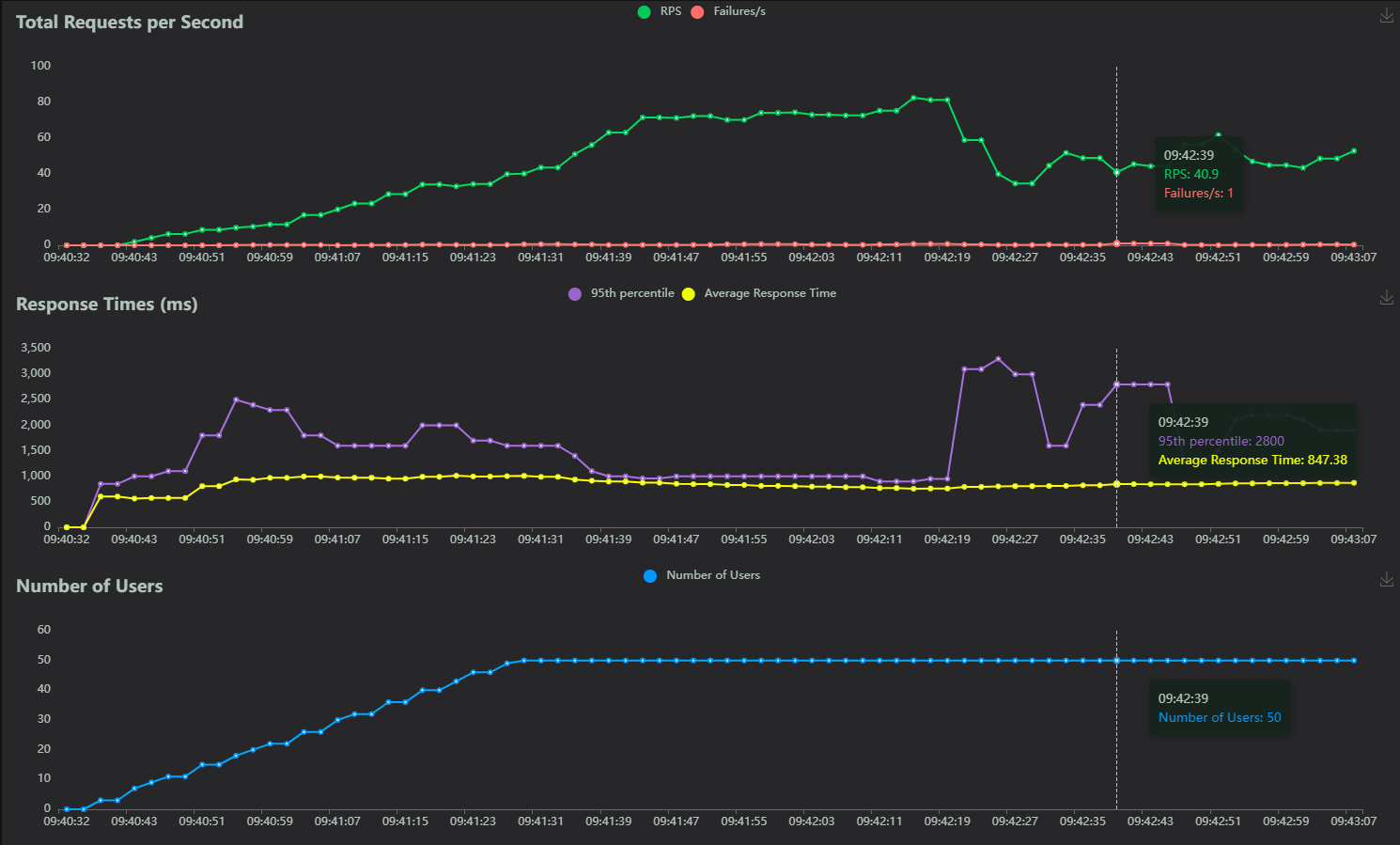
Click **START** và quan sát:



*Hình 19. Thống kê về các phương thức được test*

Tại đây người dùng có thể thấy tab thống kê trực tiếp về API mục tiêu, nó hiển thị các thông số như số lượng request mà người dùng thực hiện, lỗi khi người dùng tương tác với API và thông tin về thời gian phản hồi của API. Bên cạnh tab thống kê, có thể thấy các tab khác như biểu đồ các thông số, chi tiết về các lỗi mà người dùng gặp phải, các ngoại lệ thậm chí locust còn lưu cả các log khi tester hoặc pentester thực hiện việc kiểm thử.

