1. **What is testing?**

Kiểm thử là một quá trình đánh giá độ chính xác và chất lượng của phần mềm có đúng với những yêu cầu được chỉ định và khớp với những mong muốn của khách hàng hay không.

1. **Why is testing?**

Hầu hết chúng ta đều đã từng trải qua việc sử dụng phần mềm mà những hoạt động của nó không như mong đợi. Việc này dẫn đến nhiều vấn đề bao gồm việc hao tốn thời gian, chi phí và danh tiếng của doanh nghiệp và thậm chí có thể gây thương tích hoặc tử vong. Đó là lý do vì sao kiểm thử phần mềm được sinh ra, kiểm thử phần mềm là cách để đánh giá chất lượng phần mềm và giảm rủi ro trong hoạt động của phần mềm.

1. **Why is testing necessary? How is the test enough?**
   1. **Why is testing necessary?**

Kiểm thử giúp ngăn chặn những rủi ro, lỗi, giảm chi phí và tăng sự hài lòng của khách hàng đối với sản phẩm hoặc dịch vụ thông qua việc tìm ra việc tìm ra lỗi và lỗ hỏng có thể ảnh hưởng đến các chức năng của hệ thống.

* 1. **How is the test enough?**

Không có một định nghĩa cụ thể nào áp dụng cho mọi trường hợp kiểm thử bao nhiêu là đủ và có thể kết thúc, nó phụ thuộc vào nhiều yếu tố như là độ phức tạp của phần mềm, yêu cầu chất lượng, ràng buộc về thời gian và ngân sách, và mức độ chấp nhận rủi ro của các bên liên quan. Tuy nhiên, trong một số trường hợp thì công việc kiểm thử sẽ kết thúc: Tester đã thực hiện hết ý tưởng kiểm thử được đưa ra, thời gian cho việc kiểm thử đã hết, …

Điều quan trọng là cần phải phát triển kế hoạch kiểm thử nêu lên tổng quát về phương pháp kiểm thử, các test cases và các thứ khác để đảm bảo rằng kiểm thử có độ bao phủ lớn.

1. **Seven testing principles**
2. Kiểm thử chỉ cho thấy sự hiện diện của lỗi.

Kiểm thử chỉ cho thấy sự hiện diện của defects chứ không thể chứng minh rằng sản phẩm không còn lỗi.

1. Kiểm thử toàn diện là không thể

Việc kiểm thử toàn diện là không khả thi, thay vào đó sử dụng rủi ro và ưu tiên để tập trung nỗ lực kiểm thử.

1. Việc kiểm thử nên được thực hiện sớm nhằm tiết kiệm thời gian và chi phí

Việc kiểm thử nên được thực hiện sớm giúp giảm thiểu và loại bỏ sự thay đổi gây tốn kém.

1. Tester cần phân cụm lỗi

Một số lượng nhỏ các chức năng hoặc module chứa hầu hết các lỗi được phát hiện trong quá trình test trước khi phát hành sản phẩm.

1. Cần đề phòng ‘nghịch lý thuốc trừ sâu’

Các test cases cần được xem xét và sửa đổi thường xuyên, tạo ra nhiều test mới để tìm ra nhiều lỗi hơn.

1. Việc kiểm thử phải được thực hiện phụ thuộc theo ngữ cảnh.

Các ngữ cảnh khác nhau sẽ có các cách thực hiện kiểm thử khác nhau.

1. Không có lỗi là ngụy biện

Việc chạy mọi kiểm thử thử để tích cực tìm càng được nhiều lỗi và sửa chữa lỗi được tìm thấy để đảm bảo hệ thống hoạt động không có lỗi là hoàn toàn không thể.

1. **Test process**
2. Lập kế hoạch kiểm thử

* Input: Các tài liệu mô tả sản phẩm được kiểm thử
* Output: Kế hoạch kiểm thử mô tả kỹ thuật tiếp cận, phạm vi và tài nguyên của hoạt động kiểm thử.

1. Giám sát và kiểm soát kiểm thử

* Input: Các thông số ước tính của kế hoạch kiểm thử và các thông số của tiến độ thực tế của dự án theo kế hoạch.
* Output: Báo cáo tiến độ kiểm thử bao gồm những sai lệch so với kế hoạch và thông tin để hỗ trợ mọi quyết định dừng kiểm thử.

1. Phân tích kiểm thử

* Input: Cơ sở kiểm thử được phân tích để xác định các điều kiện kiểm thử, mục tiêu kiểm thử.
* Output: Tài liệu mô tả thông số kỹ thuật thiết kế kiểm thử ghi lại các điều kiện, mục tiêu kiểm thử.

1. Thiết kế kiểm thử

* Input: Các specification cho kiểm thử.
* Output: Môi trường kiểm thử bao gồm phần cứng, phần mềm và cấu hình mạng cần thiết cho để thực hiện các test case.

1. Triển khai kiểm thử

* Input: Môi trường kiểm thử bao gồm phần cứng, phần mềm và cấu hình mạng cần thiết cho để thực hiện các test case.
* Output: Các test case.

1. Thực thi kiểm thử

* Input: Các test case.
* Output: Kết quả kiểm thử..

1. Kết thúc kiểm thử

* Input: Kết quả kiểm thử.
* Output: Báo cáo tổng hợp kiểm thử.

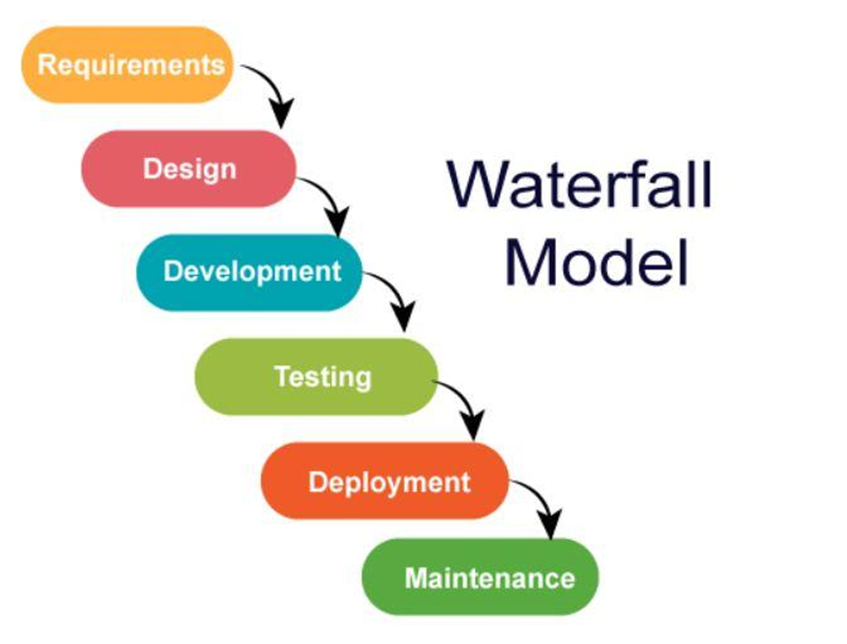
1. **What are Automation testing and Manual testing? Compare Automation and Manual testing.**

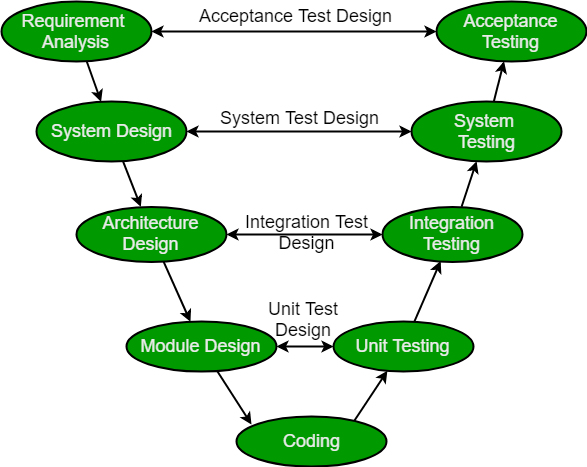
**Kiểm thử tự động** liên quan đến việc thực hiện các test cases với việc sử dụng script, code và các công cụ, phần mềm hỗ trợ. Trong khi đó, **Kiểm thử thủ công** liên quan đến việc chạy các test cases bởi con người mà không dùng đến các công cụ hoặc script.

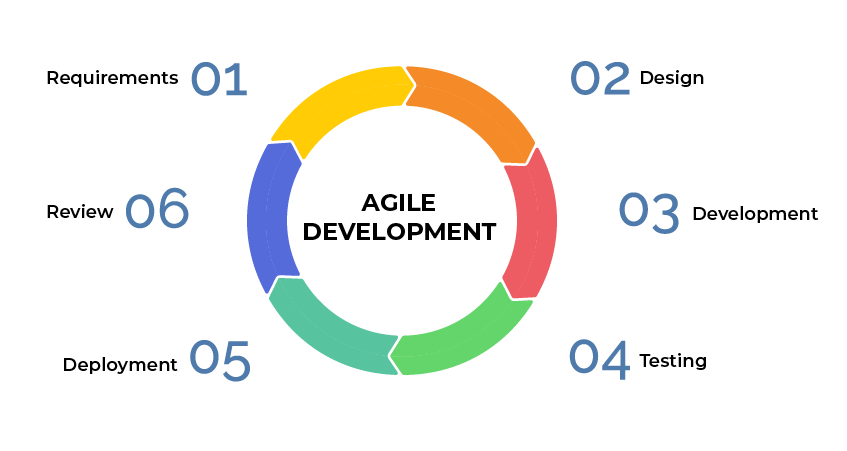
|  |  |  |
| --- | --- | --- |
|  | Automation testing | Manual testing |
| Loại kiểm thử | Hộp đen, hộp trắng | Hộp đen |
| Tốc độ thực hiện | Nhanh hơn | Chậm hơn |
| Chi phí đầu tư ban đầu | Cao | Thấp |
| Chi phí bảo trì | Thấp hơn | Cao hơn |
| Độ bao phủ kiểm thử | Cao hơn | Thấp hơn |
| Độ chính xác | Cao | Thấp |
| Kịch bản kiểm thử | Được xác định trước | Người dùng chỉ định |
| Kỹ năng yêu cầu | Kỹ năng về kỹ thuật | Kỹ năng phân tích |
| Lỗi do con người | Thấp | Cao |
| Gỡ lỗi | Dễ hơn | Khó hơn |
| Thời gian (ban đầu) | Nhiều | Thấp |

1. **Software Development Life Cycle Model: Waterfall model, V model, Agile Model.**

* **Mô hình Waterfall**
  + Là phương pháp tiếp cận tuyến tính và tuần tự tới sự phát triển phần mềm, nhấn mạnh việc hoàn thành 1 giai đoạn trước khi chuyển sang giai đoạn kế tiếp. Các giai đoạn của mô hình Waterfall thường bao gồm thu thập yêu cầu, thiết kế, phát triển, kiểm thử, bảo trì và bảo trì.



