HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Nguyễn Thị Nụ

NGHIÊN CỨU KIỂM THỬ TỰ ĐỘNG PHẦN MỀM VÀ ỨNG DỤNG TẠI CÔNG TY A1 CONSULTING

ĐỀ ÁN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI - 2025

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Nguyễn Thị Nụ

NGHIÊN CỨU KIỂM THỬ TỰ ĐỘNG PHẦN MỀM VÀ ỨNG DỤNG TẠI CÔNG TY A1 CONSULTING

CHUYÊN NGÀNH: HỆ THỐNG THÔNG TIN
MÃ SỐ: 8.48.01.04

ĐỀ ÁN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC
TS. VŨ VĂN THỎA

HÀ NỘI – 2025

LỜI CAM ĐOAN

Học viên xin cam đoan đề án "Nghiên cứu kiểm thử tự động phần mềm và ứng dụng tại công ty A1 Consulting" là công trình nghiên cứu của học viên dưới sự hướng dẫn của giảng viên hướng dẫn, không sao chép lại của người khác. Các tài liệu được đề án tham khảo, kế thừa và trích dẫn đều được liệt kê trong danh mục các tài liệu tham khảo.

Học viên xin chịu hoàn toàn trách nhiệm về lời cam đoan nêu trên.

Hà Nội, ngày tháng năm 2025

Học viên

Nguyễn Thị Nụ

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành tới các thầy cô giáo tại Học viện Công nghệ Bưu chính Viễn thông, đặc biệt là các thầy cô thuộc bộ môn Công nghệ Thông tin, những người đã luôn tận tụy giảng dạy, hỗ trợ và tạo điều kiện thuận lợi nhất cho em trong suốt quá trình học tập tại Học viện.

Em đặc biệt biết ơn TS. Vũ Văn Thoả – người thầy đã tận tâm hướng dẫn khoa học, luôn đồng hành, chỉ bảo và hỗ trợ em trong suốt thời gian thực hiện và hoàn thiện đề tài này.

Em cũng xin chân thành cảm ơn các anh chị và bạn bè trong khóa cao học đã luôn sẵn lòng giúp đỡ, chia sẻ kinh nghiệm quý báu trong quá trình em học tập và thực hiện đề tài.

Em xin trân trọng cảm ơn!

Hà Nội, ngày tháng năm 2025

Người viết

Nguyễn Thị Nụ

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	Giải thích	Tên tiếng Anh
POM	Mô hình đối tượng trang – cấu trúc tổ chức mã kiểm thử	Page Object Model
UI	Giao diện người dùng	User Interface
API	Giao diện lập trình ứng dụng	Application Programming Interface
BE	Phía backend – xử lý logic nghiệp vụ	Backend
FE	Phía frontend – giao diện người dùng	Frontend
CI/CD	Tích hợp liên tục và triển khai liên tục	Continuous Integration / Continuous Deployment
TC	Trường hợp kiểm thử	Test Case
TestNG	Công cụ điều phối kiểm thử Java	Testing Next Generation
Odoo	Hệ thống ERP mã nguồn mở	Odoo
ERP	Hệ thống hoạch định tài nguyên doanh nghiệp	Enterprise Resource Planning
HTTP	Giao thức truyền siêu văn bản	Hypertext Transfer Protocol

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH ẢNH

MỞ ĐẦU

Trong bối cảnh công nghệ số phát triển mạnh mẽ, nhiều tổ chức đang chuyển đổi các quy trình kinh doanh sang hình thức số hóa để bắt kịp với xu hướng toàn cầu. Sự chuyển đổi này không chỉ giúp tối ưu hóa quy trình làm việc mà còn mở ra cơ hội phát triển mới cho doanh nghiệp. Tuy nhiên, để đảm bảo rằng các sản phẩm và dịch vụ số đáp ứng yêu cầu về chất lượng và hiệu suất, kiểm thử phần mềm trở thành yếu tố then chốt.

Kiểm thử phần mềm đóng vai trò thiết yếu trong việc đảm bảo chất lượng và hiệu suất của các sản phẩm phần mềm. Đây là quá trình đánh giá và xác minh các chức năng của phần mềm thông qua nhiều phương pháp, bao gồm kiểm thử thủ công và tự động, cùng với việc sử dụng các công cụ như Selenium, Katalon, …

Kiểm thử không chỉ giúp phát hiện và khắc phục lỗi mà còn đảm bảo rằng sản phẩm cuối cùng đáp ứng được nhu cầu và mong đợi của người dùng, từ đó nâng cao trải nghiệm khách hàng. Từ đó, giữ vững vị thế cạnh tranh của doanh nghiệp.

Tuy nhiên, việc áp dụng kiểm thử trong môi trường chuyển đổi số cũng gặp phải nhiều thách thức, như sự phức tạp của các hệ thống tích hợp và tốc độ thay đổi nhanh chóng của công nghệ. Đồng thời, chuyển đổi số cũng mang đến cơ hội để cải tiến quy trình kiểm thử thông qua việc áp dụng công nghệ mới, như tự động hóa và phân tích dữ liệu, nhằm nâng cao hiệu quả và độ chính xác của kiểm thử, góp phần vào sự thành công của các dự án chuyển đổi số.

Vì vậy đề tài **"Nghiên cứu kiểm thử tự động phần mềm và ứng dụng tại công ty A1 Consulting"** được thực hiện với mục đích nghiên cứu về vai trò và tầm quan trọng của kiểm thử phần mềm trong bối cảnh chuyển đổi số. Đề tài sẽ phân tích các phương pháp kiểm thử hiện có, đánh giá hiệu quả của chúng trong việc đảm bảo chất lượng sản phẩm và dịch vụ số, đồng thời xác định những thách thức mà các Doanh nghiệp gặp phải trong quá trình này.

Đối tượng nghiên cứu: Tập trung vào các phương pháp và công cụ kiểm thử phần mềm tự động, so sánh với kiểm thử thủ công.

Phạm vi nghiên cứu: Tập trung vào kiểm thử tự động cho ứng dụng Odoo ERP.

Phương pháp nghiên cứu: Nghiên cứu các tài liệu và sách chuyên ngành liên quan đến kiểm thử phần mềm tự động, các công cụ và phương pháp mới nhất. Xây dựng các kịch bản kiểm thủ công, kịch bản kiểm thử tự động cho các chức năng chính của phần mềm, đảm bảo rằng chúng phản ánh đúng yêu cầu của người dùng và tiêu chuẩn chất lượng.

Từ mục tiêu, nhiệm vụ nghiên cứu, đề án sẽ được cấu trúc với ba chương nội dung chính như sau:

Chương 1: Tổng quan về kiểm thử phần mềm và các vấn đề liên quan.

Chương 2: Nghiên cứu kỹ thuật và công cụ kiểm thử tự động.

Chương 3: Ứng dụng kiểm thử tự động tại Công ty A1 Consulting.

CHƯƠNG 1: TỔNG QUAN VỀ KIỂM THỬ PHẦN MỀM VÀ CÁC VẤN ĐỀ LIÊN QUAN

Chương 1 đề án sẽ khảo sát các vấn đề liên quan đến kiểm thử phần mềm, làm cơ sở cho các nghiên cứu chuyên sâu ở các chương sau.

1.1 Tổng quan về kiểm thử phần mềm

1.1.1 Khái niệm

Kiểm thử phần mềm là quá trình đánh giá để xác định xem một sản phẩm phần mềm có đáp ứng đúng các yêu cầu đã đề ra hay không, đồng thời phát hiện và loại bỏ các lỗi hoặc sai sót có thể xảy ra. Quá trình này bao gồm việc quan sát, phân tích, kiểm tra và đánh giá nhiều khía cạnh khác nhau của phần mềm.

Người thực hiện kiểm thử (Tester) thường kết hợp sử dụng các công cụ kiểm thử thủ công và tự động nhằm nâng cao hiệu quả. Sau khi hoàn thành quá trình kiểm thử, các kết quả sẽ được báo cáo lại cho nhóm phát triển để có biện pháp xử lý kịp thời. Mục tiêu chính là phát hiện các lỗi, điểm chưa hoàn thiện hoặc những yêu cầu chưa được đáp ứng so với mong đợi ban đầu.

Nói một cách dễ hiểu, kiểm thử phần mềm là chuỗi các hoạt động nhằm đảm bảo chương trình máy tính vận hành đúng như mục đích thiết kế, đồng thời không xảy ra các hành vi ngoài ý muốn. Đây là một giai đoạn then chốt trong chu trình phát triển phần mềm, giúp cả nhà phát triển và khách hàng đánh giá được mức độ hoàn thiện và tính phù hợp của hệ thống so với các yêu cầu ban đầu. [4]

1.1.2 Mục đích của quá trình kiểm thử

Quá trình kiểm thử phần mềm nhằm các mục đích sau đây.

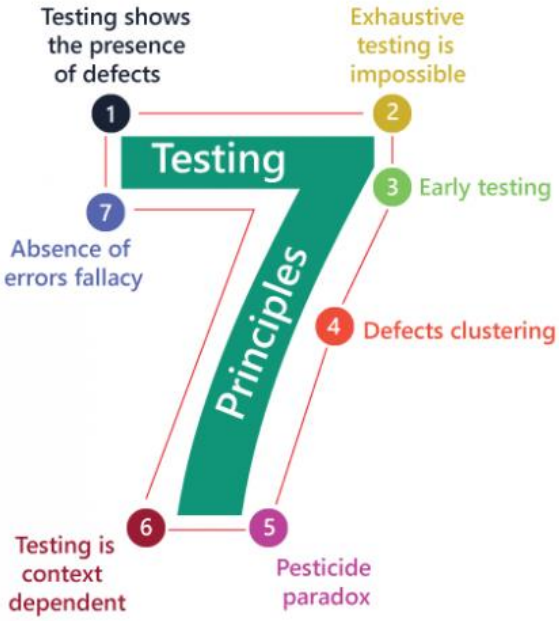
- Tìm các bug phát sinh do dev tạo ra khi code.
- Nhằm đảm bảo độ tin cậy và cung cấp cái nhìn rõ ràng về chất lượng của phần mềm.
- Để ngăn ngừa lỗi.

- Kiểm thử phần mềm giúp đảm bảo rằng sản phẩm được phát triển đúng với yêu cầu đã đề ra từ phía doanh nghiệp và người dùng cuối, đồng thời là cơ sở để tạo dựng lòng tin nơi khách hàng thông qua chất lượng phần mềm.

Kiểm thử phần mềm giúp đảm bảo rằng ứng dụng hoặc sản phẩm được phát triển phù hợp với các yêu cầu kinh doanh và mong đợi của người sử dụng. Thông qua quá trình kiểm thử, các lỗi và thiếu sót có thể được phát hiện và khắc phục kịp thời, từ đó nâng cao chất lượng sản phẩm

Kiểm thử phần mềm cho phép tạo ra những đánh giá khách quan về mức độ phù hợp của hệ thống các yêu cầu đã nêu và thông số kỹ thuật. Kiểm tra xác nhận rằng hệ thống đáp ứng các yêu cầu về chức năng, hiệu suất, độ tin cậy, an toàn, khả năng sử dụng và như vậy xác nhận này được thực hiện để đảm bảo rằng chúng tôi đang xây dựng hệ thống phù hợp. Mặt khác, các thông tin từ các kiểm thử phần mềm giúp quản lý rủi ro. [4]

1.1.3 Các nguyên tắc cơ bản của kiểm thử phần mềm



Hình 0.1: Các nguyên tắc cơ bản của kiểm thử phần mềm

Nguyên tắc 1: Kiểm thử phần mềm chứng minh sự hiện diện của lỗi

Kiểm thử chỉ có thể chứng minh được rằng sản phẩm có lỗi, nhưng không thể chứng minh rằng sản phẩm không còn lỗi. Nghĩa là sản phẩm luôn có lỗi cho dù có kiểm thử nhiều bao nhiêu. Do đó, điều quan trọng là chúng ta phải thiết kế các trường hợp kiểm thử (test case) sao cho có thể tìm được càng nhiều lỗi càng tốt.

Nguyên tắc 2: Kiểm thử toàn bộ là không khả thi

Kiểm tra cạn kiệt (kết hợp cả đầu vào và tiền điều kiện) là không khả thi ngoại trừ những trường hợp nhỏ. Trong khi hầu hết các sản phẩm ngày nay rất đa dạng và phức tạp do được phát triển trên nhiều nền tảng, công nghệ phong phú cũng như khả năng lưu trữ kết nối dữ liệu lớn, khiến việc kiểm thử trở nên khó khăn và việc kiểm thử toàn bộ là gần như không thể. Thay vì kiểm thử toàn bộ, việc phân tích rủi ro và dựa trên sự mức độ ưu tiên chúng ta có thể tập trung việc kiểm thử vào một số điểm cần thiết, có nguy cơ lỗi cao hơn.

Nguyên tắc 3: Kiểm thử càng sớm càng tốt

Nguyên tắc này yêu cầu bắt đầu thử nghiệm phần mềm trong giai đoạn đầu của vòng đời phát triển phần mềm. Các hoạt động kiểm thử phần mềm từ giai đoạn đầu sẽ giúp phát hiện bug sớm hơn. Nó cho phép chuyển giao phần mềm theo yêu cầu đúng thời gian với chất lượng dự kiến.

Nguyên tắc 4: Sự tập trung của lỗi

Thông thường, lỗi tập trung vào những module, thành phần chức năng chính của hệ thống. Nếu xác định được điều này bạn sẽ tập trung vào tìm kiếm lỗi quanh khu vực được xác định. Nó được coi là một trong những cách hiệu quả nhất để thực hiện kiểm tra hiệu quả.

Nguyên tắc 5: Nghịch lý thuốc trừ sâu

Nếu bạn sử dụng cùng một tập hợp các trường hợp kiểm thử liên tục, sau đó một thời gian các trường hợp kiểm thử không tìm thấy lỗi nào mới. Hiệu quả của các trường hợp kiểm thử bắt đầu giảm xuống sau một số lần thực hiện, vì vậy luôn luôn chúng ta phải luôn xem xét và sửa đổi các trường hợp kiểm thử trên một khoảng thời gian thường xuyên.

Nguyên tắc 6: Kiểm thử phụ thuộc vào ngữ cảnh

Theo nguyên tắc này thì việc kiểm thử phụ thuộc vào ngữ cảnh và chúng ta phải tiếp cận kiểm thử theo nhiều ngữ cảnh khác nhau. Nếu bạn đang kiểm thử ứng dụng web và ứng dụng di động bằng cách sử dụng chiến lược kiểm thử giống nhau, thì đó là sai. Chiến lược để kiểm thử ứng dụng web sẽ khác với kiểm thử ứng dụng cho thiết bị di động của Android.

Nguyên tắc 7: Không có lỗi – Sai lầm

Việc không tìm thấy lỗi trên sản phẩm không đồng nghĩa với việc sản phẩm đã sẵn sàng để tung ra thị trường. Việc không tìm thấy lỗi cũng có thể là do bộ trường hợp kiểm thử được tạo ra chỉ nhằm kiểm tra những tính năng được làm đúng theo yêu cầu thay vì nhằm tìm kiếm lỗi mới. [9]

1.2 Quy trình, kỹ thuật kiểm thử phần mềm và các vấn đề liên quan

1.2.1 Quy trình kiểm thử phần mềm

Mục tiêu chính của kiểm thử phần mềm là xây dựng một tập hợp các trường hợp kiểm thử có khả năng phát hiện lỗi một cách hiệu quả nhất. Việc thiết kế các trường hợp kiểm thử cần đảm bảo bao phủ đầy đủ các chức năng và tình huống có thể xảy ra, nhằm tăng khả năng phát hiện sai sót trong phần mềm.

Để đảm bảo chất lượng kiểm thử và đạt được kết quả đáng tin cậy, quá trình kiểm thử cần được chuẩn bị kỹ lưỡng thông qua các bước như lập kế hoạch kiểm thử, thiết kế chi tiết các ca kiểm thử và chuẩn bị dữ liệu đầu vào tương ứng. Những yếu tố này đóng vai trò quan trọng trong việc đảm bảo độ chính xác và hiệu quả của toàn bộ quy trình.

Sơ đồ dưới đây minh họa quy trình kiểm thử phần mềm với các bước cụ thể từ chuẩn bị đến thực thi. Đây là chuỗi hoạt động logic, đóng vai trò là đầu vào cho quá trình kiểm thử chính thức. Giai đoạn cuối cùng trong quy trình này là việc lập "báo cáo kiểm thử" – một tài liệu tổng hợp đầy đủ các ca kiểm thử đã thực hiện, dữ liệu sử dụng, kết quả đầu ra mong đợi và kết quả thực tế, từ đó đánh giá mức độ đạt yêu cầu của phần mềm.

Hình 0.2: Quy trình kiểm thử phần mềm.

Quy trình kiểm thử bao gồm một số giai đoạn:

Lập kế hoạch kiểm thử: Bước đầu tiên là lập kế hoạch cho tất cả các hoạt động sẽ được thực hiện và các phương pháp được sử dụng.

Vấn đề quan trọng nhất đối với kế hoạch kiểm thử:

- **Mục tiêu:** Xác định rõ phạm vi kiểm thử, phương pháp áp dụng, nguồn lực sử dụng và lịch trình thực hiện các hoạt động kiểm thử.
- **Tài liệu liên quan:** Danh sách các tài liệu được tham chiếu để xây dựng kế hoạch kiểm thử.
- **Thuật ngữ và định nghĩa:** Làm rõ các khái niệm được sử dụng trong kế hoạch để đảm bảo sự thống nhất về cách hiểu.
- **Tổng quan về xác minh và thẩm định (V&V):** Bao gồm cơ cấu tổ chức, phân bổ nguồn lực, trách nhiệm của từng bên liên quan, cũng như các công cụ, kỹ thuật và phương pháp kiểm thử được sử dụng.
- **Vòng đời xác minh và thẩm định:** Trình bày chi tiết các hoạt động, đầu vào và đầu ra tương ứng trong từng giai đoạn của vòng đời phát triển phần mềm.
- **Báo cáo V&V:** Quy định nội dung, định dạng và thời điểm cần thiết để tạo lập các báo cáo xác minh và thẩm định phần mềm.
- **Thủ tục quản lý:** Mô tả các chính sách, quy trình, tiêu chuẩn, thực hành và quy ước được sử dụng để điều hành và kiểm soát quá trình kiểm thử.

Giai đoạn bố trí nhân viên kiểm thử: Việc kiểm thử thường phải tiến hành một cách độc lập và các nhóm độc lập có trách nhiệm tiến hành các hoạt động kiểm thử, gọi là các nhóm kiểm thử.