Chuyển sang bên tab charts để thấy rõ hơn:



*Hình 20. Biểu đồ thể hiện các thông số khi thực hiện Load Test khi giữ nguyên giá trị Ramp up*

Ở trên là kết quả khi set số lượng user tối đa là 50. Người dùng có thể thấy với 50 người dùng cùng lúc thì API vẫn làm việc rất ổn định, số lương request lỗi cũng rất ít và hầu như không có sự chậm trễ về phản hồi nào. Tuy nhiên nếu người dùng thử tăng số lượng người dùng tối đa lên và set up số lượng người truy cập mỗi giây tăng lên thì kết quả có sự thay đổi. Hãy cùng nhìn hình bên dưới:



*Hình 21. Chỉnh sửa số lượng User và giá trị Ramp up*

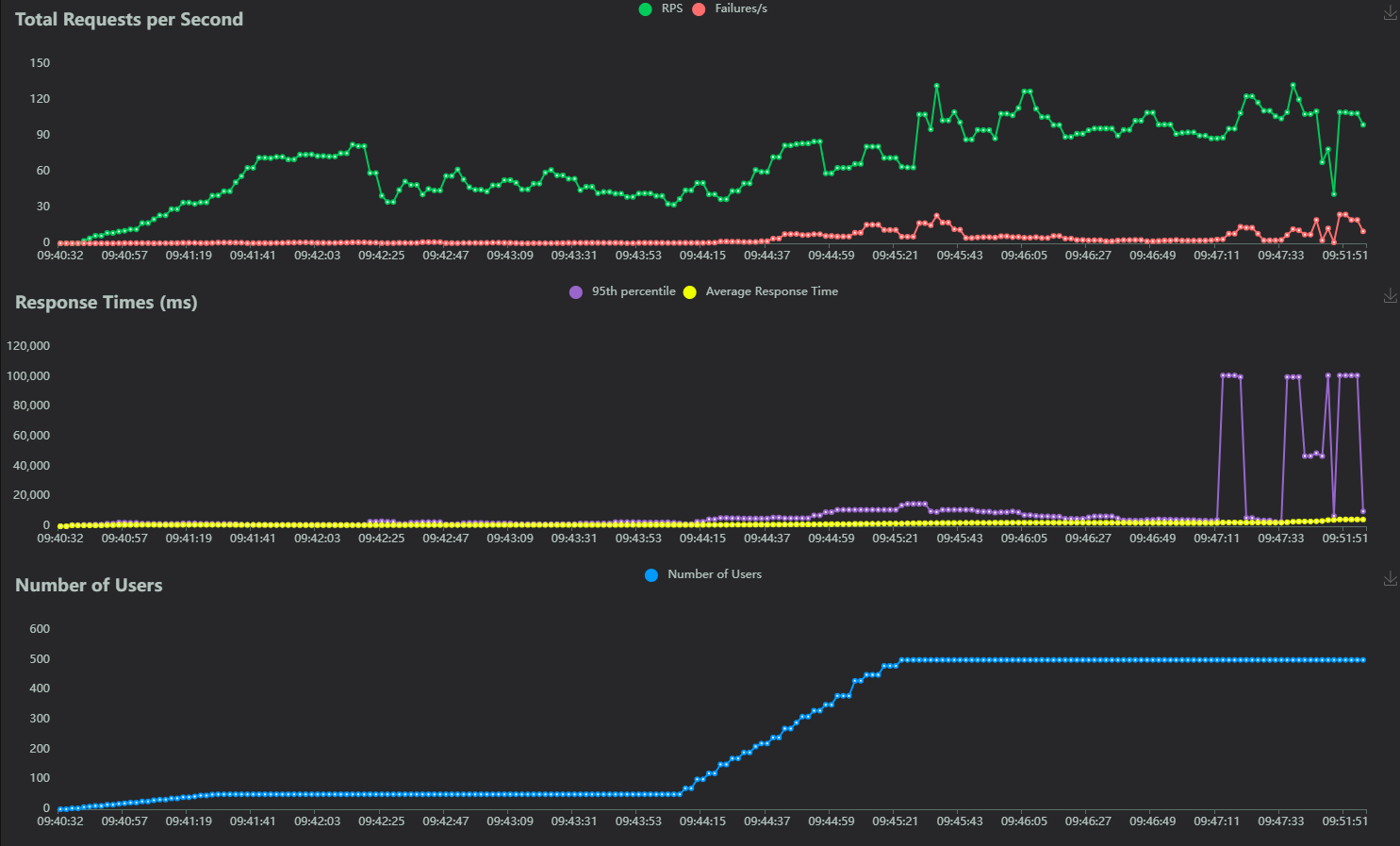
A screenshot of a computer screen

Description automatically generated

*Hình 22a. Biểu đồ thể hiện các thông số khi thực hiện Load Test khi giá trị*

*Ramp up thay đổi*

Sau khi tăng số lượng người dùng lên thì số lương request đến API cũng tăng theo và tất nhiên đi kèm với những request thành công sẽ là những request lỗi do số lương người dùng lớn. Thời gian phản hồi của API cũng tăng theo nhưng nhìn chung hoạt động của API vẫn rất ổn định không gây ra nhiều gián đoạn cho người dùng.



*Hình 22b. Biểu đồ thể hiện các thông số khi thực hiện Load Test khi giá trị*

*Ramp up thay đổi*

**KẾT LUẬN**

* Thông qua quá trình tìm hiểu về việc xây dựng công cụ kiểm thử API và các chức năng, nhóm đã đúc rút ra một số kết luận chung nhất về kết quả đạt được:
  + Hiểu biết về API và các công dụng mạng lại trong việc gọi API.
  + Hiểu biết về kiểm thử API và xây dựng công cụ kiểm thử với các kịch bản cơ bản.
