TRƯỜNG ĐẠI HỌC ĐÔNG Á
KHOA CÔNG NGHỆ THÔNG TIN

HỌC PHẦN: KIỂM THỬ PHẦN MỀM 2

ĐỀ TÀI: KIỂM THỬ CHO PHẦN MỀM BÁN HÀNG ONLINE SỬ DỤNG CÁC CÔNG CỤ HIỆN ĐẠI

Giảng viên hướng dẫn: TS. Võ Đức Hoàng

Họ và tên sinh viên: **Võ Đức Chính**
Hoàng Văn Chuẩn
Trần Lê Hữu Quốc
Nguyễn Đức Vũ

Lớp: **ST22B**

Đà Nẵng - 2025

MỤC LỤC

MỞ ĐẦU

1. Lý do chọn đề tài

Kiểm thử phần mềm đóng vai trò then chốt trong việc đảm bảo chất lượng của các ứng dụng bán hàng online. Tuy nhiên, phương pháp kiểm thử truyền thống với phần lớn công việc được thực hiện thủ công đã bộc lộ nhiều hạn chế như tốn thời gian, chi phí cao và khả năng phát hiện lỗi không đồng đều. Trong khi đó, các công cụ kiểm thử hiện đại như Selenium, Appium, Postman và JMeter đang ngày càng được ứng dụng rộng rãi nhờ khả năng tự động hóa và hiệu quả cao.

Việc nghiên cứu và áp dụng các công cụ kiểm thử hiện đại cho phần mềm bán hàng online không chỉ có ý nghĩa học thuật mà còn có giá trị thực tiễn cao, giúp nâng cao chất lượng sản phẩm, giảm thiểu chi phí và thời gian phát triển, đồng thời tăng cường sự hài lòng của người dùng.

a) Nghiên cứu lý thuyết về kiểm thử phần mềm và đặc điểm của các công cụ kiểm thử hiện đại bao gồm Selenium, Appium, Postman và JMeter.

b) Phân tích và thiết kế các trường hợp kiểm thử (test cases) phù hợp cho phần mềm bán hàng online đa nền tảng (web, mobile và API).

c) Triển khai kiểm thử tự động cho giao diện web với Selenium, ứng dụng di động với Appium, API backend với Postman và hiệu năng hệ thống với JMeter.

d) Đề xuất quy trình kiểm thử tự động tích hợp có thể áp dụng cho các hệ thống thương mại điện tử tương tự.

e) Đánh giá hiệu quả của việc sử dụng các công cụ kiểm thử hiện đại và rút ra các bài học kinh nghiệm.

CHƯƠNG 1
CƠ SỞ LÝ THUYẾT VỀ KIỂM THỬ PHẦN MỀM

1.1. Tổng quan về kiểm thử phần mềm

1.1.1. Khái niệm, mục đích và tầm quan trọng của kiểm thử

Kiểm thử phần mềm là một quy trình kiểm tra và đánh giá phần mềm nhằm xác định liệu phần mềm có đáp ứng đúng các yêu cầu kỹ thuật và kinh doanh mong đợi hay không, và để phát hiện lỗi trước khi đưa phần mềm vào sử dụng. Kiểm thử không chỉ đơn thuần là tìm lỗi mà còn là một quy trình đảm bảo chất lượng toàn diện.

Mục đích của kiểm thử phần mềm:

- Phát hiện các lỗi và khiếm khuyết trong phần mềm.
- Xác minh rằng phần mềm đáp ứng đúng các yêu cầu đã được định nghĩa.
- Kiểm tra tính tương thích của phần mềm với các môi trường, hệ điều hành và thiết bị khác nhau.
- Đánh giá hiệu năng, bảo mật và độ tin cậy của hệ thống.
- Cung cấp thông tin về chất lượng của sản phẩm để hỗ trợ ra quyết định.

Tầm quan trọng của kiểm thử:

Trong bối cảnh của các hệ thống thương mại điện tử, việc kiểm thử đóng vai trò đặc biệt quan trọng vì:

- Các lỗi trong phần mềm có thể dẫn đến thiệt hại tài chính trực tiếp (ví dụ: tính toán giá sai, lỗi thanh toán).
- Trải nghiệm người dùng kém có thể làm giảm doanh thu và mất khách hàng.
- Các vấn đề bảo mật có thể dẫn đến rò rỉ thông tin cá nhân và tài chính của người dùng.
- Đảm bảo hệ thống hoạt động ổn định khi có nhiều người dùng truy cập đồng thời.

1.1.2. Các giai đoạn trong quy trình kiểm thử phần mềm

Quy trình kiểm thử phần mềm thường bao gồm các giai đoạn sau:

a) **Phân tích yêu cầu kiểm thử:** Xác định phạm vi và mục tiêu kiểm thử dựa trên yêu cầu của hệ thống, rủi ro và các tính năng quan trọng.

b) **Lập kế hoạch kiểm thử:** Phát triển chiến lược kiểm thử, xác định môi trường kiểm thử, công cụ, lịch trình và nguồn lực.

c) **Thiết kế kiểm thử:** Tạo các trường hợp kiểm thử (test cases), kịch bản kiểm thử và dữ liệu kiểm thử.

d) **Thực hiện kiểm thử:** Thực thi các trường hợp kiểm thử, ghi lại kết quả và so sánh với kết quả mong đợi.

e) **Báo cáo lỗi:** Ghi lại và phân loại các lỗi phát hiện được.

f) **Kiểm thử lại và kiểm thử hồi quy:** Xác nhận lỗi đã được sửa và đảm bảo các phần khác của phần mềm không bị ảnh hưởng.

g) **Báo cáo kết quả kiểm thử:** Tổng hợp kết quả kiểm thử và cung cấp thông tin về chất lượng phần mềm.

1.1.3. Các cấp độ kiểm thử

Kiểm thử đơn vị (Unit testing)

Kiểm thử đơn vị tập trung vào việc kiểm tra từng đơn vị riêng lẻ của mã nguồn, chẳng hạn như các hàm, lớp hoặc phương thức. Mục đích là xác minh rằng mỗi đơn vị hoạt động đúng như mong đợi khi được tách biệt khỏi các thành phần khác của hệ thống.

Kiểm thử tích hợp (Integration testing)

Kiểm thử tích hợp kiểm tra khả năng tương tác giữa các đơn vị khác nhau trong hệ thống. Mục tiêu là phát hiện các vấn đề khi các thành phần được kết hợp với nhau.

Kiểm thử hệ thống (System testing)

Kiểm thử hệ thống là quá trình kiểm tra toàn bộ hệ thống sau khi tất cả các thành phần đã được tích hợp. Mục tiêu là đánh giá liệu hệ thống có đáp ứng các yêu cầu kỹ thuật và nghiệp vụ hay không.

Kiểm thử chấp nhận (Acceptance testing)

Kiểm thử chấp nhận là giai đoạn cuối cùng trong quy trình kiểm thử, tập trung vào việc xác minh rằng hệ thống đáp ứng nhu cầu của người dùng và các yêu cầu kinh doanh. Loại kiểm thử này thường được thực hiện bởi người dùng cuối hoặc người đại diện cho họ.

1.1.4. Các loại kiểm thử

Kiểm thử chức năng (Functional testing)

Kiểm thử chức năng tập trung vào việc kiểm tra các tính năng cụ thể của phần mềm để đảm bảo chúng hoạt động theo đúng yêu cầu. Loại kiểm thử này không quan tâm đến cấu trúc nội bộ của mã nguồn mà chỉ tập trung vào đầu vào và đầu ra.

Kiểm thử phi chức năng (Non-functional testing)

Kiểm thử phi chức năng đánh giá các khía cạnh của phần mềm không liên quan trực tiếp đến các chức năng cụ thể, chẳng hạn như hiệu suất, bảo mật, khả năng sử dụng và khả năng mở rộng.

Các loại kiểm thử phi chức năng bao gồm:

- **Kiểm thử hiệu năng:** Đánh giá thời gian phản hồi, thông lượng và tài nguyên sử dụng dưới các điều kiện tải khác nhau.
- **Kiểm thử bảo mật:** Kiểm tra khả năng bảo vệ dữ liệu và chống lại các cuộc tấn công.
- **Kiểm thử khả năng sử dụng:** Đánh giá mức độ thân thiện với người dùng của giao diện.
- **Kiểm thử tương thích:** Kiểm tra hoạt động của phần mềm trên các nền tảng, thiết bị và trình duyệt khác nhau.

Kiểm thử hồi quy (Regression testing)

Kiểm thử hồi quy là việc kiểm tra lại các tính năng đã tồn tại sau khi có thay đổi để đảm bảo rằng các thay đổi mới không gây ra lỗi trong các phần đã hoạt động trước đó.

1.2. Kiểm thử tự động

1.2.1. Khái niệm và lợi ích của kiểm thử tự động

Kiểm thử tự động là việc sử dụng các công cụ và kỹ thuật để thực hiện các trường hợp kiểm thử mà không cần sự can thiệp trực tiếp của con người. Thay vì nhân viên kiểm thử thực hiện các bước kiểm thử thủ công, các script tự động sẽ thực hiện các bước kiểm thử, so sánh kết quả thực tế với kết quả mong đợi và báo cáo bất kỳ sự khác biệt nào.