Thiết kế các trường hợp kiểm thử: Các trường hợp kiểm thử là các đặc tả đầu vào cho kiểm thử và đầu ra mong đợi của hệ thống cùng với các câu lệnh được kiểm thử. Xử lý đo lường kiểm thử bằng cách thu thập dữ liệu. Đánh giá sản phẩm phần mềm để xác nhận sản phẩm có thể sẵn sàng phát hành được chưa.[9]

1.2.2 Các kỹ thuật kiểm thử phần mềm

Trong quá trình phát triển phần mềm, việc lựa chọn kỹ thuật kiểm thử phù hợp đóng vai trò quan trọng trong việc phát hiện lỗi và đảm bảo chất lượng sản phẩm. Các kỹ thuật kiểm thử phổ biến hiện nay được phân thành ba nhóm chính: kiểm thử hộp đen (Black-box Testing), kiểm thử hộp trắng (White-box Testing) và kiểm thử hộp xám (Gray-box Testing). [9]

Kiểm thử hộp trắng (White-Box Testing), còn được gọi là kiểm thử cấu trúc hoặc kiểm thử logic, là phương pháp kiểm thử tập trung vào việc kiểm tra các thành phần bên trong của ứng dụng. Mục tiêu chính là đảm bảo rằng tất cả các câu lệnh và điều kiện trong mã nguồn đều được thực thi ít nhất một lần trong quá trình kiểm thử.

Đây là phương pháp kiểm thử dựa trên việc hiểu rõ cấu trúc bên trong và logic xử lý của phần mềm. Người kiểm thử sẽ truy cập vào mã nguồn và thiết kế các ca kiểm thử dựa trên luồng điều khiển, điều kiện rẽ nhánh, vòng lặp và cấu trúc dữ liệu nội tại. Phương pháp này còn được biết đến với các tên gọi khác như kiểm thử hộp thủy tinh (Glass-box Testing) hay kiểm thử trong suốt (Clear-box Testing).

Mục tiêu chính của kỹ thuật này là đảm bảo tất cả các đoạn mã, điều kiện và nhánh trong chương trình được thực thi ít nhất một lần trong quá trình kiểm thử. Tuy nhiên, việc bao phủ toàn bộ các đường dẫn logic của chương trình thường không khả thi, đặc biệt đối với các hệ thống lớn có nhiều điều kiện phân nhánh.

Ưu điểm của kiểm thử hộp trắng là có thể phát hiện các lỗi logic và lỗi ẩn sâu bên trong mã nguồn. Tuy nhiên, kỹ thuật này yêu cầu người kiểm thử có kiến thức

chuyên sâu về lập trình và cấu trúc hệ thống, đồng thời đòi hỏi chi phí và thời gian tương đối lớn.

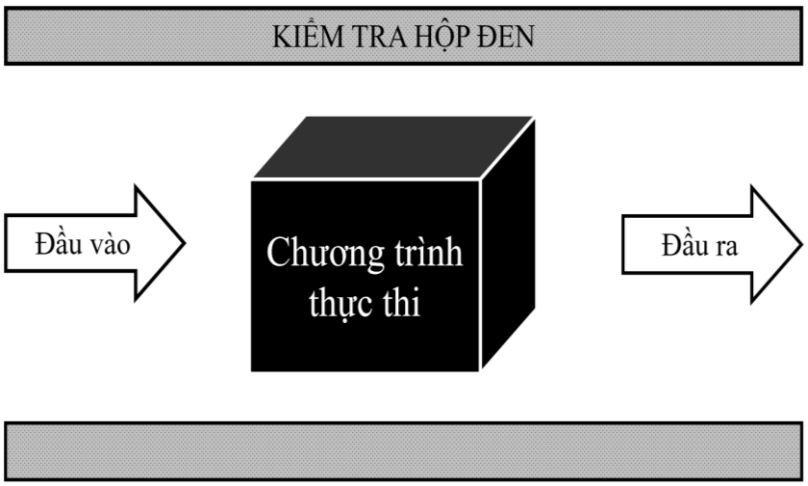
Kiểm thử hộp đen *(Black- Box Testing)* là một phương pháp kiểm thử phần mềm được thực hiện mà không biết được cấu tạo bên trong của phần mềm, là cách mà các kiểm thử viên kiểm tra xem hệ thống như một chiếc hộp đen, không có cách nào nhìn thấy bên trong của cái hộp.

Kiểm thử hộp đen là kỹ thuật kiểm thử dựa trên đặc tả bên trong của chương trình, dựa vào mã nguồn, cấu trúc chương trình. Kiểm thử hộp đen thường phát hiện các lỗi lập trình. Loại kiểm thử này khá khó thực hiện và chi phí cao. Với các module quan trọng, thực thi việc tính toán chính của hệ thống, phương pháp này là cần thiết. Các dữ liệu phục vụ kiểm thử được lấy từ các tài liệu đặc tả phần mềm ở từng cấp độ phát triển, cụ thể là: đặc tả yêu cầu cho kiểm thử hệ thống, đặc tả thiết kế cho kiểm thử tích hợp, và đặc tả chi tiết module cho kiểm thử đơn vị.

Trong kỹ thuật này, người kiểm thử xem phần mềm như là một hộp đen. Người kiểm thử không cần biết phần mềm được xây dựng như thế nào bên trong, mà chỉ cần kiểm tra xem nó có thực hiện đúng theo những gì đã được mô tả trong tài liệu yêu cầu hay không. Người kiểm thử chỉ biết những chức năng phần mềm dự kiến thực hiện và những chức năng không thực hiện, mà không thể nhìn vào bên trong xem nó hoạt động như thế nào. Vì thế dữ liệu kiểm thử sẽ xuất phát từ đặc tả.

Kiểm thử hộp đen sẽ thực hiện kiểm thử để tìm các lỗi sau:

- Thiếu chức năng
- Thiếu giao diện
- Thiếu cấu trúc dữ liệu, lỗi truy cập cơ sở dữ liệu, lỗi thi hành, lỗi khởi tạo hoặc kết thúc



Hình 0.3: Kiểm thử hộp đen.

Ưu điểm:

- Hệ thống thật sự với toàn bộ yêu cầu của nó được kiểm thử chính xác.
- Thiết kế kịch bản kiểm thử khá nhanh, ngay khi mà các yêu cầu chức năng được xác định.

Nhược điểm:

- Dữ liệu đầu vào yêu cầu một khối lượng mẫu khá lớn.
- Khó viết kịch bản thử do cần xác định tất cả các yếu tố đầu vào và thiếu cả thời gian cho việc tập hợp này.
- Khả năng để bản thân kỹ sư lạc lối trong khi kiểm thử là khá cao.

Mỗi kỹ thuật kiểm thử đều có những điểm mạnh và hạn chế riêng. Do đó, để đảm bảo chất lượng hệ thống khi triển khai đến người dùng cuối, thường cần kết hợp nhiều phương pháp kiểm thử khác nhau.

Kiểm thử hộp xám: Đây là một kỹ thuật kiểm thử mới dựa trên những đặc tính của cả kiểm thử hộp đen và kiểm thử hộp trắng. Mục tiêu chính của kiểm thử hộp xám là kiểm thử các ứng dụng trên nền web.

Ưu điểm:

- Quan điểm kiểm thử của kiểm thử hộp xám là từ quan điểm của người dùng.
- Cung cấp các lợi ích của cả thử nghiệm hộp đen và hộp trắng cùng nhau.

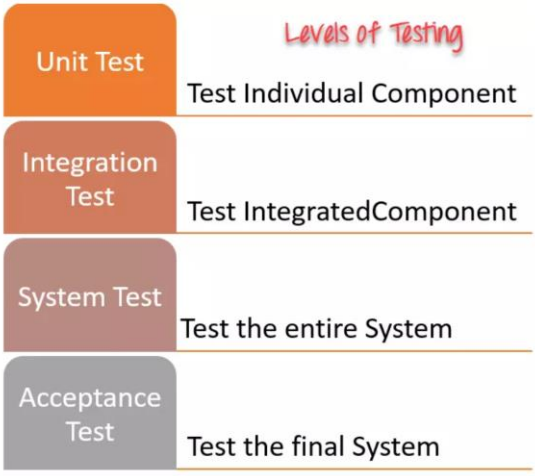
- Sẽ dựa trên các đặc tả chức năng, mô tả của người dùng và sơ đồ kiến trúc hệ thống, từ đó xác nhận các yêu cầu ngay từ ban đầu.
- Việc kiểm tra sẽ tường minh vì sẽ có nhiều sự quan tâm giữa người kiểm thử phần mềm và người thiết kế hoặc kỹ sư.

Nhược điểm:

- Kiểm tra hộp xám cũng có thể mất nhiều thời gian để kiểm tra từng đường dẫn và đôi khi điều này là không thực tế.
- Rất khó để liên kết lỗi khi thực hiện kiểm tra hộp xám cho một ứng dụng có hệ thống phân tán.
- Thông thường sẽ dẫn đến phạm vi kiểm tra thấp hơn so với thực hiện kiểm tra hộp trắng và đen riêng biệt.
- Có thể không phù hợp để thử nghiệm một số loại chức năng.

1.2.3 Mức độ kiểm thử

Hình 1.4 dưới đây mô tả các mức độ kiểm thử trong quá trình kiểm thử phần mềm.



Hình 0.4: Mức độ kiểm thử phần mềm.

Phần lớn các phương pháp thiết kế phần mềm hiện nay đều áp dụng nguyên tắc phân tách hệ thống thành các đơn vị nhỏ, gọi là "unit", mỗi đơn vị có chức năng

riêng cùng với dữ liệu đầu vào và đầu ra độc lập. Trên các đơn vị này chúng ta sẽ tiến hành kiểm thử đơn vị.

Khi phần lập trình của một đơn vị phần mềm hoàn tất và hồ sơ kiểm thử đã sẵn sàng, việc kiểm thử đơn vị sẽ được tiến hành. Việc kiểm thử dựa trên các test case đã chuẩn bị trước, và kết quả sẽ được cập nhật vào báo cáo kiểm thử.

Khi phát hiện lỗi hoặc sự không tương thích trong quá trình kiểm thử đơn vị, đơn vị phần mềm cần được chỉnh sửa và kiểm thử lại. Quá trình này sẽ lặp lại cho đến khi tất cả các ca kiểm thử được liệt kê trong tài liệu kiểm thử đơn vị đều được thực hiện thành công.

Kiểm thử đơn vị (Unit Test): Một đơn vị (Unit) là một thành phần phần mềm nhỏ nhất mà ta có thể kiểm thử được, ví dụ: các hàm (Function), thủ tục (Procedure), lớp (Class), hoặc các phương thức (Method).

Kiểm thử tích hợp (Integration Test) được thực hiện bằng cách kết hợp các đơn vị riêng biệt lại với nhau để kiểm tra sự tương tác giữa chúng. Không giống như kiểm thử đơn vị – vốn tập trung vào từng phần riêng lẻ – kiểm thử tích hợp đánh giá khả năng phối hợp và trao đổi dữ liệu giữa các thành phần trong hệ thống.

Kiểm thử hệ thống (System Test): Mục tiêu của kiểm thử hệ thống là đánh giá toàn bộ phần mềm sau khi tích hợp hoàn chỉnh để đảm bảo rằng nó đáp ứng đầy đủ các yêu cầu đã đề ra. Giai đoạn này không chỉ kiểm tra các chức năng chính mà còn kiểm tra các thuộc tính phi chức năng như độ tin cậy, hiệu năng, bảo mật và khả năng sử dụng. Kiểm thử hệ thống chỉ được tiến hành sau khi toàn bộ các thành phần đã được tích hợp thành công.

Kiểm thử chấp nhận sản phẩm (Acceptance Test): Mục tiêu của kiểm thử chấp nhận là xác minh rằng phần mềm đáp ứng các yêu cầu nghiệp vụ đã thống nhất với khách hàng và có thể được người dùng cuối chấp thuận để đưa vào sử dụng.[4]

1.2. 4 Thiết kế testcase

Biểu mẫu testcase gồm các nội dung:

- Mục đích kiểm thử
- Các bước thực hiện

- Kết quả mong muốn
- Kết quả thực tế

Các kỹ thuật thiết kế testcase:

- Phân vùng tương đương: một phương pháp kiểm thử phần mềm, trong đó các giá trị đầu vào được chia thành các nhóm hợp lệ và không hợp lệ. Từ mỗi nhóm này, chúng ta sẽ lựa chọn các giá trị đại diện để xây dựng các kịch bản kiểm thử phù hợp.
- Phân tích giá trị biên: là một kỹ thuật kiểm thử phần mềm có liên quan đến việc xác định biên (ranh giới) của điều kiện mô tả cho các giá trị đầu vào và chọn giá trị ở biên và bên cạnh giá trị biên làm dữ liệu kiểm thử.
- Bảng quyết định: là dùng bảng để hiển thị danh sách các thao tác phần mềm được quyết định trên các điều kiện khác nhau.
- Đoán lỗi: Phương pháp đoán lỗi không dựa trên quy tắc kiểm thử cụ thể nào mà phụ thuộc vào kinh nghiệm và sự linh hoạt của người kiểm thử. Các test case thường được thiết kế dựa trên tình huống thực tế hoặc theo quy trình công việc được mô tả trong tài liệu chức năng.

Mục đích của các kỹ thuật thiết kế testcase: Giảm thiểu số lượng testcase thừa, tiết kiệm thời gian kiểm thử mà vẫn đảm bảo chất lượng phần mềm. Chọn được đúng testcase đại diện cần test mà vẫn bao phủ được các trường hợp. [9]

1.3 Tổng quan về kiểm thử phần mềm trong bối cảnh chuyển đổi số

Chuyển đổi số (Digital Transformation) là quá trình áp dụng toàn diện công nghệ số vào tất cả các lĩnh vực hoạt động của tổ chức, doanh nghiệp nhằm thay đổi mô hình vận hành truyền thống sang mô hình hiện đại hơn, hiệu quả hơn. Quá trình này bao gồm ba khía cạnh chính: công nghệ, quy trình và con người. Về mặt công nghệ, các tổ chức áp dụng những nền tảng hiện đại như trí tuệ nhân tạo (AI), điện toán đám mây (Cloud Computing), dữ liệu lớn (Big Data), và Internet vạn vật (IoT) để tăng cường khả năng thu thập, phân tích và xử lý dữ liệu. Về quy trình, chuyển đổi số thúc đẩy tự động hóa, loại bỏ các thao tác thủ công, giảm thiểu sai sót và nâng cao

hiệu suất làm việc. Về con người, đòi hỏi sự thay đổi trong tư duy, kỹ năng và cách thức phối hợp giữa các bộ phận để thích ứng với mô hình vận hành mới. [1]

Trong bối cảnh đó, phần mềm giữ vai trò trung tâm như một công cụ kết nối và vận hành toàn bộ hệ sinh thái số của doanh nghiệp. Từ hệ thống quản trị nguồn lực (ERP), quản lý quan hệ khách hàng (CRM), đến các ứng dụng web, mobile phục vụ người dùng – tất cả đều phụ thuộc vào phần mềm. Khi vai trò của phần mềm ngày càng lớn, yêu cầu về chất lượng phần mềm cũng trở nên nghiêm ngặt hơn. Một lỗi nhỏ trong phần mềm có thể ảnh hưởng trực tiếp đến quy trình kinh doanh, gây mất dữ liệu, giảm trải nghiệm người dùng và ảnh hưởng đến uy tín thương hiệu.

Chính vì vậy, kiểm thử phần mềm trở thành một hoạt động không thể thiếu trong quá trình phát triển và triển khai các giải pháp số. Tuy nhiên, quá trình kiểm thử trong chuyển đổi số cũng đối mặt với nhiều thách thức. Trước hết là yêu cầu về tốc độ – phần mềm cần được kiểm thử nhanh chóng và liên tục để đáp ứng các chu kỳ phát hành ngắn. Thứ hai là môi trường hệ thống ngày càng phức tạp, tích hợp nhiều công nghệ và nền tảng khác nhau, đòi hỏi phương pháp kiểm thử linh hoạt và toàn diện. Thứ ba là kỳ vọng ngày càng cao từ phía người dùng về tính ổn định, hiệu suất và bảo mật của phần mềm.

Kiểm thử phần mềm trở thành một hoạt động không thể thiếu trong quá trình phát triển và triển khai các giải pháp số. Tuy nhiên, quá trình kiểm thử trong chuyển đổi số cũng đối mặt với nhiều thách thức.

Thách thức về chất lượng phần mềm

- **Tốc độ phát triển nhanh chóng và chu kỳ phát hành ngắn:** Áp lực từ thị trường và yêu cầu chuyển đổi số nhanh chóng dẫn đến việc phần mềm cần được phát triển và triển khai với tốc độ cao. Điều này khiến thời gian dành cho kiểm thử bị rút ngắn, dễ dẫn đến bỏ sót lỗi và ảnh hưởng đến chất lượng cuối cùng.
- **Sự phức tạp của hệ thống:** Các giải pháp số thường tích hợp nhiều công nghệ (AI, Cloud, IoT, Big Data), nền tảng và hệ thống cũ (legacy systems), tạo ra

một môi trường phức tạp. Việc đảm bảo sự tương thích và vận hành trơn tru giữa các thành phần này là một thách thức lớn về chất lượng.

- **Dữ liệu lớn và đa dạng:** Với việc áp dụng Big Data, phần mềm phải xử lý khối lượng dữ liệu khổng lồ và đa dạng. Đảm bảo tính chính xác, nhất quán và bảo mật của dữ liệu trong mọi trường hợp là một yếu tố quan trọng của chất lượng phần mềm.
- **Yêu cầu về trải nghiệm người dùng (UX) và hiệu suất:** Người dùng ngày càng có kỳ vọng cao về trải nghiệm mượt mà, tốc độ phản hồi nhanh và khả năng hoạt động ổn định của phần mềm. Bất kỳ sự chậm trễ hay gián đoạn nào cũng có thể dẫn đến sự không hài lòng và mất khách hàng.
- **Rủi ro về bảo mật:** Khi các hệ thống trở nên kết nối hơn, nguy cơ về các cuộc tấn công mạng và rò rỉ dữ liệu tăng lên. Đảm bảo an toàn thông tin là một yếu tố sống còn của chất lượng phần mềm trong bối cảnh chuyển đổi số.

