**ĐẠI HỌC ĐÀ NẴNG**

**TRƯỜNG ĐẠI HỌC KINH TẾ**

🙣 🕮 🙡

****

**BÁO CÁO CHUYÊN ĐỀ**

**NGHIÊN CỨU SỬ DỤNG KIỂM THỬ PHẦN MỀM CHO WEBSITE SMARTOS**

SVTH: Ngô Thị Minh Hiếu

Lớp: 42K21

GVHD: ThS. Nguyễn Thành Thủy

***Đà Nẵng, Tháng 6/2020.***

NHẬN XÉT CỦA DOANH NGHIỆP THỰC TẬP

1. ***Về ý thức và tinh thần trách nhiệm của sinh viên khi thực tập tại doanh nghiệp:***

1. ***Về năng lực và khả năng thích nghi của sinh viên với môi trường doanh nghiệp:***

1. ***Kiến nghị của Doanh nghiệp đối với Nhà trường để cải thiện chất lượng đào tạo:***

LỜI CẢM ƠN

Nội dung cám ơn về Thầy Cô, Gia đình, người thân, bạn bè.

Canh dòng: Paragraph\Line spacing 1.5)

**....**

**Họ và tên sinh viên**

LỜI CAM ĐOAN

*Tôi xin cam đoan:*

1. *Nội dung trong luận văn này là do tôi thực hiện dưới sự hướng dẫn trực tiếp của Thầy/Cô XXX.*
2. *Các tham khảo dùng trong luận văn đều được trích dẫn rõ ràng tên tác giả, tên công trình, thời gian, địa điểm công bố.*
3. *Nếu có những sao chép không hợp lệ, vi phạm, tôi xin chịu hoàn toàn trách nhiệm.*

*Sinh viên thực hiện*

**Họ và Tên**

DANH SÁCH HÌNH ẢNH

>> Canh dòng: 1.5 line

DANH SÁCH TỪ VIẾT TẮT

|  |  |
| --- | --- |
| **Từ viết tắt** | **Diễn giải** |
| QA | Quality Assurance |
| NPM |  |
|  |  |
|  |  |
|  |  |

Mục lục

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT 4](#_Toc42269503)

[1.1 TỔNG QUAN VỀ KIỂM THỬ 4](#_Toc42269504)

[1.1.1 Kiểm thử thủ công 4](#_Toc42269505)

[1.1.2 Kiểm thử tự động 5](#_Toc42269506)

[1.1.3 Mô hình Agile trong kiểm thử tự động 8](#_Toc42269507)

[1.1.4 Sự khác biệt giữa kiểm thử tự động và kiểm thử thủ công 10](#_Toc42269508)

[1.2 CÁC PHƯƠNG PHÁP KIỂM THỬ 14](#_Toc42269509)

[1.2.1 Kiểm thử hộp đen 14](#_Toc42269510)

[1.2.2 Kiểm thử hộp trắng 15](#_Toc42269511)

[1.2.3 Kiểm thử hộp xám. 16](#_Toc42269512)

[1.3 CÁC LOẠI KIỂM THỬ PHẦN MỀM 19](#_Toc42269513)

[1.3.1 Phân tích giá trị biên 19](#_Toc42269514)

[1.3.2 Bảng quyết định 20](#_Toc42269515)

[1.3.3 Đoán lỗi 21](#_Toc42269516)

[1.4 CÁC CHIẾN LƯỢC KIỂM THỬ PHẦN MỀM 22](#_Toc42269517)

[1.4.1 Kiểm thử dựa trên yêu cầu (requirements based test) 22](#_Toc42269518)

[1.4.2 Smoke test / Build verification test 22](#_Toc42269519)

[1.4.3 User acceptance test 23](#_Toc42269520)

[1.4.4 Regression test 23](#_Toc42269521)

[1.5. GIỚI THIỆU VỀ WEB SMARTOS 24](#_Toc42269522)

[1.5.1. Sơ lược về Web Smartos 24](#_Toc42269523)

[1.5.2. Chức năng chính của Web SmartOs 24](#_Toc42269524)

[CHƯƠNG 2: NGHIÊN CỨU PHẦN MỀM KIỂM THỬ TỰ ĐỘNG CYPRESS 26](#_Toc42269525)

[2.1. CYPRESS TRONG KIỂM THỬ TỰ ĐỘNG 26](#_Toc42269526)

[2.1.1. Định nghĩa 26](#_Toc42269527)

[2.1.2. Tại sao lại sử dụng Cypress 26](#_Toc42269528)

[2.1.3. Cài đặt Cypress Cypress 28](#_Toc42269529)

[2.2 CUCUMBER 31](#_Toc42269530)

[2.2.1 Định nghĩa về Cucumber 31](#_Toc42269531)

[2.2.2 Chu trình kiểm thử với Cucumber 32](#_Toc42269532)

[2.2.3 Ngôn ngữ Gherkin 32](#_Toc42269533)

[2.3 THIẾT LẬP MÔI TRƯỜNG 37](#_Toc42269534)

[CHƯƠNG 3: ỨNG DỤNG KIỂM THỬ TRÊN WEBSITE SMARTOS 38](#_Toc42269535)

[3.1. GIỚI THIỆU TỔNG QUAN VỀ CÔNG TY, SẢN PHẨM, MÔI TRƯỜNG KIỂM THỬ CỦA CÔNG CTY 38](#_Toc42269536)

[3.2. SOFTWARE TEST PLAN SMARTOS: 38](#_Toc42269537)

[3.3. THIẾT KẾ TRƯỜNG HỢP KIỂM THỬ TỰ DỘNG CHO CÁC CHỨC NĂNG 39](#_Toc42269538)

[3.3.1. Màn hình đăng nhập 39](#_Toc42269539)

[3.3.2 Màn hình income: 52](#_Toc42269540)

[3.3.3 Màn hình đăng xuất: 57](#_Toc42269541)

[3.4 CẤU TRÚC THƯ MỤC 63](#_Toc42269542)

[3.5 TÍCH HỢP GHERKIN 64](#_Toc42269543)

[3.6. REPORT KẾT QUẢ CHẠY TEST 64](#_Toc42269544)

[3.7 NHỮNG KHÓ KHĂN KHI TRIỂN KHAI HỆ THỐNG TRONG CÔNG TY 64](#_Toc42269545)

[3.8 HƯỚNG PHÁT TRIỂN TIẾP THEO CỦA FRAMEWORK 64](#_Toc42269546)

MỞ ĐẦU

**1.**  **Tổng quan về đề tài**

Ngày nay, tự động hóa được ứng dụng ở rất nhiều lĩnh vực, mục đích thường rất đa dạng và tùy theo nhu cầu đặc thù của từng lĩnh vực, tuy nhiên điểm chung nhất vẫn là giảm nhân lực, thời gian và sai sót. Ngành công nghệ thông tin mà cụ thể là phát triển phần mềm cũng không ngoại lệ. Như chúng ta biết, để tạo ra sản phẩm công nghệ thông tin hay phần mềm có chất lượng thì hoạt động kiểm thử phần mềm đóng vai trò rất quan trọng, trong khi đó hoạt động này lại tiêu tốn và chiếm tỷ trọng khá lớn công sức và thời gian trong một dự án. Do vậy, nhu cầu tự động hoá quy trình kiểm thử phần mềm cũng được đặt ra. Qua thực tế cho thấy, việc áp dụng kiểm thử tự động hợp lý sẽ mang lại thành công cho hoạt động kiểm thử phần mềm. Kiểm thử tự động giúp giảm bớt công sức thực hiện, tăng độ tin cậy, giảm sự nhàm chán và rèn luyện kỹ năng lập trình cho cán bộ kiểm thử. Đó là lý do em chọn đề tài “Nghiên cứu và ứng dụng công cụ kiểm thử tự động trong Kiểm thử phần mềm” làm báo cáo chuyên đề tốt nghiệp.

**2. Mục đích và ý nghĩa của đề tài**

***2.1. Mục đích***

Đề tài tìm hiểu cơ sở lý thuyết về kiểm thử cũng như cách triển khai công cụ kiểm thử phần mềm tự động để giảm chi phí nhân lực và đảm bảo chất lượng phần mềm hơn so với công việc kiểm thử bằng tay.

***2.2. Ý nghĩa***

**3. Phương pháp thực hiện**

**3. Bố cục của báo cáo**

Đồ án bao gồm các nội dung sau:

Mở đầu.

Chương 1: Trình bày cơ sở lý thuyết.

Chương 2: Trình bày nghiên cứu phần mềm kiểm thử tự động cypress.

Chương 3: Trình bày ứng dụng kiểm thử trên website smartos.

Kết luận và hướng phát triển.

# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

## TỔNG QUAN VỀ KIỂM THỬ

### Kiểm thử thủ công

#### Khái niệm:

* Kiểm thử thủ công là kiểm thử phần mềm trong đó các kiểm thử được thực hiện thủ công bởi QA Analyst. Kiểm thử thủ công được thực hiện để tìm ra các lỗi trong phần mềm đang được phát triển.
* Trong kiểm thử thủ công, tester sẽ kiểm thử tất cả các tính năng thiết yếu của ứng dụng hoặc phần mềm. Tester thực hiện các test cases và tạo báo cáo kiểm thử mà không cần sự trợ giúp của bất kỳ công cụ kiểm thử phần mềm tự động nào.
* Đây là một phương pháp kiểm thử cổ điển, giúp tìm lỗi trong các hệ thống phần mềm, thường được tiến hành bởi một tester có kinh nghiệm và thực hiện theo quy trình kiểm thử phần mềm.

#### Ưu, nhược điểm:

* **Ưu điểm:**
* Nhận phản hồi trực quan nhanh và chính xác.
* Ít tốn kém hơn vì không cần phải chi ngân sách cho các công cụ và quy trình tự động hóa.
* Sự phán đoán và trực giác của con người luôn có lợi cho yếu tố thủ công.
* Trong khi kiểm thử một thay đổi nhỏ, một kiểm thử tự động hóa sẽ yêu cầu coding có thể tốn thời gian. Trong khi kiểm thử thủ công lại không tốn thời gian.
* **Nhược điểm:**
* Phương pháp kiểm thử thủ công ít đáng tin cậy hơn vì được thực thi bởi con người. Do đó, dễ mắc sai lầm và không tìm thấy lỗi.
* Quá trình kiểm thử thủ công không thể được ghi lại, do đó không thể sử dụng lại.
* Một số phần nhất định khó thực hiện thủ công, có thể cần thêm thời gian.

### Kiểm thử tự động

#### 1.1.2.1) Khái niệm:

Kiểm thử tự động là thực hiện kiểm thử phần mềm bằng một chương trình đặt biệt với rất ít hoặc không có sự tương tác của con người, giúp cho người thực hiện kiểm thử phần mềm không phải lặp đi lặp lại các bước nhàm chán. Công cụ kiểm thử tự động có thể lấy dữ liệu từ tập tin bên ngoài nhập vào ứng dụng, so sánh kết quả mong đợi với kết quả thực tế và xuất ra báo cáo kết quả kiểm thử. Kiểm thử tự động là quá trình thực hiện tự động các bước trong một kịch bản kiểm thử nhằm rút ngắn thời gian kiểm thử, giảm công sức và chi phí, tăng độ tin cậy và tăng tính hiệu quả của quá trình kiểm thử phần mềm.

#### 1.1.2.2) Mục đích của kiểm thử tự động

Kiểm thử tự động bằng một công cụ nhằm rút ngắn thời gian kiểm thử. Mục đích của kiểm thử tự động là giảm thiểu thời gian, công sức và kinh phí, tăng độ tin cậy, tăng tính hiệu quả và giảm sự nhàm chán cho người kiểm thử trong quá trình kiểm thử sản phẩm phần mềm. Kiểm thử tự động sẽ được sử dụng khi dự án không đủ tài nguyên (thời gian, nhân lực và chi phí), phải thực hiện kiểm thử hồi quy khi sản phẩm được sửa đổi hoặc nâng cấp và cần kiểm thử lại các tính năng đã thực hiện tốt trước đó, kiểm tra khả năng vận hành của sản phẩm trong các môi trường đặc biệt (đo tốc độ xử lý trung bình ứng với mỗi yêu cầu, xác định khả năng chịu tải tối đa, xác định cấu hình tối thiểu để thực thi hệ thống, kiểm tra các cơ chế an ninh và an toàn, ...).

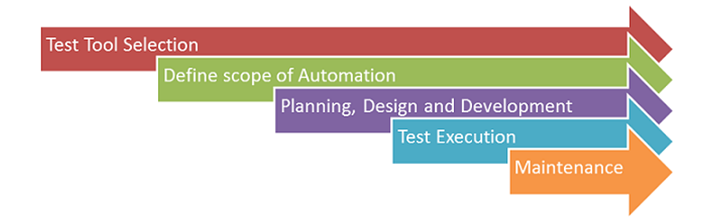
#### 1.1.2.3) Ưu nhược điểm của kiểm thử tự động

* Ưu điểm: Kiểm thử tự động giúp bạn tìm thấy nhiều lỗi hơn so với tester.
* Hầu hết các phần của quy trình kiểm thử là tự động, bạn có thể có một quy trình nhanh chóng và hiệu quả.
* Quá trình tự động có thể được ghi lại. Điều này cho phép bạn sử dụng lại và thực hiện cùng loại hoạt động kiểm thử.
* Kiểm thử tự động được thực hiện bằng các công cụ phần mềm, do đó, có thể hoạt động liên tục mà không biết mệt mỏi.
* Có thể dễ dàng tăng hiệu suất, cung cấp kết quả kiểm thử nhanh và chính xác.
* Kiểm thử tự động hỗ trợ các ứng dụng khác nhau.
* Phạm vi kiểm thử có thể được tăng lên vì công cụ kiểm thử tự động hóa không bỏ qua các đơn vị kiểm thử, ngay cả đơn vị nhỏ nhất.
* Nhược điểm:
* Không có yếu tố con người, thật khó để hiểu sâu hơn về các khía cạnh trực quan của giao diện người dùng như màu sắc, phông chữ, kích thước, độ tương phản hoặc kích thước nút bấm, layout...
* Các công cụ để chạy kiểm thử tự động có thể tốn kém, có thể làm tăng chi phí kiểm thử của dự án.
* Công cụ kiểm thử tự động chưa phải là bằng chứng đầy đủ. Mỗi công cụ tự động hóa đều có những hạn chế của chúng, làm giảm phạm vi tự động hóa.
* Gỡ lỗi test scripts là một vấn đề lớn trong kiểm thử tự động. Kiểm thử bảo trì sẽ tốn kém.