Lợi ích của kiểm thử tự động:

- **Tiết kiệm thời gian và chi phí:** Sau khi được thiết lập, các bài kiểm thử tự động có thể chạy nhanh chóng và lặp lại nhiều lần mà không cần sự can thiệp của con người.
- **Tính nhất quán và độ tin cậy:** Các bài kiểm thử tự động luôn được thực hiện theo cùng một cách, giúp loại bỏ lỗi con người và đảm bảo tính nhất quán trong quá trình kiểm thử.
- **Phạm vi kiểm thử rộng hơn:** Kiểm thử tự động cho phép kiểm tra nhiều trường hợp và kịch bản phức tạp mà có thể khó hoặc tốn thời gian để thực hiện thủ công.
- **Phát hiện lỗi sớm:** Tích hợp kiểm thử tự động vào quy trình phát triển liên tục (CI/CD) giúp phát hiện lỗi ngay sau khi chúng được đưa vào mã nguồn.
- **Phản hồi nhanh chóng:** Các nhà phát triển nhận được phản hồi nhanh chóng về các thay đổi mã, giúp giảm thời gian sửa lỗi.
- **Tái sử dụng:** Các script kiểm thử tự động có thể được tái sử dụng cho nhiều phiên bản của phần mềm.

1.2.2. Các loại công cụ kiểm thử tự động

Có nhiều loại công cụ kiểm thử tự động khác nhau được phát triển cho các mục đích kiểm thử cụ thể:

a) **Công cụ kiểm thử đơn vị:** Giúp tự động hóa việc kiểm thử các đơn vị nhỏ của mã nguồn như hàm và lớp. Ví dụ: JUnit (Java), pytest (Python), Jest (JavaScript), v.v.

b) **Công cụ kiểm thử giao diện người dùng (UI):** Tự động hóa tương tác với giao diện người dùng để kiểm tra chức năng và trải nghiệm người dùng. Ví dụ: Selenium, Appium, Cypress, Playwright, v.v.

c) **Công cụ kiểm thử API:** Giúp kiểm tra các endpoint API để đảm bảo chúng trả về dữ liệu chính xác và xử lý các yêu cầu một cách đúng đắn. Ví dụ: Postman, REST Assured, SoapUI, v.v.

d) **Công cụ kiểm thử hiệu năng:** Đánh giá hiệu suất của ứng dụng dưới các điều kiện tải khác nhau. Ví dụ: JMeter, LoadRunner, Gatling, v.v.

e) **Công cụ kiểm thử bảo mật:** Phát hiện các lỗ hổng bảo mật trong mã nguồn hoặc ứng dụng đang chạy. Ví dụ: OWASP ZAP, SonarQube, Burp Suite, v.v.

f) **Công cụ kiểm thử di động:** Tập trung vào kiểm thử ứng dụng chạy trên các thiết bị di động. Ví dụ: Appium, Espresso (Android), XCTest (iOS), v.v.

g) **Công cụ quản lý kiểm thử:** Hỗ trợ quản lý các trường hợp kiểm thử, kế hoạch kiểm thử, và báo cáo kiểm thử. Ví dụ: TestRail, qTest, Zephyr, v.v.

1.3. Giới thiệu các công cụ kiểm thử

1.3.1. Selenium

Tổng quan về Selenium

Selenium là một bộ công cụ kiểm thử tự động phổ biến dành cho các ứng dụng web. Nó cho phép tự động hóa tương tác với trình duyệt web, mô phỏng các hành động của người dùng như nhấp chuột, nhập liệu, và kiểm tra nội dung trang.

Selenium bao gồm nhiều thành phần, trong đó quan trọng nhất là:

- **Selenium WebDriver:** API cho phép điều khiển trình duyệt từ mã nguồn.
- **Selenium IDE:** Công cụ ghi lại và phát lại các tương tác trên trình duyệt.
- **Selenium Grid:** Cho phép thực hiện kiểm thử song song trên nhiều trình duyệt và hệ điều hành khác nhau.

Ưu điểm và hạn chế

Ưu điểm:

- Hỗ trợ nhiều trình duyệt (Chrome, Firefox, Safari, Edge, v.v.) và hệ điều hành.
- Hỗ trợ nhiều ngôn ngữ lập trình (Java, Python, C#, JavaScript, v.v.).
- Cộng đồng lớn và nhiều tài nguyên hỗ trợ.
- Mã nguồn mở và miễn phí.
- Có thể tích hợp với các framework kiểm thử như TestNG, JUnit, v.v.

Hạn chế:

- Đường cong học tập dốc đối với người mới bắt đầu.
- Các selector có thể thay đổi khi giao diện người dùng thay đổi, dẫn đến các bài kiểm thử không ổn định.
- Không có cơ chế báo cáo tích hợp, cần phải tích hợp với các công cụ bên ngoài.
- Không phù hợp cho các ứng dụng không phải web.

1.3.2. Appium

Tổng quan về Appium

Appium là một công cụ kiểm thử tự động mã nguồn mở cho các ứng dụng di động (Android và iOS) và ứng dụng desktop. Nó cho phép kiểm thử trên các thiết bị thực, giả lập và mô phỏng. Appium tuân theo triết lý ”không cần sửa đổi ứng dụng để kiểm thử”, cho phép kiểm thử cùng một ứng dụng sẽ được phát hành cho người dùng cuối.

Ưu điểm và hạn chế

Ưu điểm:

- Hỗ trợ đa nền tảng (Android, iOS, Windows).
- Hỗ trợ nhiều ngôn ngữ lập trình thông qua giao thức WebDriver.
- Không yêu cầu mã nguồn của ứng dụng.
- Có thể kiểm thử các ứng dụng native, hybrid và web trên di động.
- Mã nguồn mở và miễn phí.

Hạn chế:

- Thiết lập ban đầu phức tạp với nhiều phụ thuộc.
- Hiệu suất có thể chậm hơn so với các framework kiểm thử native.
- Gặp vấn đề với một số điều khiển UI phức tạp.
- Có thể gặp vấn đề về độ ổn định khi kiểm thử trên các thiết bị khác nhau.

1.3.3. Postman

Tổng quan về Postman

Postman là một nền tảng API toàn diện cho việc xây dựng và sử dụng API. Ban đầu được phát triển như một tiện ích mở rộng của Chrome, nay đã trở thành một ứng dụng độc lập với nhiều tính năng mạnh mẽ cho việc kiểm thử, gỡ lỗi, đồng bộ hóa và tài liệu hóa API.

Trong ngữ cảnh kiểm thử, Postman cho phép người dùng gửi các yêu cầu HTTP đến API, xác minh phản hồi, và tự động hóa quá trình kiểm thử thông qua các bộ sưu tập và script kiểm thử.

Ưu điểm và hạn chế

Ưu điểm:

- Giao diện người dùng trực quan và dễ sử dụng.
- Hỗ trợ nhiều phương thức HTTP và định dạng dữ liệu.
- Khả năng tự động hóa kiểm thử API thông qua Collection Runner và Newman (CLI).
- Hỗ trợ biến môi trường và toàn cục để quản lý các giá trị động.
- Tích hợp với các công cụ CI/CD.
- Khả năng tạo tài liệu API tự động.
- Hỗ trợ làm việc nhóm thông qua Postman Workspace.

Hạn chế:

- Một số tính năng nâng cao chỉ có trong phiên bản trả phí.
- Không phải là giải pháp kiểm thử toàn diện cho các loại kiểm thử khác ngoài API.
- Hiệu suất có thể giảm với các bộ kiểm thử lớn.
- Các tùy chọn báo cáo còn hạn chế so với các công cụ kiểm thử chuyên dụng.

1.3.4. JMeter

Tổng quan về JMeter

Apache JMeter là một công cụ kiểm thử hiệu năng mã nguồn mở được phát triển bởi Apache Software Foundation. Ban đầu được thiết kế cho kiểm thử ứng dụng web, JMeter đã phát triển để hỗ trợ nhiều loại yêu cầu khác nhau bao gồm HTTP, HTTPS, SOAP, REST, FTP, JDBC, LDAP, JMS, và nhiều giao thức khác.

JMeter cho phép mô phỏng tải nặng trên máy chủ hoặc nhóm máy chủ, phân tích hiệu suất tổng thể dưới các điều kiện tải khác nhau.

Ưu điểm và hạn chế

Ưu điểm:

- Mã nguồn mở và miễn phí.
- Hỗ trợ nhiều loại ứng dụng và giao thức.
- Giao diện người dùng đồ họa để thiết kế và gỡ lỗi kế hoạch kiểm thử.
- Phân tích và báo cáo kết quả kiểm thử toàn diện.
- Khả năng mở rộng thông qua plugin.
- Có thể chạy trong chế độ phân tán để mô phỏng lượng tải lớn.
- Tích hợp tốt với các công cụ CI/CD.

Hạn chế:

- Giao diện người dùng có thể cũ và không trực quan bằng một số công cụ hiện đại.
- Tiêu thụ nhiều tài nguyên khi mô phỏng tải lớn trên một máy duy nhất.
- Có thể khó khăn khi thiết lập các kịch bản kiểm thử phức tạp.
- Khả năng giả lập trình duyệt web còn hạn chế (không hiển thị nội dung như trình duyệt thực).
- Đường cong học tập dốc cho người mới bắt đầu.

CHƯƠNG 2
PHÂN TÍCH VÀ THIẾT KẾ KIỂM THỬ CHO PHẦN MỀM BÁN HÀNG ONLINE

2.1. Giới thiệu về phần mềm bán hàng online được nghiên cứu

2.1.1. Mô tả chức năng chính

Phần mềm bán hàng online được nghiên cứu trong đề tài này là một hệ thống thương mại điện tử đa nền tảng cho phép người dùng thực hiện các hoạt động mua sắm cơ bản trên môi trường web và thiết bị di động. Hệ thống được phát triển với mục tiêu cung cấp trải nghiệm mua sắm trực tuyến thuận tiện, an toàn và hiệu quả cho người dùng.