Thách thức về kiểm thử phần mềm

- **Kiểm thử liên tục trong chu trình DevOps/Agile:** Mô hình phát triển nhanh (Agile) và vận hành tích hợp (DevOps) đòi hỏi kiểm thử phải được thực hiện liên tục, từ giai đoạn đầu phát triển đến khi triển khai và vận hành. Điều này đặt ra áp lực lớn lên đội ngũ kiểm thử để tích hợp chặt chẽ vào quy trình và cung cấp phản hồi nhanh chóng.
- **Kiểm thử tích hợp và đầu cuối:** Với việc tích hợp nhiều hệ thống và công nghệ, việc kiểm thử sự tương tác giữa các module (kiểm thử tích hợp) và toàn bộ luồng nghiệp vụ từ đầu đến cuối (kiểm thử đầu cuối) trở nên phức tạp hơn rất nhiều.
- **Thách thức về môi trường kiểm thử:** Thiết lập và duy trì môi trường kiểm thử phản ánh chính xác môi trường sản phẩm với đầy đủ dữ liệu và cấu hình phức tạp là một việc tốn kém và đòi hỏi nhiều công sức. Đặc biệt khi hệ thống sử dụng các dịch vụ đám mây hoặc microservices.

- **Thiếu hụt kỹ năng kiểm thử chuyên biệt:** Các công nghệ mới như AI, IoT, Blockchain đòi hỏi các kỹ năng kiểm thử chuyên biệt. Nguồn nhân lực có đủ kiến thức và kinh nghiệm về kiểm thử các hệ thống này còn hạn chế.
- **Quản lý dữ liệu kiểm thử:** Tạo ra và quản lý dữ liệu kiểm thử phù hợp với khối lượng lớn và sự đa dạng của dữ liệu trong môi trường số là một thách thức lớn.
- **Chi phí và hiệu quả:** Việc đầu tư vào công cụ, hạ tầng và nhân lực cho kiểm thử trong bối cảnh chuyển đổi số có thể rất lớn. Đảm bảo rằng việc kiểm thử được thực hiện hiệu quả và mang lại giá trị cao là một thách thức về quản lý.

Trong bối cảnh đó, kiểm thử tự động được xem là giải pháp tối ưu để giải quyết những yêu cầu khắt khe về tốc độ, phạm vi và độ chính xác. Việc áp dụng các công cụ kiểm thử tự động như Selenium, Katalon, TestNG không chỉ giúp tiết kiệm thời gian và chi phí mà còn nâng cao độ tin cậy của phần mềm. Tự động hóa kiểm thử còn hỗ trợ triển khai các mô hình kiểm thử hiện đại như kiểm thử liên tục (Continuous Testing), tích hợp liên tục (Continuous Integration), từ đó đảm bảo phần mềm luôn được kiểm tra và sẵn sàng hoạt động trong mọi điều kiện triển khai thực tế.

Tóm lại, kiểm thử phần mềm trong bối cảnh chuyển đổi số không chỉ là một bước kỹ thuật, mà còn là một phần chiến lược trong việc đảm bảo sự thành công của các dự án số hóa. Việc lựa chọn phương pháp và công cụ kiểm thử phù hợp là yếu tố then chốt giúp doanh nghiệp duy trì chất lượng sản phẩm, tối ưu hiệu suất và nâng cao năng lực cạnh tranh trên thị trường số. [2]

Các bước triển khai chuyển đổi số

Quá trình chuyển đổi số gồm 5 bước:

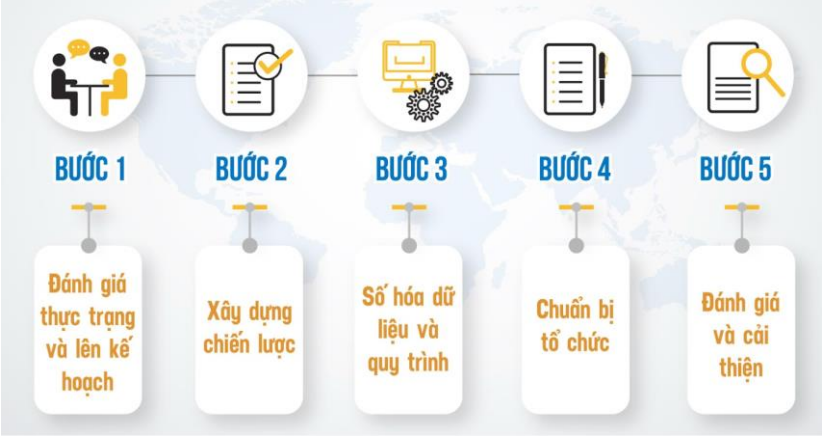
Bước 1: Đánh giá hiện trạng và lên kế hoạch để chuyển đổi số

Bước 2: Xây dựng chiến lược chuyển đổi số

Bước 3: Số hóa quy trình và dữ liệu

Bước 4: Chuẩn bị sẵn sàng tổ chức chuyển đổi số

Bước 5: Đánh giá cải thiện



Hình 0.5: Quy trình triển khai Chuyển đổi số.

1.4 Kết luận chương 1

Trong chương này, đề án đã trình bày những cơ sở lý thuyết nền tảng về kiểm thử phần mềm và chuyển đổi số, bao gồm các khái niệm, vai trò, quy trình thực hiện cũng như những thách thức liên quan trong thực tiễn.

Trên cơ sở nội dung đã nghiên cứu ở chương 1, chương tiếp theo của đề án sẽ tập trung vào việc nghiên cứu chuyên sâu về kỹ thuật và công cụ kiểm thử tự động nhằm nâng cao hiệu quả và chất lượng trong quá trình kiểm thử phần mềm.

CHƯƠNG 2: NGHIÊN CỨU KỸ THUẬT VÀ CÔNG CỤ KIỂM THỬ TỰ ĐỘNG

2.1 Tổng quan về kiểm thử tự động

2.1.1 Khái niệm

Kiểm thử tự động phần mềm (Automation Testing) là quá trình sử dụng phần mềm để điều khiển việc thực thi các kịch bản kiểm thử, so sánh kết quả thực tế với kỳ vọng, và tạo ra báo cáo mà không cần sự can thiệp trực tiếp từ người kiểm thử trong mỗi lần chạy. Tức là, sau khi thiết lập một bộ test script, chúng có thể chạy lặp lại nhiều lần để kiểm tra các chức năng một cách nhất quán, nhanh chóng và chính xác.

Khác với kiểm thử thủ công – nơi tester cần thao tác trực tiếp trên hệ thống – kiểm thử tự động có thể được lập trình để thực hiện hàng loạt bước như click chuột, nhập dữ liệu, chuyển trang, xác nhận nội dung... Điều này đặc biệt hữu ích trong kiểm thử hồi quy – nơi cần kiểm tra toàn bộ hệ thống sau mỗi lần cập nhật.

2.1.2 Vai trò

Kiểm thử tự động đóng vai trò chiến lược trong toàn bộ vòng đời phát triển phần mềm. Trong bối cảnh doanh nghiệp chuyển đổi số và các phần mềm được phát hành nhanh hơn (mỗi tuần hoặc mỗi ngày), việc kiểm thử thủ công sẽ không thể đáp ứng được tốc độ và độ tin cậy cần thiết.

Một số vai trò cốt lõi của kiểm thử tự động gồm:

- **Tối ưu hóa thời gian kiểm thử**: Thay vì phải kiểm thử lại thủ công hàng trăm ca kiểm thử sau mỗi lần cập nhật, các tập lệnh tự động có thể thực thi chỉ trong vài phút.
- **Tăng độ chính xác**: Do không có yếu tố cảm tính như kiểm thử thủ công, kiểm thử tự động mang lại kết quả ổn định và đáng tin cậy.
- **Tích hợp kiểm thử vào CI/CD**: Giúp kiểm tra phần mềm liên tục sau mỗi lần commit hoặc build.

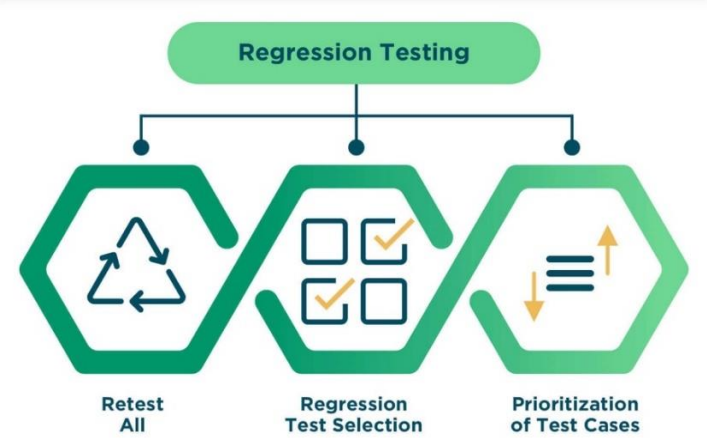
- **Giảm chi phí dài hạn**: Mặc dù chi phí ban đầu cho kiểm thử tự động cao (thiết lập công cụ, viết script, đào tạo nhân sự), nhưng về lâu dài lại tiết kiệm nhiều hơn.
- **Cải thiện độ bao phủ kiểm thử**: Do có thể tự động hóa kiểm thử trên nhiều dữ liệu, trình duyệt, thiết bị nên tăng khả năng bao phủ kiểm thử so với giới hạn của con người.

2.2 Các kỹ thuật kiểm thử tự động

Trong lĩnh vực kiểm thử phần mềm, việc áp dụng đúng kỹ thuật kiểm thử là yếu tố quan trọng để đạt được hiệu quả và tính bao phủ cao. Đặc biệt trong kiểm thử tự động, các kỹ thuật được lựa chọn không chỉ dựa trên yêu cầu kiểm tra của hệ thống mà còn phụ thuộc vào khả năng tái sử dụng, mức độ tự động hóa và mức độ phức tạp của chức năng cần kiểm thử. Dưới đây là ba kỹ thuật kiểm thử tự động tiêu biểu và phổ biến nhất trong thực tế triển khai. [9]

2.2.1. Kiểm thử hồi quy (Regression Testing)

Kiểm thử hồi quy là kỹ thuật được sử dụng nhằm đảm bảo rằng các chức năng đã hoạt động đúng trước đó sẽ tiếp tục hoạt động chính xác sau mỗi lần thay đổi mã nguồn. Sự thay đổi này có thể là do sửa lỗi, cập nhật tính năng hoặc thay đổi kiến trúc hệ thống.



Hình 0.1: Chiến lược trong kiểm thử hồi quy

Kỹ thuật này giúp kiểm soát rủi ro phát sinh lỗi không mong muốn trong quá trình phát triển và bảo trì phần mềm. Trong môi trường phát triển phần mềm hiện đại với chu kỳ cập nhật liên tục và thời gian đưa sản phẩm ra thị trường ngắn, kiểm thử hồi quy là một yêu cầu bắt buộc. Việc tự động hóa kiểm thử hồi quy cho phép chạy các bộ test case đã được xây dựng một cách nhanh chóng, nhất quán và lặp lại nhiều lần sau mỗi thay đổi.

Quy trình kiểm thử hồi quy

Quy trình kiểm thử hồi quy thường bao gồm các bước:

Bước 1: Xác định phạm vi kiểm thử hồi quy: Lựa chọn các tính năng, module hoặc luồng nghiệp vụ bị ảnh hưởng trực tiếp và gián tiếp bởi thay đổi.

Bước 2: Tái sử dụng bộ test case sẵn có**:** Sử dụng lại các kịch bản kiểm thử (test case) đã được xây dựng và xác nhận từ các chu kỳ phát triển trước.

Bước 3: Thực thi kiểm thử: Tiến hành chạy kiểm thử trên toàn bộ hoặc một phần hệ thống theo chiến lược xác định.

Bước 4: Đánh giá kết quả và phân tích lỗi**:** So sánh kết quả thực thi với kết quả mong đợi, ghi nhận lỗi phát sinh và đánh giá mức độ nghiêm trọng.

Tự động hóa trong kiểm thử hồi quy

Trong thực tế, kiểm thử hồi quy thường phải lặp lại nhiều lần mỗi khi có thay đổi, và khối lượng kiểm thử có xu hướng tăng dần theo thời gian. Do đó, việc tự động hóa kiểm thử hồi quy là giải pháp tất yếu nhằm:

- Tiết kiệm thời gian và chi phí nhân sự trong dài hạn.
- Tăng độ bao phủ kiểm thử**,** đặc biệt ở các hệ thống lớn, phức tạp.
- Nâng cao độ chính xác và tính nhất quán trong quá trình thực thi kiểm thử.
- Đáp ứng yêu cầu kiểm thử liên tục (Continuous Testing) trong môi trường DevOps.

Các công cụ phổ biến được sử dụng để tự động hóa kiểm thử hồi quy gồm có: Selenium, TestNG, Katalon Studio, JUnit, Appium, kết hợp cùng các nền tảng CI như Jenkins, GitLab CI/CD để tích hợp vào pipeline phát triển.

2.2.2. Kiểm thử API

Kiểm thử API là kỹ thuật tập trung vào việc xác minh hoạt động của các giao diện lập trình ứng dụng – nơi các thành phần trong hệ thống phần mềm tương tác và trao đổi dữ liệu với nhau. Không giống như kiểm thử giao diện người dùng, kiểm thử API tập trung vào lớp trung gian giữa giao diện và cơ sở dữ liệu hoặc logic xử lý.

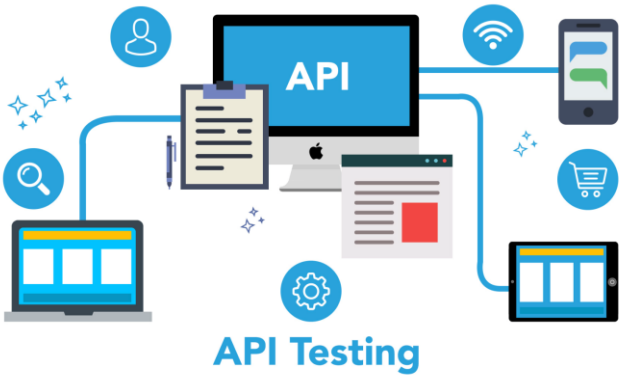
Kiểm thử API là một kỹ thuật trọng tâm trong đảm bảo chất lượng phần mềm, tập trung vào việc xác minh hoạt động của các giao diện lập trình ứng dụng (API)**.** Đây là những điểm mà các thành phần trong hệ thống phần mềm tương tác và trao đổi dữ liệu với nhau. Khác với kiểm thử giao diện người dùng (UI), kiểm thử API tập trung vào lớp trung gian nằm giữa giao diện người dùng và lớp cơ sở dữ liệu hoặc logic xử lý nghiệp vụ. [5]
Kỹ thuật này đánh giá toàn diện các khía cạnh quan trọng của hệ thống, bao gồm:

- Khả năng đáp ứng (Responsiveness): Đảm bảo API phản hồi nhanh chóng và hiệu quả.
- Độ chính xác của dữ liệu trả về (Data Accuracy): Xác minh tính đúng đắn của dữ liệu được API trả về sau mỗi yêu cầu.
- Tính toàn vẹn (Integrity): Đảm bảo dữ liệu được duy trì nhất quán và không bị sai lệch trong quá trình trao đổi.
- Bảo mật (Security): Kiểm tra các lỗ hổng bảo mật tiềm ẩn trong giao tiếp giữa các hệ thống, bảo vệ dữ liệu nhạy cảm.

Một ưu điểm nổi bật của kiểm thử API là khả năng thực hiện sớm trong chu kỳ phát triển phần mềm**.** Điều này giúp phát hiện lỗi từ giai đoạn đầu, từ đó giảm đáng kể chi phí và công sức sửa lỗi ở các giai đoạn sau.

Hơn nữa, kiểm thử API cực kỳ phù hợp để tự động hóa do những đặc tính sau:

- Không phụ thuộc vào giao diện đồ họa: Việc kiểm thử không yêu cầu giao diện người dùng, giúp loại bỏ sự phức tạp và độ trễ liên quan đến UI.
- Dễ thực hiện song song: Các kiểm thử API có thể chạy đồng thời, rút ngắn đáng kể thời gian kiểm thử tổng thể.
- Khả năng mở rộng tốt**:** Dễ dàng mở rộng phạm vi kiểm thử để bao quát nhiều kịch bản và API khác nhau.



Hình 0.2: Cách hoạt động của API

Quy trình kiểm thử API

Quy trình kiểm thử API tiêu chuẩn thường bao gồm các bước:

Bước 1: Xác định các API cần kiểm thử: Xác định rõ các endpoint, phương thức HTTP (GET, POST, PUT, DELETE...), định dạng dữ liệu, yêu cầu xác thực.

Bước 2: Thiết kế kịch bản kiểm thử (Test Case Design)**:** Bao gồm các trường hợp đúng (positive testing) và sai (negative testing), kiểm thử biên, kiểm thử dữ liệu thiếu/không hợp lệ.

Bước 3: Thực thi kiểm thử: Gửi yêu cầu (request), phân tích phản hồi (response), đối chiếu với kết quả mong đợi.

Bước 4: Đánh giá và ghi nhận kết quả: Xác định mức độ đáp ứng, lỗi phát sinh và hiệu suất của API.

2.2.3. Kiểm thử hiệu năng (Performance Testing)

Kiểm thử hiệu năng là kỹ thuật kiểm thử phi chức năng dùng để đánh giá khả năng vận hành của hệ thống phần mềm trong điều kiện thực tế, dưới nhiều tình huống tải khác nhau. Kỹ thuật này không chỉ tập trung vào tốc độ phản hồi mà còn kiểm tra tính ổn định, khả năng mở rộng và độ bền vững của hệ thống trong thời gian dài sử dụng.

Kiểm thử hiệu năng được chia thành nhiều hình thức khác nhau, mỗi hình thức phục vụ một mục tiêu cụ thể trong quá trình đánh giá hiệu suất hệ thống:

- **Kiểm thử tải (Load Testing):** Được sử dụng để đánh giá khả năng xử lý của hệ thống khi phải tiếp nhận một lượng người dùng hoặc yêu cầu truy cập ở mức giới hạn thiết kế.
- **Kiểm thử áp lực (Stress Testing):** Nhằm xác định ngưỡng chịu tải cực đại của hệ thống bằng cách gia tăng lượng tải vượt xa mức thông thường, qua đó đánh giá khả năng phản hồi và phục hồi sau quá tải.
- **Kiểm thử độ bền (Soak Testing):** Được thực hiện bằng cách cho hệ thống chạy liên tục trong thời gian dài dưới tải ổn định nhằm phát hiện các vấn đề phát sinh như rò rỉ bộ nhớ, suy giảm hiệu suất hoặc lỗi tích lũy.
- **Kiểm thử khả năng mở rộng (Scalability Testing):** Dùng để đánh giá khả năng hệ thống có thể mở rộng về số lượng người dùng, dung lượng dữ liệu hoặc tài nguyên phần cứng mà vẫn duy trì hiệu năng ổn định.