* + **Đặc điểm**
    - Là phương pháp tiếp cận tuần tự tuyến tính, mỗi giai đoạn phải được hoàn thành trước khi thực hiện giai đoạn tiếp theo.
    - Mỗi giai đoạn có mục tiêu riêng và những thay đổi trong một giai đoạn có thể khó thực hiện các giai đoạn sau.
    - Mô hình chú trọng nhiều vào tài liệu trong suốt quá trình phát triển phần mềm bao gồm requirement, thiết kế, kiểm thử và bảo trì.
    - Mô hình Waterfall thường không có sự tham gia của khách hàng trong suốt quá trình phát triển và khi có thay đổi trong yêu cầu của khách hàng có thể khó để thực hiện.
    - Mô hình không phù hợp cho dự án lớn hoặc phức tạp vì khó đoán được tất cả yêu cầu và thay đổi trong quá trình phát triển phần mềm.
  + **Ưu điểm**
    - Có cấu trúc đơn giản vì mỗi bước có một mục tiêu cụ thể và mang lại một kết quả cụ thể.
    - Kết quả có thể dự đoán trước do Waterfall là mô hình tuần tự, đảm bảo yêu cầu sản phẩm được xác định rõ ràng.
  + **Nhược điểm**
    - Không phù hợp cho dự án yêu cầu sự linh hoạt và các yêu cầu thay đổi.
    - Sự tham hạn chế của khách hàng trong dự án có thể dẫn đến sản phẩm không đáp ứng nhu cầu của khách hàng.
* **Mô hình V**
  + Là mô hình phát triển phần mềm mở rộng của mô hình Waterfall. Mô hình này được biểu diễn dưới dạng sơ đồ hình chữ V, bên trái đại diện cho giai đoạn thiết kế, bên phải đại diện cho giai đoạn kiểm thử.
  + **Đặc điểm**
    - Mô hình tổ chức quy trình phát triển phần mềm thành các giai đoạn tuần tự được thực hiện theo trình tự chữ V.
    - Mô hình nhấn mạnh việc kiểm thử sớm, tích hợp kiểm thử và phát triển, kiểm thử kỹ lưỡng để giảm thiểu các lỗi và đảm bảo rằng phần mềm đáp ứng các yêu cầu của nó.
    - Mô hình yêu cầu nhóm phát triển lập ra kỳ vọng rõ ràng cho từng giai đoạn phát triển để đảm bảo rằng quy trình phát triển phù hợp với kết quả mong đợi và giúp phát hiện sớm các lỗi tiềm năng.
    - Sử dụng validation qua việc mỗi giai đoạn phát triển phải hoàn thành trước khi chuyển sang giai đoạn khác.
  + **Ưu điểm**
    - Đơn giản và dễ dàng sử dụng.
    - Hoạt động kiểm thử cũng như lập kế hoạch và thiết kế kiểm thử diễn ra tốt trước khi coding sẽ tiết kiệm được thời gian.
    - Cơ hội thành công cao hơn Waterfall model.
    - Các lỗi được tìm thấy ở giai đoạn đầu và tránh các lỗi đi xuống.
  + **Nhược điểm**
    - Thiếu sự linh hoạt vì mỗi giai đoạn của mô hình phải hoàn thành trước khi chuyển sang giai đoạn khác.
    - Mất nhiều thời gian cho lên kế hoạch và thiết kế dẫn đến chậm trễ.
    - Không phù hợp cho dự án nhỏ yêu cầu thời gian ít.
    - Nhóm kiểm thử phải tham gia dự án từ đầu, có thể là một thách thức đối với một số tổ chức.
* **Mô hình Agile**
  + Là một phương pháp tiếp cận lặp đi lặp lại và gia tăng để phát triển phần mềm, mang lại giá trị cho khách hàng thông qua việc đánh giá liên tục và điều chỉnh các yêu cầu, kế hoạch và kết quả. Là mô hình phổ biến được dùng trong phát triển phần mềm nhưng cũng có thể áp dụng cho các lĩnh vực khách của quản lý dự án.



* + **Đặc điểm**
    - Mô hình quản lý dự án linh hoạt ủng hộ sự thay đổi và chấp nhận sự thích ứng liên tục trong suốt vòng đời của dự án.
    - Mô hình chia dự án thành các phần nhỏ và được gọi là sprint, mỗi sprint là một dự án nhỏ, tạo ra phần gia tăng của sản phẩm cung cấp giá trị cho khách hàng ở mọi bước của quá trình.
    - Khuyến khích sự hợp tác, giao tiếp giữa các thành viên, khách hàng và các bên liên quan.
    - Nhấn mạnh giá trị của mô hình và tài liệu như theo cách gọn nhẹ và hiệu quả.
    - Mục đích tạo ra giá trị cho khách hàng tại mỗi giai đoạn. Yêu cầu, kế hoạch và kết quả được đánh giá liên tục để đảm bảo tiến độ của dự án.
    - Mô hình ưu tiên sự tham gia của khách hàng trong suốt vòng đời dự án để đáp ứng các yêu cầu và mong đợi.
    - Mô hình được thiết kế để có thể linh hoạt và thích ứng với các yêu cầu thay đổi, cho phép các nhóm phản hồi đánh giá và điều chỉnh nhanh chóng.
    - Mô hình ưu tiên vào chất lượng hơn là số lượng với việc tập trung phân phối sản phẩm chất lượng cao cuối mỗi sprint.
  + **Ưu điểm**
    - Mang đến sự hài lòng của khách hàng qua việc liên tục phân phối phần mềm hữu dụng.
    - Cải thiện chất lượng sản phẩm qua cách tiếp cận lặp trong quản lý dự án.
    - Linh hoạt và thích ứng hơn trong quá trình phát triển.
    - Đề cao sự hợp tác giữa khách hàng, lập trình viên và tester.
    - Thời gian đưa ra thị trường nhanh hơn do chu kỳ phát triển ngắn.
    - Đánh giá liên tục từ khách hàng và các bên liên quan cho phép phát hiện và giải quyết vấn đề sớm.
  + **Nhược điểm**
    - Giới hạn dự đoán do tính linh hoạt và thích ứng tự nhiên của mô hình.
    - Tính phục thuộc cao vào sự tham gia khách hàng có thể là một thử thách nếu khách hàng không tham gia đầy đủ hoặc không hiểu rõ các yêu cầu.
    - Thay đổi liên tục trong yêu cầu có thể dẫn đến độ phức tạp và chi phí dự án tăng.
    - Tài liệu có thể không đầy đủ như trong các mô hình khách do nhấn mạnh vào hoạt động của phần mềm hơn tài liệu.

1. **Test Level**

Mức độ kiểm thử là các giai đoạn khác nhau của vòng đời phát triển phần mềm nơi mà việc kiểm thử được thực hiện. Mỗi mức độ kiểm thử yêu cầu một môi trường kiểm thử thích hợp. Có 4 mức độ kiểm thử:

* **Component Testing**:
  + Kiểm thử tập trung vào sự riêng lẻ. Mục tiêu của kiểm thử thành phần là:
    - Xác định các hành vi chức năng và phi chức năng của các phần có như thiết kế và chỉ định hay không.
    - Ngăn chặn defect thoát ra khỏi các mức kiểm thử cao hơn.
* **Integration Testing**:
  + Kiểm thử tập trung vào sự tương tác giữa các thành phần và hệ thống. Mục tiêu của kiểm thử tích hợp:
    - Xác định các hành vi chức năng và phi chức năng của các phần có như thiết kế và chỉ định hay không.
    - Ngăn chặn defect thoát ra khỏi các mức kiểm thử cao hơn.
  + Có 2 loại kiểm thử tích hợp:
    - Kiểm thử tích hợp thành phần: kiểm thử tập trung vào sự tương tác và giao diện giữa các thành phần được tích hợp.
    - Kiểm thử tích hợp hệ thống: kiểm thử tập trung vào sự tương tác và giao diện giữa các hệ thống, packages và microservices.
* **System Testing**:
  + Kiểm thử tập trung vào hành vi và sự tương thích của toàn bộ hệ thống hoặc sản phẩm. Mục tiêu của kiểm thử hệ thống:
    - Xác định các hành vi chức năng và phi chức năng của các phần có như thiết kế và chỉ định hay không.
    - Xác định hệ thống có được hoàn thành và hoạt động như mong đợi không.
    - Ngăn chặn defect thoát ra khỏi các mức kiểm thử cao hơn.
* **Acceptance Testing**:
  + Giống như kiểm thử hệ thống, kiểm thử chấp nhận tập trung vào hành vi và sự tương thích của toàn hệ thống hoặc sản phẩm. Mục tiêu của kiểm thử chấp nhận:
    - Xác định các hành vi chức năng và phi chức năng của các phần có như thiết kế và chỉ định hay không.
    - Xác định hệ thống có được hoàn thành và hoạt động như mong đợi không.
    - Ngăn chặn defect thoát ra khỏi các mức kiểm thử cao hơn.

