

**ĐẠI HỌC QUỐC GIA HÀ NỘI TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**Vũ Thị Thùy**

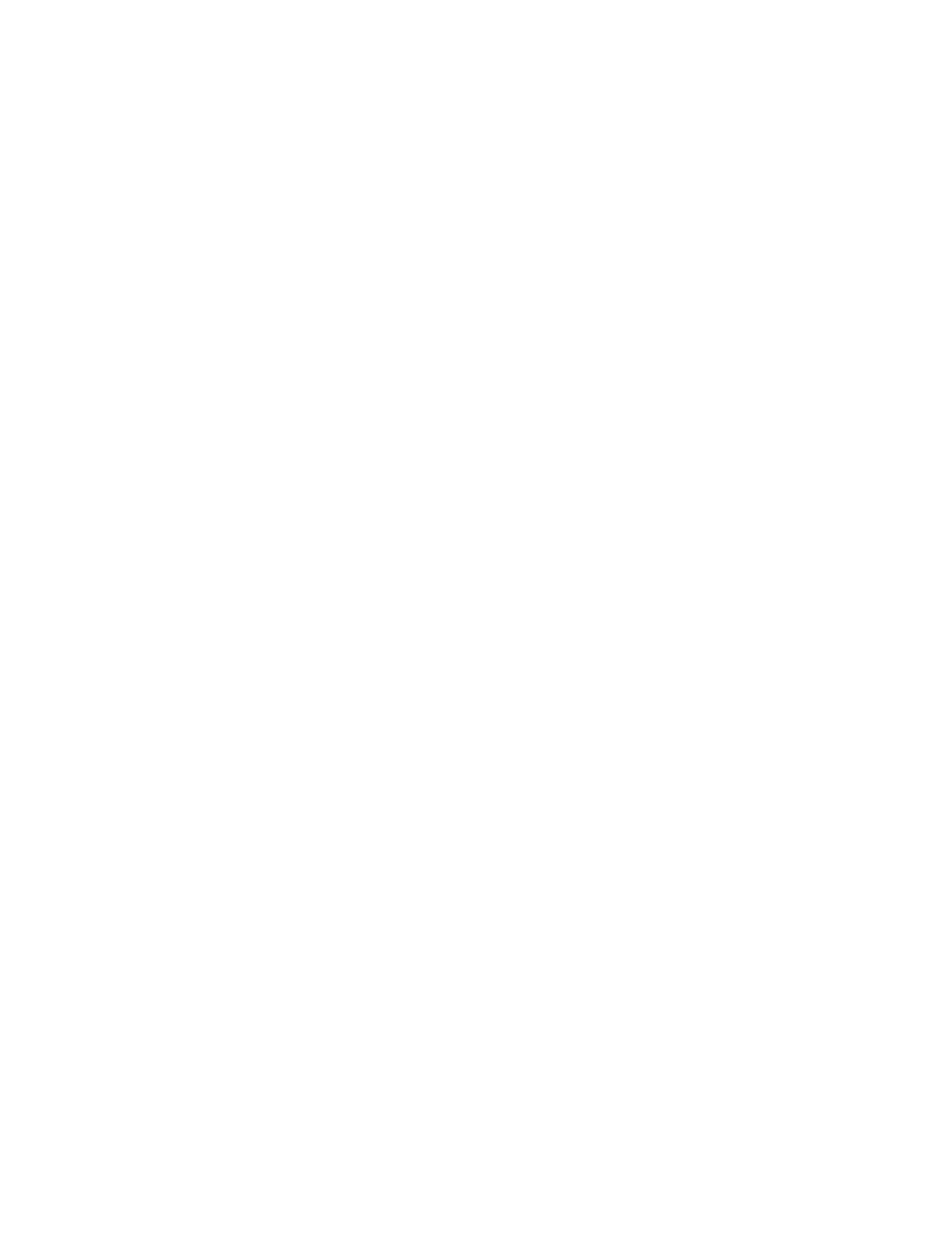
**TỰ ĐỘNG HÓA CÁC CA KIỂM THỬ CƠ BẢN**

**CỦA ỨNG DỤNG WEB VỚI CYPRESS**

**KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY**

**Ngành: Công nghệ thông tin**

**HÀ NỘI – 2018**



**ĐẠI HỌC QUỐC GIA HÀ NỘI TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**Vũ Thị Thùy**

**TỰ ĐỘNG HÓA CÁC CA KIỂM THỬ CƠ BẢN**

**CỦA ỨNG DỤNG WEB VỚI CYPRESS**

**KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY**

**Ngành: Công Nghệ Thông Tin**

**Cán bộ hướng dẫn: PSG.TS Trương Anh Hoàng**

**Cán bộ đồng hướng dẫn: ThS. Nguyễn Ngọc Khải**

**HÀ NỘI - 2018**

# LỜI CÁM ƠN

Trước tiên, em xin gửi lời cám ơn chân thành tới PGS.TS Trương Anh Hoàng – giảng viên bộ môn Công nghệ Phần mềm và ThS. Nguyễn Ngọc Khải – người đã hướng dẫn tận tình, tỉ mỉ, chu đáo cho em trong suốt quá trình làm khóa luận. Sau quãng thời gian được thầy hướng dẫn em đã đúc kết, học hỏi được nhiều kinh nghiệm về phương pháp nghiên cứu, kĩ năng giao tiếp, kĩ năng trình bày.

Ngoài ra, em xin gửi lời cám ơn chân thành đến tập thể lớp K59 - CD đã giúp đỡ em hoàn thành khóa luận sao cho đạt hiệu quả cao nhất. Các bạn đã giúp đỡ em bằng hành động, bằng lời nói mỗi khi em gặp khó khăn, thất bại. Bốn năm bên nhau không phải là dài nhưng đối với em, đây là quãng thời gian tuyệt vời nhất và không thể nào quên.

Tiếp theo, em xin gửi lời cảm ơn đến các thầy cô giảng viên Trường Đại học Công Nghệ - Đại học Quốc Gia Hà Nội – những người đã tận tâm truyền đạt những kiến thức quý báu làm nền tảng để em tiếp tục đi xa hơn nữa trong lĩnh vực công nghệ thông tin.