Việc thực hiện kiểm thử hiệu năng không chỉ giúp đảm bảo rằng phần mềm đáp ứng được yêu cầu kỹ thuật đã đề ra, mà còn góp phần nâng cao độ tin cậy, hiệu quả đầu tư và khả năng duy trì chất lượng dịch vụ (QoS – Quality of Service). Kiểm thử hiệu năng còn đóng vai trò trong việc:

- Phát hiện sớm các vấn đề hiệu suất tiềm ẩn**,** từ đó giảm thiểu rủi ro trước khi hệ thống được triển khai chính thức.
- Hỗ trợ quyết định nâng cấp hoặc tối ưu tài nguyên hệ thống**,** tránh lãng phí hoặc thiếu hụt trong vận hành.

- Cải thiện trải nghiệm người dùng, đặc biệt trong các thời điểm cao điểm như chương trình khuyến mãi, sự kiện trực tuyến, hoặc kỳ tuyển sinh.

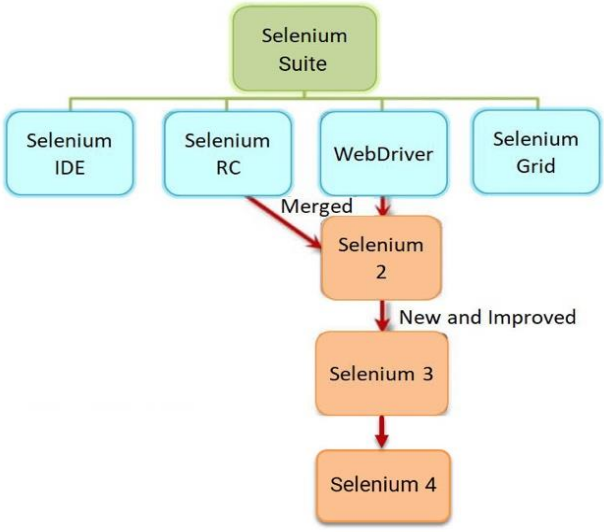
2.3. Các công cụ kiểm thử tự động

Việc lựa chọn công cụ kiểm thử phù hợp là một yếu tố quan trọng để triển khai kiểm thử tự động thành công. Trên thị trường hiện nay có nhiều công cụ hỗ trợ kiểm thử tự động, mỗi công cụ lại phục vụ cho các mục đích khác nhau như kiểm thử giao diện người dùng, kiểm thử API, hoặc kiểm thử hiệu năng. Tùy vào mục tiêu kiểm thử, quy mô hệ thống và năng lực kỹ thuật của đội ngũ mà doanh nghiệp có thể lựa chọn công cụ phù hợp.

Trong mục này, đề án Một số công cụ kiểm thử tự động phổ biến bao gồm Selenium, Katalon, Postman và Jmeter.

2.3.1 Selenium

Selenium được phát triển lần đầu vào năm 2004 bởi Jason Huggins tại ThoughtWorks với tên gọi ban đầu là “JavaScriptTestRunner”. Sau đó, công cụ được mở rộng thành một dự án mã nguồn mở có tên Selenium và phát triển qua nhiều phiên bản: Selenium RC, Selenium Gird, Selenium IDE và Selenium WebDriver – phiên bản hiện đại nhất và được sử dụng phổ biến hiện nay. [14]



Hình 0.3: Các thành phần của Selenium

Selenium IDE:

- Một tiện ích mở rộng cho trình duyệt (Firefox/Chrome), cho phép ghi – phát lại các hành động trên trình duyệt.
- Phù hợp với người mới bắt đầu và cho các test case đơn giản.

Selenium RC:

- Cho phép kiểm thử tự động trên trình duyệt web thông qua một server trung gian.
- Nó hỗ trợ nhiều ngôn ngữ lập trình như Java, Python, C#, PHP,... Tuy nhiên, RC có nhiều hạn chế như tốc độ chậm, cấu hình phức tạp và khó tương tác với các công nghệ web hiện đại.
- Hiện tại, Selenium RC đã ngừng phát triển và được thay thế hoàn toàn bởi **Selenium WebDriver**.

Selenium WebDriver:

- Là thành phần cốt lõi dùng để viết script kiểm thử tự động trên trình duyệt.
- Giao tiếp trực tiếp với trình duyệt thông qua API gốc của từng trình duyệt.
- Hỗ trợ hầu hết các trình duyệt hiện nay như Chrome, Firefox, Safari, Edge,…

Selenium Gird:

- Cho phép chạy test song song trên nhiều máy, trình duyệt và hệ điều hành khác nhau.
- Hữu ích trong kiểm thử phân tán (distributed testing) và tăng tốc quá trình kiểm thử.

Đặc điểm và chức năng chính:

- Mã nguồn mở, miễn phí**:** Selenium là công cụ kiểm thử tự động mã nguồn mở, được cộng đồng phát triển rộng rãi và liên tục cập nhật.
- Hỗ trợ kiểm thử giao diện người dùng (UI)**:** Cho phép mô phỏng hầu hết các thao tác của người dùng trên trình duyệt như click chuột, nhập liệu, kéo – thả, điều hướng giữa các trang,…

- Tương thích đa trình duyệt và đa nền tảng**:** Selenium hỗ trợ kiểm thử trên nhiều trình duyệt phổ biến (Chrome, Firefox, Safari, Edge, Internet Explorer,…) và hệ điều hành (Windows, Linux, macOS).
- Đa ngôn ngữ lập trình**:** Hỗ trợ viết test script bằng các ngôn ngữ lập trình phổ biến như Java, Python, C#, Ruby, JavaScript, giúp linh hoạt trong việc lựa chọn công nghệ phù hợp với đội ngũ phát triển.
- Khả năng tích hợp cao**:** Dễ dàng tích hợp với các công cụ CI/CD như Jenkins, GitLab CI, Bamboo,… để thực hiện kiểm thử liên tục trong chu trình DevOps.
- Khả năng mở rộng mạnh mẽ**:** Có thể kết hợp với các thư viện và framework hỗ trợ như TestNG, JUnit, Cucumber, Allure Report,… để tăng cường khả năng báo cáo, tổ chức và quản lý test case. [6]

2.3.2 Katalon Studio

Katalon Studio được phát hành chính thức năm 2015 bởi Katalon LLC, được xây dựng dựa trên nền tảng của **Selenium** và **Appium**, nhằm đơn giản hóa quy trình kiểm thử tự động cho cả kỹ sư kiểm thử và người không chuyên.

Đặc điểm và tính năng chính:

Hình 0.4: Tính năng chính của Katalon

Simple deployment: Katalon Studio cung cấp một gói cài đặt duy nhất, trong đó tích hợp đầy đủ các công cụ cần thiết để triển khai một giải pháp kiểm thử tự động

toàn diện. Người dùng không cần cài đặt thêm thành phần nào khác, giúp rút ngắn thời gian chuẩn bị hệ thống.

Quick & easy set-up: Không chỉ cài đặt nhanh, Katalon còn cho phép người kiểm thử dễ dàng thiết lập môi trường làm việc. Với các mẫu test có sẵn, cùng kho đối tượng và thư viện từ khóa được dựng sẵn, người dùng có thể khởi chạy kịch bản đầu tiên mà không mất nhiều thời gian tìm hiểu.

Faster & better results: Nhờ có cấu trúc mẫu và hướng dẫn rõ ràng, tester có thể nhanh chóng tạo và thực thi các kịch bản kiểm thử tự động. Từ khâu tạo dự án, viết kịch bản, thực hiện kiểm thử đến tạo báo cáo và bảo trì – mọi công đoạn đều được hỗ trợ hiệu quả và tối ưu về thời gian.

Flexible modes: Katalon Studio đáp ứng linh hoạt nhiều đối tượng người dùng. Với người mới, công cụ ghi thao tác và thư viện từ khóa là những tính năng hỗ trợ hiệu quả. Trong khi đó, người kiểm thử chuyên nghiệp có thể sử dụng môi trường phát triển tích hợp (IDE) để viết các kịch bản nâng cao.

Ease of use: Giao diện thân thiện và cách sử dụng đơn giản giúp ngay cả những người ít kinh nghiệm lập trình cũng có thể khai thác hiệu quả công cụ này. Việc học và sử dụng Katalon không đòi hỏi quá nhiều kỹ thuật chuyên sâu.

Cross-browser application: Katalon Studio hỗ trợ kiểm thử trên nhiều nền tảng và trình duyệt phổ biến hiện nay. Công cụ tương thích với cả hệ điều hành Windows (32-bit, 64-bit – các phiên bản 7, 8, 10) và macOS (từ phiên bản 10.5 trở lên), đảm bảo kiểm thử chéo trình duyệt diễn ra mượt mà.

2.2.3 Playwright

Playwright là một framework hiện đại cho kiểm thử tự động hóa UI, cho phép kiểm thử web trên nhiều trình duyệt khác nhau (Chromium, Firefox, WebKit). Ra đời sau Selenium, Playwright được thiết kế để khắc phục các vấn đề của các công cụ cũ như tốc độ chậm, test không ổn định. [11]

Đặc điểm nổi bật

- Kiến trúc đa nền tảng, đa trình duyệt: Playwright được thiết kế để hỗ trợ ba trình duyệt chính là Chromium, Firefox và WebKit. Điều này cho phép các tổ chức có thể thực hiện kiểm thử tương thích trình duyệt một cách toàn diện, bao gồm cả trình duyệt Safari trên macOS thông qua WebKit – điều mà nhiều công cụ khác chưa làm tốt
- Đa ngôn ngữ lập trình: Playwright hỗ trợ nhiều ngôn ngữ như JavaScript, TypeScript, Python, Java và C#, giúp các nhóm phát triển dễ dàng tích hợp với ngôn ngữ quen thuộc, đồng thời tận dụng được hệ sinh thái thư viện phong phú của các nền tảng này.
- Kiểm thử song song và đa tab: Một điểm mạnh nổi bật là khả năng kiểm thử đa tab, đa phiên (multi-context), phù hợp với các ứng dụng web có tính năng nâng cao như mở popup, xử lý session song song.
- Tự động chờ (Auto-waiting): Playwright tự động chờ các hành động như DOM load, animation complete, element visible,... giúp giảm thiểu việc viết các lệnh wait() thủ công, từ đó làm cho test case ổn định và dễ bảo trì hơn.
- Tích hợp giả lập thiết bị và kiểm thử mobile web**:** Dù không phải công cụ kiểm thử mobile native, Playwright vẫn hỗ trợ mô phỏng nhiều thiết bị di động, bao gồm thay đổi độ phân giải, user agent, và hỗ trợ kiểm thử trong chế độ giả lập mạng chậm.

Tính năng chính:

- Chụp ảnh, quay video và log test case tự động, phục vụ mục đích debug và báo cáo lỗi.
- Hỗ trợ kiểm thử component riêng biệt (Component Testing), giúp test từng phần nhỏ trong ứng dụng SPA (Single Page Application).
- Cho phép ghi lại thao tác để tạo test script ban đầu, giúp tăng tốc quá trình xây dựng kịch bản kiểm thử cho người mới.
- Khả năng tích hợp CI/CD dễ dàng thông qua CLI, cấu hình YAML, phù hợp với GitHub Actions, GitLab, Jenkins,...

- Snapshot Testing**:** So sánh giao diện thực tế và giao diện mong đợi, phát hiện lỗi giao diện (UI regression).

2.3.4 Cypress

Cypress là một **framework kiểm thử tự động giao diện người dùng (UI)** hiện đại, được thiết kế chủ yếu để kiểm thử các ứng dụng web viết bằng JavaScript. Khác với Selenium – vốn chạy bên ngoài trình duyệt và điều khiển trình duyệt từ xa – Cypress chạy trực tiếp **bên trong trình duyệt**, cùng ngữ cảnh với ứng dụng, giúp việc kiểm thử nhanh chóng và chính xác hơn. [12]

- Kiến trúc dựa trên trình duyệt thật: Cypress hoạt động bên trong trình duyệt người dùng, thay vì kiểm soát trình duyệt từ bên ngoài như Selenium. Điều này giúp tăng độ chính xác và hiệu suất của kiểm thử, đặc biệt trong việc tương tác DOM, xử lý JS async và các tình huống phức tạp trong SPA.
- Thân thiện với frontend developer: Được viết bằng JavaScript và chạy trực tiếp trong trình duyệt, Cypress rất phù hợp cho các dự án frontend hiện đại (React, Angular, Vue) và dễ dàng tích hợp vào vòng đời phát triển (Dev → Test → Deploy).
- GUI Test Runner mạnh mẽ: Cho phép xem trực tiếp từng bước kiểm thử với khả năng tua lại các bước, highlight DOM tương ứng, giúp dễ dàng debug và phát hiện nguyên nhân lỗi.
- Mock/stub dữ liệu linh hoạt: Với lệnh cy.intercept(), Cypress có thể mô phỏng các request/response API, giả lập các kịch bản khác nhau như lỗi mạng, lỗi dữ liệu,... mà không cần server backend thực sự.

Tính năng chính:

- Time Travel Debugging: Cho phép quan sát lại trạng thái ứng dụng ở từng bước kiểm thử, giống như xem lại video từng bước.

- Hỗ trợ kiểm thử API song song với UI: Cypress có thể gửi request API, validate response, sau đó thực hiện kiểm thử giao diện, đảm bảo test bao phủ toàn bộ luồng người dùng.
- Tích hợp screenshot/video khi test fail**,** hỗ trợ phân tích lỗi trong pipeline CI.
- Khả năng chạy test song song và phân tán thông qua Cypress Dashboard (có phí) để tối ưu tốc độ.
- Hệ sinh thái plugin phong phú: Hỗ trợ accessibility testing, visual testing, performance testing thông qua plugin community.

2.3.5 Appium

Giống như Selenium, Appium cũng là một công cụ kiểm tra tự động mã nguồn mở, nhưng dành cho các ứng dụng di động. Sử dụng giao thức dây JSON di động, Appium cho phép người dùng viết các bài kiểm tra giao diện người dùng tự động cho các ứng dụng di động gốc, dựa trên web và kết hợp trên cả Android và iOS. [13]

- Cross-platform test thực sự: Appium cho phép viết một bộ test script có thể chạy được trên cả Android và iOS mà không cần chỉnh sửa nhiều. Đây là ưu điểm nổi bật trong việc kiểm thử các ứng dụng di động đa nền tảng.
- Kiến trúc plugin hóa (Modular Architecture)**:** Appium 2.0 giới thiệu cơ chế plugin driver giúp người dùng có thể mở rộng, nâng cấp và cấu hình kiểm thử cho các nền tảng khác nhau như Espresso, XCUITest, Flutter, Windows, Mac,... một cách linh hoạt.
- Hỗ trợ nhiều framework test phổ biến: Appium dễ dàng tích hợp với JUnit, TestNG, Cucumber, Mocha,... và các thư viện assertion để viết test dễ dàng theo nhiều phong cách khác nhau (unit, BDD, keyword-driven).
- Hoạt động trên thiết bị thật và giả lập: Appium tương thích tốt với emulator/simulator cũng như thiết bị vật lý, phù hợp với cả kiểm thử phát triển (local) lẫn kiểm thử phân tán qua cloud (BrowserStack, Sauce Labs).

Tính năng chính:

- Tích hợp với Selenium WebDriver, kế thừa API chuẩn giúp dễ học với người đã có nền tảng automation.
- Hỗ trợ kiểm thử gesture và tương tác nâng cao như vuốt, kéo, chạm, xoay, cảm biến,...
- Khả năng kiểm tra log, ghi lại màn hình và chụp ảnh trên mobile trong quá trình kiểm thử.
- Tích hợp CI/CD dễ dàng thông qua CLI và framework hỗ trợ, chạy được trong Jenkins, GitHub Actions...
- Khả năng kiểm thử các công nghệ UI mới như React Native, Flutter, Ionic... thông qua plugin driver tương ứng.

2.3.6 Postman

Postman được phát triển vào năm 2012 bởi Abhinav Asthana như một tiện ích trên trình duyệt Chrome nhằm đơn giản hóa việc kiểm thử API. Sau đó, Postman nhanh chóng phát triển thành một nền tảng toàn diện hỗ trợ kiểm thử API RESTful, hiện có hơn 20 triệu người dùng trên toàn thế giới.

Trong bối cảnh hiện nay, các hệ thống phần mềm ngày càng được xây dựng theo kiến trúc microservices và hướng dịch vụ (SOA), các API đóng vai trò là cầu nối giao tiếp chính giữa các thành phần phần mềm. Do đó, việc kiểm thử API chính xác, hiệu quả là vô cùng quan trọng nhằm đảm bảo tính ổn định, độ tin cậy và hiệu năng của toàn bộ hệ thống. Postman đã trở thành công cụ không thể thiếu giúp các nhóm phát triển và kiểm thử thực hiện nhiệm vụ này một cách hiệu quả, giảm thiểu lỗi và tăng cường tự động hóa trong quy trình phát triển phần mềm.

Kiến trúc và các thành phần chính:

Postman được xây dựng dựa trên kiến trúc client-server với các thành phần chính bao gồm:

- **Giao diện người dùng (UI)**: Cung cấp môi trường trực quan để thiết kế, gửi các yêu cầu HTTP, và xem phản hồi. Giao diện này hỗ trợ đa nền tảng, bao gồm ứng dụng desktop trên Windows, macOS và Linux, cũng như tiện ích mở rộng trên trình duyệt.

- **Bộ xử lý logic kiểm thử**: Cho phép người dùng viết các đoạn mã kiểm thử tự động sử dụng JavaScript, giúp tự động kiểm tra các giá trị trả về, mã trạng thái, thời gian phản hồi, và các điều kiện nghiệp vụ khác.
- **Cơ chế lưu trữ và đồng bộ dữ liệu**: Hỗ trợ lưu trữ collection (bộ sưu tập các yêu cầu API), môi trường (environment) và các biến toàn cục (global variables) trên cloud hoặc local, giúp dễ dàng chia sẻ và tái sử dụng trong nhóm.
- **Tích hợp với các hệ thống CI/CD**: Thông qua công cụ dòng lệnh Newman, Postman có thể được tích hợp trong pipeline của các công cụ tự động hóa như Jenkins, GitLab CI, CircleCI nhằm chạy kiểm thử tự động liên tục.

Các tính năng nổi bật:

- Chuyên dùng cho kiểm thử API (REST, GraphQL, SOAP).
- Hỗ trợ các phương thức HTTP: GET, POST, PUT, DELETE,…
- Gửi request, kiểm tra phản hồi, kiểm tra mã trạng thái (status code), thời gian phản hồi.
- Cho phép viết kiểm thử tự động bằng JavaScript trong phần “Tests”.
- Tích hợp dễ dàng với môi trường CI/CD và hỗ trợ chạy test hàng loạt qua Newman CLI**.**

2.3.7 Apache Jmeter

Apache JMeter được phát triển bởi Stefano Mazzocchi tại Apache Software Foundation và phát hành lần đầu năm 1998. Ban đầu, JMeter được thiết kế để kiểm thử các ứng dụng web, nhưng hiện nay đã mở rộng ra nhiều giao thức khác.