#### 1.1.2.4) Khi nào cần kiểm thử tự động?

* Trường hợp kiểm thử cần thực hiện nhiều lần, thường xuyên phải thực hiện regression test, một số lượng test data lớn cần hoàn thành trong một thời gian ngắn.
* Kiểm thử cần thực hiện ở môi trường khác nhau.
* Áp dụng với những project ổn định, đặc điểm kĩ thuật được xác định trước, chức năng không thay đổi trong tương lai.
* Kiểm thử hoạt động cơ bản mà phải thực hiện lặp lại với lượng test data lớn.
* Kiểm tra nhiều màn hình trong thời gian ngắn, liên tục.
* Thực thi performance test hoặc load test thì kiểm thử tự động gần như là lựa chọn duy nhất.

#### 1.1.2.5) Quy trình kiểm thử tự động:



* Lựa chọn công cụ kiểm thử

Lựa chọn Công cụ kiểm thử phần lớn phụ thuộc vào công nghệ mà Ứng dụng Kiểm thử được xây dựng. Chẳng hạn, QTP không hỗ trợ Informatica. Vì vậy, QTP không thể được sử dụng để kiểm thử các ứng dụng của Informatica.

* Xác định phạm vi tự động

Phạm vi tự động là phạm vi Ứng dụng được Kiểm thử tự động hóa. Các điểm sau giúp xác định phạm vi tự động:

* Các tính năng quan trọng.
* Kịch bản có lượng dữ liệu lớn.
* Các chức năng chung trên ứng dụng.
* Tính khả thi về mặt kỹ thuật.
* Mức độ các thành phần được sử dụng lại.
* Độ phức tạp của test cases.
* Khả năng sử dụng các test cases tương tự để kiểm thử trên nhiều trình duyệt.
* Lập kế hoạch, thiết kế và phát triển: Tester tạo một chiến lược & kế hoạch kiểm thử tự động, chi tiết như sau:
* Công cụ kiểm thử tự động được chọn.
* Thiết kế Framework và các tính năng của Framework.
* Các mục trong phạm vi và ngoài phạm vi tự động hóa.
* Chuẩn bị kiểm thử tự động hóa.
* Lịch trình và mốc thời gian thực thi kịch bản kiểm thử.
* Bàn giao sản phẩm được kiểm thử tự động.
* Thực thi kiểm thử

Kịch bản kiểm thử tự động hóa được thực thi trong giai đoạn này. Các Scripts cần dữ liệu kiểm thử đầu vào trước khi được thiết lập để chạy. Sau khi thực hiện sẽ trả về các báo cáo kiểm thử chi tiết.

Việc thực thi có thể được thực hiện bằng cách sử dụng công cụ tự động trực tiếp hoặc thông qua công cụ Quản lý kiểm thử sẽ gọi công cụ tự động hóa.

* Bảo trì

Khi các chức năng mới được thêm vào Hệ thống được kiểm thử với các chu kỳ liên tiếp, Scripts tự động hóa cũng cần được thêm, xem xét và bảo trì cho mỗi chu kỳ phát hành. Bảo trì trở nên cần thiết để cải thiện hiệu quả của Scripts trong kiểm thử tự động hóa.

### **Mô hình Agile trong kiểm thử tự động**

#### 1.1.3.1) Khái niệm:

Mô hình phát triển linh hoạt - Agile là một loại mô hình gia tăng, phát triển dựa trên quy trình phát triển lặp.

#### 1.1.3.2) Đặc điểm

* Mỗi dự án được chia thành nhiều mảng nhỏ để dễ sử dụng và thay đổi khi khách hàng yêu cầu thay đổi.
* Từng phần nhỏ của dự án sẽ được test ngay trong quá trình làm dự án.
* Yêu cầu gặp mặt trao đổi thường xuyên vì trong Agile tại mỗi thời điểm cả nhóm phải cùng tập trung phát triển một mảng của dự án.

#### 1.1.3.3) Các nguyên tắc trong Agile

* Ưu tiên cao nhất của chúng tôi là thỏa mãn khách hàng thông qua việc chuyển giao sớm và liên tục các phần mềm có giá trị.
* Chào đón việc thay đổi yêu cầu, thậm chí rất muộn trong quá trình phát triển. Các quy trình linh hoạt tận dụng sự thay đổi cho các lợi thế cạnh tranh của khách hàng.
* Thường xuyên chuyển giao phần mềm chạy tốt tới khách hàng, từ vài tuần đến vài tháng, ưu tiên cho các khoảng thời gian ngắn hơn.
* Nhà kinh doanh và nhà phát triển phải làm việc cùng nhau hàng ngày trong suốt dự án.
* Xây dựng các dự án xung quanh những cá nhân có động lực. Cung cấp cho họ môi trường và sự hỗ trợ cần thiết, và tin tưởng họ để hoàn thành công việc.
* Phương pháp hiệu quả nhất để truyền đạt thông tin tới nhóm phát triển và trong nội bộ nhóm phát triển là hội thoại trực tiếp.
* Phần mềm chạy tốt là thước đo chính của tiến độ.
* Các quy trình linh hoạt thúc đẩy phát triển bền vững. Các nhà tài trợ, nhà phát triển, và người dùng có thể duy trì một nhịp độ liên tục không giới hạn.
* Liên tục quan tâm đến các kĩ thuật và thiết kế tốt để gia tăng sự linh hoạt.
* Sự đơn giản – nghệ thuật tối đa hóa lượng công việc chưa xong – là căn bản.
* Các kiến trúc tốt nhất, yêu cầu tốt nhất, và thiết kế tốt nhất sẽ được làm ra bởi các nhóm tự tổ chức.
* Đội sản xuất sẽ thường xuyên suy nghĩ về việc làm sao để trở nên hiệu quả hơn, sau đó họ sẽ điều chỉnh và thay đổi các hành vi của mình cho phù hợp.

#### 1.1.3.4) Ưu và nhược điểm mô hình Agile:

* Ưu điểm
* Agile là sự lựa chọn rất tốt cho những dự án nhỏ bởi những dự án nhỏ thường có những yêu cầu không được xác định rõ ràng và có thể thay đổi thường xuyên.
* Với Agile khách hàng có thể được xem trước từng phần dự án trong suốt quá trình phát triển vì Agile phát triển phần mềm theo hướng tăng dần, có thể đưa cho khách hàng xem từng phần đã thực hiện hoàn thành. Từ đó có thể bám sát dự án và luôn sẵn sàng cho bất kỳ thay đổi nào từ phía khách hàng yêu cầu về dự án.
* Agile chia dự án thành những phần nhỏ và giao cho mỗi người, hàng ngày tất cả mọi người phải họp với nhau trong khoảng thời gian ngắn để thảo luận về tiến độ và giải quyết những vấn đề nảy sinh nếu có nhằm đảm bảo đúng quy trình phát triển dự án.
* Tỉ lệ thành công của các dự án sử dụng Agile thường cao hơn các quy trình khác.
* Nhược điểm:
* Thiếu sự nhấn mạnh về thiết kế và tài liệu cần thiết.
* Quy mô nhân lực thường giới hạn từ 7 đến 10 người, sẽ có trở ngại lớn nếu nguồn nhân lực yêu cầu vượt quá con số này ví dụ trong các cuộc họp trao đổi.
* Số lượng yêu cầu có thể nhiều và khó quản lý nếu như nó bao gồm nhiều khía cạnh khác nhau về dự án.
* Số lượng nhân lực càng tăng, chất lượng càng khó kiểm soát hơn. Việc kiểm tra mã thường xuyên và thiết lập các chỉ tiêu đánh giá năng lực của lập trình viên cho phép giảm thiểu nhược điểm này.

### 1.1.4 Sự khác biệt giữa kiểm thử tự động và kiểm thử thủ công

|  |  |  |
| --- | --- | --- |
| **Thông số** | **Kiểm thử tự động** | **Kiểm thử thủ công** |
| Định nghĩa | Kiểm thử tự động sử dụng các công cụ tự động để thực hiện các trường hợp kiểm thử. | Các trường hợp kiểm thử được thực hiện bởi tester trên phần mềm |
| Thời gian xử lý | Kiểm thử tự động nhanh hơn đáng kể so với phương pháp kiểm thử thủ công. | Các trường hợp kiểm thử được thực hiện bởi tester trên phần mềm |
| Kiểm thử ngẫu nhiên (Exploratory testing) | Tự động hóa không cho phép kiểm thử ngẫu nhiên | Kiểm thử ngẫu nhiên có thể thực hiện trong kiểm thử thủ công |
| Đầu tư ban đầu | Đầu tư ban đầu trong kiểm thử tự động cao hơn. Mặc dù ROI tốt hơn về lâu dài. | Đầu tư ban đầu trong kiểm thử thủ công là tương đối thấp hơn kiểm thử tự động. ROI thấp hơn so với kiểm thử tự động trong thời gian dài. |
| Độ tin cậy | Kiểm thử tự động là một phương pháp đáng tin cậy, vì được thực hiện bởi các công cụ và scripts nên chính xác và không gây nhàm chán | Kiểm thử thủ công có thể bị nhàm chán và dễ bị lỗi. |
| Thay đổi giao diện người dùng | Đối với ngay cả một thay đổi nhỏ trong giao diện người dùng của AUT, scripts kiểm thử tự động cần phải được sửa đổi để hoạt động như mong đợi | Những thay đổi nhỏ như thay đổi trong id, class, v.v. của một nút sẽ không cản trở việc thực hiện một quá trình kiểm thử thủ công. |
| Đầu tư | Đầu tư là cần thiết cho các công cụ kiểm thử cũng như các kỹ sư kiểm thử tự động hóa | Đầu tư là cần thiết cho nguồn nhân lực. |
| Hiệu quả chi phí | Không hiệu quả nếu kiểm thử hồi quy với số lượng nhỏ | Kết quả của kiểm thử thủ công thường được ghi lại trong Excel hoặc Word |
| Báo cáo Kiểm thử | Với kiểm thử tự động, tất cả các bên liên quan có thể đăng nhập vào hệ thống tự động và xem được kết quả kiểm thử | Kết quả của kiểm thử thủ công thường được ghi lại trong Excel hoặc Word |
| Giao diện đối với người dùng | Kiểm thử tự động không liên quan đến hành động của con người. Vì vậy, không phù hợp với người dùng và trải nghiệm tích cực khách hàng. | Phương pháp kiểm thử thủ công cho phép người dùng quan sát, có thể hữu ích để cung cấp hệ thống thân thiện với người dùng. |
| Kiểm thử hiệu suất | Các loại Kiểm thử hiệu suất như Load Testing, Stress Testing, Spike Testing, v.v. bắt buộc phải được kiểm thử bằng công cụ tự động. | Kiểm thử hiệu suất là không khả thi |
| Thực hiện kiểm thử tương đồng | Có thể được thực hiện tương đồng trên các nền tảng vận hành khác nhau và giảm thời gian thực hiện kiểm thử. | Kiểm thử thủ công có thể được thực hiện tương đồng nhưng sẽ cần tăng nguồn nhân lực |
| Kiểm thử hàng loạt | Có thể chạy hàng loạt các Script, có thể thực thi vào ban đêm. | Kiểm thử thủ công không thể thực hiện hàng loạt. |
| Kiến thức lập trình | Kiến thức lập trình là bắt buộc để thực hiện kiểm thử tự động. | Không cần kiến thức lập trình trong kiểm thử thủ công. |
| Tiếp cận lý tưởng | Kiểm thử tự động rất hữu ích khi thường xuyên thực hiện cùng một bộ test cases | Kiểm thử thủ công chứng minh hữu ích khi bộ test cases chỉ cần chạy một hoặc hai lần. |
| Kiểm thử xác minh build (Build Verification Testing - BVT) | Kiểm thử tự động rất hữu ích cho BVT. | Thực hiện BVT là rất khó khăn và tốn thời gian trong kiểm thử thủ công. |
| Thời hạn (Deadlines) | Các kiểm thử tự động không có rủi ro trễ Deadlines | Kiểm thử thủ công có nguy cơ trễ Deadlines cao. |
| Framework | Kiểm thử tự động hóa sử dụng các Framework như Data Drive, Keyword, Hybrid để tăng tốc quá trình tự động hóa. | Kiểm thử thủ công không sử dụng Framework nhưng có thể sử dụng guidelines, checklists, quy trình nghiêm ngặt để tạo ra một số test case nhất định. |
| Tài liệu | Kiểm thử tự động như một tài liệu để training về những test cases unit tự động. Một developer mới có thể tham khảo và nhanh chóng hiểu yêu cầu. | Các trường hợp kiểm thử thủ công không cung cấp giá trị training |
| Test Design | Kiểm thử đơn vị tự động bắt buộc xây dựng thiết kế dựa trên kiểm thử. | Kiểm thử đơn vị thủ công không bắt buộc thiết kế trong quá trình coding. |
| DevOps | Kiểm thử tự động giúp kiểm thử xác minh build và là một phần không thể thiếu của Chu kỳ DevOps. | Kiểm thử thủ công loại bỏ nguyên lý build tự động của DevOps. |
| Khi nào nên sử dụng? | Kiểm thử tự động phù hợp cho Regression Testing, Performance Testing, Load Testing hoặc các trường hợp kiểm thử chức năng có khả năng lặp lại cao. | Kiểm thử thủ công phù hợp cho Exploratory, Usability và kiểm thử Adhoc. Nó cũng nên được sử dụng khi AUT thay đổi thường xuyên. |

## CÁC PHƯƠNG PHÁP KIỂM THỬ

Trong lĩnh vực kiểm thử phần mềm có rất nhiều các phương pháp được áp dụng hiện nay. Trong bài viết này chúng ta sẽ cùng tìm hiểu 3 phương pháp cơ bản được áp dụng một cách phổ biến và rộng rãi nhất cùng với các ưu điểm và nhược điểm của nó. Đó là: kiểm thử hộp đen, kiểm thử hộp trắng và kiểm thử hộp xám.

### 1.2.1 Kiểm thử hộp đen

Kiểm thử hộp đen là một phương pháp kiểm thử mà các tester không cần quan tâm đến các hoạt động bên trong hệ thống chạy ra sao, không cần quan tâm đến các dòng lệnh bên trong hệ thống như thế nào mà chỉ cần tập trung vào các giá trị đầu vào và các giá trị đầu ra của hệ thống có đúng với kết quả mong đợi của các trường hợp kiểm thử không để từ đó đánh giá chất lượng hệ thống.