Cuối cùng, em xin được cảm ơn gia đình đã nuôi em khôn lớn để trở thành người có ích cho xã hội, giúp em có một điểm tựa vững chắc để yên tâm học hành trong suốt bao năm qua. Em xin gửi lời cám ơn chân thành tới cha, mẹ đã luôn động viên và cổ vũ em mỗi khi em gặp khó khăn và thử thách. Mặc dù đã cố gắng để hoàn thiện đề tài tốt nhất có thể, nhưng chắc chắn không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự góp ý của các thầy cô và các bạn để báo cáo đồ án tốt nghiệp này của em được hoàn thiện hơn.

|  |  |
| --- | --- |
|  | Hà Nội, ngày 10 tháng 4 năm 2018 |
|  | Sinh viên |
|  | Vũ Thị Thùy |

# TÓM TẮT

Ngày nay, việc tự động hóa đang được nghiên cứu và ứng dụng vào nhiều lĩnh vực, trong đó không thể không nhắc tới công nghệ phần mềm nói chung và kiểm thử phần mềm nói riêng. Khi mà kiểm thử phần mềm vẫn tiêu tốn một lượng lớn thời gian, kinh phí và nhân lực trong một dự án phần mềm thì kiểm thử tự động ra đời đã đem lại những lợi ích to lớn. Trong các công cụ kiểm thử tự động, Cypress là một công cụ kiểm thử tự động mới. Cypress được biết đến là một công cụ kiểm thử tự động các ứng dụng Web, sử dụng ngôn ngữ JavaScript.

Trên cơ sở tính hữu ích, đơn giản và hiện đại của công cụ Cypress được sự gợi ý từ giáo viên hướng dẫn, em đã tìm hiểu và chọn đề tài “***Tự động hóa các ca kiểm thử cơ bản của ứng dụng Web với Cypress***.”. Nội dung khóa luận sẽ tập trung trình bày nhưng đặc điểm nổi bật, cách sử dụng công cụ kiểm thử tự động Cypress và đồng thời áp dụng những kiến thức đã tìm hiểu được vào kiểm thử một hệ thống Web thực tế.