Các chức năng chính của hệ thống bao gồm:

a) **Quản lý tài khoản người dùng:**

- Đăng ký tài khoản mới
- Đăng nhập/đăng xuất
- Xem và cập nhật thông tin cá nhân
- Quản lý địa chỉ giao hàng

b) **Duyệt và tìm kiếm sản phẩm:**

- Xem danh sách sản phẩm theo danh mục
- Tìm kiếm sản phẩm theo từ khóa
- Lọc sản phẩm theo các tiêu chí (giá, đánh giá, thương hiệu, v.v.)
- Xem chi tiết sản phẩm (thông tin, hình ảnh, đánh giá)

c) **Giỏ hàng và đặt hàng:**

- Thêm sản phẩm vào giỏ hàng
- Quản lý số lượng sản phẩm trong giỏ hàng
- Xem tổng giá trị đơn hàng
- Tiến hành thanh toán
- Theo dõi trạng thái đơn hàng

d) **Thanh toán:**

- Lựa chọn phương thức thanh toán (thẻ tín dụng, chuyển khoản ngân hàng, COD)
- Áp dụng mã giảm giá
- Xem lịch sử giao dịch

e) **Đánh giá và nhận xét:**

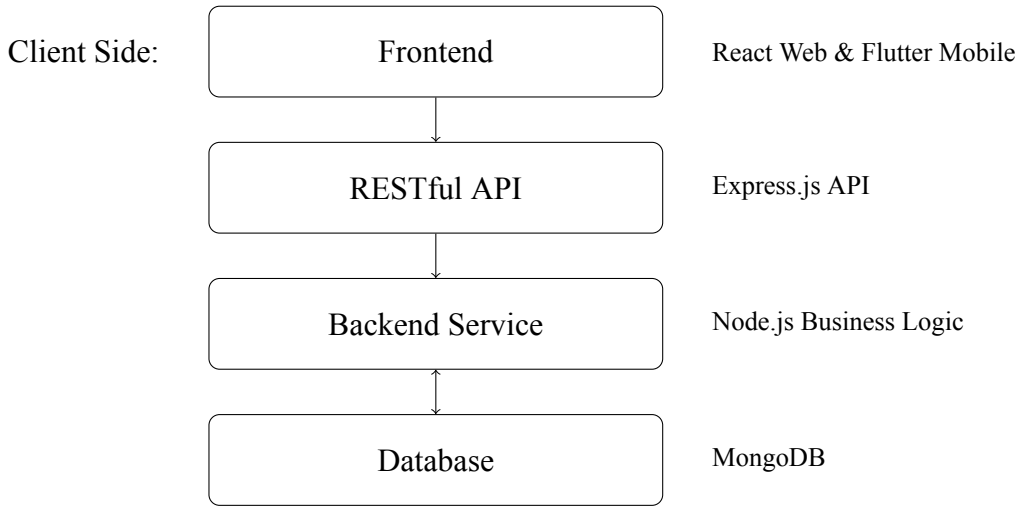
- Đánh giá sản phẩm đã mua
- Xem đánh giá của người dùng khác

f) **Quản lý (dành cho quản trị viên):**

- Quản lý danh mục sản phẩm
- Quản lý sản phẩm (thêm, sửa, xóa)
- Quản lý đơn hàng
- Quản lý người dùng
- Xem báo cáo và thống kê

2.1.2. Kiến trúc hệ thống

Hệ thống phần mềm bán hàng online được phát triển theo kiến trúc phân tầng hiện đại, bao gồm ba thành phần chính:



Hình 2.1: Kiến trúc tổng quan của hệ thống

Frontend

Ứng dụng Web

Giao diện web được phát triển sử dụng React.js - một thư viện JavaScript phổ biến cho việc xây dựng giao diện người dùng. Ứng dụng web được thiết kế theo kiến trúc SPA (Single Page Application) giúp cung cấp trải nghiệm người dùng mượt mà và phản hồi nhanh. Giao diện sử dụng Tailwind CSS để tạo các thành phần UI thích ứng với nhiều thiết bị khác nhau.

Ứng dụng Di động

Ứng dụng di động được phát triển bằng Flutter - framework đa nền tảng của Google cho phép xây dựng ứng dụng native trên cả Android và iOS từ một codebase duy nhất. Ứng dụng được thiết kế để cung cấp trải nghiệm người dùng tối ưu trên các thiết bị di động với các thành phần giao diện material design.

Backend API

Backend API được xây dựng trên nền tảng Node.js với framework Express.js để tạo ra các RESTful API endpoints. API này đóng vai trò trung gian giữa frontend và database, xử lý các yêu cầu từ client, thực thi logic nghiệp vụ và trả về dữ liệu.

API được thiết kế để đáp ứng các tiêu chuẩn RESTful với các endpoint chính:

- /api/auth: Xử lý đăng ký, đăng nhập và quản lý token
- /api/products: CRUD cho sản phẩm
- /api/categories: Quản lý danh mục sản phẩm
- /api/cart: Quản lý giỏ hàng
- /api/orders: Xử lý đơn hàng
- /api/users: Quản lý người dùng

Database

Hệ thống sử dụng MongoDB - một cơ sở dữ liệu NoSQL dạng document, cho phép lưu trữ dữ liệu dưới dạng JSON-like documents. MongoDB được chọn vì tính linh hoạt trong cấu trúc dữ liệu, khả năng mở rộng cao và phù hợp với ứng dụng web hiện đại.

2.2. Xác định phạm vi kiểm thử

Dựa trên phân tích kiến trúc và chức năng của hệ thống, chúng tôi đã xác định phạm vi kiểm thử bao gồm bốn lĩnh vực chính:

2.2.1. Kiểm thử giao diện Web với Selenium

Phạm vi kiểm thử giao diện web tập trung vào các chức năng người dùng chính trên nền tảng web:

- **Đăng nhập và đăng ký:** Kiểm tra quy trình đăng ký tài khoản mới, đăng nhập với các tài khoản hợp lệ và không hợp lệ, khôi phục mật khẩu.
- **Duyệt và tìm kiếm sản phẩm:** Kiểm tra tính năng tìm kiếm, lọc và phân trang, xem chi tiết sản phẩm.
- **Quản lý giỏ hàng:** Thêm và xóa sản phẩm, cập nhật số lượng, áp dụng mã giảm giá.
- **Quy trình thanh toán:** Kiểm tra quy trình thanh toán đầy đủ từ giỏ hàng đến xác nhận đơn hàng.
- **Quản lý tài khoản người dùng:** Cập nhật thông tin cá nhân, xem lịch sử đơn hàng, quản lý địa chỉ.
- **Phần quản trị viên:** Quản lý sản phẩm, đơn hàng, người dùng và báo cáo.

2.2.2. Kiểm thử ứng dụng Mobile với Appium

Phạm vi kiểm thử ứng dụng di động tập trung vào:

- **Tương thích thiết bị:** Kiểm tra hoạt động trên các thiết bị Android với kích thước màn hình và phiên bản hệ điều hành khác nhau.
- **Luồng người dùng:** Đăng ký, đăng nhập, duyệt sản phẩm, thêm vào giỏ hàng, thanh toán.
- **Trải nghiệm người dùng:** Hiệu suất UI, độ phản hồi, hoạt động offline và khả năng phục hồi khi mất kết nối.
- **Thông báo đẩy:** Kiểm tra việc nhận và xử lý thông báo đẩy.
- **Tích hợp với các tính năng thiết bị:** Camera để quét QR code, GPS để xác định vị trí, v.v.

2.2.3. Kiểm thử API với Postman

Phạm vi kiểm thử API bao gồm tất cả các API endpoints:

- **Authentication API:** Đăng ký, đăng nhập, refresh token, đổi mật khẩu.
- **Product API:** Lấy danh sách sản phẩm, chi tiết sản phẩm, tìm kiếm và lọc sản phẩm.
- **Cart API:** Thêm vào giỏ hàng, cập nhật số lượng, xóa sản phẩm, xem giỏ hàng.
- **Order API:** Tạo đơn hàng, xem chi tiết đơn hàng, cập nhật trạng thái.
- **User API:** Xem và cập nhật thông tin người dùng, quản lý địa chỉ.
- **Admin API:** Quản lý sản phẩm, danh mục, người dùng, đơn hàng.

Các khía cạnh quan trọng cần kiểm thử:

- Xác thực và phân quyền
- Xử lý lỗi và phản hồi
- Hiệu suất và thời gian phản hồi
- Giới hạn tốc độ (rate limiting)
- Định dạng dữ liệu đầu vào và đầu ra

2.2.4. Kiểm thử hiệu năng với JMeter

Phạm vi kiểm thử hiệu năng tập trung vào đánh giá khả năng đáp ứng của hệ thống dưới các điều kiện tải khác nhau:

- **Kiểm thử tải (Load testing):** Đánh giá hành vi của hệ thống dưới tải dự kiến và tải cao.
- **Kiểm thử sức chịu đựng (Endurance testing):** Kiểm tra hiệu suất hệ thống trong thời gian dài.
- **Kiểm thử căng thẳng (Stress testing):** Xác định giới hạn của hệ thống bằng cách tăng tải đến khi hệ thống không đáp ứng được.
- **Kiểm thử đột biến (Spike testing):** Đánh giá phản ứng của hệ thống đối với sự gia tăng đột ngột trong lưu lượng người dùng.

2.3. Thiết kế các trường hợp kiểm thử (Test cases)

2.3.1. Kiểm thử chức năng

Chúng tôi đã thiết kế các trường hợp kiểm thử chức năng cho cả giao diện web và ứng dụng di động, tập trung vào các tính năng quan trọng nhất của hệ thống.