Chính vì cơ chế như vậy nên phương pháp này có các ưu nhược điểm như sau:

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhựợc điểm** |
| - Rất phù hợp và hiệu quả khi mà số lượng các dòng lệnh của hệ thống là lớn. | - Bị giới hạn ở độ bao phủ của các trường hợp kiểm thử. |
| - Không cần truy cập vào các dòng lệnh. | - Sẽ không hiệu quả bởi thực tế các tester bị giới hạn kiến thức về hệ thống. |
| - Phân biệt được rõ ràng quan điểm của người dùng với quan điểm của nhà phát triển. | - Độ bao phủ sẽ bị thiếu vì tester không kiểm tra được các đoạn lệnh của hệ thống hoặc tập trung vào các dòng lệnh dễ xảy ra lỗi. |
| - Không cần đòi hỏi những kiến thức về ngôn ngữ lập trình ở các tester để có thể kiểm thử hệ thống. | - Sẽ khó để có thể thiết kế đầy đủ các trường hợp kiểm thử. |

### 1.2.2 Kiểm thử hộp trắng

Kiểm thử hộp trắng là việc nghiên cứu cụ thể chi tiết từng luồng hoạt động cũng như các dòng lệnh bên trong hệ thống. Kiểm thử hộp trắng cũng được gọi dưới các cái tên khác như: Glass testing hay open-box testing. Kiểm thử hộp trắng đòi hỏi tester phải có kiến thức về ngôn ngữ lập trình. Tester sẽ cần phải nghiên cứu vào bên trong hệ thống cụ thể là các dòng lệnh để tìm hiểu chúng có chạy đúng hay không.

Dưới đây là các ưu nhược điểm của phương pháp này:

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| - Đối với những tester có kiến thức về ngôn ngữ lập trình sẽ rất dễ dàng để phát hiện ra những lỗi ở trong các dòng lệnh. | - Trên thực tế việc sử dụng các tester có kiến thức về ngôn ngữ lập trình sẽ làm gia tăng giá thành để phát triển phần mềm. |
| - Giúp tối ưu hóa các dòng lệnh của hệ thống. | - Đôi lúc sẽ là không khả thi khi kiểm tra chi tiết từng dòng lệnh để có thể từ đó phát hiện ra các lỗi tiềm ẩn của hệ thống, có rất nhiều các luồng không thể kiểm tra được. |
| - Các dòng lệnh không cần thiết hoặc các dòng lệnh có khả năng mang đến các lỗi tiềm ẩn sẽ bị loại bỏ. | - Rất khó để duy trì phương pháp này liên tục, cần phải có những tool chuyên biệt như tool về phân tích code hay tool về phát hiện lỗi và sửa lỗi. |
| - Các tester có kiến thức về ngôn ngữ lập trình sau khi đã thực hiện phương pháp này thì sẽ dễ dàng đạt được độ bao phủ lớn nhất khi thực hiện thiết kế các trường hợp kiểm thử sau này. |  |

### 1.2.3 Kiểm thử hộp xám.

Kiểm thử hộp xám là một phương pháp kiểm thử mà đòi hỏi tester phải có một lượng kiến thức nhất định về các luồng hoạt động ở bên trong hệ thống. Khác với kiểm thử hộp đen, phương pháp mà tester chỉ quan tâm duy nhất để việc kiểm thử thông qua giao diện người dùng, kiểm thử hộp xám đòi hỏi tester phải truy cập vào các tài liệu thiết kế hệ thống cũng như hệ thống cơ sở dữ liệu của hệ thống. Do đó mà tester có thể chuẩn bị tốt hơn những dữ liệu cho việc kiểm thử cũng như các trường hợp kiểm thử trong quá trình lên kế hoạch kiểm thử hệ thống.

|  |  |
| --- | --- |
| **Ưu điểm** | **Nhược điểm** |
| - Vì là sự kết hợp giữa kiểm thử hộp trắng và kiểm thử hộp đen nên có được ưu điểm của cả hai phương pháp này. | - Vì phương pháp này không dựa trên việc truy cập code của hệ thống nên sẽ không tránh được việc độ bao phủ của các trường hợp kiểm thử bị giới hạn. |
| - Các tester sử dụng phương pháp này không dựa vào các dòng lệnh của hệ thống mà chủ yếu dựa trên các tài liệu định nghĩa giao diện cũng như các tài liệu đặc tả chức năng. | - Khi sử dụng phương pháp này thì nhiều trường hợp kiểm thử có thể bị dư thừa nếu mà những nhà thiết kế phần mềm đã chạy các trường hợp kiểm thử này trước đó. |
| - Trong phương pháp này các tester có thể thiết kế nên những trường hợp kiểm thử đặc biệt xung quanh các giao thức kết nối và các loại dữ liệu khác nhau. | - Việc kiểm tra tất cả các luồng đầu vào của hệ thống là không thể thực hiện được vì bị giới hạn về mặt thời gian kiểm thử và sẽ dẫn đến có rất nhiều các luồng hoạt động của hệ thống không được kiểm tra. |
| - Việc kiểm thử được hoàn thành từ góc nhìn của người dùng chứ không phải từ nhà thiết kế. |  |

1. **Bảng so sánh giữa các phương pháp kiểm thử.**

|  |  |  |
| --- | --- | --- |
| **Kiểm thử hộp đen.** | **Kiểm thử hộp xám.** | **Kiểm thử hộp trắng** |
| Không cần quan tâm đến các luồng hoạt động trong hệ thống. | Cần có kiến thức nhất định về các luồng hoạt động bên trong hệ thống. | Cần nắm được toàn bộ các luồng hoạt động bên trong hệ thống. |
| Được biết đến với các tên gọi khác như: closed-box testing, data-driven testing hoặc functional testing. | Được biết đến với các tên gọi khác như: translucent testing. | Được biết đến với các tên gọi khác như: clear-box testing hoặc code-based testing. |
| Được thực hiện bởi người dùng cuối, kiểm thử viên và lập trình viên. | Được thực hiện bởi người dùng cuối, kiểm thử viên và lập trình viên. | Thường thì được hoàn thành bởi kiểm thử viên và lập trình viên. |
| Việc kiểm thử dựa trên kết quả mong muốn và kết quả thực tế mà hệ thống trả về. | Việc kiểm thử dựa trên các sơ đồ về cơ sở dữ liệu và sơ đồ về các luồng dữ liệu. | Dựa trên toàn bộ kiến thức về các luồng hoạt động bên trong hệ thống và các bộ dữ liệu kiểm thử phù hợp mà các kiểm thử viên tự thiết kế. |
| Vì chỉ quan tâm đến các giá trị đầu vào, kết quả đầu ra và kết quả mong đợi nên đây là phương pháp tốn ít thời gian nhất cũng như đô bao phủ các trường hợp không đầy đủ nhất. | Mức độ đầy đủ của các trường hợp kiểm thử ở mức vừa phải và mức độ tốn thời gian là vừa phải. | Đầy đủ nhất và tốn nhiều thời gian nhất. |
| Không thích hợp để kiểm tra các thuật toán trong hệ thống. | Không thích hợp để kiểm tra các thuật toán trong hệ thống. | Thích hợp để kiểm tra các thuật toán trong hệ thống. |
| Phương pháp này sẽ được hoàn thành bởi cơ chế phát hiện lỗi. | Các miền dữ liệu và các giới hạn có thể sẽ được test nếu các tester có kiến thức về nó. | Các miền dữ liệu và các giới hạn sẽ được test. |
| Không thích hợp để kiểm tra các thuật toán trong hệ thống. | Không thích hợp để kiểm tra các thuật toán trong hệ thống. | Thích hợp để kiểm tra các thuật toán trong hệ thống. |
| Phương pháp này sẽ được hoàn thành bởi cơ chế phát hiện lỗi. | Các miền dữ liệu và các giới hạn có thể sẽ được test nếu các tester có kiến thức về nó. | Các miền dữ liệu và các giới hạn sẽ được test. |

## CÁC LOẠI KIỂM THỬ PHẦN MỀM

### 1.3.1 Phân tích giá trị biên

Phân tích giá trị biên dựa trên việc kiểm thử tại các ranh giới giữa các phân vùng, bao gồm các ranh giới tối đa, tối thiểu, bên trong hoặc bên ngoài, các giá trị điển hình và giá trị lỗi.

Chúng ta thường thấy rằng một số lượng lớn lỗi xảy ra tại các ranh giới của các giá trị đầu vào được xác định thay vì các giá trị giữa, còn được gọi là các giá trị biên. Từ đó đưa ra lựa chọn các test cases thực hiện giá trị đầu vào các giá trị biên.

Kỹ thuật thiết kế test cases này bổ sung cho phân vùng tương đương. Kỹ thuật kiểm thử phần mềm này dựa trên nguyên tắc: Nếu một hệ thống hoạt động tốt với các giá trị biên thì nó sẽ hoạt động tốt cho tất cả các giá trị nằm giữa hai giá trị biên.

**Cách thực hiện kỹ thuật phân tích giá trị biên:**

* Nếu một điều kiện đầu vào bị giới hạn giữa các giá trị x và y, thì các test cases phải được thiết kế với các giá trị x và y cũng như các giá trị ở trên và dưới x và y.
* Nếu một điều kiện đầu vào là một số lượng lớn các giá trị, test cases cần được thiết kế với dữ liệu là các số min và max, các giá trị trên và dưới giá trị min và max cũng được kiểm thử.
* Áp dụng 2 cách thực hiện trên cho các điều kiện đầu ra, đầu ra mong đợi phản ánh giá trị min và giá trị max cũng như các giá trị dưới hoặc trên.
* Phân vùng tương đương

Phân vùng tương đương cho phép chia tập hợp điều kiện kiểm thử thành một phân vùng được coi là giống nhau.

Phương pháp kiểm thử phần mềm này chia miền đầu vào của chương trình thành các lớp dữ liệu mà từ đó các test cases sẽ được thiết kế.

Test cases của một giá trị đại diện thuộc mỗi lớp bằng với kiểm thử của bất kỳ giá trị nào khác của cùng một lớp đó, cho phép xác định các lớp tương đương hợp lệ cũng như không hợp lệ.

### 1.3.2 Bảng quyết định

Bảng quyết định còn được gọi là bảng Nguyên nhân – Ảnh hưởng (Cause-Effect). Kỹ thuật kiểm thử phần mềm này được sử dụng cho các chức năng đáp ứng sự kết hợp của các yếu tố đầu vào các biến cố. Ví dụ: Nút **Submit**phải được enable nếu người dùng đã nhập tất cả các trường bắt buộc.

Nhiệm vụ đầu tiên là xác định các chức năng trong đó đầu ra phụ thuộc vào sự kết hợp của các dữ liệu đầu vào. Nếu có tập hợp kết hợp đầu vào lớn, thì hãy chia nó thành các tập hợp nhỏ hơn hữu ích cho việc quản lý bảng quyết định.

Đối với mọi chức năng, cần tạo một bảng và liệt kê tất cả các loại kết hợp đầu vào và đầu ra tương ứng. Điều này giúp xác định các điều kiện bị tester bỏ qua.

**Các bước để tạo bảng quyết định:**

* Nhập đầu vào theo hàng.
* Nhập tất cả các quy tắc trong cột.
* Điền vào bảng với sự kết hợp của đầu vào.
* Trong hàng cuối cùng, ghi chú đầu ra so với kết hợp đầu vào.

### 1.3.3 Đoán lỗi

Đoán lỗi là một kỹ thuật kiểm thử phần mềm dựa trên việc đoán lỗi có thể chiếm ưu thế trong code. Đây là một kỹ thuật dựa trên kinh nghiệm, trong đó nhà phân tích kiểm thử sử dụng kinh nghiệm của mình để đoán phần có vấn đề hoặc có lỗi của ứng dụng kiểm thử.

Kỹ thuật xác định danh sách các lỗi có thể xảy ra hoặc các tình huống dễ xảy ra lỗi. Sau đó, người kiểm thử viết test cases để tìm kiếm những lỗi đó. Để thiết kế các test cases dựa trên kỹ thuật kiểm thử phần mềm này, nhà phân tích có thể sử dụng các kinh nghiệm trong quá khứ để xác định các điều kiện.

**Cách đoán lỗi:**

* Tester nên sử dụng kinh nghiệm trước đây để kiểm thử các ứng dụng tương tự.
* Hiểu biết về hệ thống đang kiểm thử.
* Kiến thức về các lỗi thực hiện điển hình.
* Nhớ những chức năng phức tạp trước đây.
* Đánh giá lịch sử dữ liệu và kết quả kiểm thử.

## CÁC CHIẾN LƯỢC KIỂM THỬ PHẦN MỀM

### 1.4.1 Kiểm thử dựa trên yêu cầu (requirements based test)

Sử dụng các đặc tả yêu cầu của hệ thống làm cơ sở để design test. Một cách tốt để bắt đầu là sử dụng bảng nội dung của đặc tả yêu cầu như một danh sách các mục kiểm thử và không kiểm thử. Chúng ta nên xét độ ưu tiên của yêu cầu dựa trên các tiêu chí rủi ro và sử dụng độ ưu tiên để kiểm thử. Điều này sẽ đảm bảo những phần quan trọng nhất sẽ được kiểm thử.

### 1.4.2 Smoke test / Build verification test

Thuật ngữ smoke test được bắt đầu trong ngành điện tử, phần cứng. Đây là hoạt động kiểm thử đầu tiên cần phải thực hiện khi kỹ sư bật công tắc hay cắm nguồn điện để xem….có “khói bốc lên cao hay không”. Nếu không có khói (nghĩa là sản phẩm ok để test tiếp), nếu có khói (sản phẩm đã chết) thì phải sửa ngay tức khắc.

Tương tự, trong phát triển phần mềm thì smoke test là loại test nhằm đánh giá xem sản phẩm được xây dựng bởi dev có lỗi gì nghiêm trọng hay không để có thể tiếp tục các hoạt động khác.

Loại kiểm thử này chỉ nhằm mục đích đánh giá sơ khởi xem build nhận được có ok để test tiếp hay không. Lí do ta phải sử dụng smoke test là việc phát hiện sớm những lỗi quan trọng sẽ giúp tránh lãng phí khi chúng ta dành thời gian cho những hoạt đông kiểm thử khác.