1. **Test types**

* **Functional Testing**:
  + Kiểm thử chức năng của hệ thống liên quan đến các kiểm thử đánh giá các chức năng mà hệ thống sẽ thực hiện.
  + Các chức năng là “những gì” hệ thống đang làm.
* **Non-Functional Testing**:
  + Kiểm thử phi chức năng đánh giá các đặc tính của hệ thống và phần mềm như khả năng sử dụng, hiệu quả hoạt động hoặc bảo mật.
  + Kiểm thử phi chức năng là kiểm thử “hệ thống hoạt động tốt như thế nào”.
  + Một số kiểm thử phi chức năng phổ biến:
    - *Performance Testing*:
      * Là loại kiểm thử đánh giá cách hệ thống hoạt động dưới một khối lượng công việc hoặc mức tải cụ thể.
      * Mục tiêu là xác định các vấn đề về hiệu suất như tốc độ, khả năng mở rộng và tính ổn định của hệ thống trong các điều kiện khác nhau.
    - *Stress Testing*:
      * Là loại kiểm thử được sử dụng để xác nhận tính ổn định và độ tin cậy của hệ thống trong các điều kiện khắc nghiệt ngoài điểm vận hành bình thường.
      * Mục tiêu chính của stress testing là xác định các giới hạn mà hệ thống phần mềm hoặc phần cứng bị hỏng.
    - *Load Testing*:
      * Là loại kiểm thử hiệu năng mô phỏng tải trong thế giới thực trên phần mềm, ứng dụng hoặc trang web.
      * Mục tiêu của load testing là xác định các vấn đề như độ trễ của hệ thống, thời gian tải trang chậm hoặc sự cố khi các mức lưu lượng truy cập khác nhau đang truy cập vào hệ thống trong quá trình phát triển thay vì sau khi phát hành.
    - *Volume Testing*:
      * Là loại kiểm thử tập trung vào kiểm thử ứng dụng phần mềm với một lượng lớn dữ liệu để đánh giá hiệu suất của nó khi tải dữ liệu lớn.
      * Volume Testing được thực hiện nhằm mục đích xác định cách phần mềm hoạt động khi chịu một khối lượng dữ liệu lớn và liệu nó có thể xử lý và duy trì hiệu suất hay không.
    - *Security Testing*:
      * Là loại kiểm thử được dùng để xác định các rủi ro và lỗ hổng bảo mật tiềm ẩn trong các ứng dụng, hệ thống và mạng.
      * Thực hiện security testing nhằm mục đích phát hiện ra bất kỳ điểm yếu nào có thể bị kẻ tấn công khai thác để truy cập trái phép vào dữ liệu nhạy cảm hoặc làm gián đoạn hoạt động bình thường của hệ thống.
    - *Usability Testing*:
      * Là phương pháp để đánh giá sản phẩm hoặc dịch vụ bằng các kiểm thử nó với người dùng đại diện để đo lường mức độ đáp ứng mong đợi của họ và mức độ dễ sử dụng.
      * Mục tiêu của phương pháp này là xác định mọi vấn đề về khả năng sử dụng, thu thập dữ liệu định tính, định lượng về hành vi và nhận thức của người dùng và xác định mức độ hài lòng chung của người dùng đối với sản phẩm hoặc dịch vụ.
    - *Reliability Testing*:
      * Là loại kiểm thử nhằm đo lường xác suất hoạt động của phần mềm không bị lỗi trong một môi trường cụ thể.
      * Mục tiêu của phương pháp là đảm bảo phần mềm đáng tin cậy, đáp ứng mục đích đã định của nó và có khả năng hiển thị hoạt động không có lỗi.
* **White-box Testing**
  + Kiểm thử hộp trắng đứa ra các kiểm thử dựa trên cấu trúc hoặc triển khai bên trong hệ thống. Cấu trúc bên trong hệ thống bao gồm code, kiến trúc, luồng công việc và/ hoặc dòng dữ liệu trong hệ thống.
* **Change-related Testing**
  + Khi một xảy ra một thay đổi trong hệ thống do sửa lỗi, chức năng mới hoặc thay đổi, việc kiểm thử nên được thực hiện để xác nhận sự thay sửa lỗi hoặc thực thi chức năng đúng đắn và chưa gây ra bất kỳ bất lợi nào không lường trước.
  + Có 2 loại kiểm thử thay đổi liên quan:
    - Kiểm thử xác nhận: xác nhận rằng lỗi ban đầu được sửa chữa thành công.
    - Kiểm thử hồi quy: liên quan đến việc chạy kiểm thử để tìm ra những các lỗi ngoài ý muốn như vậy.

1. **What are static testing and dynamic testing methods? Compare these two testing techniques.**

* **Kiểm thử tĩnh** là được thực hiện mà không thực thi code của ứng dụng phần mềm. Kiểm thử tĩnh là phương pháp kiểm thử để phát hiện lỗi bằng cách xem xét code, các tài liệu yêu cầu, thiết kế,… Kiểm thử tĩnh được thực hiện từ những giai đoạn đầu của quá trình phát triển để tránh lỗi và dễ dàng tìm ra nguồn gốc sự cố, có thể khắc phục dễ dàng.
* **Kiểm thử động** được thực hiện trong suốt quá trình thực thi code. Kiểm thử động giúp tìm và sửa lỗi hiện diện trong phần mềm. Kiểm thử động đảm bảo phần mềm hoạt động đúng theo yêu cầu và mong đợi của người dùng. Nó được thực hiện sau khi thực thi code.

|  |  |
| --- | --- |
| Static testing | Dynamic testing |
| Tập trung vào việc ngăn chặn lỗi. | Tập trung vào việc phát hiện và sửa chữa lỗi. |
| Cung cấp đánh giá về code và tài liệu. | Xác định lỗi và tắc nghẽn trong hệ thống phần mềm. |
| Liên quan đến 1 checklist và quy trình phải tuân thủ. | Yêu cầu test cases để thực thi. |
| Thực hiện trước khi phát hành code. | Thực hiện sau khi phát hành code. |
| Ít tốn kém chi phí. | Tốn nhiều chi phí. |

1. **Test techniques? Compare black box and white box?**

Mục đích của kỹ thuật kiểm thử là giúp cho việc xác định ra điều kiện kiểm thử, test cases và dữ liệu test.

Có 3 kỹ thuật kiểm thử chính:

* **Kiểm thử hộp đen**: Là kỹ thuật kiểm thử dựa trên yêu cầu và chỉ định của phần mềm, tập trung chủ yếu vào đầu vào và đầu ra mà không quan tâm đến cấu trúc bên trong.
* **Kiểm thử hộp trắng**: Là kỹ thuật kiểm thử dựa vào khả năng phân tích kiến trúc, thiết kế chi tiết, cấu trúc bên trong hoặc code. Kiểm thử hộp trắng tập trung vào cấu trúc và tiến trình trong đối tượng kiểm thử.
* **Kiểm thử dựa vào kinh nghiệm**: là kỹ thuật kiểm thử tận dụng kinh nghiệm của lập trình viên, tester và người dùng để thiết kế, thực thi và thực hiện kiểm thử
* So sánh black box và white box testing:

|  |  |  |
| --- | --- | --- |
|  | Black-box testing | White-box testing |
| Định nghĩa | Là kỹ thuật kiểm thử chỉ quan tâm đến đầu vào và đầu ra của đối tượng kiểm thử mà không quan tâm đến kiến trúc bên trong. | Là kỹ thuật tập trung vào kiến trúc và tiến trình bên trong đối tượng kiểm thử. Yêu cầu kiến thức về code, logic và kiến trúc bên trong của hệ thống. |
| Mục tiêu kiểm thử | Xác thực các chức năng của phần mềm mà không yêu cầu bất kỳ kiến thức gì về hoạt động bên trong. | Xác thực các hoạt động bên trong của phần mềm như code, logic và kiến trúc của nó. |
| Mức độ kiểm thử | Thường được thực hiện ở mức Kiểm thử hệ thống, Kiểm thử chấp nhận. | Thường có thể được thực hiện ở tất cả các mức kiểm thử. |
| Ưu điểm | Kiểm thử được thực hiện dưới góc nhìn của người dùng và kỹ thuật này rất dễ sử dụng ngay cả đối với người không có kiến thức về kỹ thuật. | Các test case bao phủ mọi khía cạnh của phần mềm bao gồm các hoạt động bên trong. Có thể phát hiện lỗi bị xót ở kiểm thử hợp đen. |
| Nhược điểm | Các test case không bao phủ hết mọi khía cạnh của phần mềm. Có thể không tìm được các yêu cầu về kiến thức của các hoạt động bên trong của phần mềm. | Có thể phức tạp và tốn thời gian, yêu cầu tester có yêu cầu có kiến thức về hoạt động bên trong của phần mềm. |

1. **Compare verification and validation**

|  |  |
| --- | --- |
| Verification | Validation |
| Tập trung vào quá trình đảm bảo rằng sản phẩm phần mềm đáp ứng các thông số kỹ thuật và yêu cầu. | Tập trung vào việc đánh giá sản phẩm phần mềm để xác định nó có đáp ứng nhu cầu và mong đợi của khách hàng hay không. |
| Sử dụng kỹ thuật kiểm thử tĩnh. | Sử dụng kỹ thuật kiểm thử động. |
| Thực hiện ở giai đoạn đầu của vòng đời phát triển phần mềm như requirements, thiết kế và coding. | Thực hiện ở những giai đoạn cuối của vòng đời phát triển phần mềm như kiểm thử hệ thống và kiểm thử chấp nhận. |
| Mục tiêu của verification là đảm bảo sản phẩm phần mềm được xây dựng theo các thông số kỹ thuật yêu cầu. | Mục tiêu của validation là đảm bảo sản phẩm phần mềm đáp ứng nhu cầu và mong đợi của khách hàng. |
| Thực hiện bởi lập trình viên, tester, BA và QA. | Thực hiện bởi đội kiểm thử gồm testers, BA và khách hàng. |
| Là hoạt động định hướng quá trình. | Là hoạt động định hướng sản phẩm. |

1. **Compare Authentication and Authorization**

|  |  |  |
| --- | --- | --- |
|  | Authentication | Authorization |
| Definition | Quá trình xác minh danh tính của người dùng hoặc thiết bị. | Quá trình cung cấp hoặc từ chối quyền truy cập tài nguyên hoặc hành động dựa trên quyền của người dùng được xác thực. |
| Focus | Thiết lập danh tính của người dùng hoặc thiết bị. | Xác định những gì người dùng được phép truy cập và những hành động họ có thể thực hiện. |
| Verification method | Tên người dùng, mật khẩu, xác minh dấu vân tay, xác thực 2 yếu tố. | Kiểm soát truy cập dựa trên vai trò,kiểm soát truy cập theo thuộc tính, kiểm soát truy cập dựa trên yêu cầu. |