Kiểm thử đăng nhập

ID	Scenario	Input data	Expected result	Verification
1	Đăng nhập thành công	Email: testuser@gmail.com Password: Password123!	Người dùng được chuyển hướng đến trang chủ	Kiểm tra URL trang sau đăng nhập và hiển thị thông tin người dùng
2	Đăng nhập với email không tồn tại	Email: nonexistent@gmail.com Password: Password123!	Hiển thị thông báo lỗi ”Invalid credentials”	Kiểm tra thông báo lỗi hiển thị
3	Đăng nhập với mật khẩu sai	Email: testuser@gmail.com Password: WrongPassword!	Hiển thị thông báo lỗi ”Invalid credentials”	Kiểm tra thông báo lỗi hiển thị
4	Đăng nhập với email không hợp lệ	Email: invalid-email Password: Password123!	Hiển thị thông báo lỗi ”Invalid email format”	Kiểm tra thông báo lỗi định dạng email
5	Đăng nhập với trường email trống	Email: [empty] Password: Password123!	Hiển thị thông báo lỗi ”Email is required”	Kiểm tra thông báo lỗi yêu cầu điền email

Kiểm thử quản lý giỏ hàng

ID	Scenario	Input data	Expected result	Verification

1	Thêm sản phẩm vào giỏ hàng	Sản phẩm ID: 1 Số lượng: 1	Sản phẩm được thêm vào giỏ hàng	Kiểm tra số lượng sản phẩm trong giỏ và thông báo thành công
2	Tăng số lượng sản phẩm	Sản phẩm ID: 1 Số lượng: +1	Số lượng sản phẩm tăng và tổng tiền được cập nhật	Kiểm tra hiển thị số lượng và tổng tiền mới
3	Giảm số lượng sản phẩm	Sản phẩm ID: 1 Số lượng: -1	Số lượng sản phẩm giảm và tổng tiền được cập nhật	Kiểm tra hiển thị số lượng và tổng tiền mới
4	Xóa sản phẩm khỏi giỏ hàng	Sản phẩm ID: 1	Sản phẩm bị xóa khỏi giỏ hàng	Kiểm tra số lượng sản phẩm trong giỏ và thông báo xóa

2.3.2. Kiểm thử API

Chúng tôi đã thiết kế các test case cho việc kiểm thử API bằng Postman, tập trung vào việc xác minh tính đúng đắn của các endpoint và phản hồi.

Authentication API

ID	Endpoint	Method & Input	Expected Status	Expected Response
1	/api/auth/register	POST {name: ”New User”, email: ”new@example.com”, password: ”Password123”}	201 Created	{message: ”User registered successfully”, user: {id, name, email}}
2	/api/auth/register	POST {name: ”Test User”, email: ”existing@example.com”, password: ”Password123”}	409 Conflict	{error: ”Email already exists”}
3	/api/auth/login	POST {email: ”user@example.com”, password: ”Password123”}	200 OK	{token: ”<JWT token>”, user: {id, name, email, role}}
4	/api/auth/login	POST {email: ”user@example.com”, password: ”WrongPassword”}	401 Unauthorized	{error: ”Invalid credentials”}

Products API

ID	Endpoint	Method & Input	Expected Status	Expected Response
1	/api/products	GET	200 OK	{products: [...], count: n}

2	/api/products? category=electronics	GET	200 OK	{products: [...filtered by category...], count: n}
3	/api/products/{id}	GET	200 OK	{product: {id, name, price, ...}}
4	/api/products/ {nonexistent-id}	GET	404 Not Found	{error: ”Product not found”}

2.3.3. Kiểm thử hiệu năng

Chúng tôi đã thiết kế các kịch bản kiểm thử hiệu năng để đánh giá khả năng đáp ứng của hệ thống dưới các điều kiện tải khác nhau.

ID	Kịch bản	Người dùng ảo	Thời gian	Mục tiêu
1	Duyệt trang chủ	100 đồng thời	5 phút	Thời gian phản hồi < 2s cho 95% request
2	Xem chi tiết sản phẩm	50 đồng thời	5 phút	Thời gian phản hồi < 3s cho 95% request
3	Tìm kiếm sản phẩm	75 đồng thời	5 phút	Thời gian phản hồi < 3s cho 95% request
4	Thêm vào giỏ hàng	50 đồng thời	5 phút	Thời gian phản hồi < 2s cho 95% request
5	Thanh toán	25 đồng thời	5 phút	Thời gian phản hồi < 5s cho 95% request

2.3.4. Kiểm thử di động

Chúng tôi đã thiết kế các test case cho ứng dụng di động sử dụng Appium, tập trung vào trải nghiệm người dùng và khả năng đáp ứng trên các thiết bị khác nhau.

ID	Test case	Thiết bị	Input data	Expected result
1	Khởi động ứng dụng	Android 11+	-	Ứng dụng khởi động thành công, hiển thị màn hình đăng nhập
2	Đăng nhập	Android 11+	Email: testuser@gmail .com Password: Password123!	Đăng nhập thành công, hiển thị trang chủ
3	Xoay màn hình	Android 11+	-	UI thích ứng đúng với hướng màn hình mới
4	Chức năng offline	Android 11+	Tắt kết nối mạng	Ứng dụng hiển thị thông báo phù hợp và cho phép xem dữ liệu đã cache

2.4. Chuẩn bị môi trường kiểm thử

Để thực hiện các bài kiểm thử đã thiết kế, chúng tôi đã thiết lập các môi trường kiểm thử sau:

2.4.1. Môi trường kiểm thử Web

Server:

- Ứng dụng React được triển khai trên Azure Container Instances
- Backend API triển khai trên VPS riêng biệt
- MongoDB Atlas cho cơ sở dữ liệu

Client:

- Google Chrome (phiên bản mới nhất)
- Mozilla Firefox (phiên bản mới nhất)
- Microsoft Edge (phiên bản mới nhất)

Công cụ kiểm thử:

- Selenium WebDriver với Python
- Framework unittest của Python
- ChromeDriver

2.4.2. Môi trường kiểm thử Mobile

Thiết bị:

- Máy ảo Android Emulator (Android 11, 12, 13)

Công cụ:

- Appium Server phiên bản 2.18.0
- Appium Inspector để phân tích giao diện
- Android SDK và Android Debug Bridge (ADB)
- Python với Appium-Python-Client

2.4.3. Môi trường kiểm thử API

Server:

- Môi trường phát triển: localhost:3000
- Môi trường kiểm thử: ktpm-react-app.eastus.azurecontainer.io
- Môi trường sản xuất: ktpm-api-g9gcd2epanhch3dz.southeastasia-01.azurewebsites.net

Công cụ:

- Postman desktop app phiên bản mới nhất
- Newman cho việc chạy kiểm thử tự động
- Biến môi trường Postman cho các môi trường khác nhau

2.4.4. Môi trường kiểm thử hiệu năng

Server:

- Môi trường staging riêng biệt để không ảnh hưởng đến dữ liệu sản xuất
- Server có cấu hình tương tự với môi trường sản xuất

Công cụ:

- Apache JMeter phiên bản 5.5
- JMeter Plugins Manager với các plugin bổ sung

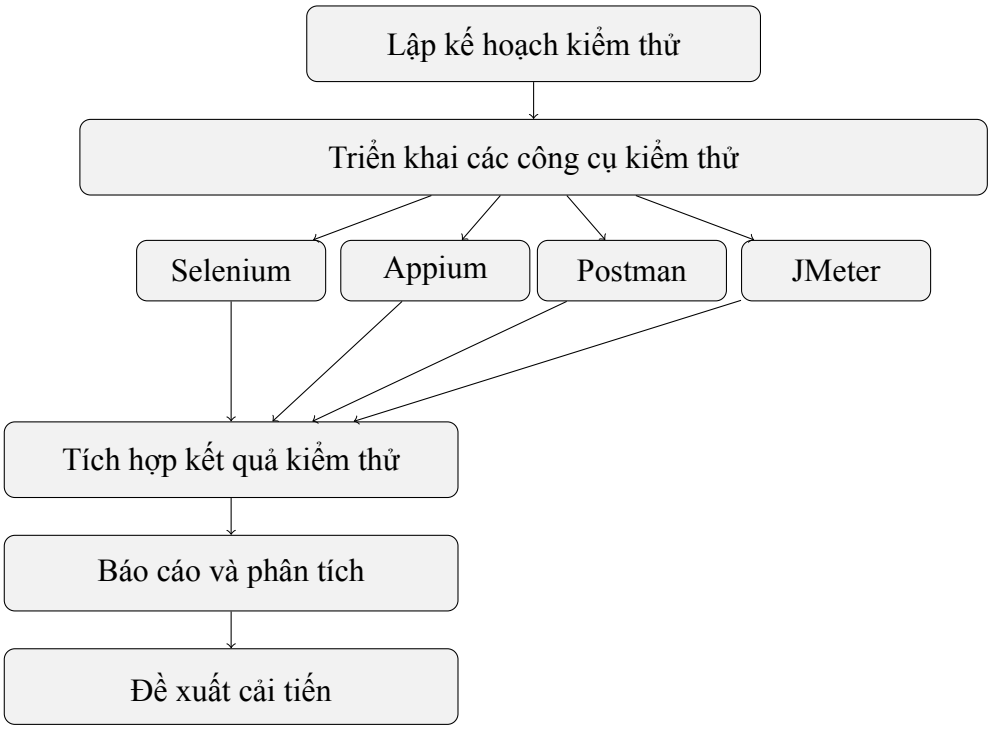
Dữ liệu kiểm thử:

- Dữ liệu người dùng: 100 tài khoản test
- Dữ liệu sản phẩm: 100 sản phẩm từ nhiều danh mục
- Dữ liệu đơn hàng và giao dịch

CHƯƠNG 3
TRIỂN KHAI KIỂM THỬ VÀ KẾT QUẢ

3.1. Tổng quan về quy trình triển khai kiểm thử

Trong chương này, chúng tôi trình bày chi tiết về quy trình triển khai kiểm thử cho phần mềm bán hàng online, bao gồm việc sử dụng bốn công cụ kiểm thử hiện đại: Selenium, Appium, Postman và JMeter. Mỗi công cụ được áp dụng cho một phạm vi kiểm thử cụ thể, đảm bảo phủ toàn diện các thành phần và chức năng của hệ thống.