Smoke test (một số nơi có thể gọi là sanity test, build validation test, build acceptance test) thường là một bộ kiểm thử đơn giản và chứa một số các test case cơ bản đi qua những tính năng quan trọng nhất của sản phẩm. Khi bạn làm việc với sản phẩm bạn sẽ phải biết được những tính năng nào là quan trọng nhất (nghĩa là những tính năng này là giá trị gốc, là sống còn đối với sản phẩm hoặc công ty). Nếu bạn vẫn chưa biết thì tốt nhất là tìm hiểu hoặc hỏi ngay.

Khi nào nên sử dụng smoke test?

Khi dev giao build cho đội test thì việc trước tiên là thực thi bộ smoke test này. Bộ smoke test thường nhỏ nên bạn sẽ thường mất khoảng 1-2 giờ để thực thi. Nếu build fail, bạn báo ngay cho sếp, developer hoặc các bên liên quan để đánh giá tình hình. Trả build về và không nên test tiếp những tính năng khác.

### 1.4.3 User acceptance test

- User Acceptance Testing (UAT) là gì?

User Acceptance Testing là quá trình xác nhận rằng một giải pháp / phần mềm đã tạo ra có hoạt động phù hợp với người dùng cuối hay không.

* Khi nào và tại sao cần User Acceptance Testing?

Acceptance test (kiểm thử chấp nhận) có thể được hiểu là một cách để kiểm tra xem “hợp đồng” đã được xác định trước đó giữa các nhà phát triển và khách hàng có đi đúng hướng hay không.

Chạy các thử nghiệm chấp nhận cũng để đảm bảo rằng không có thay đổi yêu cầu nào xảy ra trong thời gian đó và mọi thứ đều phải đúng để làm hài lòng khách hàng.

* Acceptance test khá hữu dụng vì:
* Chúng nắm bắt các yêu cầu của người dùng bằng cách kiểm chứng trực tiếp.
* Chúng tìm ra những vấn đề mà ở unit hay integration test có thể đã để lọt.
* Và chúng cung cấp 1 cái nhìn tổng quan về kết quả hệ thống đạt được.

Khi nhìn vào quá trình phát triển phần mềm, chúng ta có thể thấy rằng UAT được sử dụng để xác định và xác minh nhu cầu của khách hàng.

### 1.4.4 Regression test

Giống như kiểm thử xác nhận kiểm thử hồi quy liên quan đến việc thực hiện các trường hợp kiểm thử đã được thực hiện trước đó. Sự khác biệt là đối với kiểm thử hồi quy, các trường hợp kiểm thử có thể đúng ở lần cuối cùng chúng được thực thi.

Mục đích của kiểm thử hồi quy để xác minh rằng các sửa đổi trong phần mềm hoặc môi trường không gây ra bất lợi ngoài ý muốn và hệ thống vẫn đáp ứng các yêu cầu của nó.

Bộ kiểm thử hồi quy hoặc gói kiểm tra hồi quy là một tập hợp các trường hợp kiểm thử được sử dụng đặc biệt để kiểm tra hồi quy. Chúng được thiết kế để thực hiện hầu hết các chức năng trong một hệ thống nhưng không chi tiết bất kỳ chức năng nào. Tất cả các trường hợp trong bộ kiểm thử hồi quy sẽ được thực thi mỗi khi một phiên bản mới của phần mềm được phát hành và điều này làm cho chúng trở lên lý tưởng cho tự động hóa.

Kiểm thử hồi quy được thực hiện khi phần mềm thay đổi, do sửa lỗi, chức năng mới. Nó cũng là một ý tưởng tốt để thực thi chúng khi một vài khía cạnh của môi trường thay đổi.

## **1.5. GIỚI THIỆU VỀ WEB SMARTOS**

### 1.5.1. Sơ lược về Web Smartos

SMARTOS là một hệ thống quản lý giúp theo dõi tất cả các hoạt động trong không gian làm việc chung bao gồm: quản lý đặt phòng, quản lý chi phí, doanh thu một cách hiệu quả giúp nhân viên và quản lý có thể tự động hóa các tác vụ thủ công, tối đa hóa hiệu quả và tăng sự hài lòng của thành viên.

SMARTOS hỗ trợ xây dựng một cộng đồng mạnh mẽ, quản lý các nhiệm vụ, tạo các đề xuất và hợp đồng, gửi hóa đơn và được thanh toán trực tuyến, và nhiều hơn nữa.

### 1.5.2. Chức năng chính của Web SmartOs

+ Quản lý khách hàng: Giúp khách hàng dễ dàng đăng kí, cập nhật hồ sơ trong suốt quá trình đặt phòng, trả phòng và đưa ra sơ đồ báo cáo lượng khách hàng tháng để đưa ra các đề xuất thích hợp để tăng sự hài lòng khách hàng.

+ Quản lý đặt phòng: Dễ dàng quản lý đặt phòng, báo cáo về thanh toán, phòng đang được sử dụng.

+ Quản lý tài nguyên: Theo dõi việc sử dụng và bố trí không gian, hàng tồn kho. Cho phép quản lý nhân viên, quản lý thu nhập và các phí dịch vụ khác từ đó giúp quản lý chi phí hiệu quả hơn.

+ Kết nối cộng đồng: Giúp các thành viên kết nối với nhau khi làm việc

# **CHƯƠNG 2: NGHIÊN CỨU PHẦN MỀM KIỂM THỬ TỰ ĐỘNG CYPRESS**

## 2.1. CYPRESS TRONG KIỂM THỬ TỰ ĐỘNG

### 2.1.1. Định nghĩa

Cypress là front end testing tool được xây dựng cho các ứng dụng Web hiện đại. Đây là một công cụ hỗ trợ hữu hiệu cho developers và QA trong kiểm thử ứng dụng Web hiện đại. Cypress được xây dựng trên một kiến trúc mới và chạy trong cùng vòng lặp chạy khi ứng dụng đang được thử nghiệm. Cypress thường được so sánh với Selenium. Nhưng Cypress hoàn toàn khác biệt với Selenium và nó không gặp phải các hạn chế như Selenium. Điều này giúp chúng ta thực thi các Test Cases một cách đơn giản và dễ dàng hơn.

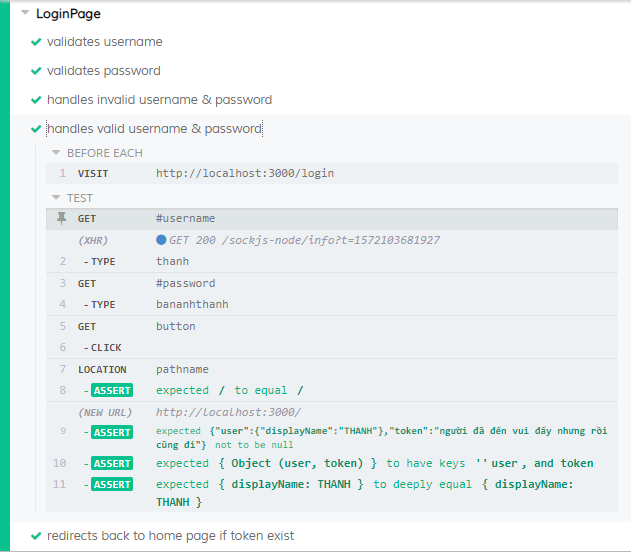
Cypress hỗ trợ chúng ta:

* Set up tests
* Write tests
* Run tests
* Debug Tests

Đằng sau Cypress là một máy chủ Node.js. Quá trình Cypress và Node.js liên tục liên lạc, đồng bộ hóa và thực hiện các nhiệm vụ thay mặt cho nhau. Có quyền truy cập vào cả hai phần (front and back) cho phép khả năng phản hồi các sự kiện của ứng dụng trong thời gian thực, đồng thời hoạt động bên ngoài trình duyệt cho các tác vụ yêu cầu đặc quyền cao hơn.

### 2.1.2. Tại sao lại sử dụng Cypress

* Setup cực kì đơn giản, chạy độc lập luôn cũng được.
* Báo lỗi (debug) tốt.



Như các bạn thấy đối với mỗi test case, **Cypress** đều lưu lại step-by-step cũng như dữ liệu của từng bước thực hiện. Điều này giúp chúng ta đánh giá được khi test case failed thì lỗi do người viết test hay do chính ứng dụng gây ra. Ngoài ra, **Cypress** còn có thêm snapshot (chụp màn hình BEFORE & AFTER) ngay kế bên để bạn có thể kiểm tra luôn giao diện lúc đó liệu có đúng không. Đối với bug contributor như mình thì điều này vô cùng hữu ích.

* Có auto-reload khi viết test

Giống như watch mode ở một số testing framework khác, **Cypress** sẽ detect được khi bạn thay đổi test case và chạy lại toàn bộ các test case.

* Nhanh, chính xác và thông minh

**Cypress** chạy rất nhanh. Tốc độ test của **Cypress** chỉ phụ thuộc vào tốc độ render của website và thời gian phản hồi (response time) của API.

Biết tự chờ đối với các thao tác liên quan tới network (gọi API, chuyển trang, ...). Điều này giảm thiểu thời gian cần thiết để viết test case. Bạn sẽ không phải quan tâm đến việc setup để chờ page/API load nữa.

Điểm cuối cùng là **Cypress** có hỗ trợ autocomplete trong lúc code. Nghe có vẻ bình thường nhưng đối với một framework lớn như **Cypress** thì điều này rất quan trọng. Nó vừa giúp developer code dễ hơn, vừa tránh được việc chạy test case rồi mới biết mình gõ sai syntax ở trong test case.

### 2.1.3. Cài đặt Cypress Cypress

* Cần phải cài đặt npm trên máy tính của bạn để có thể cài đặt Cypress. NPM là trình quản lý gói cho JavaScript và đăng ký phần mềm lớn nhất thế giới.
* Sau đó chạy lệnh sau để cài đặt Cypress



Hãy chắc chắn rằng bạn đã chạy npm init hoặc có một thư mục node\_modules hoặc tệp package.json trong thư mục gốc của dự án để đảm bảo Cypress được cài đặt trong thư mục chính xác.

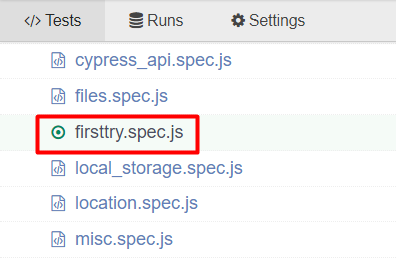
* Làm thế nào để mở Cypress?
* Chạy lệnh bên dưới để mở Cypress



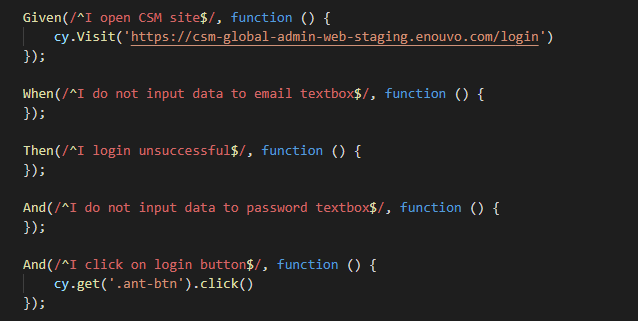
* Hoặc sử dụng lệnh:

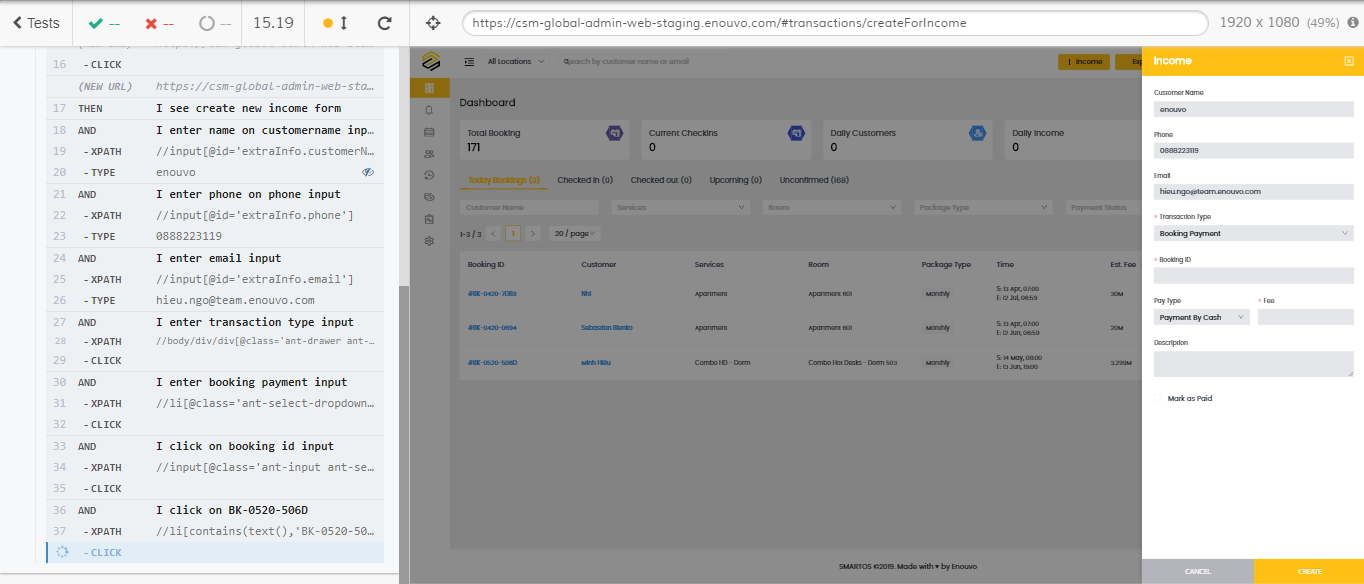


* Làm thế nào để viết một bài Test?
* Bước đầu tiên là cần duyệt đến thư mục cài đặt cypress và tạo một tệp javascript. Sau đó, sẽ thấy tệp javascript(js) mới được tạo trong cypress



* Bạn có thể chỉnh sửa tệp js để đưa ra các test action phù hợp với yêu cầu của ứng dụng.
* Có thể tham khảo đoạn code dưới đây



* Có thể thấy, có một đối tượng "cy" mà chúng ta luôn sử dụng. Đối tượng "cy" cho phép bạn tương tác với trình duyệt. Nó được sử dụng trước mỗi lệnh. Dưới đây là các action mà đoạn code trên thực hiện trên ứng dụng Web.
  + cy.viewport (HEIGHT, WIDTH):Lệnh này thay đổi kích thước màn hình theo giá trị được cung cấp.
  + cy.visit ('URL'): Phương thức này là phương thức điều hướng cho cypress. Nó gọi URL đã cho.
  + cy.get ('locator'): Phương thức này lấy một đối số là trình định vị CSS của phần tử web mà chúng ta muốn tương tác.
  + cy.get ('locator'). type ('INPUT'): Phương thức này cho phép bạn điền vào các trường đầu vào.
  + cy.get ('locator'). click (): Phương thức này cho phép bạn click vào một đối tượng có thể click ( Ví dụ như 1 button).
  + cy.get ('locator'). contains('EXPECTED\_VALUE'): Phương thức này đưa ra sự xác nhận cho thành phần web mà bạn đã chỉ định.
* Sau đó bạn chỉ cần mở cypress và click vào bài test mà bạn muốn thực thi. Giao diện thực thi bao gồm 2 phần : viewport cho execute commands và preview part cho Website. 