Apache Jmeter là công cụ kiểm thử hiệu năng phổ biến và mạnh mẽ. JMeter có thể mô phỏng hàng trăm đến hàng nghìn người dùng truy cập cùng lúc để đo lường tốc độ phản hồi, khả năng chịu tải và độ ổn định của hệ thống dưới áp lực cao.

Tùy thuộc vào nhu cầu thực tế, tổ chức có thể lựa chọn một hoặc kết hợp nhiều công cụ nhằm đảm bảo độ bao phủ kiểm thử và đạt được hiệu quả tối ưu trong việc đánh giá chất lượng phần mềm.

Trong các hệ thống phần mềm hiện đại, đặc biệt là các ứng dụng web và dịch vụ trực tuyến, hiệu năng đóng vai trò then chốt trong việc đảm bảo trải nghiệm người dùng, duy trì tính cạnh tranh và đáp ứng yêu cầu kinh doanh. Apache JMeter là công cụ phổ biến và đáng tin cậy trong việc đánh giá hiệu năng hệ thống thông qua các bài kiểm thử mô phỏng tải (load testing), kiểm thử sức bền (endurance testing), kiểm thử áp lực (stress testing), và kiểm thử khối lượng (volume testing).

JMeter giúp các tổ chức và nhóm phát triển đánh giá khả năng chịu tải, thời gian phản hồi, thông lượng và độ ổn định của hệ thống khi phải xử lý đồng thời hàng trăm đến hàng nghìn yêu cầu truy cập, từ đó phát hiện sớm các vấn đề hiệu suất và tối ưu hạ tầng kỹ thuật.

Các chức năng chính:

Apache JMeter cung cấp một tập hợp phong phú các tính năng phục vụ cho kiểm thử hiệu năng:

- **Kiểm thử tải (Load Testing)**: Mô phỏng nhiều người dùng ảo truy cập vào hệ thống trong cùng một khoảng thời gian để đánh giá hiệu suất xử lý.
- **Kiểm thử sức bền (Soak Testing)**: Đánh giá độ ổn định của hệ thống trong thời gian dài khi duy trì tải cao.
- **Hỗ trợ đa giao thức**: Bao gồm HTTP/HTTPS (web), JDBC (cơ sở dữ liệu), SOAP/REST (dịch vụ web), FTP, SMTP (email), v.v.
- **Thu thập và hiển thị số liệu hiệu năng**: Bao gồm thời gian phản hồi trung bình, số lượng request mỗi giây (Throughput), tỷ lệ lỗi (Error Rate), thời gian tối đa/tối thiểu, phần trăm phần trăm (percentile), v.v.
- **Kết xuất báo cáo**: Hỗ trợ xuất báo cáo kiểm thử dưới dạng bảng dữ liệu, đồ thị, và HTML tương tác phục vụ cho việc trình bày và phân tích kết quả.
- **Khả năng mở rộng cao**: Có thể chạy kiểm thử phân tán trên nhiều máy để tăng số lượng người dùng mô phỏng.

- **Tích hợp CI/CD**: Dễ dàng tích hợp với Jenkins, GitLab CI, hoặc các công cụ DevOps khác để thực hiện kiểm thử hiệu năng định kỳ hoặc tự động sau mỗi lần triển khai.

2.4 So sánh và đánh giá các công cụ kiểm thử

Mỗi công cụ kiểm thử tự động có những ưu điểm và hạn chế riêng. Để đưa ra quyết định lựa chọn công cụ phù hợp, cần tiến hành so sánh dựa trên các tiêu chí cơ bản như: mức độ dễ sử dụng, khả năng mở rộng, hỗ trợ đa nền tảng, cộng đồng hỗ trợ, tích hợp CI/CD, chi phí và tính năng kỹ thuật.

Bảng so sánh tổng hợp một số công cụ phổ biến:

Bảng 0-1: So sánh công cụ kiểm thử

Công cụ	Ưu điểm	Hạn chế
Selenium	Mã nguồn mở, linh hoạt, hỗ trợ đa trình duyệt và ngôn ngữ, tích hợp CI dễ	Yêu cầu kỹ năng lập trình, không có giao diện người dùng trực quan
Katalon	Giao diện thân thiện, hỗ trợ đa nền tảng, dễ học	Một số tính năng nâng cao chỉ có ở bản thương mại
Postman	Dễ sử dụng, mạnh về kiểm thử API, hỗ trợ viết test script và xuất báo cáo	Không hỗ trợ kiểm thử giao diện người dùng
JMeter	Mạnh về kiểm thử tải, hiệu năng, có khả năng mở rộng tốt	Giao diện chưa thân thiện, cần thời gian làm quen

Trong số các công cụ trên, **Selenium** nổi bật với tính năng mạnh mẽ trong kiểm thử UI, khả năng mở rộng và tùy biến cao. Đây là công cụ được đề án lựa chọn vì phù hợp với mục tiêu kiểm thử giao diện web, dễ tích hợp vào quy trình DevOps và có cộng đồng người dùng đông đảo, tài liệu phong phú hỗ trợ triển khai thực tế.

2.5 Lợi ích và thách thức khi áp dụng kiểm thử tự động

2.5.1 Lợi ích

- **Tăng hiệu quả kiểm thử**: Với khả năng thực hiện hàng trăm, thậm chí hàng nghìn trường hợp kiểm thử trong thời gian ngắn, kiểm thử tự động giúp rút ngắn đáng kể thời gian và công sức so với kiểm thử thủ công.
- **Tăng tính nhất quán và độ tin cậy**: Các tập lệnh kiểm thử khi được lập trình chính xác sẽ luôn cho ra kết quả đồng nhất, loại bỏ sai sót do yếu tố con người gây ra.
- **Hỗ trợ kiểm thử hồi quy hiệu quả**: Các test case cũ có thể được chạy lại nhiều lần sau mỗi thay đổi phần mềm mà không cần tốn thêm tài nguyên.
- **Tích hợp vào quy trình DevOps/CI-CD**: Kiểm thử có thể được kích hoạt tự động mỗi khi có thay đổi mã nguồn, đảm bảo phần mềm luôn được kiểm tra trước khi đưa vào môi trường thực tế.
- **Tiết kiệm chi phí về lâu dài**: Dù chi phí đầu tư ban đầu có thể cao, nhưng về dài hạn sẽ giảm thiểu nhân sự kiểm thử, tiết kiệm thời gian và hạn chế lỗi.

2.5.2 Thách thức

- **Chi phí triển khai ban đầu cao**: Bao gồm việc lựa chọn công cụ phù hợp, thiết lập môi trường kiểm thử, và đào tạo đội ngũ kiểm thử.
- **Yêu cầu kỹ năng kỹ thuật cao**: Kiểm thử tự động không chỉ đòi hỏi kiến thức kiểm thử mà còn cần khả năng lập trình, hiểu biết về hệ thống, và kỹ năng sử dụng công cụ.
- **Bảo trì test script tốn thời gian**: Khi giao diện hoặc logic của phần mềm thay đổi, các tập lệnh kiểm thử cần được cập nhật để đảm bảo tính chính xác.
- **Không phù hợp với tất cả loại kiểm thử**: Một số dạng kiểm thử yêu cầu đánh giá trải nghiệm người dùng, thẩm mỹ giao diện, hoặc cảm nhận hành vi không thể tự động hóa hiệu quả.

- **Sai lầm khi lạm dụng tự động hóa**: Nếu không được thiết kế hợp lý, hệ thống kiểm thử tự động có thể trở thành gánh nặng và gây ra chi phí bảo trì lớn hơn so với kiểm thử thủ công.

2.6 Kết luận chương 2

Chương 2 đã trình bày tổng quan về kiểm thử tự động phần mềm, bao gồm khái niệm, vai trò, các kỹ thuật kiểm thử phổ biến như kiểm thử hồi quy, kiểm thử API và kiểm thử hiệu năng. Đồng thời, chương cũng phân tích các công cụ kiểm thử tự động đang được sử dụng rộng rãi như Selenium, Katalon Studio, Postman và Apache JMeter, kèm theo đánh giá so sánh dựa trên nhiều tiêu chí thực tiễn.

Ngoài ra, chương cũng chỉ ra những lợi ích nổi bật khi áp dụng kiểm thử tự động trong quy trình phát triển phần mềm hiện đại, song song với những thách thức mà tổ chức có thể gặp phải khi triển khai.

Những nội dung trong chương 2 sẽ là nền tảng để chương tiếp theo tập trung vào **ứng dụng kiểm thử tự động với công cụ Selenium trong thực tế**, từ đó đánh giá hiệu quả và đề xuất hướng áp dụng phù hợp trong chuyển đổi số tại doanh nghiệp.

CHƯƠNG 3: ỨNG DỤNG KIỂM THỬ TỰ ĐỘNG PHẦN MỀM TẠI CÔNG TY A1 CONSULTING

3.1 Giới thiệu về Công ty A1 Consulting

3.1.1 Lịch sử hình thành và phát triển

A1 Consulting là doanh nghiệp hoạt động chuyên sâu trong lĩnh vực tư vấn và triển khai hệ thống quản trị doanh nghiệp dựa trên nền tảng mã nguồn mở Odoo. Công ty có lịch sử phát triển đáng chú ý với quá trình chuyển đổi và mở rộng chiến lược trong suốt hơn một thập kỷ.

Tiền thân của A1 Consulting là **Onnet Consulting**, một trong những đơn vị tiên phong tại Việt Nam trong việc đưa Odoo – phần mềm ERP mã nguồn mở phổ biến toàn cầu – đến gần hơn với cộng đồng doanh nghiệp Việt. Onnet được biết đến như một trong số ít đơn vị đầu tiên tại Đông Nam Á tham gia vào hệ sinh thái đối tác chính thức của Odoo Bỉ, với nhiều thành tựu triển khai Odoo tại thị trường nội địa.

Sau một giai đoạn sáp nhập và vận hành dưới thương hiệu **AHT Tech**, công ty tiếp tục mở rộng quy mô hoạt động, cải tiến năng lực phát triển sản phẩm và chuẩn hóa quy trình triển khai dự án. Đến năm **2024**, công ty chính thức đổi tên thành **A1 Consulting** như một bước chuyển mình chiến lược để khẳng định vai trò mới: không chỉ là nhà cung cấp dịch vụ triển khai Odoo mà còn là đối tác tư vấn tổng thể cho chuyển đổi số doanh nghiệp tại Việt Nam và khu vực. [3]

3.1.2. Lĩnh vực hoạt động và vị thế hiện tại

Hiện nay, A1 Consulting tập trung chủ yếu vào các dịch vụ:

- Triển khai và tùy biến hệ thống ERP Odoo;
- Tư vấn quy trình nghiệp vụ doanh nghiệp (tài chính – kế toán, sản xuất, kho vận, nhân sự...);
- Xây dựng cổng tích hợp dữ liệu, giải pháp phần mềm mở rộng (POS, E-learning, eInvoice...);
- Dịch vụ bảo trì, hỗ trợ kỹ thuật và đào tạo vận hành Odoo.

Với đội ngũ chuyên gia chuyên sâu cả về nghiệp vụ và kỹ thuật, A1 Consulting hiện được công nhận là **đơn vị dẫn đầu thị trường Việt Nam về triển khai Odoo**, và là một trong những nhà cung cấp Odoo có số lượng dự án thành công nhiều nhất khu vực Đông Nam Á. Hệ thống khách hàng của công ty bao gồm hàng loạt doanh nghiệp từ vừa đến lớn, trải dài nhiều lĩnh vực: sản xuất, phân phối, dịch vụ logistics, giáo dục, y tế, bán lẻ…

3.1.3. Thực trạng kiểm thử phần mềm tại A1 Consulting

Do đặc thù là công ty triển khai giải pháp phần mềm có mức độ tùy biến cao, đội ngũ kỹ sư tại A1 Consulting thường xuyên phát triển, sửa đổi và mở rộng các module theo yêu cầu cụ thể của từng khách hàng. Quá trình này đòi hỏi khả năng kiểm thử linh hoạt và chính xác, đặc biệt trong các hệ thống ERP có nhiều luồng nghiệp vụ phức tạp và tích hợp đa chiều. [3]

Tuy nhiên, cho đến gần đây, phần lớn quá trình kiểm thử tại A1 Consulting vẫn được thực hiện theo hướng thủ công – tức tester thực hiện trực tiếp từng bước kiểm thử, ghi nhận kết quả và báo cáo lỗi theo cách truyền thống. Hạn chế chính của phương pháp này bao gồm:

- Chi phí nhân sự lớn khi cần kiểm thử lặp lại hàng chục lần.
- Khó duy trì tính nhất quán trong môi trường có nhiều tester và nhiều khách hàng.
- Không thể đáp ứng kiểm thử hồi quy ở quy mô lớn, đặc biệt khi hệ thống thường xuyên cập nhật.
- Thiếu tính tích hợp với quy trình DevOps và các công cụ CI/CD đang được áp dụng trong các nhóm phát triển Agile.

Với xu hướng chuyển đổi số trong nội bộ, cùng mục tiêu nâng cao chất lượng dịch vụ và tối ưu chi phí kiểm thử, A1 Consulting đã triển khai áp dụng kiểm thử tự động trong quy trình kiểm thử phần mềm – bắt đầu từ các hệ thống web được phát triển nội bộ dựa trên Odoo, và mở rộng dần ra các dự án khách hàng.

Việc ứng dụng kiểm thử tự động được kỳ vọng sẽ giúp:

- Tăng tốc độ kiểm thử trong các đợt phát hành (release) liên tục;
- Cải thiện chất lượng phần mềm bằng cách giảm lỗi phát sinh do kiểm thử thiếu sót;
- Giảm chi phí kiểm thử dài hạn;
- Tăng khả năng tích hợp và giám sát quy trình kiểm thử trong chuỗi DevOps – CI/CD hiện có.

3.2 Giới thiệu phần mềm lựa chọn để thực nghiệm kiểm thử

Trong phạm vi đề án này, phần mềm được lựa chọn để thực nghiệm kiểm thử tự động là **Odoo**, một hệ thống ERP mã nguồn mở có kiến trúc mô-đun linh hoạt và được sử dụng rộng rãi trong các tổ chức và doanh nghiệp trên toàn thế giới. Đề án tập trung kiểm thử trên hai phân hệ cốt lõi: **Sales (Bán hàng)** và **Purchase (Mua hàng)** - hai phân hệ đại diện cho quy trình vận hành chính trong chuỗi cung ứng của doanh nghiệp.

3.2.1. Giới thiệu phân hệ Sales và Purchase trong Odoo

Nền tảng quản trị doanh nghiệp **Odoo** (trước đây là TinyERP) được phát triển bởi công ty Odoo S.A., có trụ sở tại Bỉ, ra mắt phiên bản đầu tiên vào năm 2005. Từ năm 2014, phần mềm chính thức được đổi tên thành Odoo và liên tục phát triển, trở thành một hệ thống Hoạch định Nguồn lực Doanh nghiệp (ERP) toàn diện với khả năng mở rộng cao. Đến nay, Odoo đã phát hành tới phiên bản 17, tích hợp hàng trăm mô-đun phục vụ đa dạng các lĩnh vực quản trị doanh nghiệp như Kế toán, Bán hàng, Mua hàng, Quản lý kho, Nhân sự, Sản xuất, v.v. Các phân hệ này được thiết kế để hoạt động liền mạch, tạo nên một hệ sinh thái đồng bộ giúp tối ưu hóa các quy trình vận hành và nâng cao hiệu quả kinh doanh.

3.2.1.1 Phân hệ Sales (Bán hàng)

Phân hệ **Sales (Bán hàng)** trong Odoo là một cấu phần trọng tâm, cho phép doanh nghiệp quản lý toàn bộ chu trình bán hàng một cách hiệu quả, từ giai đoạn tiếp nhận yêu cầu từ khách hàng cho đến khi hoàn tất xuất hóa đơn. Phân hệ này có mối

liên kết chặt chẽ với các phân hệ khác trong hệ thống Odoo, đặc biệt là **Kho (Inventory)**, **Kế toán (Accounting)** và **Quản lý quan hệ khách hàng (CRM)**, đảm bảo luồng dữ liệu và quy trình nghiệp vụ được đồng bộ xuyên suốt.

Các chức năng chính của phân hệ Sales bao gồm:

- **Quản lý thông tin khách hàng và điều khoản thanh toán:** Hệ thống cho phép lưu trữ chi tiết hồ sơ khách hàng, bao gồm thông tin liên hệ, lịch sử giao dịch và các điều khoản thanh toán cụ thể áp dụng cho từng đối tượng khách hàng.
- **Tạo, gửi và xác nhận báo giá:** Người dùng có thể dễ dàng tạo các báo giá tùy chỉnh, gửi trực tiếp cho khách hàng qua email và theo dõi trạng thái phản hồi. Khi báo giá được chấp thuận, có thể chuyển đổi trực tiếp thành đơn hàng bán.
- **Quản lý đơn hàng bán (Sales Order):** Quản lý tập trung tất cả các đơn hàng bán, từ lúc khởi tạo đến khi hoàn thành, với khả năng điều chỉnh số lượng, giá cả và các mặt hàng theo yêu cầu.
- **Theo dõi trạng thái đơn hàng và tình trạng giao hàng:** Phân hệ cung cấp cái nhìn tổng quan về trạng thái hiện tại của từng đơn hàng, bao gồm quá trình chuẩn bị hàng trong kho, đóng gói, vận chuyển và xác nhận giao hàng, tích hợp với phân hệ Kho vận.
- **Tự động hóa việc tạo hóa đơn và liên kết với phân hệ Kế toán:** Dựa trên đơn hàng bán và tình trạng giao hàng, Odoo tự động tạo hóa đơn, đảm bảo tính chính xác và đồng bộ dữ liệu với phân hệ Kế toán, giúp giảm thiểu sai sót thủ công.
- **Báo cáo và phân tích bán hàng:** Cung cấp các báo cáo đa dạng về doanh thu, các sản phẩm bán chạy nhất, hiệu suất của đội ngũ bán hàng và phân tích xu hướng bán hàng theo thời gian, hỗ trợ ra quyết định kinh doanh.

3.2.1.2 Phân hệ Purchase (Mua hàng)

Phân hệ **Purchase (Mua hàng)** trong Odoo được thiết kế để quản lý toàn bộ quy trình mua sắm hàng hóa và dịch vụ của doanh nghiệp, từ khâu phát sinh yêu cầu mua cho đến khi hàng hóa được nhận vào kho và hóa đơn thanh toán được xử lý. Phân hệ này đóng vai trò quan trọng trong việc tối ưu hóa chi phí, nâng cao hiệu quả quản lý nhà cung cấp và đảm bảo sự tích hợp trực tiếp với các phân hệ **Kho (Inventory)** và **Kế toán (Accounting)**.