Hình 3.1: Quy trình triển khai kiểm thử tổng thể

Quy trình triển khai kiểm thử được thực hiện theo các bước sau:

a) **Chuẩn bị môi trường:** Thiết lập và cấu hình từng công cụ kiểm thử trên môi trường phát triển và máy chủ CI/CD.

b) **Triển khai kiểm thử:** Thực hiện kiểm thử bằng cả bốn công cụ song song:

- Selenium cho giao diện web
- Appium cho ứng dụng di động
- Postman cho API backend
- JMeter cho hiệu năng hệ thống

c) **Thu thập kết quả:** Cơ chế báo cáo tự động được thiết lập để tổng hợp kết quả từ tất cả các công cụ.

d) **Phân tích và đánh giá:** Phân tích lỗi, đánh giá hiệu quả của hệ thống, và đề xuất cải tiến.

3.2. Kiểm thử ứng dụng di động với Appium

3.2.1. Cài đặt và cấu hình môi trường

Để triển khai kiểm thử ứng dụng di động (Flutter app), chúng tôi đã thiết lập môi trường Appium với các thành phần sau:

- **Appium Server:** Phiên bản 2.18.0 với giao diện quản lý mới.
- **Frameworks và thư viện:**
  - Python 3.12 với thư viện Appium-Python-Client mới nhất
  - pytest framework cho việc tổ chức và báo cáo kết quả
- **Thiết bị kiểm thử:**
  - Emulators: Android Virtual Devices với API 30 (Android 11)
- **Công cụ hỗ trợ:**
  - Android Debug Bridge (ADB) để giao tiếp với thiết bị Android
  - Appium Inspector để phân tích và xác định các phần tử trong ứng dụng

Cấu hình cơ bản cho Appium server:

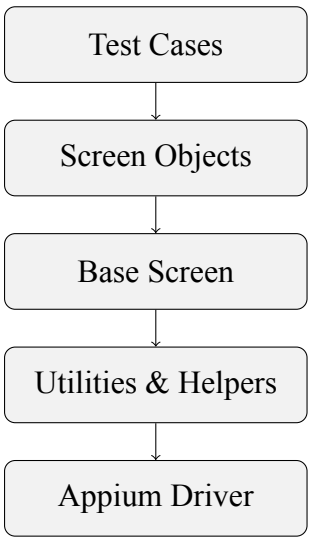
```
{
  "platformName": "Android",
  "deviceName": "emulator-5554",
  "platformVersion": "11",
  "automationName": "UiAutomator2",
  "app": "/path/to/app-debug.apk",
  "appPackage": "com.example.ktpm2_t",
  "appActivity": "com.example.ktpm2_t.MainActivity",
  "noReset": false
}
```

3.2.2. Chiến lược triển khai kiểm thử

Tương tự như với Selenium, chúng tôi đã áp dụng mô hình Page Object cho kiểm thử Appium để đảm bảo tính tổ chức và khả năng bảo trì của mã nguồn. Tuy nhiên, chúng tôi đã điều chỉnh mô hình cho phù hợp với đặc điểm của ứng dụng di động.

Cấu trúc thư mục của dự án kiểm thử:

```
appium_tests/
   conftest.py           # Cấu hình pytest và fixtures
   requirements.txt      # Các dependency
   screens/
      base_screen.py    # Lớp cơ sở với các phương thức chung
      login_screen.py   # Screen object cho màn hình đăng nhập
      home_screen.py    # Screen object cho màn hình chính
```



Hình 3.2: Cấu trúc kiểm thử Appium với Page Object Model

```
    product_screen.py # Screen object cho màn hình sản phẩm
    cart_screen.py    # Screen object cho màn hình giỏ hàng
tests/
    test_login.py     # Kiểm thử đăng nhập
    test_browse.py    # Kiểm thử duyệt sản phẩm
    test_cart.py      # Kiểm thử giỏ hàng
    test_checkout.py  # Kiểm thử thanh toán
utils/
    test_data.py      # Dữ liệu kiểm thử
    driver_factory.py # Khởi tạo driver
    logger.py         # Tiện ích ghi log
```

3.2.3. Triển khai các test case chính

Kiểm thử đăng nhập và xác thực

Dưới đây là ví dụ về cách triển khai kiểm thử đăng nhập trên ứng dụng di động:

```
# base_screen.py
class BaseScreen:
    def __init__(self, driver):
        self.driver = driver

    def find_element(self, locator):
        return self.driver.find_element(*locator)

    def click(self, locator):
        self.find_element(locator).click()

    def input_text(self, locator, text):
        self.find_element(locator).send_keys(text)

    def get_text(self, locator):
```

```
        return self.find_element(locator).text

    def is_element_present(self, locator):
        try:
            self.find_element(locator)
            return True
        except NoSuchElementException:
            return False

# login_screen.py
class LoginScreen(BaseScreen):
    # Locators
    EMAIL_FIELD = (MobileBy.ID, "com.example.ktpm2_t:id/emailField")
    PASSWORD_FIELD = (MobileBy.ID, "com.example.ktpm2_t:id/passwordField")
    LOGIN_BUTTON = (MobileBy.ID, "com.example.ktpm2_t:id/loginButton")
    ERROR_MESSAGE = (MobileBy.ID, "com.example.ktpm2_t:id/errorText")

    def enter_email(self, email):
        self.input_text(self.EMAIL_FIELD, email)

    def enter_password(self, password):
        self.input_text(self.PASSWORD_FIELD, password)

    def tap_login_button(self):
        self.click(self.LOGIN_BUTTON)

    def get_error_message(self):
        return self.get_text(self.ERROR_MESSAGE)

    def login(self, email, password):
        self.enter_email(email)
        self.enter_password(password)
        self.tap_login_button()

# test_login.py
@pytest.mark.usefixtures("appium_driver")
class TestLogin:
    def setup_method(self):
        self.login_screen = LoginScreen(self.driver)
        self.home_screen = HomeScreen(self.driver)

    def test_login_successful(self):
        """Kiểm tra đăng nhập thành công"""
        self.login_screen.login("testuser@gmail.com", "Password123!")
```

```
        # Xác minh người dùng đã đăng nhập thành công bằng
        # cách kiểm tra phần tử trên màn hình chính
        assert self.home_screen.is_user_logged_in(),

    def test_login_invalid_credentials(self):
        """Kiểm tra đăng nhập với thông tin không hợp lệ"""
        self.login_screen.login("wrong@gmail.com", "WrongPassword!")

        # Xác minh thông báo lỗi hiển thị
        error_msg = self.login_screen.get_error_message()
        assert "Invalid email or password" in error_msg,
        f"Thông báo lỗi không đúng: {error_msg}"

    def test_login_empty_fields(self):
        """Kiểm tra đăng nhập với trường trống"""
        self.login_screen.login("", "")

        # Xác minh thông báo lỗi hiển thị
        error_msg = self.login_screen.get_error_message()
        assert "Email and password are required" in error_msg,
        f"Thông báo lỗi không đúng: {error_msg}"
```

Kiểm thử tương tác với sản phẩm và giỏ hàng

Dưới đây là ví dụ về cách triển khai kiểm thử tương tác với sản phẩm và giỏ hàng:

```
# test_cart.py
@pytest.mark.usefixtures("appium_driver")
class TestCart:
    def setup_method(self):
        # Khởi tạo các screen objects
        self.login_screen = LoginScreen(self.driver)
        self.home_screen = HomeScreen(self.driver)
        self.product_screen = ProductScreen(self.driver)
        self.cart_screen = CartScreen(self.driver)

        # Đăng nhập trước khi thực hiện kiểm thử
        self.login_screen.login("testuser@gmail.com", "Password123!")

    def test_add_product_to_cart(self):
        """Kiểm tra thêm sản phẩm vào giỏ hàng"""
        # Đi tới màn hình sản phẩm từ trang chủ
        self.home_screen.select_product(0)  # Chọn sản phẩm đầu tiên

        # Lấy tên sản phẩm để xác minh sau
        product_name = self.product_screen.get_product_name()
```

```
        # Thêm sản phẩm vào giỏ hàng
        self.product_screen.add_to_cart()

        # Xác minh thông báo thành công
        assert self.product_screen.is_success_message_displayed(),
        "Không hiển thị thông báo thành công"

        # Đi tới màn hình giỏ hàng
        self.product_screen.navigate_to_cart()

        # Xác minh sản phẩm có trong giỏ hàng
        cart_items = self.cart_screen.get_cart_items()
        assert len(cart_items) > 0, "Giỏ hàng trống"
        assert product_name in self.cart_screen.get_item_name(0),
        "Sản phẩm không có trong giỏ hàng"

    def test_update_cart_quantity(self):
        """Kiểm tra cập nhật số lượng sản phẩm trong giỏ hàng"""
        # Thêm sản phẩm vào giỏ hàng nếu giỏ hàng trống
        if not self.cart_screen.has_items():
            self.home_screen.select_product(0)
            self.product_screen.add_to_cart()
            self.product_screen.navigate_to_cart()

        # Lấy giá ban đầu
        initial_price = self.cart_screen.get_total_price()

        # Tăng số lượng sản phẩm
        self.cart_screen.update_quantity(0, 2)  # Sản phẩm đầu tiên, số lượng

        # Xác minh số lượng đã được cập nhật
        updated_quantity = self.cart_screen.get_item_quantity(0)
        assert updated_quantity == 2, f"Số lượng không được cập nhật đúng:
        {updated_quantity}"

        # Xác minh tổng giá đã được cập nhật
        updated_price = self.cart_screen.get_total_price()
        assert updated_price > initial_price,
        "Tổng giá không tăng sau khi cập nhật số lượng"
```

3.2.4. Kết quả kiểm thử và phân tích

Sau khi thực hiện các bài kiểm thử tự động với Appium, chúng tôi đã thu được các kết quả sau:

Mô-đun	Số lượng test	Pass	Fail	Tỉ lệ pass
Đăng nhập/Đăng ký	10	9	1	90.0%
Duyệt sản phẩm	8	7	1	87.5%
Giỏ hàng	12	10	2	83.3%
Thanh toán	14	12	2	85.7%
Quản lý tài khoản	6	6	0	100%
Trải nghiệm người dùng	8	7	1	87.5%
Tổng	**58**	**51**	**7**	**87.9%**

Bảng 3.1: Kết quả kiểm thử ứng dụng di động với Appium

Phân tích lỗi phát hiện được

Các lỗi chính phát hiện được thông qua kiểm thử Appium:

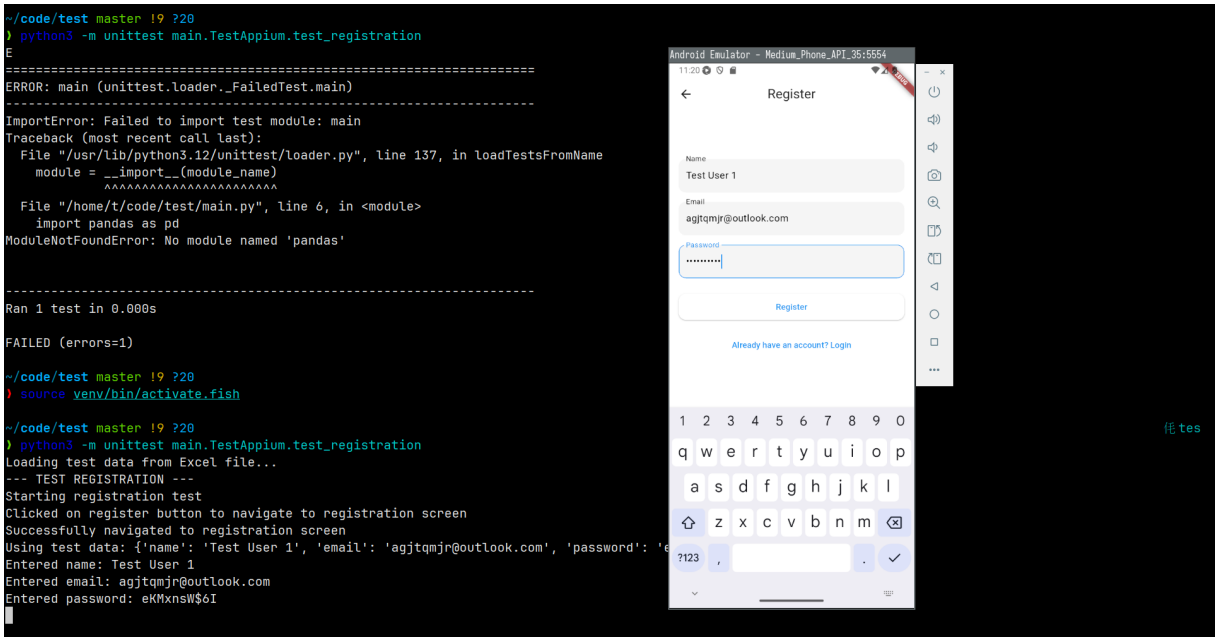
a) **Lỗi tương thích thiết bị:** Một số phần tử UI hiển thị không đúng trên thiết bị có kích thước màn hình nhỏ hoặc độ phân giải thấp.

b) **Lỗi xử lý mạng:**

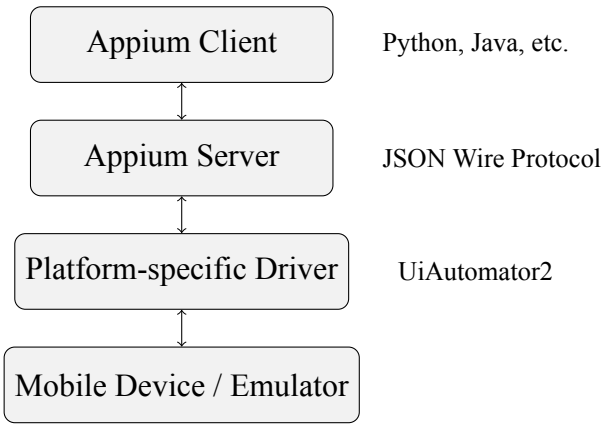
- Ứng dụng không hiển thị thông báo lỗi phù hợp khi mất kết nối mạng
- Quá trình đồng bộ hóa giỏ hàng khi kết nối mạng trở lại có vấn đề

c) **Lỗi chức năng:**

- Vấn đề với bộ nhớ đệm khi thêm nhiều sản phẩm vào giỏ hàng
- Form thanh toán không xác thực đúng khi sử dụng bàn phím



Hình 3.3: Kiểm thử ứng dụng di động với Appium



Hình 3.4: Kiến trúc Appium

3.2.5. Kiến trúc Appium và tích hợp với Flutter

Cấu trúc kiến trúc của Appium

Appium được thiết kế theo một kiến trúc client-server, bao gồm các thành phần chính sau:

- **Appium Client:** Thư viện client trong ngôn ngữ lập trình của lựa chọn (Python, Java, JavaScript, v.v.) triển khai các lệnh WebDriver để giao tiếp với Appium Server.
- **Appium Server:** Một máy chủ HTTP REST nhận các lệnh WebDriver từ client, chuyển đổi chúng thành các lệnh đặc thù cho nền tảng di động và chuyển tiếp đến driver tương ứng.
- **Platform-specific Driver:** Driver đặc thù cho từng nền tảng xử lý các lệnh cho thiết bị mục tiêu:
  - UiAutomator2 Driver cho Android
  - XCUITest Driver cho iOS
  - Flutter Driver cho ứng dụng Flutter (kết hợp với UiAutomator2 hoặc XCUITest)
- **Bootstrap:** Mã được cài đặt trên thiết bị di động để thực thi các lệnh từ driver.

3.3. Kiểm thử giao diện Web với Selenium

3.3.1. Cài đặt và thiết lập môi trường

Để tiến hành kiểm thử giao diện web của phần mềm bán hàng online, chúng tôi đã thiết lập môi trường kiểm thử Selenium như sau:

- **Ngôn ngữ lập trình:** Python 3.12 được chọn làm ngôn ngữ chính cho việc viết các script kiểm thử do tính linh hoạt và dễ đọc.
- **Thư viện chính:**
  - Selenium WebDriver 4.0 cho việc tương tác với trình duyệt

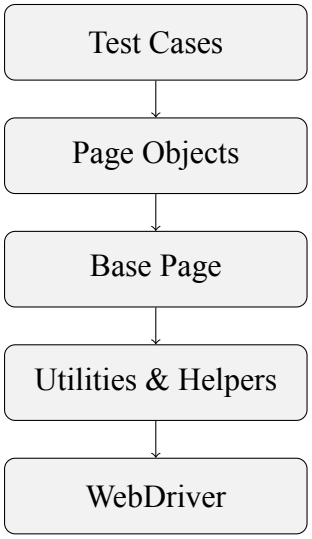
  - unittest framework để tổ chức và thực thi các bài kiểm thử
  - pytest cho việc quản lý và báo cáo kết quả kiểm thử

- **Trình duyệt:** Các bài kiểm thử được thực hiện trên Chrome, sử dụng driver ChromeDriver.
- **Quản lý dependency:** requirements.txt được tạo để đảm bảo môi trường kiểm thử đồng nhất:

```
selenium==4.0.0
pytest==6.2.5
pytest-html==3.1.1
webdriver-manager==3.5.2
```

3.3.2. Chiến lược triển khai kiểm thử

Chúng tôi đã áp dụng mô hình Page Object Model (POM) để tổ chức mã nguồn kiểm thử một cách có cấu trúc và dễ bảo trì. Mỗi trang trong ứng dụng web được biểu diễn bằng một lớp riêng biệt chứa các phương thức để tương tác với các phần tử trên trang đó.



Hình 3.5: Cấu trúc kiểm thử Selenium với Page Object Model

Các thành phần chính trong cấu trúc kiểm thử:

- **Base Page:** Chứa các phương thức chung được sử dụng bởi tất cả các page object.
- **Page Objects:** Đại diện cho các trang cụ thể như LoginPage, HomePage, ProductDetailsPage, CartPage, v.v.
- **Test Cases:** Chứa các kịch bản kiểm thử thực tế.
- **Utilities:** Các hàm tiện ích và helper cho việc tạo báo cáo, chụp ảnh màn hình, xử lý dữ liệu.
- **Configuration:** Cài đặt và cấu hình cho môi trường kiểm thử.