**Trong trường hợp có lỗi thì Cypress sẽ cảnh báo qua viewport** 

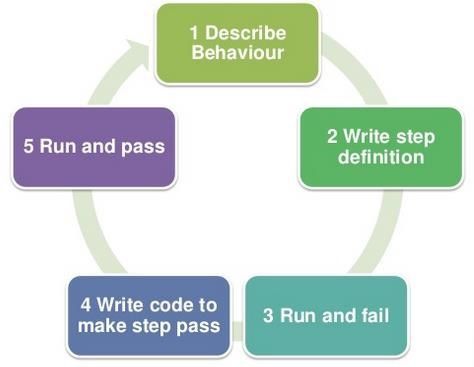
## 2.2 CUCUMBER

### 2.2.1 Định nghĩa về Cucumber

Cucumber là một công cụ phần mềm được sử dụng bởi các lập trình viên để thực hiện test. Nó chạy các câu lệnh acceptance tests tự động được viết theo behavior-driven development (BDD) style. Trọng tâm của phương pháp tiếp cận BDD Cucumber là trình phân tích cú pháp ngôn ngữ thô gọi là Gherkin. Nó thực thi các câu lệnh bằng một ngôn ngữ logic mà khách hàng có thể hiểu. Như vậy, Cucumber cho phép các người sử dụng nó viết các câu lệnh lập trình với cú pháp như ngôn ngữ nói thông thường.

Cucumber được viết dựa trên ngôn ngữ lập trình Ruby. Và ban đầu được sử dụng độc quyền cho kiểm thử trên Ruby như là một bổ sung cho việc viết RSpec BDD.Cucumber bây giờ hỗ trợ một loạt các ngôn ngữ lập trình khác nhau thông qua các implementations khác nhau. Ví dụ, Cuke4php và Cuke4Lua là các phần mềm kết nối cho phép thử nghiệm các dự án PHP và Lua, tương ứng. Các implementations khác chỉ đơn giản có thể là đòn bẩy phân tích cú pháp Gherkin trong khi thực hiện phần còn lại của khuôn khổ thử nghiệm bằng ngôn ngữ đích.

### 2.2.2 Chu trình kiểm thử với Cucumber



1. Mô tả hành vi dưới dạng văn bản (sử dụng ngôn ngữ Gherkin).

2. Định nghĩa các bước.

3. Chạy kiểm thử và xem các bước không thành công.

4. Viết mã nguồn làm cho các bước được thành công.

5. Chạy lại kiểm thử và xem những bước đã thành công.

6. Lặp lại các bước cho đến khi các bước đã chạy thành công.

Sau khi kịch bản kiểm thử chạy, Cucumber sẽ đọc mã Gherkin từ tệp với phần mở rộng là \*.feature, sau đó sẽ tìm đoạn mã trong tệp mô tả cách thực thi các bước đúng với hành động trong tệp \*.feature và thực hiện đoạn mã nguồn, ở bước chạy mã nguồn Cucumber có thể kết hợp với các khung phần mềm khác như Rubyon Rail, Selenium, Spring,…

### 2.2.3 Ngôn ngữ Gherkin

#### 2.2.3.1) Khái niệm:

Gherkin là định dạng cho thông số kỹ thuật cucumber. Đây là một ngôn ngữ cụ thể của miền giúp bạn mô tả hành vi nghiệp vụ mà không cần phải đi sâu vào chi tiết thực hiện. Văn bản này hoạt động như tài liệu và bộ khung của các bài kiểm tra tự động của bạn. Gherkin dựa trên Ngữ pháp TreeTop tồn tại trong hơn 37 ngôn ngữ. Do đó, bạn có thể viết cucumber của mình bằng hơn 37 ngôn ngữ .

#### 2.2.3.2) Mục đích:

* Cung cấp dữ liệu.
* Kiểm thử tự động.

#### 2.2.3.3) Quy tắc viết Gherkin:

* Mỗi tệp Gherkin chỉ mô tả cho một tệp feature.
* Tệp mã nguồn của Gherkin là .feature

#### 2.2.3.4) Cú pháp Gherkin

* Giống như Python và YAML, Gherkin là một ngôn ngữ kịch bản được sử dụng để định nghĩa logic theo cấu trúc. Cũng giống như Ruby, nên thay thế kí tự tab bằng các kí tự space, dòng comment sẽ có kí tự # ở đầu dòng.
* Bắt đầu một file sẽ là Feature, sau đó đến scenarios và steps. Khi chạy file source “.feature” mỗi step sẽ match với một Ruby code block được định nghĩa sẵn trước đó gọi là “Step Definitions”.

**File Gherkin thường như sau:**

*/\*Em nghĩ đoạn ni nên giải thích đoạn mã ở dưới dùng để làm gì, người đọc không chuyên ngành về Test sẽ dễ hiểu hơn\*/*

**Feature:** Selendroid Test

This is support to fill user’s localization

**Scenario Outline:** Create the default locate

**Given** I open the app

**When** I verify the app

**Then** I click text phone number

**And** I verify input phone number screen

**And** I input Username as <phonenumber>

**And** I click next button

**Ngữ pháp của Gherkin chia đầu vào thành các feature, scenario và step**

* Feature:
* Tất cả các file "\*.feature" được quy ước chỉ bao gồm một feature.
* Dòng bắt đầu với từ khóa ”**Feature**:” theo sau bởi 3 dòng lùi vào.
* Một Feature thường gồm một list các Scenario. Có thể viết bất cứ gì mà bạn muốn cho đến khi bắt đầu Scenario đầu tiên (khi dòng mới bắt đầu bằng từ “Scenario:”).
* Có thể sử dụng tags để nhóm các feature và scenario lại với nhau, không lệ thuộc vào file và cấu trúc thư mục.
* Mỗi scenario bao gồm một list các steps, các steps bắt đầu bằng các keyword như (Given, When, Then, But hoặc And).
* Thêm vào đó, feature có thể bao gồm scenario outline và background.
* **Scenario**
* Scenario là nòng cốt trong cấu trúc Gherkin.
* Mọi Scenario đều bắt đầu với từ khóa “Scenario:” theo sau bởi một tiêu đề tùy ý.
* Mỗi Feature có thể có một hoặc nhiều Scenario, và mỗi Scenario bao gồm một hay nhiều steps.
* Background
* Background cho phép thêm một vài context vào tất cả các scenario trong một feature.
* Một background giống như một Scenario không có tiêu đề, bao gồm nhiều số steps.
* Sự khác nhau là khi chạy: background chạy trước mỗi scenario, nhưng sau Hook
* Step
* Feature bao gồm Step như (Given, When, Then…).
* Cucumber không phân biệt giữa 3 loại steps.
* Tuy nhiên cần sử dụng cẩn thận và đúng mục đích đối với những từ này.
* Given
* Được sử dụng để mô tả ngữ cảnh ban đầu của hệ thống.
* Mục đích của Given là đưa hệ thống vào một trạng thái đã biết trước khi sử dụng (hoặc hệ thống bên ngoài) bắt đầu tương tác với hệ thống (trong bước When).
* Nếu bạn đã làm việc với use case, Givens là điều kiện tiên quyết.
* Khi Cucumber thực thi bước Given, nó sẽ cấu hình hệ thống để được một trạng thái rõ ràng như là: tạo, cấu hình các đối tượng hoặc thêm dữ liệu test vào cơ sở dữ liệu.
* Nên sử dụng Given với một table, cách này giúp đọc tất cả các scenario tại cùng một chỗ mà không cần nhảy giữa các file
* When
* Mục đích của When là để mô tả các sự kiện, hành động chính mà người dùng sử dụng.
* Then
* Mục đích của Then là quan sát kết quả. Các quan sát phải được liên quan đến các giá trị kinh doanh / lợi ích trong việc mô tả feature. Các quan sát phải kiểm tra đầu ra của hệ thống (một báo cáo, giao diện người dùng, tin nhắn,...)
* And, But
* Khi có nhiều Given, When, Then thì có thể viết: */\*Đoạn ni em nghĩ nên chia ra, khi có ít Given, When, Then thì file feature sẽ được viết như thế này:… Xong rồi viết ra đoạn feature: Khi cod nhiều G, W, T như chị viết\*/*

**Given** I open the app

**When** I verify the app

**Then** I click text phone number

**And** I verify input phone number screen

**And** I input Username as <phonenumber>

**And** I click next button

* Comment
* Bắt đầu dòng bằng dấu # cho biết dòng đó là dòng comment và không được thực thi
* Step Definition:
* Cucumber không thể biết làm thế nào để thực thi được scenario. Nó cần Step Definition để biên dịch nguyên văn các bước Gherkin thành các hành động cái mà có thể tương tác với hệ thống.
* Khi Cucumber thực thi các steps trong Scenario nó sẽ tìm kiếm các Step Definition phù hợp để thực thi.
* Một Step Definition là một phần nhỏ của code với một pattern đính kèm.
* Pattern được sử dụng để liên kết các Step definition với tất cả các Step phù hợp, và code là cái mà Cucumber sẽ thực thi khi thấy Gherkin Step.

## 2.3 THIẾT LẬP MÔI TRƯỜNG

Để có thể kiểm thử tự động trên Web, cần thiết lập môi trường để viết và chạy test cript. Bao gồm:

* Download và cài đặt Java Script.
* Cài đặt Nodejs.
* Add-ons cho Google chrome: Tidy Gherkin.
* Thiết lập Virtual Private Server.
* Cài đặt cypress.

# 

# **CHƯƠNG 3: ỨNG DỤNG KIỂM THỬ TRÊN WEBSITE SMARTOS**

## 3.1. GIỚI THIỆU TỔNG QUAN VỀ CÔNG TY, SẢN PHẨM, MÔI TRƯỜNG KIỂM THỬ CỦA CÔNG CTY

- Enouvo là một công ty phần mềm cung cấp một Platform bao gồm nhiều ứng dụng dành cho doanh nghiệp, như quản lý doanh nghiệp, ứng dụng kế toán,… Công ty liên tục phát triển những feature mới để phục vụ cho các yêu cầu và xu hướng của khách hàng. Sản phẩm của công ty là ứng dụng web, chạy trên nhiều môi trường khác nhau (windows, Android, IOS …)

- Hàng năm, công ty luôn có kế hoạch cải tiến sản phẩm và các tính năng hiện tại đồng thời tạo ra các sản phẩm và tính năng mới. Mỗi khi một sản phẩm hay tính năng mới nào được xuất bản ra ngoài thị trường, sản phẩm và tính năng đó cũng như những sản phẩm và tính năng cũ đều phải được tiến hành kiểm thử trên các môi trường mà sản phẩm đó hỗ trợ.

- Các ứng dụng được phát triển và cài đặt trên các môi trường khác nhau

* Hệ điều hành: Windows, Android, IOS …
* Browser: IE, Firefox, Chrome …
* Database: MySQL, HSQL, Oracle, PSQL …

## 3.2. SOFTWARE TEST PLAN SMARTOS:

- Giới thiệu:

Mục đích của kế hoạch kiểm thử là lên kế hoạch cụ thể nhằm đảm bảo việc kiểm tra đối chiếu chức năng, giao diện của dự án SmartOs theo đúng yêu cầu của khách hàng.

* Kế hoạch kiểm thử

Đối tượng kiểm thử:

* Hệ thống Smartos được chia làm 2 phần: giao diện trang chủ và giao diện admin.

Những đối tượng kiểm thử ở giao diện admin gồm: đăng nhập, luồng income, luồng expense, luồng booking, luồng logout. Kế hoạch kiểm thử sau đây tập trung cho giao diện admin.

Phạm vi kiểm thử:

Thực hiện functional testing cho các module sau:

* Đăng nhập
* Luồng Income
* Luồng Expense
* Luồng Booking
* Luồng Logout

- Thiết bị: Desktop

- Trình duyệt: Chrome

- Test design

## 3.3. THIẾT KẾ TRƯỜNG HỢP KIỂM THỬ TỰ DỘNG CHO CÁC CHỨC NĂNG

### 3.3.1. Màn hình đăng nhập

#### 3.3.1.1) Chức năng:

Sau khi được Admin cấp cho một tài khoản với đầy đủ thông tin bao gồm email và mật khẩu, mỗi người dùng sẽ có mỗi email khác nhau để đăng nhập.Người dùng chỉ có thể sử dụng tài khoản đó để đăng nhập vào trang web và truy cập vào hệ thống để sử dụng. Điều đó cũng giúp hệ thống được bảo mật và tránh sự truy cập bất hợp pháp.