Các chức năng chính của phân hệ Purchase bao gồm:

- **Quản lý danh sách nhà cung cấp và điều khoản thanh toán:** Hệ thống cho phép duy trì một cơ sở dữ liệu toàn diện về các nhà cung cấp, bao gồm thông tin liên hệ, lịch sử giao dịch và các điều khoản thanh toán, chiết khấu đặc biệt.
- **Tạo và xác nhận yêu cầu báo giá (Request for Quotation – RFQ):** Người dùng có thể dễ dàng tạo và gửi các yêu cầu báo giá đến nhiều nhà cung cấp khác nhau để so sánh giá và điều kiện, đảm bảo lựa chọn được nhà cung cấp tối ưu.
- **Tự động chuyển đổi RFQ thành đơn mua hàng (Purchase Order):** Sau khi nhận được báo giá và lựa chọn được nhà cung cấp, hệ thống cho phép chuyển đổi trực tiếp yêu cầu báo giá thành đơn mua hàng chính thức, tiết kiệm thời gian và giảm thiểu sai sót.
- **Theo dõi quá trình nhận hàng và ghi nhận hóa đơn mua:** Phân hệ cung cấp khả năng theo dõi chặt chẽ quá trình giao nhận hàng từ nhà cung cấp, tích hợp với phân hệ Kho để cập nhật tồn kho tức thời. Sau khi hàng được nhận, hệ thống hỗ trợ ghi nhận hóa đơn mua hàng một cách nhanh chóng.
- **Tích hợp với phân hệ Kho để cập nhật tồn kho:** Mọi hoạt động nhập hàng thông qua phân hệ Mua hàng đều được tự động cập nhật vào phân hệ Kho, đảm bảo dữ liệu tồn kho luôn chính xác và hỗ trợ việc quản lý hàng hóa hiệu quả.

- **Phân tích hiệu suất nhà cung cấp và chi phí mua hàng:** Cung cấp các công cụ báo cáo và phân tích giúp đánh giá hiệu suất của từng nhà cung cấp (thời gian giao hàng, chất lượng sản phẩm) và tổng hợp chi phí mua hàng, từ đó hỗ trợ việc đàm phán và tối ưu hóa chi phí.

3.2.2 Yêu cầu và công cụ thử nghiệm

Để đánh giá hiệu quả của kiểm thử tự động trong môi trường triển khai thực tế, đề án tiến hành thực nghiệm trên hai phân hệ **Sales** và **Purchase** của hệ thống Odoo. Việc lựa chọn hai phân hệ này nhằm mục tiêu bao phủ toàn bộ chu trình đơn hàng – từ mua sắm đầu vào đến bán hàng đầu ra – vốn là quy trình nghiệp vụ cốt lõi trong hầu hết các tổ chức.

Yêu cầu thử nghiệm

Để đảm bảo chất lượng và độ ổn định của hệ thống Odoo, các yêu cầu thử nghiệm cần được đáp ứng bao gồm:

- **Tự động hóa kiểm thử giao diện người dùng (UI):**

  Các thao tác trên giao diện như nhập liệu, đăng nhập, thao tác với đơn hàng, điều hướng giữa các màn hình phải được kiểm thử tự động nhằm tăng hiệu quả, giảm sai sót và tiết kiệm thời gian so với kiểm thử thủ công. Tự động hóa cũng giúp thực hiện các kiểm thử lặp lại nhiều lần trong các vòng phát triển khác nhau.

- **Quản lý và tổ chức bộ test case linh hoạt:**

Cần một công cụ hỗ trợ phân nhóm, quản lý test case và chạy theo các cấu hình test suit khác nhau để dễ dàng thực hiện kiểm thử theo từng giai đoạn hoặc theo module của hệ thống.

- **Kiểm thử API và backend:**

Để xác minh tính đúng đắn của các chức năng nghiệp vụ liên quan đến dữ liệu và trạng thái hệ thống, cần kiểm thử các API nội bộ. Việc này giúp phát hiện lỗi từ tầng backend trước khi giao diện hiển thị cho người dùng cuối.

- **Dữ liệu thử nghiệm nhất quán và có kiểm soát:**

Việc sử dụng bộ dữ liệu thử nghiệm mẫu chuẩn bao gồm khách hàng, nhà cung cấp, sản phẩm, đơn hàng sẽ giúp đảm bảo tính nhất quán và tái sử dụng trong các lần kiểm thử khác nhau. Điều này làm giảm rủi ro do dữ liệu không hợp lệ hoặc không đồng nhất ảnh hưởng đến kết quả kiểm thử.

- **Báo cáo kết quả chi tiết và dễ hiểu:**

Cần có các báo cáo tổng hợp kết quả test, thông tin chi tiết về các test case đã chạy, lỗi phát sinh để nhanh chóng phân tích và xử lý.

Công cụ thử nghiệm được sử dụng

Để đáp ứng yêu cầu kỹ thuật và đảm bảo khả năng mở rộng trong tương lai, đề án sử dụng kết hợp các công cụ kiểm thử sau: [10]

Bảng 0-1: Công cụ thử nghiệm được sử dụng

Công cụ	Vai trò và ứng dụng
Selenium WebDriver	Tự động hóa các thao tác giao diện trên trình duyệt, đặc biệt hữu ích cho các bước nhập liệu, xác nhận đơn hàng và điều hướng giao diện người dùng trong Odoo
TestNG (Java)	Hỗ trợ quản lý và tổ chức bộ test case, cấu hình kiểm thử theo nhóm (suite), cung cấp báo cáo kết quả chi tiết
Postman (bổ trợ)	Được sử dụng để kiểm thử một số API nội bộ hoặc khi cần xác minh trạng thái hệ thống ở backend, chẳng hạn như xác thực việc tạo đơn hàng hoặc truy xuất dữ liệu sản phẩm

Dữ liệu thử nghiệm mẫu	Bao gồm các tập dữ liệu về khách hàng, nhà cung cấp, sản phẩm và đơn hàng để đảm bảo thử nghiệm lặp lại, nhất quán và dễ kiểm soát.

3.3 Triển khai thực nghiệm

Sau khi đánh giá tổng thể các đặc điểm của hệ thống cần kiểm thử – cụ thể là phân hệ Sales trong Odoo, với giao diện web động, có nhiều thao tác chuột, biểu mẫu phức tạp và luồng nghiệp vụ xuyên suốt – công cụ **Selenium** đã được lựa chọn để triển khai kiểm thử tự động. [5]

Selenium được ưu tiên do:

- Hỗ trợ kiểm thử tương tác người dùng trực tiếp trên trình duyệt;
- Tương thích tốt với hệ thống Odoo;
- Hỗ trợ nhiều ngôn ngữ lập trình (Java, Python, C#, v.v.);
- Tích hợp dễ dàng vào quy trình CI/CD (thông qua Jenkins, GitLab CI...).

Bên cạnh việc lựa chọn công cụ, mô hình Page Object Model (POM) cũng được áp dụng trong quá trình thiết kế bộ kiểm thử tự động. POM là mô hình tổ chức mã kiểm thử theo hướng tách biệt giữa phần thao tác giao diện người dùng và phần logic kiểm thử, nhờ đó giúp:

- **Tăng khả năng bảo trì**: thay đổi giao diện chỉ cần cập nhật ở một nơi (trong lớp Page);
- **Tái sử dụng cao**: nhiều test case có thể dùng chung hàm thao tác;
- **Quản lý rõ ràng**: dễ tách, quản lý, mở rộng khi có nhiều chức năng kiểm thử khác nhau.
- **Tăng khả năng bảo trì**: thay đổi giao diện chỉ cần cập nhật ở một nơi (trong lớp Page);

Kiến trúc kiểm thử áp dụng:

- **Page Classes**: định nghĩa các locator và hành động trên từng trang cụ thể (ví dụ: LoginPage.java, CreateOrderPage.java);

- **Test Classes**: thực hiện kiểm thử theo luồng nghiệp vụ bằng cách gọi lại các Page đã định nghĩa;
- **Base & Utility Classes**: xử lý khởi tạo WebDriver, thiết lập báo cáo, log, chờ đợi;
- **Test Framework**: sử dụng TestNG để tổ chức test suite, báo cáo kết quả và thực hiện assertion.

Việc lựa chọn Selenium kết hợp mô hình POM là giải pháp phù hợp với thực trạng dự án tại A1 Consulting: cần kiểm thử thường xuyên, khối lượng lớn, giao diện có thể thay đổi nhưng luồng nghiệp vụ tương đối ổn định. Cách tiếp cận này cũng tạo nền tảng tốt để tích hợp về sau vào quy trình DevOps hiện có tại công ty. [14]

3.3.1 Xây dựng kịch bản kiểm thử thủ công các tính năng ứng dụng kiểm thử tự động

Trước khi phát triển hệ thống kiểm thử tự động, cần thực hiện tiến hành xây dựng đầy đủ các **kịch bản kiểm thử thủ công** để xác định luồng nghiệp vụ, kiểm tra độ ổn định của hệ thống, và đảm bảo rằng các bước kiểm thử có thể chuyển hóa chính xác thành test script tự động. [8]

Quy trình xây dựng kịch bản kiểm thử thủ công bao gồm:

- **Phân tích yêu cầu nghiệp vụ** từ đặc tả phần mềm (SRS) và tài liệu quy trình nội bộ của hệ thống Odoo. Việc phân tích giúp liệt kê toàn bộ các chức năng chính và phụ, từ đó xác định các luồng nghiệp vụ cần kiểm thử.
- **Xác định các luồng kiểm thử chính**: Sau khi đã hoàn tất bước phân tích yêu cầu nghiệp vụ, bước tiếp theo trong quá trình xây dựng kịch bản kiểm thử thủ công là xác định các luồng kiểm thử chính cho từng chức năng. Việc xác định luồng kiểm thử giúp đảm bảo hệ thống được kiểm tra một cách toàn diện ở cả tình huống lý tưởng lẫn các tình huống ngoại lệ, từ đó giảm thiểu rủi ro và tăng độ tin cậy của phần mềm.

Phân loại luồng kiểm thử

- **Luồng chuẩn (Happy Path)**: Là luồng nghiệp vụ chính, trong đó người dùng thao tác đúng theo quy trình được định nghĩa và hệ thống phản hồi đúng như mong đợi. Đây là cơ sở để kiểm tra tính đúng đắn cơ bản của hệ thống.
- **Luồng ngoại lệ (Exception Path)**: Là những tình huống sai lệch có thể xảy ra trong quá trình sử dụng hệ thống, như thiếu thông tin bắt buộc, dữ liệu không hợp lệ, thao tác sai thứ tự... Việc kiểm tra luồng ngoại lệ giúp đảm bảo hệ thống có khả năng xử lý lỗi một cách hợp lý và an toàn.

Để đảm bảo xây dựng bộ test case có độ bao phủ đầy đủ nhưng vẫn tối ưu về số lượng và chi phí kiểm thử, cần kết hợp các kỹ thuật thiết kế kiểm thử đã được chuẩn hóa trong thực tiễn. Các kỹ thuật này không chỉ giúp xác định được đầy đủ các luồng kiểm thử cần thiết (bao gồm cả luồng chuẩn và luồng ngoại lệ), mà còn góp phần phát hiện các lỗi tiềm ẩn có thể không được nêu rõ trong tài liệu yêu cầu. Các kỹ thuật được áp dụng gồm: Phân vùng tương đương, Phân tích giá trị biên, Đoán lỗi. Việc kết hợp linh hoạt giữa các kỹ thuật trong quá trình xây dựng test case giúp tăng cường tính bao phủ của kiểm thử mà vẫn đảm bảo hiệu quả về chi phí và thời gian. Đây là những kỹ thuật thiết yếu, đặc biệt quan trọng trong bối cảnh kiểm thử thủ công đóng vai trò nền tảng để phát triển hệ thống kiểm thử tự động sau này.

- **Thiết lập dữ liệu kiểm thử**: gồm khách hàng mẫu, sản phẩm, mức giá, phương thức thanh toán, trạng thái đơn hàng.
- **Viết test case chi tiết**: mỗi bước kiểm thử bao gồm mục tiêu, điều kiện đầu vào, thao tác thực hiện và kết quả mong đợi.

Các chức năng chính được đưa vào kiểm thử gồm:

- Đăng nhập hệ thống
- Tạo mới yêu cầu bán hàng
- Xác nhận yêu cầu bán hàng
- Tìm kiếm đơn hàng
- Xác nhận phiếu Xuất kho
- Tạo yêu cầu mua hàng

- Xác nhận đơn mua hàng
- Xác nhận phiếu Nhập kho
- Quy trình Trả hàng

Kịch bản kiểm thử thủ công không chỉ là cơ sở để phát triển test script tự động, mà còn là tài liệu tham chiếu để đánh giá độ chính xác của hệ thống kiểm thử sau khi được tự động hóa.

Kịch bản FE

Bảng 0-2: Kịch bản kiểm thử FE

Mã kịch bản	Tên kịch bản	Các bước thực hiện	Kết quả mong đợi
TC01	Đăng nhập bỏ trống Tên đăng nhập và Mật khẩu	1. Truy cập vào hệ thống 2. Để trống cả hai trường "Email" và "Mật khẩu" 3. Nhấn nút "Đăng nhập"	Hệ thống hiển thị thông báo lỗi yêu cầu điền thông tin.
TC02	Đăng nhập khi nhập sai Tên đăng nhập	1. Truy cập vào hệ thống 2. Nhập "Email" sai và "Mật khẩu" đúng 3. Nhấn nút "Đăng nhập"	Hiển thị thông báo lỗi
TC03	Đăng nhập khi nhập sai Mật khẩu	1. Truy cập vào hệ thống 2. Nhập "Email" đúng và "Mật khẩu" sai 3. Nhấn nút "Đăng nhập"	Hiển thị thông báo lỗi
TC04	Đăng nhập thành công	1. Truy cập vào hệ thống 2. Nhập "Email và "Mật khẩu" hợp lệ 3. Nhấn nút "Đăng nhập"	Đăng nhập thành công

TC05	Tạo yêu cầu mua hàng thành công	1. Truy cập vào phân hệ Mua hàng 2. Nhấn nút Mới 3. Nhập thông tin hợp lệ: Chọn Nhà cung cấp, chọn sản phẩm, nhập SL 4. Nhấn nút Lưu	Tạo đơn mua hàng thành công Tự động sinh mã đơn hàng
TC06	Tìm kiếm yêu cầu mua hàng thành công	1. Vào màn hình danh sách yêu cầu mua hàng 2. Nhập mã đơn hàng hợp lệ 3. Nhấn Tìm kiếm	Hiển thị đơn hàng tương ứng
TC07	Tìm kiếm yêu cầu mua hàng với mã không hợp lệ	1. Vào màn hình danh sách yêu cầu mua hàng 2. Nhập mã đơn hàng không hợp lệ 3. Nhấn Tìm kiếm	Hiển thị thông báo không có kết quả tìm kiếm
TC08	Xác nhận yêu cầu mua hàng thành công	1. Vào màn hình danh sách yêu cầu mua hàng 2. Chọn đơn hàng cần xác nhận 3. Nhấn nút Xác nhận	Đơn chuyển trạng thái “Đơn bán hàng”, tạo phiếu nhập kho
TC09	Xác nhận phiếu Nhập kho có tách kiện	1. Mở phiếu nhập kho 2. Nhập SL hoàn tất < Nhu cầu 3. Nhấn nút Xác nhận 4. Chọn tách kiện	Phiếu kho cập nhật sang trạng thái Hoàn tất Đồng thời sinh phiếu kho mới với SL còn lại

TC10	Xác nhận phiếu Nhập kho không tách kiện	1. Mở phiếu nhập kho 2. Nhập SL hoàn tất < Nhu cầu 3. Nhấn nút Xác nhận 4. Chọn Không tách kiện	Phiếu kho cập nhật sang trạng thái Hoàn tất Không sinh phiếu kho mới với SL còn lại
TC11	Xác nhận phiếu Nhập kho khi SL hoàn tất = SL yêu cầu	1. Mở phiếu nhập kho 2. Nhập SL hoàn tất = Nhu cầu 3. Nhấn nút Xác nhận	Phiếu kho cập nhật sang trạng thái Hoàn tất
TC12	Tạo yêu cầu bán hàng thành công	1. Truy cập vào phân hệ Bán hàng 2. Nhấn nút Mới 3. Nhập thông tin hợp lệ: Chọn Khách hàng, chọn sản phẩm, nhập SL 4. Nhấn nút Lưu	Tạo đơn bán hàng thành công Tự động sinh mã đơn hàng
TC13	Tìm kiếm yêu cầu bán hàng thành công	1. Vào màn hình danh sách yêu cầu bán hàng 2. Nhập mã đơn hàng hợp lệ 3. Nhấn Tìm kiếm	Hiển thị đơn hàng tương ứng
TC14	Tìm kiếm yêu cầu bán hàng với mã không hợp lệ	1. Vào màn hình danh sách yêu cầu bán hàng 2. Nhập mã đơn hàng không hợp lệ 3. Nhấn Tìm kiếm	Hiển thị thông báo không có kết quả tìm kiếm

TC15	Xác nhận yêu cầu bán hàng thành công	1. Vào màn hình danh sách yêu cầu bán hàng 2. Chọn đơn hàng cần xác nhận 3. Nhấn nút Xác nhận	Đơn chuyển trạng thái "Đơn bán hàng", tạo phiếu nhập kho
TC16	Xác nhận phiếu xuất kho có tách kiện	1. Mở phiếu xuất kho 2. Nhập SL hoàn tất < Nhu cầu 3. Nhấn nút Xác nhận 4. Chọn tách kiện	Phiếu kho cập nhật sang trạng thái Hoàn tất Đồng thời sinh phiếu kho mới với SL còn lại
TC17	Xác nhận phiếu xuất kho không tách kiện	1. Mở phiếu xuất kho 2. Nhập SL hoàn tất < Nhu cầu 3. Nhấn nút Xác nhận 4. Chọn Không tách kiện	Phiếu kho cập nhật sang trạng thái Hoàn tất Không sinh phiếu kho mới với SL còn lại
TC18	Tạo đơn trả hàng	1. Truy cập đơn giao hàng đã hoàn tất 2. Nhấn nút Trả hàng 3. Nhập số lượng cho từng sản phẩm 4. Nhấn nút Xác nhận	Tạo đơn trả hàng thành công.