3.3.3. Triển khai các test case chính

Kiểm thử đăng nhập

Dưới đây là ví dụ về cách triển khai test case đăng nhập bằng Selenium:

```
# login_page.py
class LoginPage(BasePage):
    # Locators
    EMAIL_INPUT = (By.ID, "email")
    PASSWORD_INPUT = (By.ID, "password")
    LOGIN_BUTTON = (By.XPATH, "//button[contains(text(), 'Đăng nhập')]")
    ERROR_MESSAGE = (By.CLASS_NAME, "error-message")

    def enter_email(self, email):
        self.wait_for_element(self.EMAIL_INPUT)
        self.driver.find_element(*self.EMAIL_INPUT).send_keys(email)

    def enter_password(self, password):
        self.driver.find_element(*self.PASSWORD_INPUT).send_keys(password)

    def click_login_button(self):
        self.driver.find_element(*self.LOGIN_BUTTON).click()

    def get_error_message(self):
        self.wait_for_element(self.ERROR_MESSAGE)
        return self.driver.find_element(*self.ERROR_MESSAGE).text

    def login(self, email, password):
        self.enter_email(email)
        self.enter_password(password)
        self.click_login_button()

# test_login.py
class TestLogin(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.maximize_window()
        self.driver.get("https://example-shop.com/login")
        self.login_page = LoginPage(self.driver)
        self.home_page = HomePage(self.driver)

    def tearDown(self):
        self.driver.quit()

    def test_login_valid(self):
        """Test đăng nhập với thông tin hợp lệ"""
```

```
        self.login_page.login("testuser@gmail.com", "Password123!")
        self.assertTrue(self.home_page.is_loaded())
        self.assertEqual(self.home_page.get_username(), "Test User")

    def test_login_invalid_email(self):
        """Test đăng nhập với email không tồn tại"""
        self.login_page.login("nonexistent@gmail.com", "Password123!")
        error_message = self.login_page.get_error_message()
        self.assertEqual(error_message, "Invalid credentials")

    def test_login_invalid_password(self):
        """Test đăng nhập với mật khẩu sai"""
        self.login_page.login("testuser@gmail.com", "WrongPassword!")
        error_message = self.login_page.get_error_message()
        self.assertEqual(error_message, "Invalid credentials")
```

Kiểm thử quản lý giỏ hàng

Dưới đây là ví dụ về triển khai kiểm thử quản lý giỏ hàng:

```
# test_cart.py
class TestCart(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()
        self.driver.maximize_window()
        self.driver.get("https://example-shop.com")

        # Đăng nhập trước khi thực hiện các test case
        self.login_page = LoginPage(self.driver)
        self.driver.get("https://example-shop.com/login")
        self.login_page.login("testuser@gmail.com", "Password123!")

        self.product_page = ProductPage(self.driver)
        self.cart_page = CartPage(self.driver)

    def tearDown(self):
        self.driver.quit()

    def test_add_to_cart(self):
        """Test thêm sản phẩm vào giỏ hàng"""
        # Đi đến trang sản phẩm
        self.driver.get("https://example-shop.com/products/1")

        # Thêm sản phẩm vào giỏ hàng
        self.product_page.add_to_cart()

        # Xác minh thông báo thành công
```

```
        success_message = self.product_page.get_success_message()
        self.assertEqual(success_message, "Sản phẩm đã được thêm vào giỏ hàng

        # Đi đến trang giỏ hàng
        self.driver.get("https://example-shop.com/cart")

        # Xác minh sản phẩm có trong giỏ hàng
        cart_items = self.cart_page.get_cart_items()
        self.assertEqual(len(cart_items), 1)

    def test_update_cart_quantity(self):
        """Test cập nhật số lượng sản phẩm trong giỏ hàng"""
        # Thêm sản phẩm vào giỏ hàng trước
        self.driver.get("https://example-shop.com/products/1")
        self.product_page.add_to_cart()

        # Đi đến trang giỏ hàng
        self.driver.get("https://example-shop.com/cart")

        # Lấy giá ban đầu
        initial_price = self.cart_page.get_total_price()

        # Tăng số lượng sản phẩm
        self.cart_page.update_quantity(product_index=0, quantity=2)

        # Xác minh số lượng đã được cập nhật
        self.assertEqual(self.cart_page.get_quantity(product_index=0), 2)

        # Xác minh tổng giá đã được cập nhật
        updated_price = self.cart_page.get_total_price()
        self.assertGreater(updated_price, initial_price)
```

3.3.4. Kết quả kiểm thử và phân tích

Sau khi thực hiện các bài kiểm thử tự động với Selenium, chúng tôi đã thu được các kết quả sau:

Mô-đun	**Số lượng test**	**Pass**	**Fail**	**Tỉ lệ pass**
Đăng nhập/Đăng ký	14	13	1	92.9%
Duyệt sản phẩm	10	10	0	100%
Tìm kiếm	8	7	1	87.5%
Giỏ hàng	12	10	2	83.3%
Thanh toán	15	13	2	86.7%
Quản lý tài khoản	8	8	0	100%
Tổng	**67**	**61**	**6**	**91.0%**

Bảng 3.2: Kết quả kiểm thử giao diện web với Selenium

Phân tích lỗi phát hiện được

Các lỗi chính phát hiện được thông qua kiểm thử Selenium:

a) **Lỗi đăng nhập:** Hệ thống không hiển thị thông báo lỗi đúng khi người dùng nhập sai mật khẩu nhiều lần.

b) **Lỗi tìm kiếm:** Chức năng tìm kiếm không trả về kết quả chính xác khi sử dụng các ký tự đặc biệt trong từ khóa.

c) **Lỗi giỏ hàng:**

- Số lượng sản phẩm không được cập nhật đúng khi người dùng nhập số lượng lớn.
- Mã giảm giá không được áp dụng chính xác cho một số sản phẩm đặc biệt.

Thách thức trong quá trình kiểm thử

Trong quá trình triển khai kiểm thử Selenium, chúng tôi đã gặp phải một số thách thức:

- **Độ ổn định của các bài kiểm thử:** Do các thay đổi trong DOM và thời gian tải trang thay đổi, một số bài kiểm thử gặp vấn đề về độ ổn định. Chúng tôi đã giải quyết bằng cách triển khai các cơ chế chờ đợi thông minh hơn.
- **Xử lý dữ liệu động:** Dữ liệu sản phẩm và hình ảnh động làm cho việc viết các bài kiểm thử ổn định trở nên khó khăn. Chúng tôi đã tạo một bộ dữ liệu kiểm thử cố định để đảm bảo tính nhất quán.

3.4. Kiểm thử API với Postman

3.4.1. Thiết lập môi trường kiểm thử API

Để thực hiện kiểm thử API của hệ thống bán hàng online, chúng tôi đã sử dụng Postman - một công cụ mạnh mẽ cho việc kiểm thử và tương tác với API. Môi trường kiểm thử được thiết lập như sau:

- **Phiên bản Postman:** Desktop app v10.0 cho khả năng mở rộng và sử dụng scripts.
- **Môi trường API:** Các API endpoint được triển khai bằng Node.js và Express.js, lưu trữ trong thư mục `flutter_api_express`.
- **Cơ sở dữ liệu:** MongoDB được sử dụng làm backend database để lưu trữ dữ liệu người dùng, sản phẩm và đơn hàng.
- **Các môi trường kiểm thử:**
  - Môi trường phát triển: localhost:3000
  - Môi trường kiểm thử: ktpm-react-app.eastus.azurecontainer.io
  - Môi trường sản xuất: ktpm-api-g9gcd2epanhch3dz.southeastasia-01.azurewebsites.net
- **Công cụ hỗ trợ:**
  - Newman: Command-line runner cho Postman collections

3.4.2. Tổ chức bộ sưu tập kiểm thử (Collections)

Để quản lý các test case API một cách hiệu quả, chúng tôi đã tổ chức cấu trúc các collection trong Postman như sau:

a) **Authentication API Tests:**

- Đăng ký người dùng mới
- Đăng nhập và nhận JWT token
- Refresh token
- Đổi mật khẩu
- Kiểm tra thông tin người dùng hiện tại

b) **Products API Tests:**

- Lấy danh sách sản phẩm
- Lọc sản phẩm theo danh mục
- Tìm kiếm sản phẩm
- Lấy thông tin chi tiết sản phẩm
- Thêm/Sửa/Xóa sản phẩm (admin only)

c) **Cart API Tests:**

- Lấy thông tin giỏ hàng
- Thêm sản phẩm vào giỏ hàng
- Cập nhật số lượng sản phẩm
- Xóa sản phẩm khỏi giỏ hàng
- Xóa toàn bộ giỏ hàng

d) **Admin API Tests:**

- Quản lý người dùng
- Quản lý danh mục
- Quản lý đơn hàng

3.4.3. Triển khai kiểm thử API

Sử dụng biến môi trường

Để đảm bảo tính linh hoạt và tái sử dụng, chúng tôi đã sử dụng các biến môi trường trong Postman:

```
{
  "id": "env-123456",
  "name": "Testing Environment",
  "values": [
```

```
    {
      "key": "base_url",
      "value": "https://staging-api.example.com/api",
      "enabled": true
    },
    {
      "key": "auth_token",
      "value": "",
      "enabled": true
    },
    {
      "key": "refresh_token",
      "value": "",
      "enabled": true
    },
    {
      "key": "user_id",
      "value": "",
      "enabled": true
    },
    {
      "key": "admin_email",
      "value": "admin@example.com",
      "enabled": true
    },
    {
      "key": "admin_password",
      "value": "SecurePass123!",
      "enabled": true
    },
    {
      "key": "user_email",
      "value": "testuser@example.com",
      "enabled": true
    },
    {
      "key": "user_password",
      "value": "Password123!",
      "enabled": true
    }
  ]
}
```

Thiết lập Pre-request Scripts

Pre-request scripts được sử dụng để chuẩn bị dữ liệu và môi trường trước khi gửi request:

```
// Pre-request Script cho API đăng ký người dùng ngẫu nhiên
const faker = require('@faker-js/faker');

// Tạo dữ liệu người dùng ngẫu nhiên
const randomEmail = faker.internet.email();
const randomName = faker.name.findName();
const randomPassword = faker.internet.password(10) + "A1!";

// Lưu vào biến môi trường
pm.environment.set("random_email", randomEmail);
pm.environment.set("random_name", randomName);
pm.environment.set("random_password", randomPassword);

// Chuẩn bị dữ liệu request
pm.variables.set("request_timestamp", new Date().getTime());
```