#### 3.3.1.2) Thiết kế Testcase cho chức năng đăng nhập:

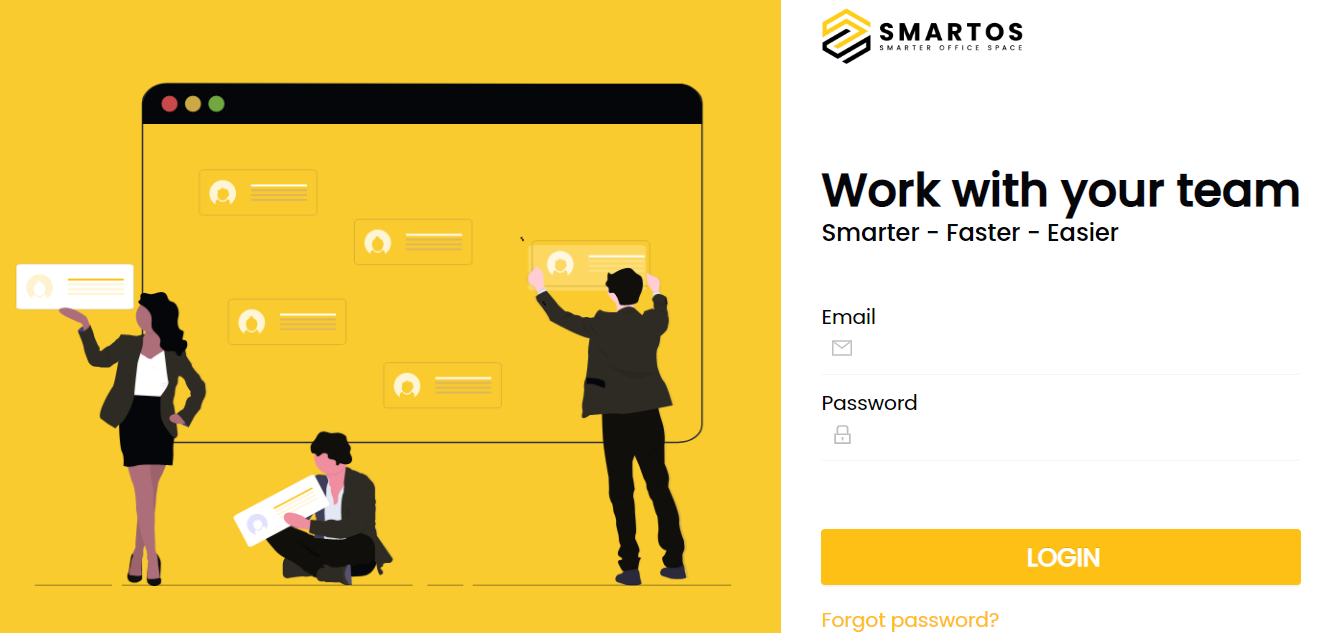


Figure 3.1: Màn hình đăng nhập

Chức năng đăng nhập vào hệ thống gồm có 7 trường hợp để thực hiện kiểm thử.

Trường hợp 1: Xác minh giao diện của trang đăng nhập xem vị trí, kích thước, phông chữ, màu sắc có giống như thiết kế không.

Bảng 4.1: Trường hợp kiểm thử 1 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-01 |
| Tiêu đề | Xác minh giao diện của trang đăng nhập |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Kiểm tra vị trí của tất cả yếu tố và kích thước phông chữ, loại phông chữ, màu sắc với bản thiết kế |
| Môi trường | Kiểm thử |
| Mong đợi | Tất cả các yếu tố “ vị trí, kích thước, phông chữ, loại phông chữ, màu sắc” giống như thiết kế |
| Kiểm tra dữ liệu |  |

Trường hợp 2: Xác minh hệ thống khi người dùng để trống Email và Password. Sau đó nhấn nút “Login”

Bảng 4.2: Trường hợp kiểm thử 2 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-02 |
| Tiêu đề | Xác minh hệ thống khi người dùng để trống Email và Password. Sau đó nhấn nút “Login” |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Nhấn nút Login |
| Môi trường | Kiểm thử |
| Mong đợi | Hiển thị thống báo lỗi   * Email bắt buộc nhập * Password bắt buộc nhâp   Không thể đăng nhập vào trang Admin |
| Kiểm tra dữ liệu | Email: Không nhập  Password: Không nhập |

Trường hợp 3: Xác minh hệ thống khi người dùng để trống Email và nhập đúng Password. Sau đó nhấn nút “Login”

Bảng 4.4: Trường hợp kiểm thử 3 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-03 |
| Tiêu đề | Xác minh hệ thống khi người dùng để trống Email và nhập đúng Password. |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Nhập đúng Password  3. Nhấn nút Login |
| Môi trường | Kiểm thử |
| Mong đợi | Hiển thị thống báo lỗi   * “Please input your password!”   Không thể đăng nhập vào trang Admin |
| Kiểm tra dữ liệu | Email: không nhập  Password: Enouvo123 |

Trường hợp 4: Xác minh hệ thống khi người dùng nhập đúng Email, để trống Password sau đó nhấn nút “Login”

Bảng 4.4: Trường hợp kiểm thử 4 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-04 |
| Tiêu đề | Xác minh hệ thống khi người dùng nhập đúng Email, để trống Password sau đó nhấn login |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Nhập đúng Email  3. Nhấn nút Login |
| Môi trường | Kiểm thử |
| Mong đợi | Hiển thị thống báo lỗi   * “Password is required”   Không thể đăng nhập vào trang Admin |
| Kiểm tra dữ liệu | Email: admin@enouvo.com  Password: Không nhập |

Trường hợp 5: Xác minh hệ thống khi người dùng nhập đúng Email và sai Password. Sau đó nhấn nút “Login”

Bảng 4.4: Trường hợp kiểm thử 5 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-05 |
| Tiêu đề | Xác minh hệ thống khi người dùng nhập sai Email và sai Password |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Nhập đúng Email và sai Password  3. Nhấn nút Login |
| Môi trường | Kiểm thử |
| Mong đợi | Hiển thị thống báo lỗi   * “User not found”   Không thể đăng nhập vào trang Admin |
| Kiểm tra dữ liệu | Email: admin@enouvo.com  Password: hieuenouvo123 |

Trường hợp 6: Xác minh hệ thống khi người dùng nhập sai Email và đúng Password

Bảng 4.4: Trường hợp kiểm thử 6 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-06 |
| Tiêu đề | Xác minh hệ thống khi người dùng nhập sai Email và đúng Password |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Nhập sai Email và đúng Password  3. Nhấn nút Login |
| Môi trường | Kiểm thử |
| Mong đợi | Hiển thị thống báo lỗi   * “User not found”   Không thể đăng nhập vào trang Admin |
| Kiểm tra dữ liệu | Email: hieu.ngo@team.enouvo.com  Password: enouvo123 |

Trường hợp 6: Xác minh hệ thống khi người dùng nhập sai Email và Password

Bảng 4.4: Trường hợp kiểm thử 6 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-06 |
| Tiêu đề | Xác minh hệ thống khi người dùng nhập sai Email và Password |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Nhập sai Email và Password  3. Nhấn nút Login |
| Môi trường | Kiểm thử |
| Mong đợi | Hiển thị thống báo lỗi   * “User not found”   Không thể đăng nhập vào trang Admin |
| Kiểm tra dữ liệu | Email: hieu.ngo@team.enouvo.com  Password: hieuenouvo123 |

Trường hợp 7: Xác minh hệ thống khi người dùng nhập email, mật khẩu đúng. Sau đó nhấn nút “Login”

Bảng 4.2: Trường hợp kiểm thử 7 cho màn hình đăng nhập

|  |  |
| --- | --- |
|  | Log-07 |
| Tiêu đề | Xác minh hệ thống khi người dùng nhập email, password đúng. Sau đó nhấn chọn nút “Login” |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Nhập đúng email và password  3. Nhấn chọn nút Login |
| Môi trường | Kiểm thử |
| Mong đợi | Điều hướng đến trang SmartOs admin |
| Kiểm tra dữ liệu | Email: [admin@enouvo.com](mailto:admin@enouvo.com)  Password: enouvo123 |

#### 3.3.1.3) Gherkin

Dựa trên testcase sau đó viết ra feature và đặt nó vào feature files ở cypress/integration/login.feature của cây thư mục

**- Feature File */\*warning message Và notice message giống hay khác để dùng 1 cái thôi\*/***

Feature: visit login page in CSM

    Scenario: Login with empty data by desktop */\*empty data cho trường nào\*/*

       Given I open CSM site by desktop screen

       When I see the logo

       And I enter empty data to email input

       And I enter empty data to password input

       And I click on submit button

       Then I see warning message below user input

       And I see warning message below password input

    Scenario: Login with empty email data by desktop

       Given I open CSM site by desktop screen

       When I see the logo

       And I enter empty user data to email input

       And I enter correct data to password input

       And I click on submit button

       Then I see warning message below user input */\*below email chứ ạ \*/*

    Scenario: Login with empty password data by desktop

       Given I open CSM site by desktop screen

       When I see the logo

       And I enter correct data to email input

       And I enter empty data to password input

       And I see warning message below password *input /\*chưa click submit sao thấy message\*/*

       And I click on submit button

       Then I see notification error message

    Scenario: Login with incorrect password data by desktop

       Given I open CSM site by desktop sreen

       When I see the logo

       And I enter correct data to email input

       And I enter incorrect password data to password input

       And I click on submit button

       Then I see notification error *message /\*notification hay message, dùng 1 cái thì khi đọc sẽ dễ hơn\*/*

    Scenario: Login with incorrect email data by desktop

       Given I open CSM site by desktop screen

       When I see the logo

       And I enter incorrect data to user input /\*incorrect cho email chứ ạ\*/

       And I enter correct data to password input

       And I click on submit button

       Then I see notification error message

   Scenario: Login with incorrect email data and incorrect passwword data by deskop

       Given I open CSM site by desktop screen

       When I see the logo

       And I enter incorrect data to email input

       And I enter incorrect data to password input

       And I click on submit button

       Then I see notification error message

    Scenario: Verify when inputing Email and Password are registered

        Given I open CSM site by desktop screen

        When I see the logo

        And I enter correct data to email input

        And I enter correct data to password input

        And I click on submit button

        Then I see Menu icon at dashboard page

Step Definition để biên dịch nguyên văn các bước Gherkin thành các hành động tương tác với hệ thống qua WebDriver

Khi Cucumber thực thi các Step trong Scenario, nó sẽ tìm kiếm các Step Definition phù hợp của trang để thực thi thông qua các Annotation tương ứng

**- Step Definition**

*/// <reference types="Cypress" />*

import { Then, Given, When, And } from "cypress-cucumber-preprocessor/steps";

*//Login empty data by desktop*

**Given**(/^I open CSM site by desktop screen$/, function () {

    cy.**visit**('https://csm-global-admin-web-staging.enouvo.com')

});

**When**(/^I see the logo$/, function () {

 });

**Then**(/^I see warning message below user input$/, function () {

});

**And**(/^I enter empty data to email input$/, function () {

});

**And**(/^I enter empty data to password input$/, function ( ) {

 });

**And**(/^I see warning message below password input$/, function () {

 });

**And**(/^I click on submit button$/, function () {

 cy.**get**('.login-form-button').**click**()

});

*//Login with empty user data by desktop*

**Given**(/^I open CSM site by desktop screen$/, function () {

  cy.**visit**('https://csm-global-admin-web-staging.enouvo.com')

});

**When**(/^I see the logo$/, function () {

  });

**Then**(/^I see warning message below user input$/, function () {

});

**And**(/^I enter empty user data to email input$/, function () {

  });

**And**(/^I enter correct data to password input$/, function ( ) {

  cy.**get**('#password').**type**('enouvo123')

  });

**And**(/^I see warning message below user input$/, function () {

  });

**And**(/^I click on submit button$/, function () {

  cy.**get**('.login-form-button').**click**()

});

*//Login with empty password data by desktop*

**Given**(/^I open CSM site by desktop screen$/, function () {

cy.**visit**('https://csm-global-admin-web-staging.enouvo.com')

});

**When**(/^I see the logo$/, function () {

});

**Then**(/^I see notification error message$/, function () {

});

**And**(/^I enter correct user data to email input$/, function () {

cy.**get**('#email').**type**('admin@enouvo.com')

});

**And**(/^I enter empty data to password input$/, function ( ) {

});

**And**(/^I see warning message below password input$/, function () {

});

**And**(/^I click on submit button$/, function () {

  cy.**get**('.login-form-button').**click**()

});

*//Login with incorrect password data by desktop*

**Given**(/^I open CSM site by desktop sreen$/, function () {

  cy.**visit**('https://csm-global-admin-web-staging.enouvo.com')

});

**When**(/^I see the logo$/, function () {

});

**Then**(/^I see notification error message$/, function () {

});

**And**(/^I enter correct data to email input$/, function () {

  cy.**get**('#email').**type**('admin@enouvo.com')

});

**And**(/^I enter incorrect password data to password input$/, function () {

  cy.**get**('#password').**type**('eenouvo123')

});

**And**(/^I click on submit button$/, function () {

});

*//Login with incorrect user data by desktop*

**Given**(/^I open CSM site by desktop screen$/, function () {

  cy.**visit**('https://csm-global-admin-web-staging.enouvo.com')

});

**When**(/^I see the logo$/, function () {

});

**Then**(/^I see notification error message$/, function () {

});

**And**(/^I enter incorrect data to user input$/, function () {

  cy.**get**('#email').**type**('123@enouvo.com')

});

**And**(/^I enter correct data to password input$/, function ( ) {

  cy.**get**('#password').**type**('enouvo123')

  });

**And**(/^I click on submit button$/, function () {

  cy.**get**('.login-form-button').**click**()

});

*//Login with incorrect user data and incorrect passwword data by desktop*

**Given**(/^I open CSM site by desktop screen$/, function () {

  cy.**visit**('https://csm-global-admin-web-staging.enouvo.com')

});

**When**(/^I see the logo$/, function () {

});

**Then**(/^I see notification error message$/, function () {

});

**And**(/^I enter incorrect data to email input$/, function () {

  cy.**get**('#email').**type**('123@enouvo.com')

});

**And**(/^I enter incorrect data to password input$/, function ( ) {

  cy.**get**('#password').**type**('123')

  });

**And**(/^I click on submit button$/, function () {

  cy.**get**('.login-form-button').**click**()

});

*// Verify when inputing Email and Password are registered*

**Given**(/^I open CSM site by desktop screen$/, function () {

  cy.**visit**('https://csm-global-admin-web-staging.enouvo.com')

});

**When**(/^I see the logo$/, function () {

});

**Then**(/^I see Menu icon at dashboard page$/, function () {

});

**And**(/^I enter correct data to email input$/, function () {

  cy.**get**('#email').**type**('admin@enouvo.com')

});

**And**(/^I enter correct data to password input$/, function ( ) {

  cy.**get**('#password').**type**('enouvo123')