1. **Some popular Automation test tools.**

* **Katalon**
  + Là một công cụ kiểm thử tự động cho phép người dùng tạo ra cà thực hiện kiểm thử tự động cho các ứng dụng desktop, API, web và thiết bị di động.
  + **Đặc trưng**
    - Katalon Studio được thiết kế để tạo ra và tái sử dụng các tập lệnh kiểm thử tự động cho giao diện người dùng mà không cần code.
    - Nó có một giao diện môi trường phát triển tích hợp cho kiểm thử ứng dụng web, API, di động và desktop.
    - Nó cho phép chạy kiểm thử tự động các yếu tố giao diện người dùng bao gồm pop-up, iframe và wait-time.
    - Nó hỗ trợ nhiều ngôn ngữ lập trình như Groovy, Java, Javascript.
    - Nó cũng hỗ trợ khung làm việc định hướng phát triển hành vi (BDD) như Cucumber và JBehave.
    - Nó cho phép dễ dàng tích hợp với các công cụ khác như Git, Jenkins và JIRA giúp sự hợp tác trong nhóm tốt hơn.
    - Nó có các tính năng như kho lưu trữ đối tượng, thử nghiệm theo hướng dữ liệu và tạo báo cáo.
  + **Ưu điểm**
    - Katalon có giao diện người dùng thân thiện và các tính năng dễ sử dụng, phù hợp cho người mới bắt đầu hoặc không có kiến thức lập trình.
    - Cung cấp nhiều tính năng báo cáo có thể tùy chỉnh theo yêu cầu người dùng không giống như Selenium.
    - Không giống như Selenium, Katalon Studio hỗ trợ kiểm thử tự động cho các ứng dụng không dựa trên web.
    - Katalon bao gồm một tập hợp đầy đủ cho một giải pháp kiểm thử tự động hoàn chỉnh.
  + **Nhược điểm**
    - Không giống Selenium, Katalon Studio chỉ hỗ trợ tạo ra tập lệnh sử dụng Java và Groovy điều này có thể là giới hạn cho một số người dùng.
    - Không hỗ trợ cho kiểm thử phân tán, đây có thể là một hạn chế lớn đối với các dự án quy mô lớn.
    - Không thể tự động hóa ứng dụng desktop.
* **Selenium** 
  + Là công cụ kiểm thử tự động phổ biến hỗ trợ đa ngôn ngữ lập trình và nền tảng, và chủ yếu được sử dụng kiểm thử ứng dụng web. Selenium cho phép lập trình viên viết lệnh kiểm thử với các ngôn ngữ lập trình như Java, Python và C#. Nó có thể được dùng cho kiểm thử chức năng, hồi quy và tải của các ứng dụng web. Có 4 thành phần:
    - *Selenium IDE:*
    - *Selenium WebDriver*:
    - *Selenium Grid*:
    - *Selenium Standalone ServerUI:*
  + **Đặc trưng**
    - Selenium có thể được sử dụng ở đa nền tảng như Window, Mac và Linux.
    - Selenium hỗ trợ đa ngôn ngữ: Java, Python và C#
    - Selenium hỗ trợ nhiều web browser: Google Chrome, Firefox, Safari vaf Microsoft Edge…
    - Selenium có thể dễ dàng tích hợp với các công cụ và framework khác: TestNG, JUnit và Cucumber.
  + **Ưu điểm**
    - Selenium là công cụ miễn phí và nguồn mở.
    - Hỗ trợ đa ngôn ngữ, web browser và có thể tích hợp với các công cụ khác để thực hiện kiểm thử tự động.
  + **Nhược điểm**
    - Selenium được thiết kế cơ bản cho kiểm thử web tự động và nó hạn chế để kiểm thử trên thiết bị di động.
    - Kiểm thử Selenium có thể tốn nhiều thời gian đặc biệt khi kiểm thử ứng dụng web phức tạp.
    - Selenium yêu cầu kỹ thuật lập trình để viết lệnh kiểm thử, điều này có thể là cản trở cho một số người dùng không có nền tảng kỹ thuật.
    - Không có tính năng báo cáo tích hợp, có thể bất lợi cho báo cáo kết quả kiểm tra.
* **Jmeter**
  + Là công cụ miễn phí mã nguồn mở chủ yếu được sử dụng cho kiểm thử hiệu năng, kiểm thử tải và kiểm thử căng thẳng của ứng dụng web.
  + **Đặc trưng**
    - Là công cụ miễn phí và nguồn mở có thể được sử dụng cho các loại ứng dụng khác nhau như web, dịch vụ web, cơ sở dữ liệu, shell script…
    - Có gia diện thân thiện người dùng dễ sử dụng và hỗ trợ nhiều ngôn ngữ như Java JavaScript, BeanShell và Groovy.
  + **Ưu điểm**
    - Công cụ kiểm thử miễn phí phù hợp với mọi quy mô doanh nghiệp.
    - Một công cụ nền tảng độc lập vì nó được viết bởi Java và có thể hoạt động trên mọi nền tảng Java hỗ trợ.
    - Hỗ trợ các loại kiểm thử khác nhau như kiểm thử chức năng, tự động và tải.
    - Nó kết hợp tốt với các công cụ khác như Selenium, Eclípe, Jenkins và TeamCity.
  + **Nhược điểm**
    - JMeter có thể có một chút phức tạp để cài đặt và yêu cầu một số kiến thức kỹ thuật để sử dụng hiệu quả.
    - JMeter yêu cầu nhiều bộ nhớ và tài nguyên để vận hành ví thế nó có thể không phù hợp với máy cấu hình thấp.

1. **SQL Fundamental**

* **SQL** (Ngôn ngữ truy vấn có cấu trúc) là ngôn ngữ tiêu chuẩn được sử dụng để quản lý cơ sở dữ liệu quan hệ và thực hiện các hoạt động khác nhau trên dữ liệu được lưu trữ trong đó.
* **SELECT**
  + Câu lệnh SELECT được dùng để chọn dữ liệu từ cơ sở dữ liệu, Dữ liệu trả về được lưu trữ trong bảng kết quả được gọi là tập kết quả.
  + Cú pháp: SELECT column1, ….

FROM table\_name

WHERE condition

* + SELECT DISTINCTđược được dùng để chỉ trả về các giá trị riêng biệt.
* **INSERT**
  + Câu lệnh INSERT được dùng để thêm bản ghi mới vào bảng.
  + Cú pháp: INSERT INTO table\_name

VALUES (value1, value2,...)

* **UPDATE**
  + Câu lệnh UPDATE được dùng để sửa đổi các bản ghi hiện có trong một bảng.
  + Cú pháp: UPDATE table\_name

SET column1 = value1, column2 = value2,...

WHERE condition

* **DELETE**
  + Câu lệnh DELETE được dùng để xóa các bản ghi có trong bảng.
  + Cú pháp: DELETE FROM table\_name

WHERE condition

* **SQL injection**
  + Là một kiểu tấn công mạng khai thác lỗ hổng trong phần mềm của ứng dụng để chèn mã SQL độc hại vào trường nhập, cho phép kẻ tấn công thực thi các lệnh SQL trái phép và giành quyền truy cập vào thông tin nhạy cảm hoặc thao tác dữ liệu.
  + Các cuộc tấn công SQL injection có thể được sử dụng để trích xuất hoặc xóa dữ liệu, bỏ qua xác thực và kiểm soát truy cập, sửa đổi bản ghi cơ sở dữ liệu hoặc thậm chí thực thi các lệnh trên máy chủ bên dưới.
  + Để ngăn chặn các cuộc tấn công SQL injection, các nhà phát triển có thể sử dụng các truy vấn được tham số hóa hoặc các thủ tục được lưu trữ để xác thực đầu vào của người dùng và làm sạch dữ liệu đầu vào trước khi thực hiện các câu lệnh SQL. Ngoài ra, việc cập nhật phần mềm và các biện pháp bảo mật có thể làm giảm khả năng bị tấn công.
  + Một số ví dụ:

SELECT \*

FROM users

WHERE username=’admin OR 1=1;

SELECT \*

FROM users

WHERE username=’admin’ --’;

SELECT \*

FROM users

WHERE username=’admin’; DROP TABLE users;

1. **What is Test Case? Necessary components of a test case?**

* **Test cases** là tập hợp các điều kiện hoặc các bước được thiết kế cho việc kiểm thử tính năng hoặc chức năng cụ thể của một ứng ùng phần mềm, với mục đích xác minh rằng ứng dụng đó hoạt động như mong đợi và đáp ứng các yêu cầu của người dùng cuối.
* **Necessary components of a test case?**

Một test case thường chứa tiêu đề, mô tả, điều kiện tiên quyết, các bước kiểm thử, kết quả mong đợi, kết quả thực tế (khi đã được kiểm thử) và trạng thái của test case.

1. **Equivalence partitioning and Boundary value analysis.**

* **Phân vùng tương đương**: kỹ thuật liên quan đến việc chia các dữ liệu đầu vào thành các phân vùng dữ liệu, bằng cách chọn ra một giá trị đại diện cho từng phân vùng, kỹ thuật này nhằm mục đích giảm các số test cases cần thiết.
* **Phân tích giá trị biên**: là kỹ thuật mở rộng của phân vùng tương đương nhưng chỉ có thể dùng cho phân vùng có thứ tự. Ý tưởng của kỹ thuật này là việc kiểm thử tại các khu vực của biên có khả năng tìm lỗi cao.

1. **What cases does a test case file need to cover?**

* Một file test case nên bao quát tất cả các chức năng, phi chức năng của ứng dụng phần mềm, bao gồm giao diện người dùng, hiệu suất, bảo mật,…

1. **How is Test Case called effective?**

* Test case được gọi là hiệu quả khi:
  + Test case chỉ được sử dụng cho một mục đích.
  + Các test case phải độc lập với nhau.
  + Case đơn giản, rõ ràng.
  + Có thể tái sử dụng.
  + Có thể theo dõi.