***Từ khóa***: *Tự động hóa, kiểm thử tự động, ứng dụng Web, Cypress.*

# LỜI CAM ĐOAN

Em xin cam đoan rằng những nghiên cứu về kiểm thử tự động bằng Cypress được trình bày trong luận án này là của riêng em và được sự hướng dẫn của PGS.TS Trương Anh Hoàng cùng ThS Nguyễn Ngọc Khải. Các kết quả nêu trong khóa luận là hoàn toàn trung thực. Nôi dung khóa luận có tham khảo và sử dụng một số thông tin, tài liệu từ các nguồn sách, tạp chí được liệt kê đầy đủ trong danh mục các tài liệu tham khảo.

Em xin cam đoan công cụ kiểm thử tự động em trình bày trong khoá luận là do em tự phát triển, không sao chép mã nguồn của người khác. Nếu sai em hoàn toàn chịu trách nhiệm theo quy định của trường Đại học Công Nghệ - Đại học Quốc Gia Hà Nội.

|  |  |
| --- | --- |
|  | Hà Nội, ngày 27 tháng 3 năm 2018 |
|  | Sinh viên |
|  | Vũ Thị Thùy |

MỤC LỤC

[Chương 1: Mở đầu 1](#_Toc512469212)

[1.1 Lý do chọn đề tài 1](#_Toc512469213)

[1.2 Mục tiêu của đề tài 1](#_Toc512469214)

[1.3 Phạm vi và cấu trúc đề tài 2](#_Toc512469215)

[1.4 Phương pháp nghiên cứu 2](#_Toc512469216)

[Chương 2: Tổng quan về kiểm thử phần mềm 4](#_Toc512469217)

[2.1 Định nghĩa về kiểm thử phần mềm 4](#_Toc512469218)

[2.2 Mục đích của kiểm thử phần mềm 4](#_Toc512469219)

[2.3 Các kỹ thuật kiểm thử phần mềm 4](#_Toc512469220)

[2.3.1 Kiểm thử hộp đen 4](#_Toc512469221)

[2.3.2 Kiểm thử hộp trắng 5](#_Toc512469222)

[2.3.3 Kiểm thử hộp xám 5](#_Toc512469223)

[2.4 Các mức kiểm thử 5](#_Toc512469224)

[2.5 Quy trình kiểm thử 5](#_Toc512469225)

[2.6 Kiểm thử tự động và các công cụ hỗ trợ 6](#_Toc512469226)

[2.6.1 Tổng quát về kiểm thử tự động 6](#_Toc512469227)

[2.6.2 Ưu điểm, nhược điểm của kiểm thử tự động 8](#_Toc512469228)

[2.6.3 Một số công cụ kiểm thử tự động 9](#_Toc512469229)

[Chương 3: Công cụ kiểm thử tự động Cypress 15](#_Toc512469230)

[3.1 Tìm hiểu công cụ kiểm thử tự động Cypress 15](#_Toc512469231)

[3.2 Các hệ điều hành và trình duyệt hỗ trợ 15](#_Toc512469232)

[3.2.1 Các hệ điều hành hỗ trợ ứng dụng 15](#_Toc512469233)

[3.2.2 Các trình duyệt được hỗ trợ 15](#_Toc512469234)

[3.3 Đặc điểm của Cypress 16](#_Toc512469235)

[3.4 Ngôn ngữ sử dụng viết mã lệnh 18](#_Toc512469236)

[3.5 Sự khác biệt giữa Cypress và Selenium WebDriver 18](#_Toc512469237)

[3.5.1 So sánh giữa Cypress và Selenium WebDriver 18](#_Toc512469238)

[3.5.2 Ưu và nhược điểm của công cụ Cypress 20](#_Toc512469239)

[3.6 Những lệnh cơ bản trong Cypress 21](#_Toc512469240)

[3.7 Tạo một kịch bản kiểm thử 22](#_Toc512469241)

[Chương 4: Ứng dụng Cypress cho một số chức năng trên hệ thống Quản lý điều trị thuốc Methadone 25](#_Toc512469242)

[4.1 Giới thiệu bài toán 25](#_Toc512469243)

[4.1.1 Mục đích bài toán 25](#_Toc512469244)

[4.1.2 Giới thiệu chung về dự án 25](#_Toc512469245)

[4.2 Hướng giải quyết 26](#_Toc512469246)

[4.2.1 Tổng quát 26](#_Toc512469247)

[4.2.2 Chi tiết từng bước thực hiện 28](#_Toc512469248)

[Chương 5: Kết luận 43](#_Toc512469249)