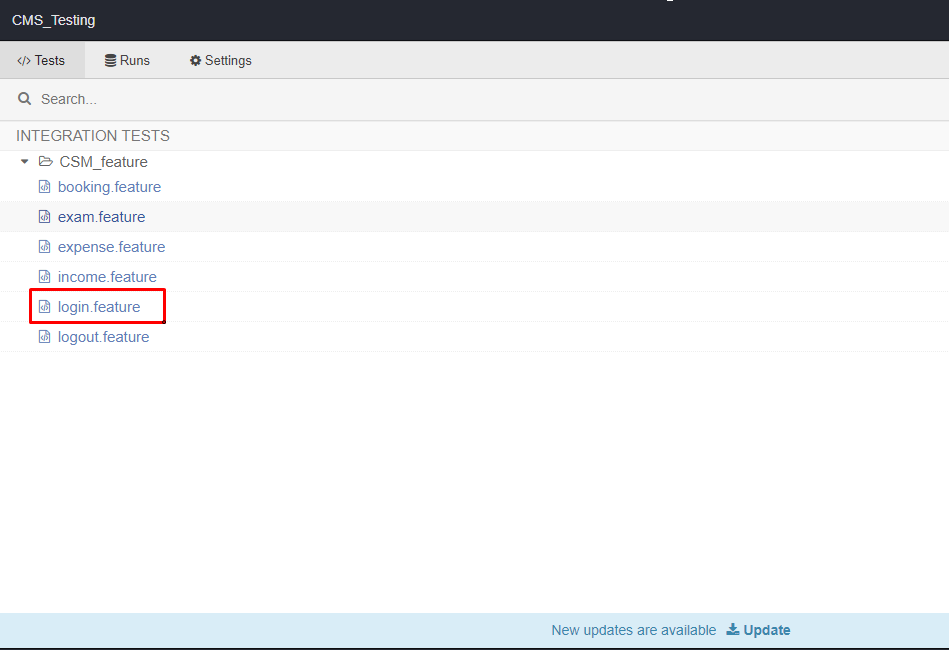
  });

**And**(/^I click on submit button$/, function () {

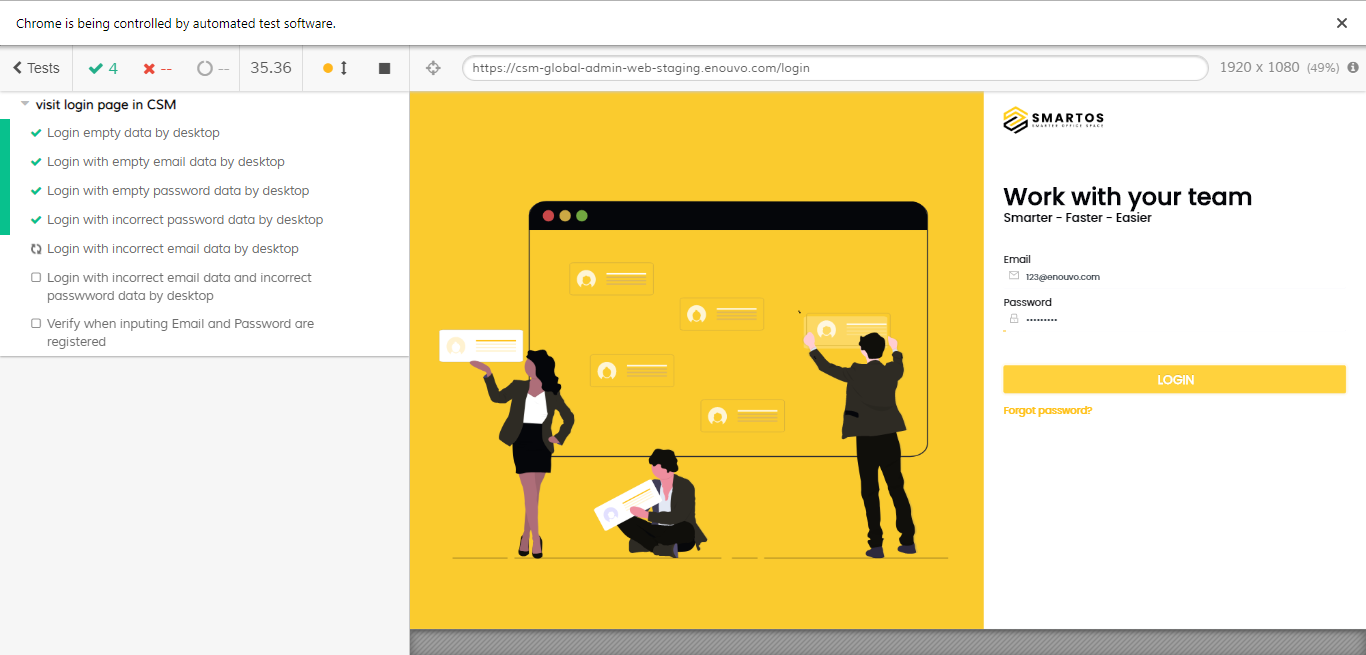
**cy.get('.login-form-button').click()**

});

Chạy ./node\_modules/.bin/cypress open để mở cypress và click vào Login test để thực thi



Giao diện kết quả bao gồm 2 phần : viewport cho execute commands và preview part cho Website.



Nếu có lỗi thì có thể click vào lỗi và thông tin về lỗi sẽ được hiển thị trên console log

#### 3.3.1.4) Kết quả

Running: CSM\_feature\login.feature... (5 of 6)

visit login page in CSM

√ Login empty data by desktop (3313ms)

√ Login with empty email data by desktop (2432ms)

√ Login with empty password data by desktop (1686ms)

√ Login with incorrect password data by desktop (2100ms)

√ Login with incorrect email data by desktop (1593ms)

√ Login with incorrect email data and incorrect passwword data by desktop (2123ms)

√ Verify when inputing Email and Password are registered (2111ms)

7 passing (54s)

(Results)

┌─────────────────────────────────────────┐

│ Tests: 7 │

│ Passing: 7 │

│ Failing: 0 │

│ Pending: 0 │

│ Skipped: 0 │

│ Screenshots: 0 │

│ Video: true │

│ Duration: 53 seconds │

│ Spec Ran: CSM\_feature\login.feature │

└─────────────────────────────────────────┘

(Video)

- Started processing: Compressing to 32 CRF

- Compression progress: 83%

- Finished processing: F:\CMS\_Testing\cypress\videos\CSM\_feature\login.feature.mp4 (12 seconds)

🡪 Dựa vào kết quả test cho thấy rằng: Tổng số Testcase của chức năng đăng nhập là 7 testcase, và tất cả các testcase đều passed

### Màn hình income:

#### 3.3.2.1) Chức năng:

Chức năng Income được tạo ra nhằm mục đích thống kê tổng tiền thu vào của doanh nghiệp. Bao gồm các fields như: Customer Name, Phone, Email, Transaction Type, Booking ID, Pay Type, Fee, Decription, Checkbox: Mark as Paid

#### 3.3.2.2) Thiết kế Testcase cho màn hình Income:

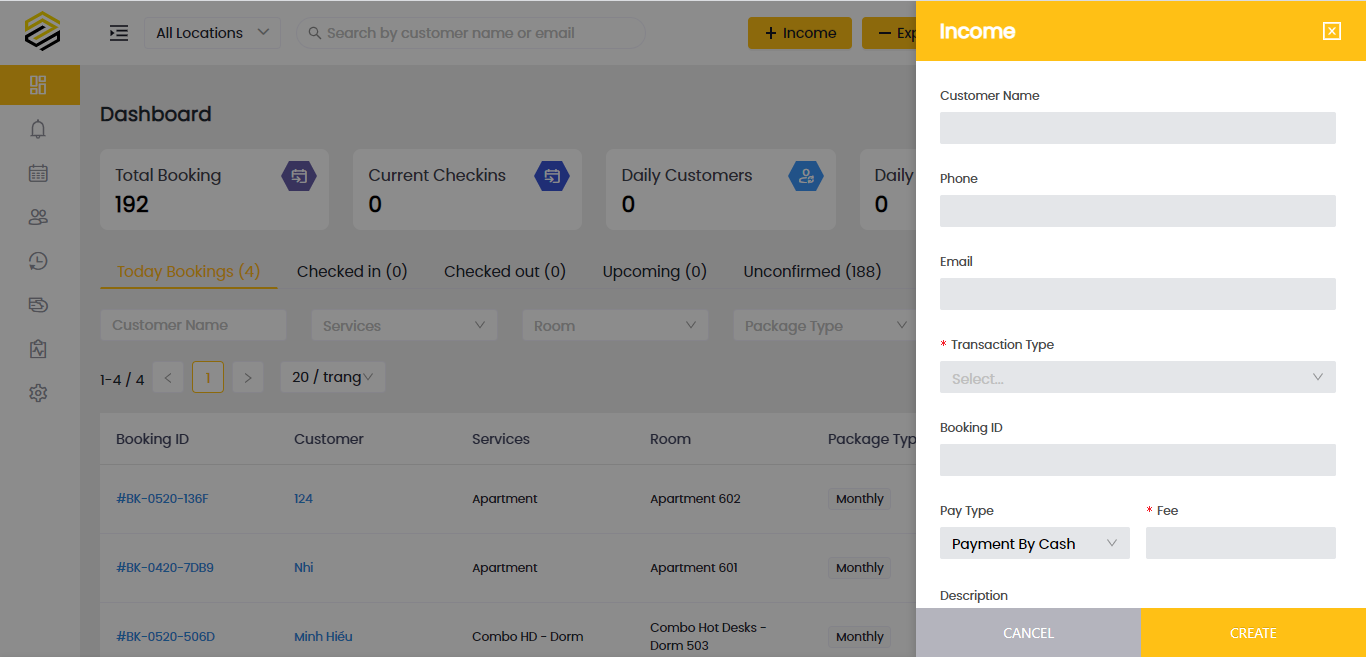


Figure 3.1: Màn hình Income

Màn hình Income gồm có trường hợp để thực hiện kiểm thử.

Trường hợp 1: Xác minh giao diện của màn hình Income xem vị trí, kích thước, phông chữ, màu sắc có giống như thiết kế không.

Bảng 4.1: Trường hợp kiểm thử 1 cho màn hình Income

|  |  |
| --- | --- |
|  | Log-01 |
| Tiêu đề | Xác minh giao diện của màn hình Income |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Login thành công  3. Chọn Income button  4. Hiển thị Income Popup |
| Môi trường | Kiểm thử |
| Mong đợi | Tất cả các yếu tố “ vị trí, kích thước, phông chữ, loại phông chữ, màu sắc” giống như thiết kế |
| Kiểm tra dữ liệu |  |

Trường hợp 2: Xác minh hệ thống khi người dùng không nhập thông tin vào tất cả các fields được yêu cầu và nhấn nút “ Create ”

Bảng 4.2: Trường hợp kiểm thử 2 cho màn hình Income

|  |  |
| --- | --- |
|  | Log-02 |
| Tiêu đề | Xác minh hệ thống khi người dùng không nhập thông tin vào tất cả các fields được yêu cầu và nhấn nút “ Create ” |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Login thành công  3. Click “Income” button  4. Hiển thị Income Popup  5. Không nhập dữ liệu đến tất cả các fields được bắt buộc  6. Click “ Create” button |
| Môi trường | Kiểm thử |
| Mong đợi | Hiển thị thông báo lỗi dưới các fields được bắt buộc nhập  Hiển thị thông báo lỗi:  -Tại transaction Type: “ This field is required”  - Tại Fee: “This field is required” |
| Kiểm tra dữ liệu | * Transaction Type: Không nhập * Fee: Không nhập |

Trường hợp 3: Xác minh hệ thống khi người dùng không nhập thông tin vào các fields không bắt buộc và nhấn nút “ Create ”

Bảng 4.2: Trường hợp kiểm thử cho 3 màn hình Income

|  |  |
| --- | --- |
|  | Log-03 |
| Tiêu đề | Xác minh hệ thống khi người dùng không nhập thông tin vào các fields không bắt buộc và nhấn nút “ Create ” |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Login thành công  3. Click “Income” button  4. Hiển thị Income Popup  5. Không nhập dữ liệu đến tất cả các fields không được bắt buộc  6. Click “ Create” button |
| Môi trường | Kiểm thử |
| Mong đợi | Tạo Income thành công |
| Kiểm tra dữ liệu | * Customer name: Không nhập * Phone: Không nhập * Email: Không nhập * Booking ID: Không nhập * Description: Không nhập |

Trường hợp 4: Xác minh hệ thống khi người dùng không nhập thông tin vào các fields bắt buộc, nhập giá trị hợp lệ đến các field con lại và nhấn nút “ Create ”

Bảng 4.2: Trường hợp kiểm thử cho 4 màn hình Income

|  |  |
| --- | --- |
|  | Log-04 |
| Tiêu đề | Xác minh hệ thống khi người dùng không nhập thông tin vào các fields bắt buộc và nhấn nút “ Create ” |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Login thành công  3. Click “Income” button  4. Hiển thị Income Popup  5. Không nhập dữ liệu đến tất cả các fields bắt buộc  6. Click “ Create” button |
| Môi trường | Kiểm thử |
| Mong đợi | Tạo Income thành công |
| Kiểm tra dữ liệu | * Transaction Type: Không nhập * Fee: Không nhập |

Trường hợp 5: Xác minh hệ thống khi người dùng nhập giá trị không hợp lệ đến tất cả các fields và kích nút “ Create ”

Bảng 4.2: Trường hợp kiểm thử cho 5 màn hình Income

|  |  |
| --- | --- |
|  | Log-05 |
| Tiêu đề | Xác minh hệ thống khi người dùng giá trị không hợp lệ đến tất cả các fields và kícc nút “ Create ” |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Login thành công  3. Click “Income” button  4. Hiển thị Income Popup  5. Nhập giá trị không hợp lệ đến tất cả các fields  6. Click “ Create” button |
| Môi trường | Kiểm thử |
| Mong đợi | Tạo Income không thành công và hiển thị thông báo lỗi dưới các field với data không hợp lệ |
| Kiểm tra dữ liệu |  |

Trường hợp 6: Xác minh hệ thống khi người dùng nhập giá trị hợp lệ đến tất cả các fields và kích vào “ Create” button

Bảng 4.4: Trường hợp kiểm thử 6 cho màn hình Income

|  |  |
| --- | --- |
|  | Log-06 |
| Tiêu đề | Xác minh hệ thống khi người dùng không nhập thông tin hợp lệ vào tất cả các fields và nhấn nút “ Create ” |
| Các bước thực hiện | 1.Mở trang web “SmartOs”  2. Login thành công  3. Click “Income” button  4. Hiển thị Income Popup  5. Nhập giá trị hợp lệ đến tất cả các fields  6. Kích “ Create” button |
| Môi trường | Kiểm thử |
| Mong đợi | Tạo “Income” thành công và hiển thị thông báo : “Create Income successful” |
| Kiểm tra dữ liệu |  |