1. **Report bug is called effective? Difference between bug, defect, error, fail, failure.**

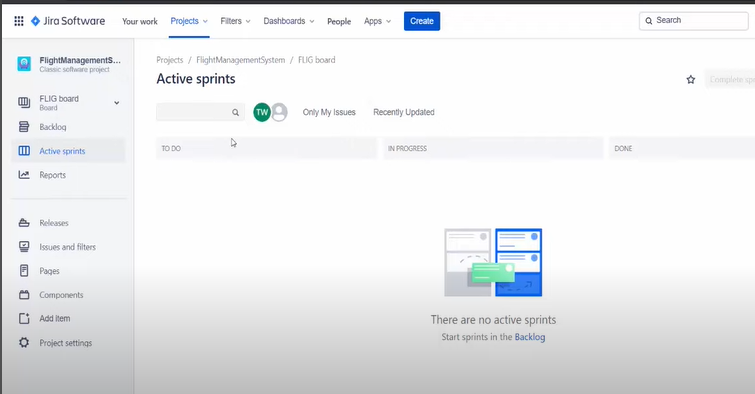
* **Report bug** được gọi là hiệu quả khi:
  + Bug có thể tái tạo:
    - Nếu bug không thể tái tạo thì nó sẽ không bao giờ được sửa chữa.
    - Nên đề cập rõ ràng các bước tài tạo bug.
    - Không giả định hoặc bỏ qua bất kỳ bước tái tạo nào.
    - Bug được mô tả từng bước dễ tái tạo và sửa chữa.
  + Báo cáo bug cần ngắn gọn và cụ thể:
    - Nên viết báo cáo cụ thể vào thẳng vấn đề.
    - Cố gắng tóm tắc vấn đề bằng các từ tối thiểu theo cách hiệu quả.
    - Không kết hợp nhiều vấn đề ngay cả khi chúng có vẻ giống nhau.
* **Bug, defect, error, fail, failure**
  + **Bug** 
    - Là lỗi trong quá trình viết code hoặc một vấn đề không mong muốn khiến phần mềm hoạt động bất thường.
    - Bug có thể xảy ra do lỗi lập trình hoặc lỗi trong thiết kế phần mềm khiến phần mềm bị trục trặc và hoạt động không chính xác.
  + **Defect** 
    - Là một lỗ hổng trong phần mềm có thể khiến phần mềm không đáp ứng được các thông số kỹ thuật dự kiến.
    - Defect có thể phát sinh do code lỗi, các vấn đề về thiết kế hoặc yêu cầu phần mềm.
  + **Error** 
    - Là lỗi do lập trình viên mắc phải khi viết code khiến phần mềm hoạt động không như mong muốn. Một error có thể dẫn đến bug hoặc defect trong phần mềm.
    - Lỗi có thể xảy ra do thiếu hiểu biết về ngôn ngữ lập trình hoặc yêu cầu của phần mềm.
  + **Fault**
    - Là một defect hoặc một error trong phần mềm, có thể làm nó hoạt động bất thường hoặc không đáp ứng các thông số kỹ thuật dự kiến.
    - Fault có thể phát sinh do lỗi phần cứng, sự cố thiết kế phần mềm hoặc sự cố với môi trường phần mềm.
  + **Failure**
    - Là lỗi xảy ra khi phần mềm không thực hiện chức năng dự định của nó.
    - Failure xảy ra do defect hoặc fault trong phần mềm

1. **Management tool: Jira, Redmine, Azure DevOps, Trello.**

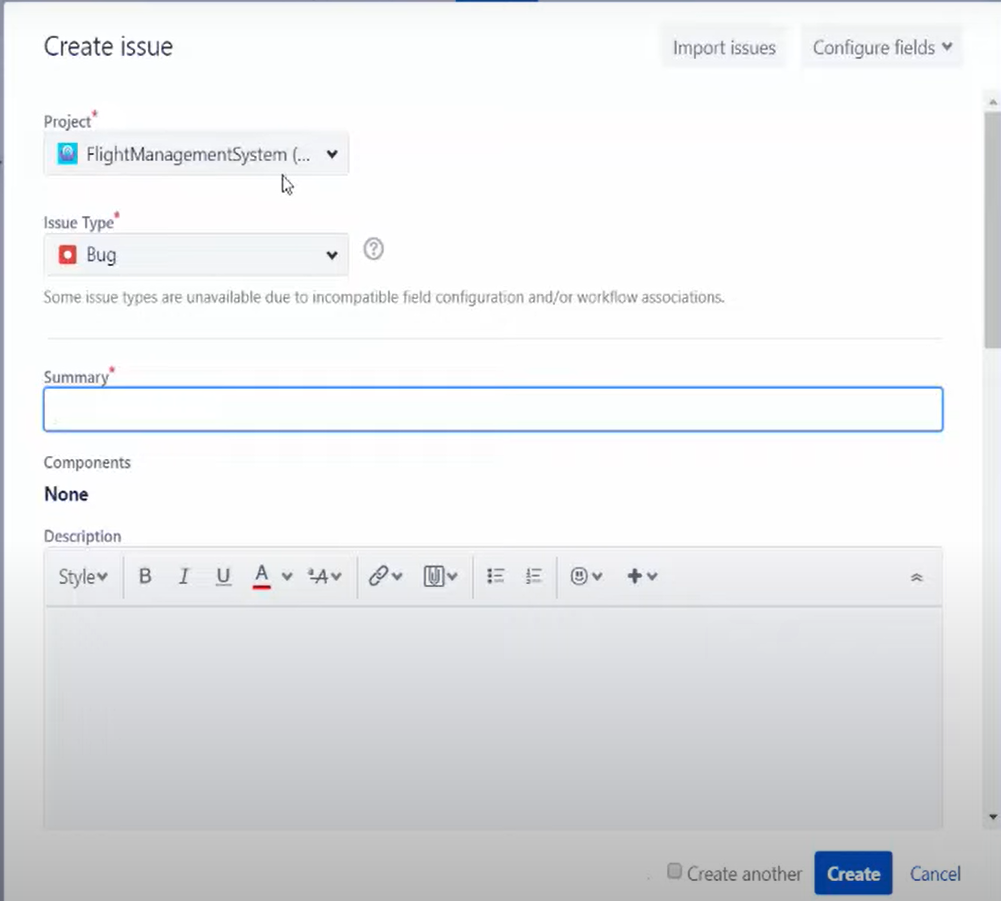
**Testing tool: Postman, JMeter.**

* **Management tools:**
* **Jira**
  + Là công cụ được dùng phổ biến trong kiểm thử phần mềm để theo dõi bug, lỗi và quản lý các tác vụ trong dự án. Nó là một công cụ theo dõi bug cho phép nhà phát triển lập kế hoạch, theo dõi và làm việc nhanh chóng trong dự án phần mềm. Jira cho phép các nhà phát triển tổ chức các tác vụ tài liệu, theo dõi tài liệu quá trình và lên kế hoạch cho tính năng mới được thêm vào hoặc bug được sửa chữa trong lần phát hành tới.
  + Jira được dùng cho các mục đích:
    - Quản lý các yêu cầu và test case.
    - Trong phương pháp Agile.
    - Quản lý dự án.
    - Phát triển phần mềm.
    - Quản lý sản phẩm.
    - Quản lý tác vụ.
    - Theo dõi bug.
  + **Ưu điểm**
    - Giúp theo dõi nhiều tác vụ không quan tâm đó là bug, một số thứ cần theo dõi hoặc bất kỳ lỗi khác.
    - Thuận tiện để giao việc cho các nhóm và để theo dõi tiến độ cũng như các vấn đề.
    - Cung cấp những hiểu biết kịp thời và các dữ liệu khác giúp phát triển các sản phẩm thành công.
    - Có thể tích hợp với nhiều ứng dụng, do đó mở rộng khả năng của công cụ.
    - Hỗ trợ quản lý công việc linh hoạt bằng Scrum và Kanban.
  + **Nhược điểm**
    - Cấu hình phức tạp vì người bắt đầu phải chọn trong các số rất nhiều tính năng để chọn.
    - Không phù hợp cho một tác vụ đơn giản.
    - Thiếu khả năng quản lý chi phí hoặc đánh giá rủi ro.
  + **Báo cáo một bug trên JIRA**

1. Đi đến màn hình của dự án.
2. Chọn ‘Create’ trên thanh công cụ.



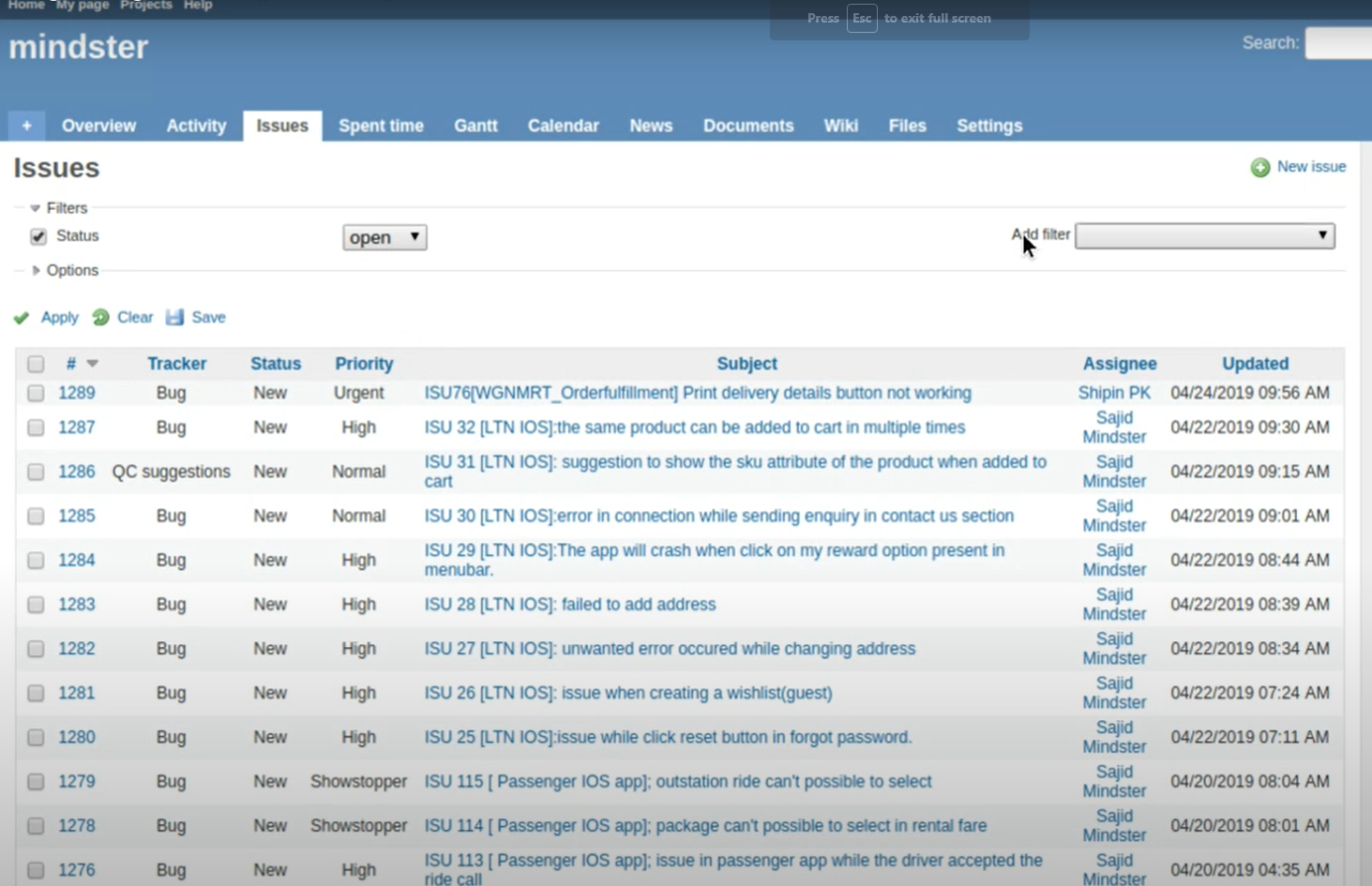
1. Ở phần Issue Type chọn ‘Bug’.