  + Nâng cao tính tự động của công cụ và linh hoạt trong các môi trường.
  + Có thể truy xuất ra báo cáo để có những đánh giá, cái nhìn tổng thể về các testcase.
* Tuy nhiên bên cạch những điểm đã thu hoạch được thì trong quá trình tìm hiểu vẫn còn một số hạn chế về kĩ năng và kiến thức công cụ được nhóm xây dựng cũng chưa đi sâu khai thác được toàn bộ vấn đề và chưa hoàn toàn tối ưu trong việc kiểm thử.
* Định hướng phát triển trong thời gian tới, nhóm muốn phát triển hơn trong bộ công cụ kiểm thử API có thể đi sâu để khắc phục những hạn chế còn tồn đọng.Mong thầy đưa ra những nhận xét và góp ý để chúng em có thể hoàn thiện hơn nữa những kĩ năng, kiến thức nói chung và bài báo cáo nói riêng.
* Source Code: https://github.com/nbn-03/testing\_api/tree/main/api-testing

**TÀI LIỆU THAM KHẢO**

[1] <https://anhtester.com/blog/blog/so-luoc-ve-api-testing>

[2] [Distributed load generation — Locust 2.24.2.dev17 documentation](https://docs.locust.io/en/latest/running-distributed.html)

[3] [API là gì? - Giải thích về Giao diện lập trình ứng dụng - AWS (amazon.com)](https://aws.amazon.com/vi/what-is/api/)

[4][What Is API Testing? | Types, Tools, Frameworks (altexsoft.com)](https://www.altexsoft.com/blog/api-testing/)

[5][What is API Load Testing? | LoadNinja Blog](https://loadninja.com/articles/what-is-api-load-testing/)

[6] [HTTP Request là gì? Các phương thức HTTP request (viblo.asia)](https://viblo.asia/p/http-request-la-gi-cac-phuong-thuc-http-request-6J3Zgy6A5mB)