Kịch bản BE

Bảng 0-3: Kịch bản kiểm thử BE

Mã kịch bản	Tên kịch bản	Các bước thực hiện	Kết quả mong đợi

TC01	Đăng nhập bỏ trống Tên đăng nhập và Mật khẩu	1. Truyền vào
		{
		"params": {
		"login": "",
		"password": ""
		}
		}
		2. Call API:
		http://localhost:8069/web/login
			Lỗi xác thực "Missing login or password".

TC02	Đăng nhập khi nhập sai Tên đăng nhập	1. Truyền vào
		{
		"params": {
		"login": "admin",
		"password": "admin1123"
		}
		}
		2. Call API:
		http://localhost:8069/web/login
			Hiển thị thông báo lỗi: "Wrong login/password".

TC03	Đăng nhập khi nhập sai Mật khẩu	1. Truyền vào
		{
		"params": {
		"login": "admin123 ",
		"password": "admin"
		}
		}
			Hiển thị thông báo lỗi: "Wrong login/password".

		2. Call API: http://localhost:8069/web/login	

TC04	Đăng nhập thành công	1. Truyền vào
		{
		"params": {
		"login": "admin",
		"password": "admin"
		}
		}
		2. Call API:
		http://localhost:8069/web/login
			Đăng nhập thành công: Trả về session_id, user_id, thành công.

TC05	Tạo yêu cầu mua hàng thành công	1. Truyền vào
		{
		"partner_id": 5,
		"order_line": [
		{
		"product_id": 12,
		"product_qty": 10,
		"price_unit": 15000
		}
		]
		}2. Call API:
		http://localhost:8069/purchase/order/create
			- Tạo mới yêu cầu mua hàng thành công
			- Response trả về ID đơn hàng

TC06	Tạo yêu cầu bán hàng thành công	1. Truyền vào { "customers_id": 5,	- Tạo mới yêu cầu bán hàng thành công

		"order_line": [
		{
		"product_id": 12,
		"product_qty": 10,
		"price_unit": 15000
		}
		]
		}2. Call API:
		http://localhost:8069/purchase/order/create
			- Response trả về ID đơn hàng

TC07	Tìm kiếm yêu cầu mua hàng thành công	1. Truyền vào { "name": "PO0001" } 2. Call API: http://localhost:8069/purchase/order/search	Response hiển thị chính xác kết quả tìm kiếm theo Mã đơn hàng
TC08	Tìm kiếm yêu cầu bán hàng thành công	1. Truyền vào { "name": "SO0001" } 2. Call API: http://localhost:8069/purchase/order/search	Response hiển thị chính xác kết quả tìm kiếm theo Mã đơn hàng
TC09	Tìm kiếm yêu cầu mua hàng khi nhập Mã đơn hàng không hợp lệ	1. Truyền vào { "name": "PO000" } 2. Call API: http://localhost:8069/purchase/order/search	Response trả về mảng trống ([])
TC10	Xác nhận yêu cầu mua hàng thành công	1. Truyền vào: { "order_id": 102 } 2. Call API: /sale/order/confirm.	- Hệ thống cập nhật yêu cầu mua hàng sang trạng thái Đơn bán hàng

			- Sinh ra phiếu Nhập kho tương ứng.

3.3.2 Xây dựng kịch bản kiểm thử tự động theo các kịch bản thủ công đã xây dựng.

Sau khi hoàn thiện hệ thống kịch bản kiểm thử thủ công cho từng chức năng, **kịch bản kiểm thử tự động** được xây dựng dựa trên các test case đã xác thực. Quá trình chuyển đổi từ kiểm thử thủ công sang kiểm thử tự động đảm bảo rằng:

- Các bước nghiệp vụ được mô phỏng đầy đủ và chính xác;
- Logic xử lý và dữ liệu đầu vào giống hệt kịch bản thủ công;
- Đầu ra (kết quả mong đợi) được kiểm tra tự động thông qua điều kiện assert.

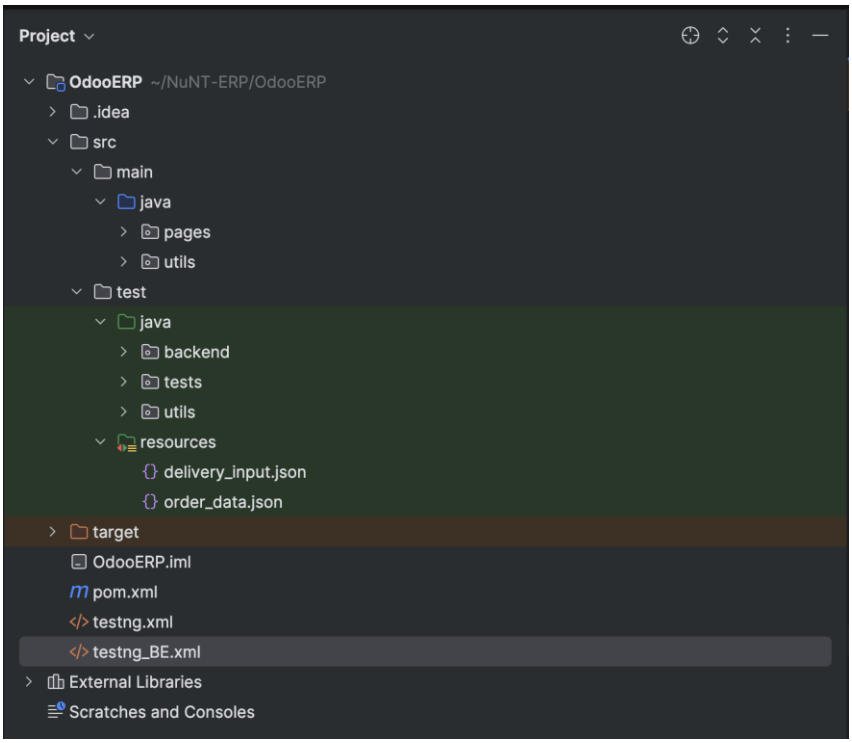
Các kịch bản được chia thành hai nhóm chính:

- **Kiểm thử frontend (UI)**: Các thao tác thông qua trình duyệt, thực hiện bởi người dùng cuối, bao gồm: đăng nhập, tạo đơn hàng, xác nhận phiếu kho, tìm kiếm báo giá...
- **Kiểm thử backend (API)**: Các luồng xử lý hệ thống, không qua giao diện người dùng, bao gồm: login API, tạo đơn hàng qua API, xác nhận đơn, xử lý phiếu kho, trả hàng...

Bước 1: Khởi tạo cấu trúc dự án kiểm thử tự động

Hệ thống kiểm thử chức năng tự động trong đề án được xây dựng dựa trên mô hình tổ chức chuẩn Maven. Việc chia tách rõ ràng giữa phần kiểm thử giao diện (frontend) và phần kiểm thử nghiệp vụ (backend) giúp đảm bảo mã nguồn kiểm thử dễ đọc, dễ mở rộng. Cấu trúc dự án được chia thành các phần rõ ràng, mỗi phần đảm nhận một vai trò cụ thể trong toàn bộ luồng kiểm thử.

Việc tổ chức cấu trúc kiểm thử theo chuẩn Maven kết hợp với phân tầng rõ ràng giữa kiểm thử giao diện (FE) và kiểm thử API (BE) là yếu tố then chốt giúp thực hiện kiểm soát tốt mã nguồn kiểm thử. Nhờ đó, các test case được phát triển nhanh chóng, dễ quản lý và có thể dễ dàng mở rộng, tích hợp với hệ thống kiểm thử liên tục về sau.[10]

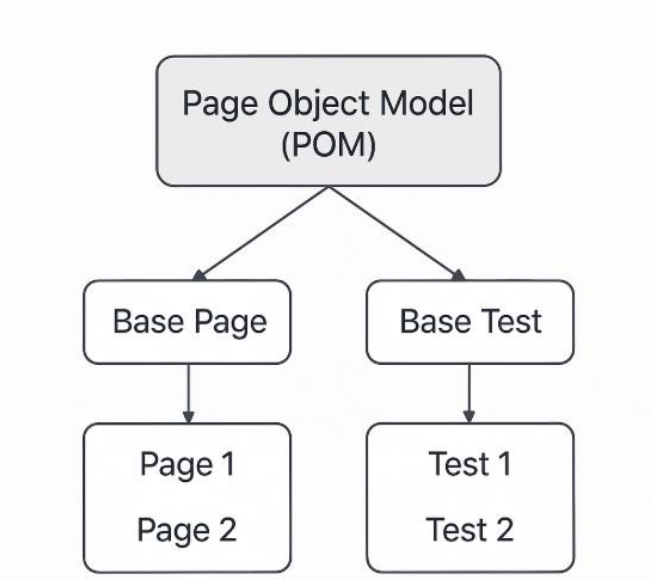


Hình 0.1: Cấu trúc thư mục dự án Odoo ERP

Bước 2: Xây dựng lớp Page Object và lớp Test Class

Trong kiểm thử giao diện người dùng (frontend), **Page Object Model (POM)** – là một trong những mô hình chuẩn kiến trúc được khuyến nghị trong kiểm thử tự động với Selenium. Mỗi **màn hình (giao diện)** sẽ được ánh xạ thành một **lớp (class)** riêng, nơi tập trung định nghĩa:

- Các phần tử giao diện (locator),
- Các hành động tương tác (hàm click, nhập dữ liệu, lấy kết quả...).

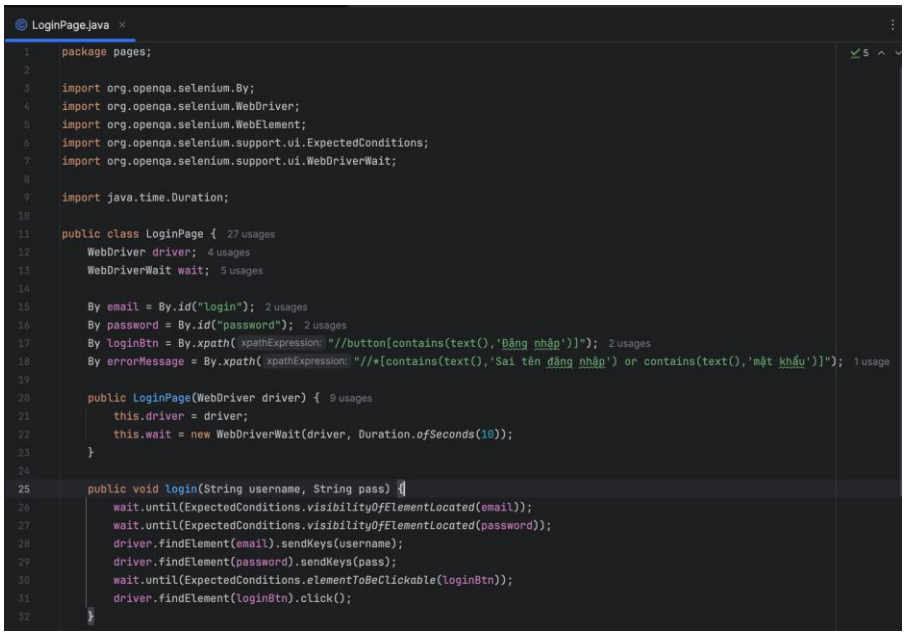


Hình 0.2: Mô hình các thành phần trong dự án kiểm thử tự động.

Sơ đồ trên minh họa mô hình tổ chức kiểm thử giao diện người dùng theo kiến trúc Page Object Model (POM). Việc áp dụng mô hình này giúp phân tách rõ ràng giữa phần xử lý giao diện (Page) và phần kiểm thử (Test), từ đó nâng cao tính tái sử dụng mã nguồn và đơn giản hóa quy trình bảo trì. [14]

Khi có thay đổi trong giao diện người dùng, chỉ cần cập nhật lớp tương ứng trong phần Page mà không cần điều chỉnh các lớp kiểm thử. Cách tổ chức này giúp giảm thiểu rủi ro phát sinh lỗi khi sửa đổi hệ thống và tiết kiệm đáng kể thời gian bảo trì.

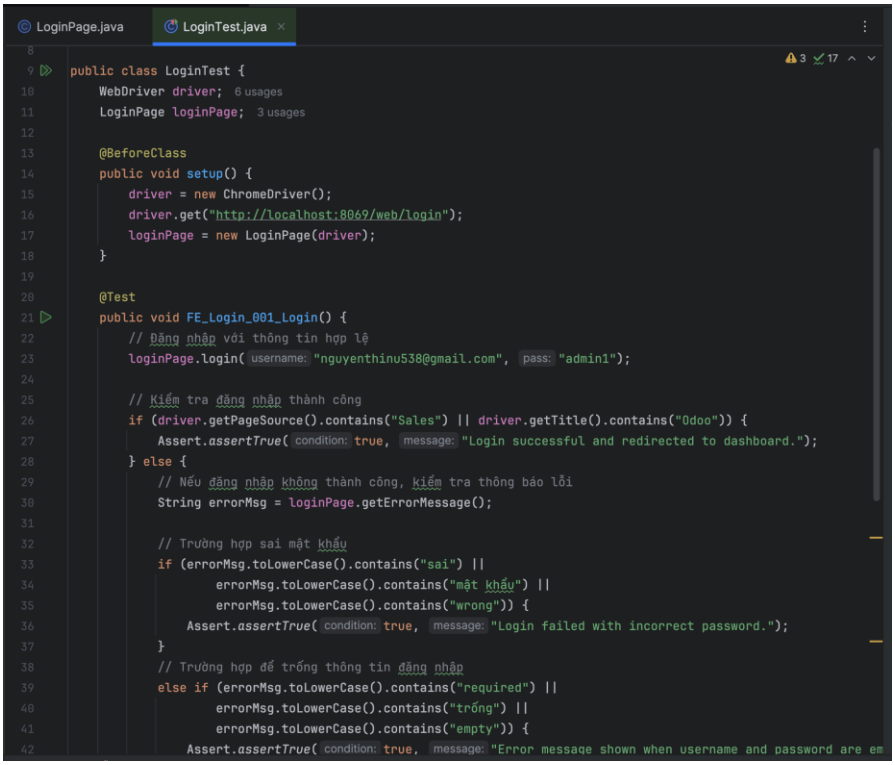
Bên cạnh đó, việc mở rộng phạm vi kiểm thử cũng trở nên dễ dàng hơn. Cụ thể, chỉ cần xây dựng thêm một lớp Page kế thừa từ lớp cơ sở và định nghĩa các phần tử cùng hành vi tương ứng. Sau đó, triển khai các lớp kiểm thử tương ứng kế thừa từ lớp kiểm thử cơ sở. Cách tiếp cận này mang lại sự linh hoạt, đồng thời đảm bảo tính hệ thống và khả năng mở rộng cho bộ kiểm thử.



```java
package pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

public class LoginPage {
    WebDriver driver;
    WebDriverWait wait;

    By email = By.id("login");
    By password = By.id("password");
    By loginBtn = By.xpath("//button[contains(text(),'Đăng nhập')]");
    By errorMessage = By.xpath("//*[contains(text(),'Sai tên đăng nhập') or contains(text(),'mật khẩu')]");

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        this.wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    }

    public void login(String username, String pass) {
        wait.until(ExpectedConditions.visibilityOfElementLocated(email));
        wait.until(ExpectedConditions.visibilityOfElementLocated(password));
        driver.findElement(email).sendKeys(username);
        driver.findElement(password).sendKeys(pass);
        wait.until(ExpectedConditions.elementToBeClickable(loginBtn));
        driver.findElement(loginBtn).click();
    }
```

Hình 0.3: Kiểm thử chức năng đăng nhập trong lớp LoginPage

Từng luồng nghiệp vụ sẽ được ánh xạ thành **một lớp kiểm thử (Test Class)** riêng biệt. Trong lớp này sẽ gọi các hành động từ Page Object để kiểm tra đầy đủ luồng giao diện.



```java
public class LoginTest {
    WebDriver driver;
    LoginPage loginPage;

    @BeforeClass
    public void setup() {
        driver = new ChromeDriver();
        driver.get("http://localhost:8069/web/login");
        loginPage = new LoginPage(driver);
    }

    @Test
    public void FE_Login_001_Login() {
        // Đăng nhập với thông tin hợp lệ
        loginPage.login(username: "nguyenthinu538@gmail.com", pass: "admin1");

        // Kiểm tra đăng nhập thành công
        if (driver.getPageSource().contains("Sales") || driver.getTitle().contains("Odoo")) {
            Assert.assertTrue(condition: true, message: "Login successful and redirected to dashboard.");
        } else {
            // Nếu đăng nhập không thành công, kiểm tra thông báo lỗi
            String errorMsg = loginPage.getErrorMessage();

            // Trường hợp sai mật khẩu
            if (errorMsg.toLowerCase().contains("sai") ||
                    errorMsg.toLowerCase().contains("mật khẩu") ||
                    errorMsg.toLowerCase().contains("wrong")) {
                Assert.assertTrue(condition: true, message: "Login failed with incorrect password.");
            }
            // Trường hợp để trống thông tin đăng nhập
            else if (errorMsg.toLowerCase().contains("required") ||
                    errorMsg.toLowerCase().contains("trống") ||
                    errorMsg.toLowerCase().contains("empty")) {
                Assert.assertTrue(condition: true, message: "Error message shown when username and password are em
```

Hình 0.4: Kiểm thử chức năng đăng nhập trong lớp LoginTest

Trong mô hình Page Object Model, mỗi lớp kiểm thử (Test Class) không thao tác trực tiếp với các phần tử giao diện (locator). Thay vào đó, tất cả các thao tác như nhập dữ liệu, click nút, lấy thông báo lỗi đều được gọi thông qua các hàm xử lý đã được định nghĩa trong Page Object.