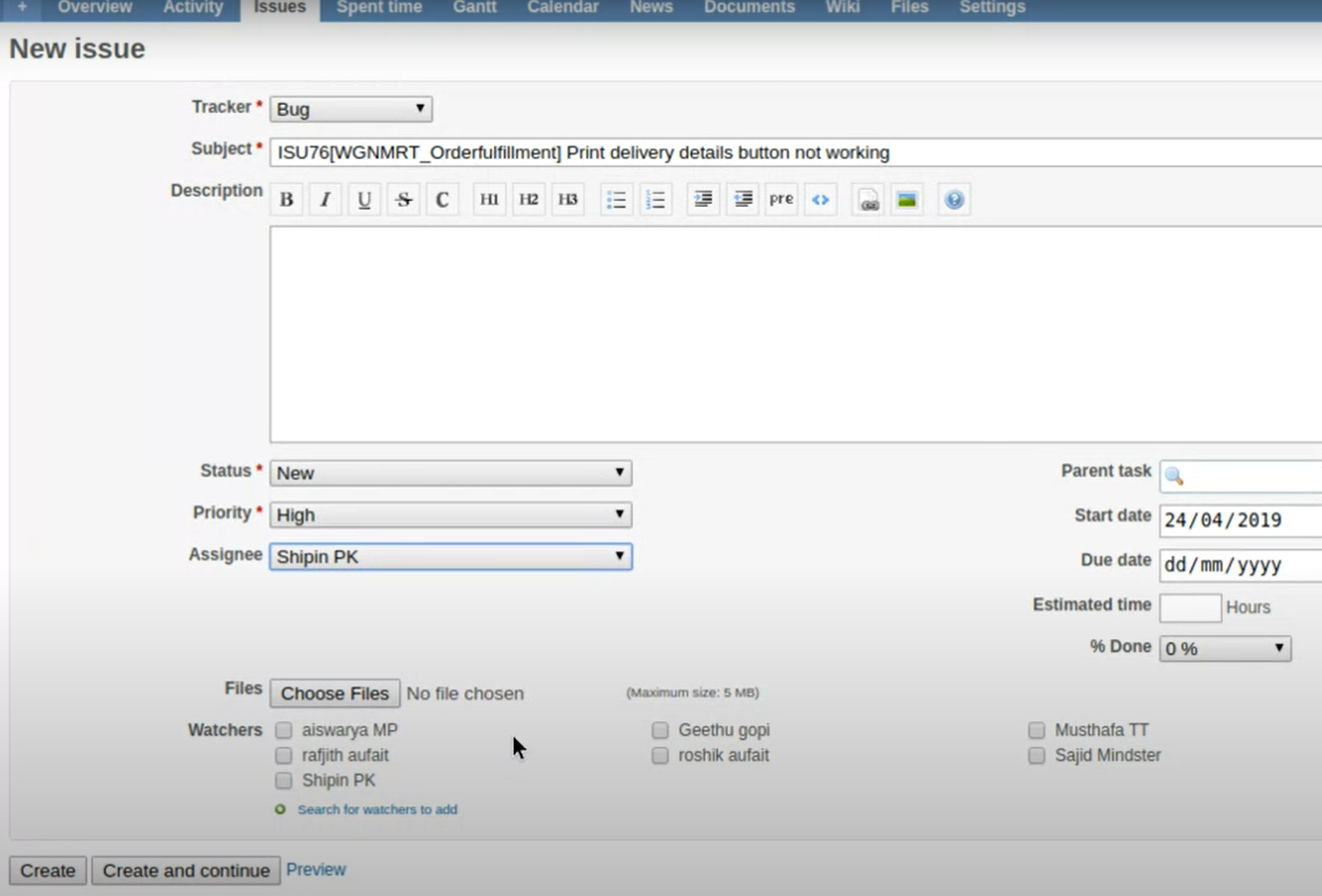
1. Điền các thông tin cần thiết vào các trường, liên kết tới .
2. Chọn ‘Create’.

* **Redmine**
  + Là công cụ quản lý dự án và theo dõi miễn phí nguồn mở. Nó được thiết kế để hỗ trợ người dùng trong việc quản lý nhiều dự án và liên kết các dự án con, cung cấp các tính năng như wiki theo dự án, diễn đàn, theo dõi thời gian và kiểm soát truy cập dựa trên vai trò.
  + **Ưu điểm**
    - Sở hữu nhiều tính năng và dễ dàng quản lý.
    - Công cụ miễn phí nguồn mở.
    - Linh hoạt và có khả năng tùy biến cao.
    - Nền tảng bao gồm theo dõi thời gian, wiki, diễn đàn, kiểm soát truy cập và tích hợp nhiều phiên bản hệ thống điều khiển.
    - Có thể được dùng như một cổng thông tin hỗ trợ cho hệ thống issue raising và ticket.
  + **Nhược điểm**
    - Sử dụng giao diện không trực quan với một số người dùng.
    - Tập trung vào theo dõi bug và issue nên có thể không phải là sự lựa chọn tốt nhất.
  + **Báo cáo bug trên Redmine**

1. Đi đến tab Issues.



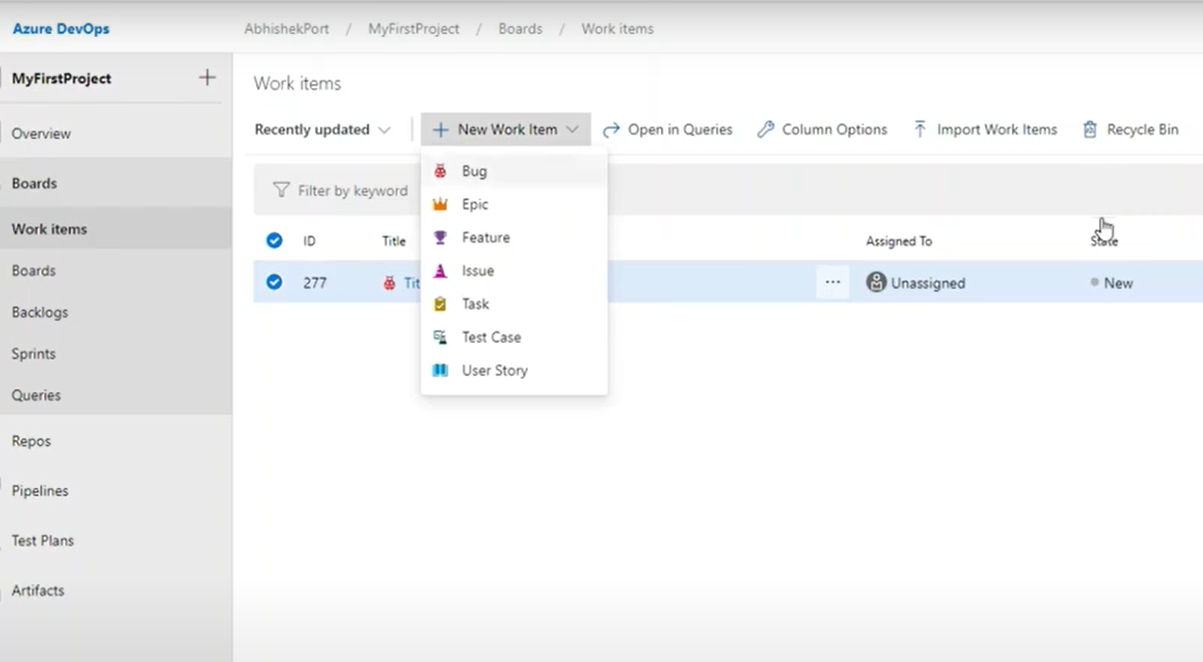
1. Chọn ‘New issues’.