Trường hợp 7: Xác minh hệ thống khi người dùng nhập giá trị hợp lệ đến tất cả các fields và kích vào “ Cancel” button

Bảng 4.4: Trường hợp kiểm thử 7 cho màn hình Income

|  |  |
| --- | --- |
|  | Log-07 |
| Tiêu đề | Xác minh hệ thống khi người dùng không nhập thông tin hợp lệ vào tất cả các fields và nhấn nút “ Create ” |
| Các bước thực hiện | 1.Mở trang web “SmartOs”  2. Login thành công  3. Click “Income” button  4. Hiển thị Income Popup  5. Nhập giá trị hợp lệ đến tất cả các fields  6. Kích “ Cancel” button |
| Môi trường | Kiểm thử |
| Mong đợi | Hủy bỏ thao tác và thêm Income không thành công |
| Kiểm tra dữ liệu |  |

#### 3.3.2.3. Gherkin

Dựa trên testcase sau đó viết ra feature và đặt nó vào feature files ở cypress/integration/Income.feature của cây thư mục

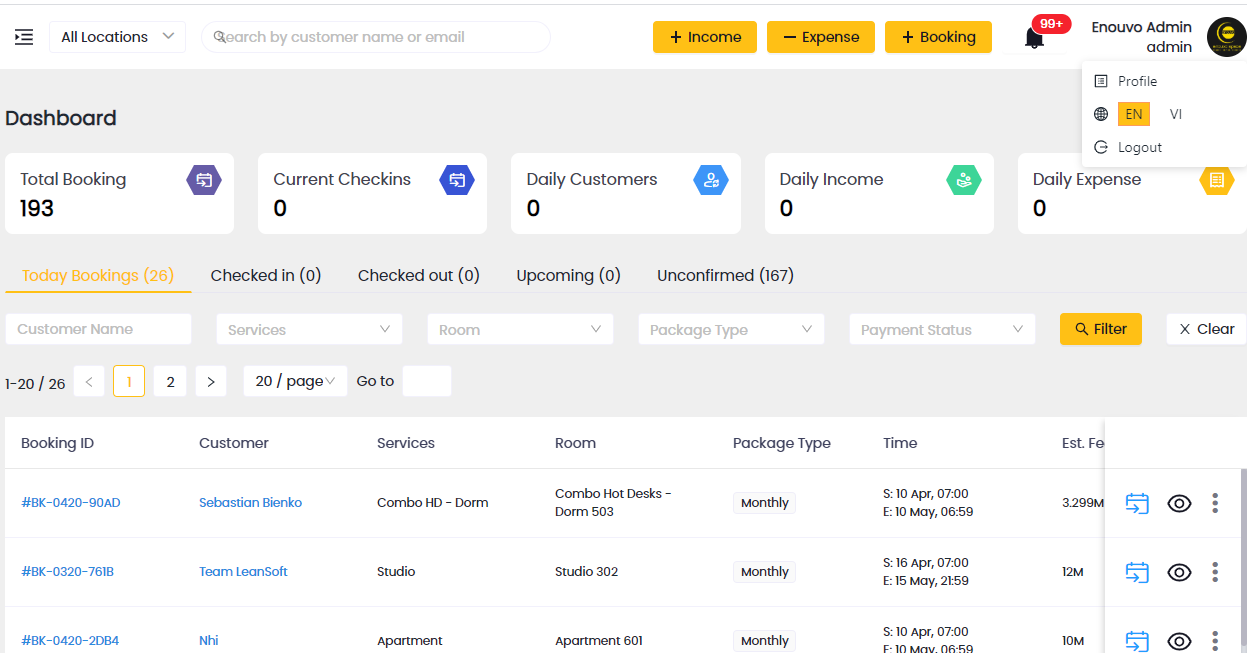
### Màn hình đăng xuất:

#### *3.3.3.1. Chức năng:*

Giúp đăng xuất người dùng khỏi hệ thống để đăng nhập một người dùng mới và để bảo mật thông tin người dùng.  
Bước 1: Từ màn hình chính của hệ thống, kích chuột vào profile. Hệ thống sẽ hiển thị nút Logout

Bước 2: Click vào nút logout thì tài khoản tiến hành đăng xuất khỏi hệ thống để quay về trang chủ

#### *3.3.3.2. Thiết kế Testcase cho chức năng đăng xuất:*



Hình 3.2: Màn hình đăng xuất

Chức năng đăng xuất khỏi hệ thống gồm 2 trường hợp có thể để thực hiện kiểm thử

Trường hợp 1: Xác minh giao diện của trang đăng nhập xem vị trí, kích thước, phông chữ, màu sắc có giống như thiết kế không.

Bảng 4.1: Trường hợp kiểm thử cho một màn hình đăng xuất

|  |  |
| --- | --- |
|  | Log-01 |
| Tiêu đề | Xác minh giao diện của trang đăng nhập |
| Các bước thực hiện | 1. Mở trang web “SmartOs”  2. Kiểm tra vị trí của tất cả yếu tố và kích thước phông chữ, loại phông chữ, màu sắc với bản thiết kế |
| Môi trường | Kiểm thử |
| Mong đợi | Tất cả các yếu tố “ vị trí, kích thước, phông chữ, loại phông chữ, màu sắc” giống như thiết kế |
| Kiểm tra dữ liệu |  |

Trường hợp 2: Xác minh hệ thống khi người dùng nhấn vào nút “Log out”

Bảng 4.11: Trường hợp kiểm thử hai cho màn hình đăng xuất

|  |  |
| --- | --- |
|  | Log-02 |
| Tiêu đề | Xác minh hệ thống khi người dùng nhấn vào nút Logout |
| Các bước thực hiện | 1.Mở trang web “SmartOs”  2. Login thành công  3. Nhấn chọn nút Logout |
| Môi trường | Kiểm thử |
| Mong đợi | Khi login thành công thì điều hướng đến trang SmartOs Admin  Logout thành công thì điều hướng đến trang Login |
| Kiểm tra dữ liệu |  |

#### *3.3.3.3. Gherkin*

Dựa trên testcase sau đó viết ra feature và đặt nó vào feature files ở cypress/integration/logout.feature của cây thư mục

**- Feature File**

Feature: Logout

        Scenario:Logout

            Given I open CSM site by desktop screen

              When I see the logo

               And I enter "gmailIncorrect" data to gmail input

               And I enter "enouvo123" data to password input

               And I click on submit button

               And I click on profile button

               And I click on logout button

              Then I see login page

**Step Definition**

*// / <reference types="Cypress" />*

import { Then, When, And, Given } from "cypress-cucumber-preprocessor/steps";

**Given**(/^I open CSM site by desktop screen$/, function () {

cy.**visit**('https://csm-global-admin-web-staging.enouvo.com/')

});

**When**(/^I see the logo$/, function () {

});

**Then**(/^I see login page$/, function () {

});

**And**(/^I enter \"([^\"]\*)\" data to gmail input$/, function () {

cy.**get**('#email').**type**('admin@enouvo.com')

});

**And**(/^I enter \"([^\"]\*)\" data to password input$/, function () {

cy.**get**('#password').**type**('enouvo123')

});

**And**(/^I click on submit button$/, function () {

cy.**get**('.login-form-button').**click**()

});

**And**(/^I click on profile button$/, function () {

cy.**xpath**("//span[@class='ant-avatar ant-avatar-lg ant-avatar-circle ant-avatar-image ant-avatar-icon']//img").**click**()

 });

**And**(/^I click on logout button$/, function () {

cy.**get**(':nth-child(3) > .div-menu-item').**click**()

});

#### *3.3.3.4. Kết quả:*

Running: CSM\_feature\logout.feature... (6 of 6)

Logout

√ Logout (8596ms)

1 passing (17s)

(Results)

┌──────────────────────────────────────────┐

│ Tests: 1 │

│ Passing: 1 │

│ Failing: 0 │

│ Pending: 0 │

│ Skipped: 0 │

│ Screenshots: 0 │

│ Video: true │

│ Duration: 17 seconds │

│ Spec Ran: CSM\_feature\logout.feature │

└──────────────────────────────────────────┘

(Video)

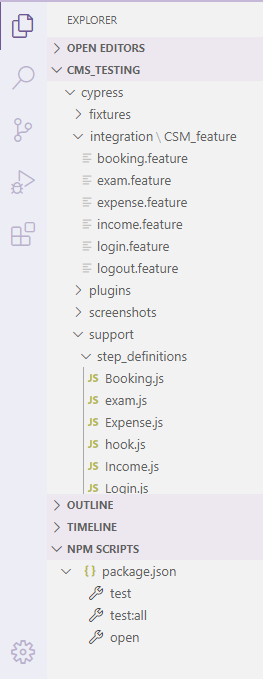
- Started processing: Compressing to 32 CRF

- Finished processing: F:\CMS\_Testing\cypress\videos\CSM\_feature\logout.feature.mp4 (6 seconds)

|  |  |
| --- | --- |
| Feature | Số lượng Testcase |
| Login | 20 |
| Income | 37 |
| Expense | 41 |
| Booking | 34 |
| Logout | 2 |

* Để kiểm thử cho tất cả ứng dụng, hệ điều hành, trình duyệt và cơ sở dữ liệu khác nhau, số lượng kịch bản kiểm thử sẽ rất lớn. Bên cạnh đó các feature mới cũng phải tiến hành kiểm thử song song với các feature cũ. Có thể thấy nguồn lực cho kiểm thử hồi quy (Regression test) và các kiểm thử cho feature mới sẽ rất lớn.
* Từ những lợi ích của cypress, tôi và các bạn đồng nghiệp đã xây dựng Framework tích hợp điểm mạnh của cypress để hỗ trợ quá trình kiểm thử, build dự án, phát hành sản phẩm với chất lượng tốt hơn.
* Dưới đây là kết quả cài đặt, chạy thử và đánh giá kết quả của framework. Do quy định về luật bảo mật của dự án và quy định vê luật bảo mật của công ty, tôi sẽ không sử dụng mã nguồn chi tiết trong dự án đang triển khai ở công ty.

## CẤU TRÚC THƯ MỤC



* /fixtures: chứa data mẫu
* /integration: nơi chứa E2E test chính. Thư mục /examples sẽ được tạo ra khi bạn chạy **Cypress** lần đầu.
* /plugins: hỗ trợ cho **Cypress**
* /support/index.js: nơi để setup cho mọi test case.
* support/commands.js: cho phép override là các hàm có sẵn của **Cypress**

## TÍCH HỢP GHERKIN

Dự án có sử dụng Maven để chạy test tích hợp giữa Cypress và Jenkins

## 3.6. REPORT KẾT QUẢ CHẠY TEST

- Sign in

## 3.7 NHỮNG KHÓ KHĂN KHI TRIỂN KHAI HỆ THỐNG TRONG CÔNG TY

* Việc tìm nhân lực vừa có kỹ năng kiểm thử vừa có kỹ năng lập trình gặp khó khăn bởi nhiều bạn nhân viên kiểm thử có kỹ năng kiểm thử tốt nhưng tư duy lập trình không tốt, các bạn có kỹ năng lập trình tốt thì không muốn làm kiểm thử
* Khi một feature có nhiều sự thay đổi về cách tổ chức UI thì mất nhiều thời gian để maintain mã nguồn của framework.
* Kiểm thử tự động sẽ chỉ có hiệu quả cao nhất đối với dự án dài, có regression test nhiều. Các dự án ngắn, đòi hỏi release sớm thì việc triển khai kiểm thử tự động sẽ không hiệu quả
* Hệ thống hiện tại đang chạy ổn định trên firefox và chrome, vẫn còn một số lỗi khi chạy trên IE do việc xử lý trên IE và Firefox có một số điểm khác nhau.

## 3.8 HƯỚNG PHÁT TRIỂN TIẾP THEO CỦA FRAMEWORK

* Xây dựng thêm phần DataDriven trong framework. DataDriven giúp cho hệ thống có thể đọc được dữ liệu đầu vào từ hệ thống quản lý file hoặc từ một hệ quản trị cơ sở dữ liệu bất kỳ
* Cải tiến framework để hỗ trợ cho các bạn kiểm thử không có kỹ năng lập trình có thể sử dụng dễ dàng hơn.

**KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

Trong quá trình tìm hiểu và nghiên cứu, báo cao thực tập đã đưa ra những khái niệm và hướng áp dụng của Cypress và các công cụ liên quan khác trong kiểm thử phần mềm. Dựa vào kết quả nghiên cứu và áp dụng kiểm thử tự động tại công ty Enouvo, tôi thấy rằng việc ứng dụng cypress và các công cụ liên quan trong các dự án phần mềm là hoàn toàn khả thi. Do thời gian có hạn, tôi chỉ đưa ra một framework cơ bản nhất để có thể áp dụng và chạy thử nghiệm luôn, framework vẫn còn nhiều phần cần phải cải tiến và cập nhật thêm. Việc cải tiến framework sẽ được nghiên cứu và cập nhật trong thời gian tới.

TÀI LIỆU THAM KHẢO

**Internet**

**Tiếng Việt**

<https://vntesters.com/hoc-gi-de-tro-thanh-automation-tester/>

<https://viblo.asia/p/cach-tao-tai-lieu-chien-luoc-kiem-thu-test-strategy-document-oOVlYGkzK8W>

<https://toidicodedao.com/2018/01/16/tong-quan-testing-3-cong-viec-cua-tester-va-automation-test/>

<https://toidicodedao.com/2018/01/09/tong-quan-ve-testing-phan-2-phan-biet-cac-loai-testing-va-thuat-ngu-chuyen-nganh-test/>

<https://toidicodedao.com/2018/01/16/tong-quan-testing-3-cong-viec-cua-tester-va-automation-test/>

<https://vntesters.com/tu-hoc-kiem-thu-phan-mem/>

**Tiếng Anh**

<https://www.cypress.io/>

<https://docs.cypress.io/guides/getting-started/installing-cypress.html#System-requirements>

PHỤ LỤC

Phần này bao gồm những nội dung cần thiết nhằm minh họa hoặc bổ trợ cho nội dung luận văn như số liệu, mẫu biểu, tranh ảnh... Phụ lục không được dày hơn phần chính của luận văn. Phụ lục đuợc đánh số trang tiếp với đồ án.

1. Nội dung mã nguồn 01

2. Nội dung mã nguồn 02