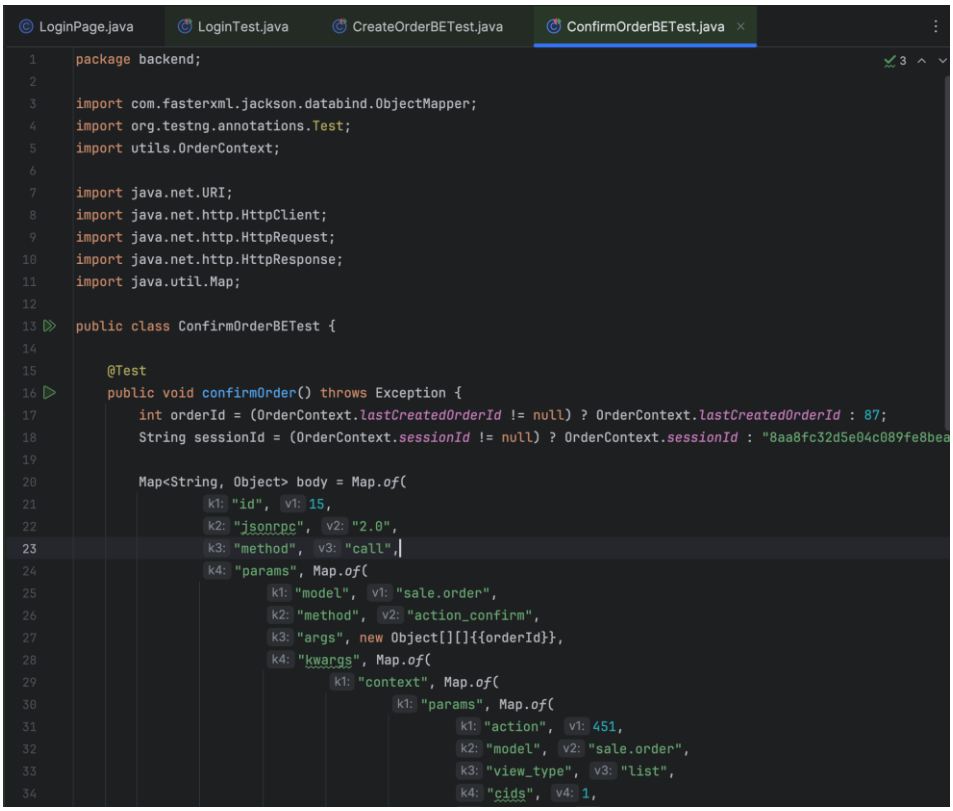
Cách tiếp cận này giúp tách biệt rõ ràng giữa logic giao diện (giao tiếp với hệ thống) và logic kiểm thử (kiểm tra nghiệp vụ). Nhờ đó, khi giao diện thay đổi, chỉ cần sửa tại một nơi (Page Object), còn toàn bộ mã kiểm thử vẫn giữ nguyên.

Bước 3: Xây dựng kiểm thử backend (API)

Bên cạnh kiểm thử giao diện người dùng (frontend), hệ thống còn được kiểm thử ở tầng nghiệp vụ xử lý phía máy chủ (backend). Mục tiêu của việc kiểm thử backend là đảm bảo các chức năng lõi như tạo đơn, xác nhận đơn hàng, xử lý kho, trả hàng... vận hành đúng logic, dữ liệu phản hồi chính xác, và không có lỗi xảy ra ở tầng sâu của hệ thống, ngay cả khi không có giao diện người dùng.

Trong phạm vi đề án này, kiểm thử backend được thực hiện thông qua việc gửi trực tiếp các yêu cầu HTTP (GET, POST) đến các API nội bộ của hệ thống Odoo, xử lý phản hồi, và đối chiếu với kết quả mong đợi bằng các lệnh kiểm tra tự động (assert).

Tất cả các lớp kiểm thử backend được đặt trong thư mục src/test/java/backend/, mỗi lớp đại diện cho một nghiệp vụ chính, được tách biệt để dễ bảo trì. Mỗi lớp kiểm thử gồm nhiều phương thức @Test, mỗi phương thức là một tình huống kiểm thử (test case) cụ thể, được ánh xạ trực tiếp từ danh sách kiểm thử thủ công.



Hình 0.5: Kiểm thử BE trong lớp Test

3.3.4 Thực thi kiểm thử

Sau khi hoàn thiện các lớp kiểm thử chức năng tự động cho cả giao diện người dùng (frontend - FE) và xử lý phía máy chủ (backend - BE), hệ thống kiểm thử được tổ chức thực thi theo từng khối chức năng tương ứng. Quá trình thực thi được cấu hình thông qua tệp testng.xml, cho phép kiểm thử được triển khai linh hoạt theo từng lớp kiểm thử, từng chức năng nghiệp vụ riêng lẻ, hoặc toàn bộ hệ thống. Sau khi thực thi, hệ thống tự động tạo báo cáo ghi nhận kết quả kiểm thử.

Đề án sử dụng framework **TestNG** để điều phối quá trình kiểm thử. Đây là công cụ hỗ trợ tổ chức và quản lý test case một cách hiệu quả, cho phép thiết lập các kịch bản chạy theo từng đơn vị kiểm thử (<test>), theo từng lớp (<class>) hoặc theo trình tự nghiệp vụ được xác định trước.

Các lớp kiểm thử được phân chia rõ ràng theo mục đích kiểm thử, bao gồm:

- **Frontend (FE):** kiểm thử các thao tác của người dùng thông qua trình duyệt với sự hỗ trợ của Selenium. [6]

- **Backend (BE):** kiểm thử logic nghiệp vụ bằng cách gửi trực tiếp các yêu cầu HTTP (API) tới hệ thống Odoo.

Tất cả các lớp kiểm thử đều được khai báo và quản lý tập trung trong tệp testng.xml, đóng vai trò như tệp điều phối chính của toàn bộ quy trình kiểm thử tự động.

3.4 Kết quả kiểm thử

Sau khi hoàn thiện 28 kịch bản kiểm thử tự động cho các chức năng trọng yếu như đăng nhập, xử lý đơn hàng, kiểm tra kho và xác nhận dữ liệu, quá trình thực nghiệm được triển khai trong **8 tuần** với tần suất thực thi **2 lần/tuần**, tổng cộng **16 đợt chạy kiểm thử** toàn bộ hệ thống.

Tổng kết kết quả kiểm thử tự động như sau:

- **Tổng số lượt kiểm thử thực hiện:** 448 lượt (28 test case × 16 lần).
- **Tổng số lỗi được phát hiện:** 12 lỗi, trong đó:
    - **4 lỗi liên quan đến giao diện người dùng (UI)**: bao gồm mất nút, giao diện hiển thị lệch, không phản hồi đúng khi thực hiện thao tác.
    - **5 lỗi liên quan đến xử lý logic hệ thống**: bao gồm điều kiện sai, không kiểm tra dữ liệu trống, xử lý đầu vào không hợp lệ gây treo hệ thống tạm thời.
    - **3 lỗi thuộc nhóm điều hướng và thông báo sai**: không chuyển trang sau thao tác, hiển thị thông báo không chính xác hoặc không có thông báo khi thao tác lỗi.
- **Tỷ lệ thành công sau lần chạy thứ 5:** đạt **100%**, cho thấy hệ thống đã ổn định sau khi phát hiện và khắc phục các lỗi ban đầu.
- **Thời gian thực thi trung bình mỗi lượt:** ~7 phút.
- **Thời gian kiểm thử thủ công tương đương:** khoảng 65–70 phút/lượt

→ Như vậy, hệ thống giúp **tiết kiệm trung bình 58–63 phút mỗi lượt**, tương đương **928–1.008 phút (~15–17 giờ)** trong toàn bộ quá trình thử nghiệm

Các test case được thực hiện bằng công cụ Selenium WebDriver kết hợp với TestNG, tổ chức mã nguồn theo mô hình POM để tối ưu khả năng bảo trì và tái sử dụng.

3.5 Đánh giá kết quả

Trước khi có kiểm thử tự động, mỗi lần cập nhật hệ thống (upcode) đều yêu cầu đội kiểm thử chạy lại toàn bộ các kịch bản chức năng để đảm bảo không có lỗi phát sinh ngoài ý muốn (regression). Trung bình, mỗi lần như vậy cần:

- **6 giờ làm việc** để kiểm thử toàn bộ bằng tay (3 giờ trên môi trường staging, 3 giờ trên production).
- Với tần suất 2 lần cập nhật/tuần → 12 giờ/tuần, tức **96 giờ trong 8 tuần** chỉ riêng cho hồi quy.

Sau khi áp dụng kiểm thử tự động:

- Thời gian hồi quy giảm còn **2 giờ/lần**, gồm 1 giờ chạy test tự động và 1 giờ test thủ công bổ sung.
- Tổng thời gian còn lại là **32 giờ/8 tuần**.

→ **Giảm được 64 giờ (66.7%)** thời gian kiểm thử hồi quy, từ đó tăng tốc độ phát hành và giảm áp lực cho đội kiểm thử**.**

Quá trình xây dựng và bảo trì testcase

Về xây dựng testcase:

Việc xây dựng test case ban đầu đòi hỏi thời gian và nguồn lực lớn do đội kiểm thử phải làm quen công cụ (Selenium + TestNG) và mô hình Page Object Model (POM). Tuy nhiên, hiệu suất được cải thiện đáng kể sau mỗi tuần:

- **Tuần 1**: xây dựng được 13 test case đơn giản (chủ yếu là đăng nhập và điều hướng).
- **Tuần 2 và 3**: hoàn thiện toàn bộ 28 test case ban đầu theo kế hoạch.
- **Thời gian trung bình xây dựng 1 test case**: từ 1.5–2 giờ lúc đầu → giảm còn ~1 giờ khi quen quy trình.

Về bảo trì testcase:

- **Giai đoạn đầu**: mất trung bình 6 giờ để điều chỉnh lại 20 test case sau khi hệ thống thay đổi.
- **Giai đoạn tuần 8**: toàn bộ 28 test case chỉ cần 4 giờ bảo trì, nhờ tổ chức code hợp lý, tái sử dụng tốt theo mô hình POM.

→ **Thời gian bảo trì giảm hơn 33%** so với giai đoạn đầu, đảm bảo tính bền vững khi mở rộng quy mô.

Bảng 0-4: Đánh giá kết quả

Tiêu chí	Trước khi áp dụng kiểm thử tự động	Sau khi áp dụng kiểm thử tự động	Mức cải thiện
Thời gian kiểm thử hồi quy mỗi lần	6 giờ (3h staging + 3h production)	2 giờ (1h tự động + 1h thủ công)	Giảm 66.7% (4 giờ/lần)
Tần suất kiểm thử hồi quy	2 lần/tuần	2 lần/tuần	Không đổi
Tổng thời gian kiểm thử hồi quy 8 tuần	96 giờ	32 giờ	Giảm 64 giờ
Số test case xây dựng tuần 1	13 test case đơn giản	-	-
Tổng số test case hoàn thiện tuần 3	0	28 test case	Hoàn thiện toàn bộ
Thời gian xây dựng trung bình/test case	1.5–2 giờ	~1 giờ	Tiết kiệm 0.5–1 giờ/test case
Thời gian bảo trì ban đầu (20 test case)	6 giờ	-	-
Thời gian bảo trì tuần 8 (28 test case)	-	4 giờ	Giảm >33%

3.6 Kết luận chương 3

Trong chương này, đề án trình bày quá trình ứng dụng kiểm thử tự động vào hệ thống Odoo ERP trong bối cảnh chuyển đổi số, với mục tiêu đánh giá hiệu quả và rút ra bài học thực tế. Kết quả thực nghiệm cho thấy kiểm thử tự động giúp phát hiện

lỗi sớm**,** tăng độ ổn định hệ thống, và tiết kiệm thời gian so với phương pháp thủ công.

KẾT LUẬN

Kết quả đạt được của đề án

Đề án **"Nghiên cứu kiểm thử tự động phần mềm và ứng dụng tại công ty A1 Consulting"** đã bước đầu tiếp cận và triển khai kiểm thử tự động trong môi trường doanh nghiệp, góp phần nâng cao hiệu quả kiểm thử, rút ngắn thời gian và hỗ trợ kiểm soát chất lượng phần mềm tốt hơn. Cụ thể đề án đã đạt được các kết quả sau:

- **Tiết kiệm thời gian kiểm thử hồi quy:** Thời gian kiểm thử hồi quy giảm từ 6 giờ mỗi lần xuống còn 2 giờ, tương đương giảm 66.7%, tiết kiệm 64 giờ trong 8 tuần. Điều này giúp tăng tốc độ phát hành sản phẩm và giảm tải công việc cho đội ngũ kiểm thử.
- **Hiệu quả xây dựng và bảo trì test case:** Sử dụng mô hình Page Object Model phối hợp với Selenium và TestNG giúp tối ưu hóa việc tổ chức và tái sử dụng mã nguồn test case, giảm thời gian bảo trì hơn 33% sau 8 tuần vận hành, đồng thời nâng cao độ ổn định và khả năng mở rộng của bộ test.
- **Phát hiện lỗi sớm, nâng cao chất lượng:** Kiểm thử tự động hỗ trợ phát hiện lỗi sớm trong quá trình phát triển, giảm thiểu lỗi phát sinh và cải thiện độ ổn định của hệ thống.
- **Giảm thiểu công sức kiểm thử thủ công:** Giảm đáng kể khối lượng công việc thủ công, giúp đội kiểm thử tập trung vào các kịch bản phức tạp và sáng tạo hơn.
- **Đánh giá tính khả thi của công cụ và mô hình:** Mô hình Page Object Model và công cụ Selenium được chứng minh phù hợp và hiệu quả khi áp dụng trong môi trường thực tế doanh nghiệp.

Hướng phát triển tiếp theo

Để tiếp tục nâng cao hiệu quả và mở rộng phạm vi áp dụng của hệ thống kiểm thử tự động, đề án đề xuất một số hướng phát triển trong tương lai như sau:

- **Mở rộng phạm vi kiểm thử**: Phát triển thêm các test case để bao phủ toàn diện hơn các chức năng, bao gồm các tình huống đặc biệt (edge cases), dữ liệu biên (boundary values), cũng như các bài kiểm thử hiệu năng và bảo mật.
- **Tích hợp quy trình kiểm thử vào hệ thống CI/CD**: Thiết lập cơ chế kích hoạt kiểm thử tự động ngay khi có thay đổi trong mã nguồn (continuous testing), đảm bảo phản hồi sớm và liên tục trong quá trình phát triển phần mềm.
- **Tối ưu hóa thời gian thực thi**: Nghiên cứu áp dụng kiểm thử song song (parallel testing) và đa trình duyệt (cross-browser testing) nhằm rút ngắn thời gian thực hiện và đảm bảo khả năng tương thích của hệ thống trên nhiều môi trường khác nhau.
- **Nâng cao trực quan hóa kết quả kiểm thử**: Tích hợp các công cụ báo cáo nâng cao như Allure Report hoặc ExtentReports để hiển thị kết quả kiểm thử một cách trực quan, dễ theo dõi và thuận tiện cho việc đánh giá và phân tích.
- **Xây dựng hệ thống kiểm thử hoàn toàn tự động**: Tiến tới triển khai quy trình kiểm thử khép kín bao gồm khởi tạo dữ liệu đầu vào, thực thi kiểm thử, xác minh kết quả và dọn dẹp môi trường sau khi kiểm thử, qua đó giảm thiểu tối đa sự can thiệp thủ công.

Các định hướng trên không chỉ giúp hoàn thiện hệ thống kiểm thử tự động hiện có mà còn tạo nền tảng để nhân rộng và áp dụng ở quy mô lớn hơn, đáp ứng yêu cầu phát triển phần mềm hiện đại trong bối cảnh DevOps và Agile ngày càng phổ biến, nơi mà tốc độ và chất lượng đều là những yếu tố then chốt.

DANH MỤC TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Hồ Tú Bảo (2019) – "Việt Nam thời Chuyển đổi số", Nhà xuất bản Thế giới.

[2] Cẩm nang chuyển đổi số do Bộ Thông tin và Truyền thông ban hành.

[3] https://www.a1consulting.vn/

Tiếng Anh

[4] Murali Chemuturi (2010) – "Mastering Software Quality Assurance: Best Practices, Tools and Techniques for Software Developers"

[5] David Burns (2012) – "Selenium 2 Testing Tools: Beginner's Guide"

[6] David Burns (2012) – "API Testing and Test Automation"

[7] Mark Collin (2015) – "Mastering Selenium WebDriver", Packt Publishing

[8] David R. Kuhn, Renee Bryce, Feng Duan, Laleh Ghandehari, Yu Lei, and Raghu N. Kacker (2015). "Combinatorial Testing: Theory and Practice".

[9] Dorothy Graham, Rex Black, and Erik van Veenendaal (2020). "Foundations of Software Testing''.

[10] Rahman, B. (2016). *Test Automation Using Selenium WebDriver with Java*. CreateSpace Independent Publishing.

[11] https://playwright.dev/docs/intro

[12] https://docs.cypress.io/app/get-started/why-cypress

[13] https://appium.io/docs/en/latest/

[14] https://www.selenium.dev/documentation/test_practices

BẢN CAM ĐOAN

Tôi xin cam đoan đã thực hiện kiểm tra mức độ trùng lặp nội dung của đề án thông qua phần mềm kiểm tra tài liệu một cách trung thực, và kết quả cho thấy mức độ trùng lặp là 12**%** so với toàn bộ nội dung của đề án tốt nghiệp.

Bản đề án đã được kiểm tra bằng phần mềm là bản cứng chính thức đã nộp để bảo vệ trước Hội đồng. Tôi xin hoàn toàn chịu trách nhiệm về tính trung thực của cam kết này. Nếu có sai sót, tôi xin chấp nhận mọi hình thức xử lý theo quy định hiện hành của Học viện.

Hà Nội, ngày …… tháng …… năm ……

HỌC VIÊN CAO HỌC

(Ký và ghi rõ họ tên)

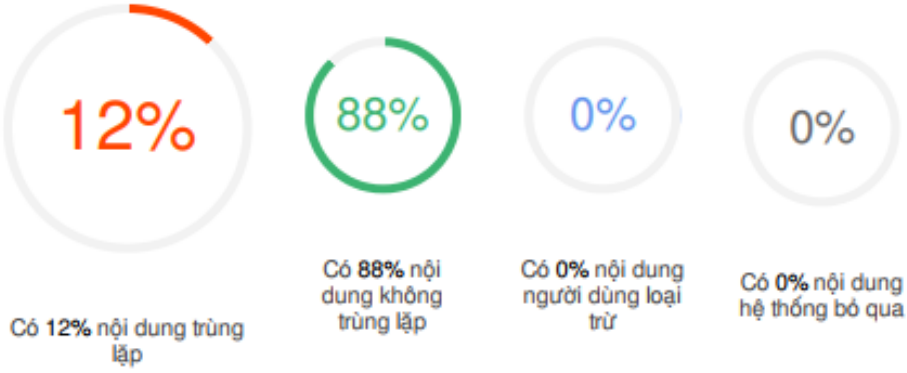
KiểmTraTàiLiệu

BÁO CÁO KIỂM TRA TRÙNG LẶP

Thông tin tài liệu

Tên tài liệu:	Đề án - Nguyễn Thị Nụ - B23CHIS056
Tác giả:	Nguyễn Thị Nụ
Điểm trùng lặp:	12
Thời gian tải lên:	16:39 30/07/2025
Thời gian sinh báo cáo:	16:42 30/07/2025
Các trang kiểm tra:	75/75 trang

Kết quả kiểm tra trùng lặp



Có **12%** nội dung trùng lặp

Có **88%** nội dung không trùng lặp

Có **0%** nội dung người dùng loại trừ

Có **0%** nội dung hệ thống bỏ qua

Nguồn trùng lặp tiêu biểu

123docz.net tailieu.vn testmentor.vn