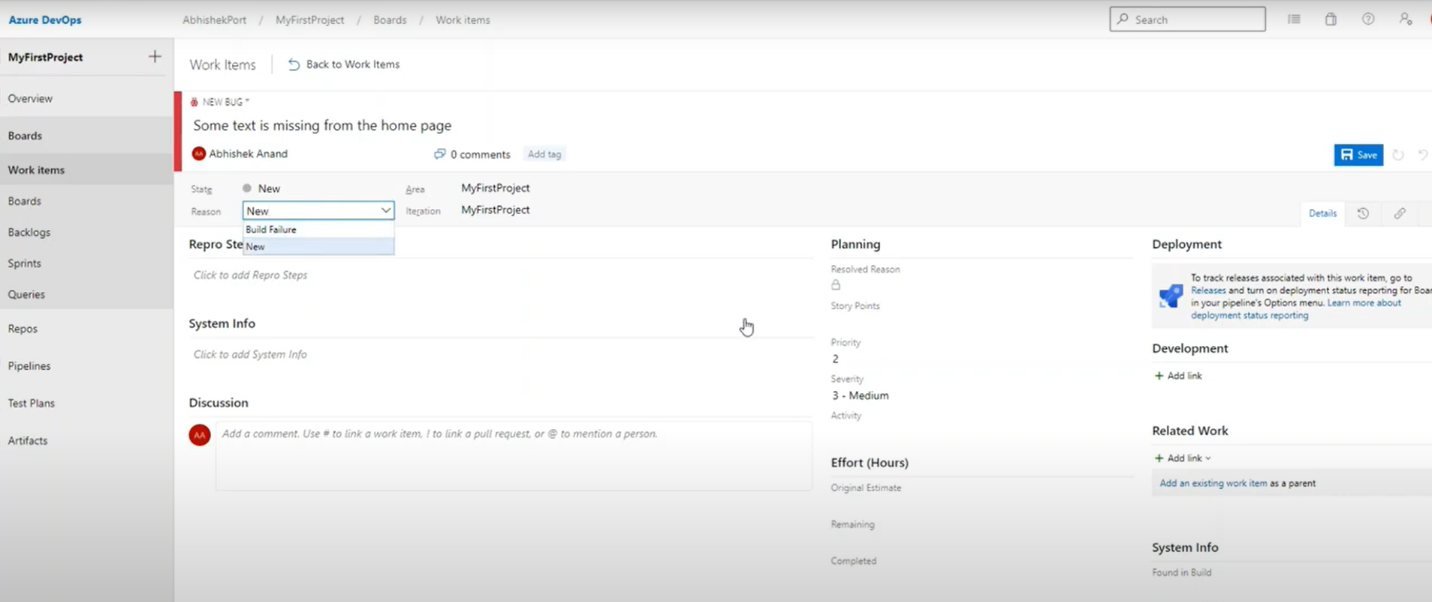
1. Điền thông tin cần thiết.
2. Chọn ‘Create’.

* **Azure DevOps**
  + Là công cụ quản lý dự án và cộng tác dựa trên đám mây do Microsoft phát triển. Nó cung cấp một bộ dịch vụ bao gồm kiểm soát phiên bản, tích hợp và phân phối liên tục, lập kế hoạch linh hoạt và kiểm thử tự động.
  + **Ưu điểm**
    - Cung cấp khả năng bảo vệ cấp cao.
    - Cung cấp các tính năng mạnh mẽ như quản lý và bảo mật git repos, xem xét các yêu cầu pull, giao tiếp với các thành viên trong nhóm và triển khai ở mọi nơi.
    - Có tính sẵn sàng cao của các trung tâm dữ liệu trên toàn cầu.
  + **Nhược điểm**
    - Việc học hỏi để sử dụng có thể phức tạp ở lúc bắt đầu.
    - Tích hợp hạn chế các công cụ và nền tảng khác.
  + **Báo cáo bug trên Azure DevOps**

1. Tại màn hình home của dự án, chọn ‘Boards’.



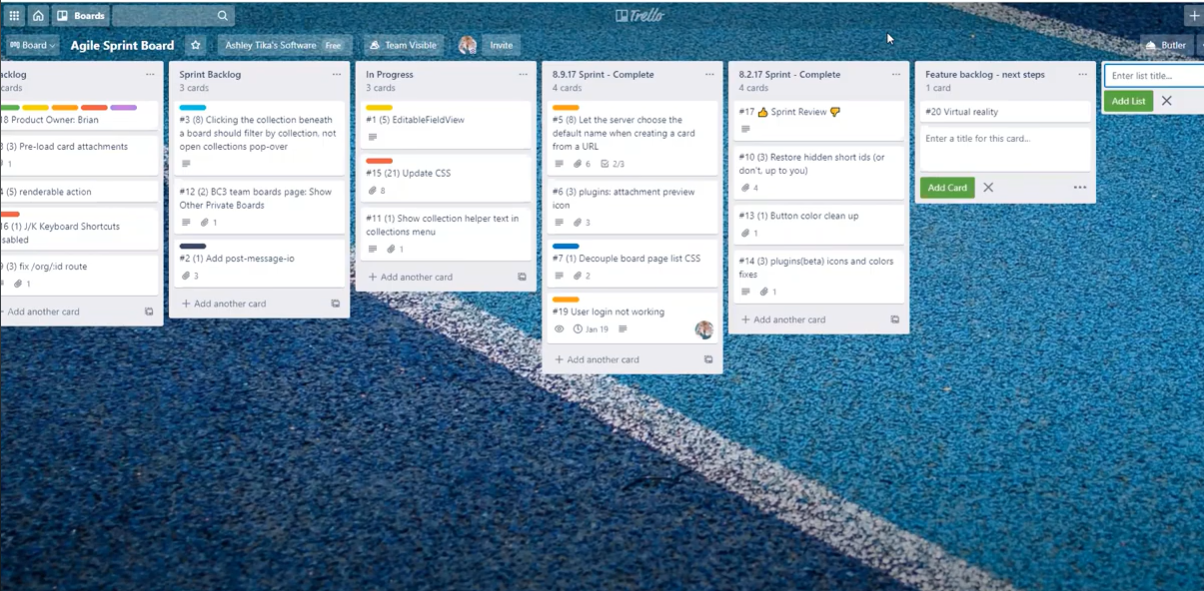
1. Tại trang ‘Work items’, chọn ‘New Work Item’.
2. Chọn ‘Bug’.

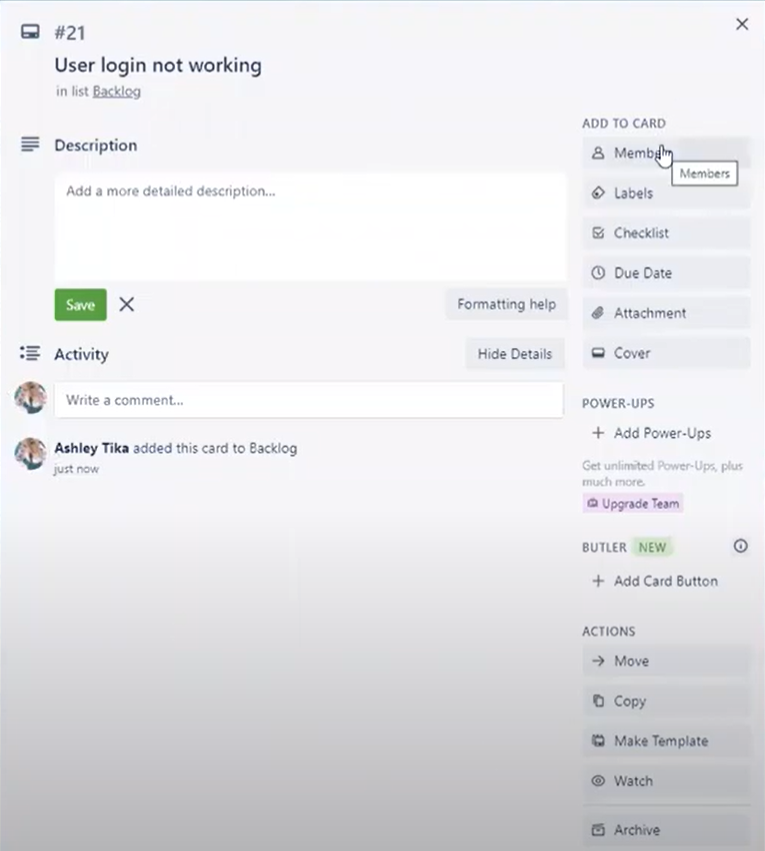
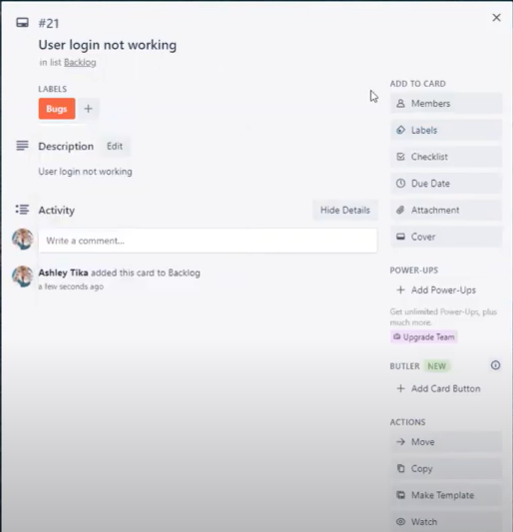


1. Điền đầy đủ các thông tin cần thiết.
2. Chọn ‘Save’.

* **Trello**
  + Là công cụ cộng tác dựa trên Kanban framework giúp người dùng quản lý dự án và tác vụ bằng các tổ chức chúng thành các bảng. Mỗi bảng đại diện cho một dự án và được điền vào bằng danh sách các tác vụ hoặc ghi chú, mỗi danh sách đại diện 1 nhiệm vụ cho người dùng và nhóm của họ.
  + **Ưu điểm**
    - Dễ dàng áp dụng phương pháp Kanban.
    - Giao diện thân thiện người dùng.
    - Cho phép cộng tác và chia sẻ bảng và thẻ giữa những người dùng hoặc nhóm.
  + **Nhược điểm**
    - Giới hạn một số chức năng ở phiên bản miễn phí.
    - Giao diện có thể cồng kềnh với những dự án lớn.
    - Khó học hỏi đối với các dự án phức tạp có số lượng bảng và thẻ lớn.
  + **Báo cáo bug trên Trello**