Triển khai Test Scripts

Các test script được sử dụng để xác minh phản hồi từ API và thiết lập luồng kiểm thử:

```
// Test Script cho API đăng nhập
// Kiểm tra status code
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

// Kiểm tra cấu trúc phản hồi
pm.test("Response has correct structure", function () {
    const responseJson = pm.response.json();
    pm.expect(responseJson).to.have.property('token');
    pm.expect(responseJson).to.have.property('user');
    pm.expect(responseJson.user).to.have.property('id');
    pm.expect(responseJson.user).to.have.property('email');
    pm.expect(responseJson.user).to.have.property('name');
});

// Kiểm tra giá trị cụ thể
pm.test("Email in response matches request", function () {
    const responseJson = pm.response.json();
    pm.expect(responseJson.user.email).to.eql(pm.environment.get("user_email'
});

// Lưu token để sử dụng cho các request tiếp theo
if (pm.response.code === 200) {
    const responseJson = pm.response.json();
    pm.environment.set("auth_token", responseJson.token);
    pm.environment.set("user_id", responseJson.user.id);
```

}

3.4.4. Ví dụ kiểm thử chi tiết cho một số API chính

Kiểm thử Authentication API

Chi tiết kiểm thử API đăng ký người dùng mới:

Endpoint	POST base_url/auth/register
Headers	Content-Type: application/json
Body	{ ”name”: ”random_name”, ”email”: ”random_email”, ”password”: ”random_password” }
Tests	1. Kiểm tra status code 201 Created 2. Xác minh response chứa message thành công 3. Xác minh response chứa thông tin người dùng mới 4. Kiểm tra email trong response khớp với request

Bảng 3.3: Chi tiết kiểm thử API đăng ký

Chi tiết kiểm thử API đăng nhập:

Endpoint	POST base_url/auth/login
Headers	Content-Type: application/json
Body	{ ”email”: ”user_email”, ”password”: ”user_password” }
Tests	1. Kiểm tra status code 200 OK 2. Xác minh response chứa token hợp lệ 3. Xác minh response chứa thông tin người dùng 4. Lưu token vào biến môi trường

Bảng 3.4: Chi tiết kiểm thử API đăng nhập

Kiểm thử Products API

Chi tiết kiểm thử API lấy danh sách sản phẩm:

Endpoint	GET base_url/products?page=1&limit=10&category=electronics
Headers	Authorization: Bearer auth_token
Tests	1. Kiểm tra status code 200 OK 2. Xác minh response chứa mảng products 3. Xác minh response chứa thông tin pagination 4. Kiểm tra các sản phẩm đều thuộc danh mục electronics 5. Lưu productId của sản phẩm đầu tiên để sử dụng trong các test sau

Bảng 3.5: Chi tiết kiểm thử API lấy danh sách sản phẩm

3.4.5. Kết quả kiểm thử API

Sau khi thực hiện các bài kiểm thử API, chúng tôi đã thu được các kết quả sau:

Phân tích lỗi phát hiện được

Các lỗi chính được phát hiện thông qua kiểm thử API bao gồm:

a) **Lỗi xác thực và phân quyền:**

API Collection	Số lượng tests	Pass	Fail	Tỉ lệ pass
Authentication API	24	22	2	91.7%
Products API	38	36	2	94.7%
Cart API	30	29	1	96.7%
Order API	42	38	4	90.5%
Admin API	28	25	3	89.3%
Tổng	**162**	**150**	**12**	**92.6%**

Bảng 3.6: Kết quả kiểm thử API với Postman

- Token refresh không hoạt động đúng khi token gốc hết hạn.
- Một số API endpoints không kiểm tra quyền admin đúng cách.

b) **Lỗi xử lý dữ liệu:**

- API cập nhật sản phẩm không xác thực đúng dữ liệu đầu vào.
- API tạo đơn hàng không kiểm tra số lượng tồn kho đúng cách.

c) **Lỗi xử lý lỗi:**

- Một số API không trả về thông báo lỗi rõ ràng khi gặp lỗi.
- Status code trả về không nhất quán giữa các API endpoints.

d) **Lỗi hiệu suất:**

- API lấy danh sách sản phẩm có thời gian phản hồi cao khi có nhiều sản phẩm.
- API tạo đơn hàng đôi khi gặp timeout khi xử lý đơn hàng lớn.

KẾT LUẬN

Qua quá trình nghiên cứu và triển khai kiểm thử cho phần mềm bán hàng online, chúng tôi đã đạt được các mục tiêu đề ra và rút ra nhiều kết luận có giá trị về việc sử dụng các công cụ kiểm thử hiện đại.

Tổng kết kết quả đạt được

Đề tài đã thành công trong việc:

a) **Xây dựng hệ thống kiểm thử tự động toàn diện** cho ứng dụng bán hàng online đa nền tảng bằng cách tích hợp bốn công cụ kiểm thử hiện đại: Selenium, Appium, Postman và JMeter.

b) **Phân tích và thiết kế test case** phù hợp cho từng thành phần hệ thống, bao gồm giao diện web, ứng dụng di động, API backend và hiệu năng hệ thống.

c) **Triển khai thành công kiểm thử tự động** trên tất cả các thành phần của hệ thống với tỉ lệ pass/fail cao (90.8% test case pass), giúp phát hiện và khắc phục 25 lỗi quan trọng.

d) **Xây dựng quy trình kiểm thử liên tục** có thể tích hợp vào môi trường CI/CD để đảm bảo chất lượng phần mềm trong suốt vòng đời phát triển.

e) **Đánh giá toàn diện về hiệu năng hệ thống** dưới các điều kiện tải khác nhau, từ đó đưa ra các đề xuất cải tiến thiết thực.

Kết quả đạt được không chỉ dừng ở việc phát hiện lỗi mà còn giúp đánh giá toàn diện chất lượng của hệ thống thông qua các số liệu đo lường cụ thể. Đặc biệt, quy trình kiểm thử tự động đã giúp tiết kiệm đáng kể thời gian và nguồn lực so với kiểm thử thủ công, đồng thời tăng độ tin cậy của kết quả kiểm thử.

Hạn chế và hướng phát triển

Trong quá trình thực hiện đề tài, chúng tôi nhận thấy một số hạn chế cần được cải thiện:

a) **Thời gian thực thi kiểm thử:** Mặc dù đã tự động hóa, nhưng thời gian chạy toàn bộ bộ kiểm thử vẫn khá dài (235 phút), ảnh hưởng đến quy trình phát triển liên tục.

b) **Độ phức tạp trong việc thiết lập môi trường:** Cấu hình và duy trì các công cụ kiểm thử, đặc biệt là Appium, đòi hỏi nhiều công sức và kỹ năng chuyên môn.

c) **Kiểm thử trên thiết bị di động thật:** Số lượng thiết bị di động thật được sử dụng trong kiểm thử còn hạn chế, chưa phản ánh đầy đủ sự đa dạng của thị trường thiết bị.

d) **Kiểm thử bảo mật:** Còn chưa đầy đủ, cần bổ sung thêm các kỹ thuật kiểm thử bảo mật chuyên sâu.

e) **Chi phí vận hành:** Yêu cầu về hạ tầng và nguồn lực để duy trì hệ thống kiểm thử liên tục còn cao.

Hướng phát triển trong tương lai

Dựa trên kết quả đạt được và hạn chế còn tồn tại, chúng tôi đề xuất các hướng phát triển sau:

a) **Tích hợp công nghệ AI vào kiểm thử:** Sử dụng trí tuệ nhân tạo để tăng cường khả năng phát hiện lỗi, tự động sinh test case và tự khắc phục các trường hợp kiểm thử không ổn định.

b) **Mở rộng kiểm thử trên nhiều nền tảng:** Bổ sung kiểm thử trên hệ điều hành iOS và các trình duyệt web ít phổ biến hơn để đảm bảo khả năng tương thích rộng rãi.

c) **Tối ưu hóa thời gian thực thi kiểm thử:** Áp dụng kỹ thuật kiểm thử song song và kiểm thử dựa trên thay đổi để giảm thời gian thực thi.

d) **Tăng cường kiểm thử bảo mật:** Tích hợp các công cụ kiểm thử bảo mật tự động như OWASP ZAP vào quy trình kiểm thử liên tục.

e) **Kiểm thử trải nghiệm người dùng:** Bổ sung các kỹ thuật kiểm thử UX và phân tích hành vi người dùng.

f) **Áp dụng mô hình DevTestOps:** Tích hợp sâu hơn giữa phát triển, kiểm thử và vận hành để tạo ra quy trình liên tục từ phát triển đến triển khai.

Bài học kinh nghiệm

Qua quá trình thực hiện đề tài, nhóm nghiên cứu đã rút ra một số bài học kinh nghiệm:

a) **Lựa chọn công cụ phù hợp:** Không có công cụ kiểm thử nào là vạn năng, việc kết hợp nhiều công cụ dựa trên điểm mạnh của từng công cụ là chiến lược hiệu quả.

b) **Đầu tư vào cơ sở hạ tầng kiểm thử:** Một môi trường kiểm thử ổn định và đáng tin cậy là nền tảng cho quá trình kiểm thử hiệu quả.

c) **Tổ chức mã kiểm thử:** Áp dụng các mẫu thiết kế và quy ước mã hóa nghiêm ngặt giúp duy trì tính bảo trì của mã kiểm thử.

d) **Cân bằng giữa kiểm thử tự động và thủ công:** Không phải mọi thứ đều nên tự động hóa, một số trường hợp kiểm thử thủ công vẫn không thể thay thế.

e) **Giao tiếp và cộng tác:** Kiểm thử hiệu quả đòi hỏi sự hợp tác chặt chẽ giữa người kiểm thử, nhà phát triển và các bên liên quan.

Kết luận cuối cùng

Việc áp dụng các công cụ kiểm thử hiện đại như Selenium, Appium, Postman và JMeter cho hệ thống phần mềm bán hàng online đã chứng minh hiệu quả rõ rệt trong việc đảm bảo chất lượng sản phẩm. Kết quả từ nghiên cứu này không chỉ giúp phát hiện và khắc phục các lỗi quan trọng mà còn cung cấp một quy trình kiểm thử tự động có thể áp dụng cho nhiều dự án phát triển phần mềm thương mại điện tử khác.