1. Tại màn hình dự án, chọn ‘Add another card’ ở bảng cần tạo báo cáo bug.
2. Đặt tên cho thẻ và chọn thẻ.

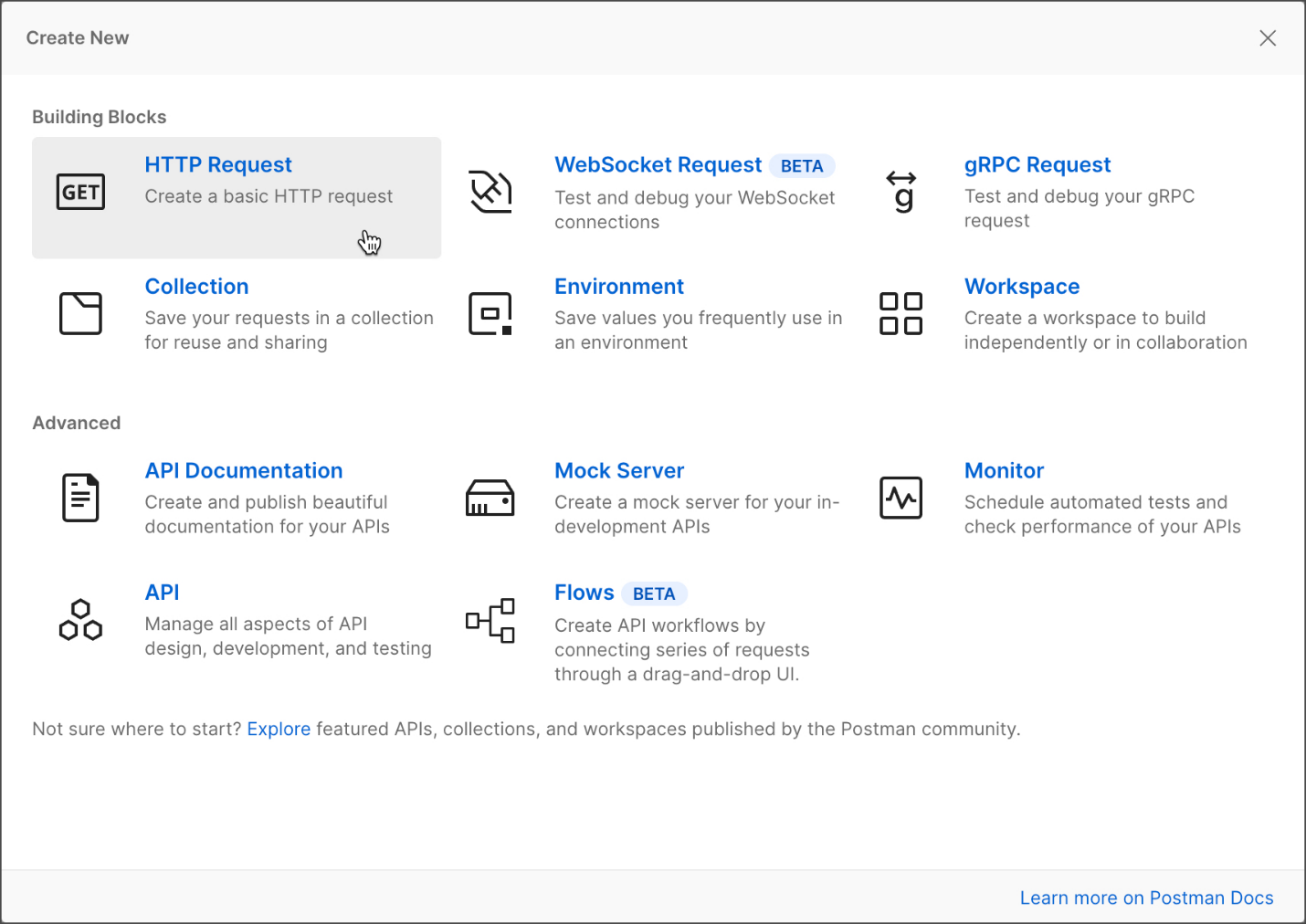


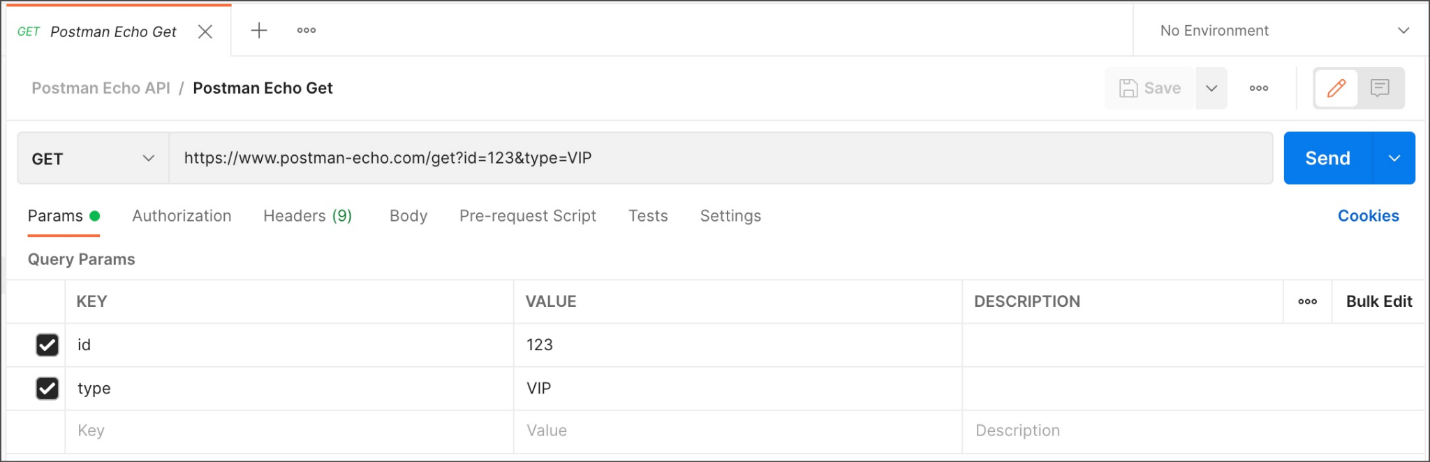
1. Điền tất cả thông tin cần thiết.
2. Chọn ‘Label’ để thêm nhãn cho báo cáo.

* **Testing tools:**
* **Postman**
  + Là nền tảng API toàn diện cung cấp một loạt các công cụ giúp đẩy nhanh vòng đời API bao gồm thiết kế, kiểm thử, lập tài liệu và chia sẻ API.
  + **Ưu điểm**
    - Giao diện người dùng thân thiện.
    - Có khả năng tự động hóa.
    - Sở hữu bộ công cụ toàn diện.
    - Có cộng đồng người dùng lớn và cung cấp nhiều tài liệu.
  + **Nhược điểm**
    - Hạn chế về các tùy chọn tùy chỉnh.
    - Không hỗ trợ cho một vài tính năng nâng cao.
    - Người dùng phải trả phí cho việc sử dụng các tính năng nâng cao.
  + **Cách sử dụng Postman trong API testing**

1. Tạo requests.

Tại màn hình home chọn **New > HTTP Request** hoặc chọn ‘+’ để mở một tab mới.



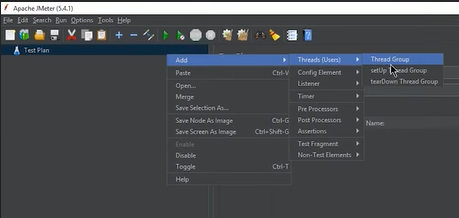
1. Thêm các thông tin cần thiết
   1. Điền URL đã được cấp.
   2. Chọn phương thức request

* Các phương thức thường sử dụng nhất:
* GET: lấy dữ liệu
* POST: thêm dữ liệu mới
* PUT: thay thế dữ liệu.
* DELETE: xóa dữ liệu.
* PATCH: cập nhật các trường dữ liệu.
  1. Cài đặt các thông tin cấu hình khác.

1. Chọn ‘Send’ để thực hiện request.

* **JMeter**
  + Là công cụ kiểm thử hiệu suất mã nguồn mở phổ biến có thể giả lập các tải nặng trên máy chủ, nhóm các máy chủ, mạng và các đối tượng để kiểm tra sức mạnh và hiệu suất tổng thể của chúng dưới các loại tải trọng khác nhau.
  + **Ưu điểm**
    - Là công cụ miễn phí mã nguồn mở.
    - Có thể được sử dụng để kiểm thử nhiều loại ứng dụng như ứng dụng web, dịch vụ web, cơ sở dữ liệu, …
    - Giao diện đồ họa người dùng thân thiện.
    - Có thể tích hợp với nhiều công cụ khác như Selenium, Eclipse, Jenkins, TeamCity,...
  + **Nhược điểm**
    - Cần thời gian để học cách sử dụng công cụ hiệu quả.
    - Khả năng báo cáo hạn chế , và khó để tạo báo cáo chi tiết.
    - Chiếm dụng nhiều tài nguyên như bộ nhớ, CPU.
  + **Cách sử dụng JMeter cho Performance testing**

1. Tạo một thread group bằng cách **click chuột phải vào Test plan > Add > Threads > Thread Group.**

****

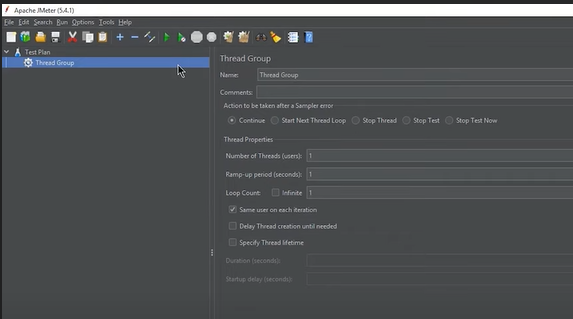
1. Thêm các thông cho Thread group.

* Tên Thread Group.
* Number of Threads (or Users): là số lượng người dùng giả lập truy cập vào ứng dụng.
* Loop Count: Số lần lặp cho mỗi người dùng.
* Ramp-Up Period: Thời gian (tính bằng giây) để đạt được Number of Threads.
* Ví dụ: Number of threads: 20

Ram-Up period: 10

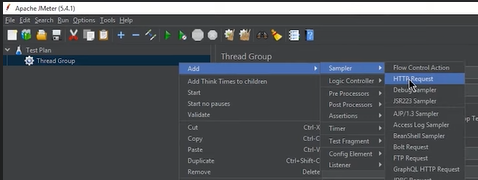
Loop Count: 2

→ Mất 10s để 20 thread truy cập vào hệ thống. Mỗi thread sẽ bắt đầu sau 0.5s sau khi thread trước đó bắt đầu. Tổng cộng có 40 request được thực hiện (do Loop Count = 2).



1. Thêm HTTP Request.

* **Click chuột phải vào Thread group > Add > Sampler > HTTP request** và điền URL trang web vào Server Name.



1. Tạo báo cáo kiểm thử.

* **Click chuột phải vào Thread group > Add > Listener > View Result Tree.**

1. Chạy kiểm thử bằng cách chọn Run trên thanh công cụ.