# DANH MỤC HÌNH ẢNH

[Hình 2.1 Kiến trúc chung của một bộ kiểm thử tự động 7](#_Toc512432896)

[Hình 2.2 Công cụ kiểm thử tự động QuickTest Professional 10](#_Toc512432897)

[Hình 2.3 Mô tả công cụ kiểm thử hiệu năng Jmeter 11](#_Toc512432898)

[Hình 2.4 Công cụ kiểm thử tự động LoadRunner 12](#_Toc512432899)

[Hình 2.5 Công cụ kiểm thử tự động Selenium 13](#_Toc512432900)

[Hình 2.6 Công cụ kiểm thử tự động Junit 14](#_Toc512432901)

[Hình 3.1 Sự khác biệt giữa Cypress với các công cụ kiểm thử tự động khác 18](#_Toc512432902)

[Hình 4.1 Giao diện chính của hệ thống Methadone 26](#_Toc512432903)

[Hình 4.2 Giao diện màn hình chức năng thêm mới bệnh nhân 29](#_Toc512432904)

[Hình 4.3 Màn hình chức năng tìm kiếm 33](#_Toc512432905)

[Hình 4.4 Màn hình kết quả sau khi chạy mã lệnh đăng nhập 37](#_Toc512432906)

[Hình 4.5 Màn hình kết quả sau khi chạy mã lệnh thất bại 38](#_Toc512432907)

[Hình 4.6 Cấu trúc mã nguôn cho các ca kiểm thử chức năng 38](#_Toc512432908)

[Hình 4.7 Cấu trúc mã nguồn cho các ca kiểm thử giao diện 39](#_Toc512432909)

[Hình 4.8 Màn hình khi mã lệnh chạy lỗi 1 40](#_Toc512432910)

[Hình 4.9 Màn hình khi mã lệnh chạy lỗi 2 41](#_Toc512432911)

# DANH MỤC BẢNG

[Bảng 4.1 Mô phỏng cách đưa các trường hợp kiểm thử thành mã lệnh 27](#_Toc512432301)

[Bảng 4.2 Phân tích mã lệnh lỗi để tìm nguyên nhân 28](#_Toc512432302)

[Bảng 4.3 Các ca kiểm thử giao diện cơ bản 29](#_Toc512432303)

[Bảng 4.4 Các ca kiểm thử chức năng tìm kiếm 34](#_Toc512432304)

[Bảng 4.5 Bảng báo cáo kết quả kiểm thử các chức năng đã thực hiện 41](#_Toc512432305)

# DANH SÁCH KÝ HIỆU, TỪ VIẾT TẮT

|  |  |
| --- | --- |
| DOM | Document Object Model |
| XHR | XML HTTP request |
| HTML | HyperText Markup Language |
| IE | Internet Explorer |
| QA | Quality Assurance |
| TDD | Test-driven development |
| API | Application Programming Interface |

# Chương 1: Mở đầu

## Lý do chọn đề tài

Trong giai đoạn phát triển của công nghệ thông tin, ngành công nghệ phần mềm đang chiếm một vị trí hết sức quan trọng trong xu hướng phát triển kinh tế công nghiệp hóa, hiện đại hóa của nước ta. Cùng với sự phát triển ấy là những yêu cầu, thách thức về việc đảm bảo chất lượng phần mềm ngày một tăng cao. Từ đó, kiểm thử phần mềm ra đời nhằm nâng cao chất lượng cũng như tính năng chuyên nghiệp của dự án. Nhưng thực tế cho thấy, kiểm thử phần mềm là giai đoạn chiếm đến hơn 40% thời gian, kinh phí, và nhân lực trong quá trình phát triển dự án phần mềm [7]. Vậy làm thế nào để có thể kiểm tra dự án phần mềm hoạt động đúng, đạt được hiệu quả cao nhưng cũng tiết kiệm thời gian cũng như kinh phí trong quá trình kiểm thử là một điều thiết yếu đối với các nhà kiểm thử.

Ngày nay, việc tự động hóa đang được áp dụng rộng rãi trong nhiều lĩnh vực, trong đó công nghệ phần mềm nói chung và kiểm thử phần mềm nói riêng cũng không ngoại lệ. Do việc kiểm thử thủ công tiêu tốn một lượng không nhỏ thời gian, kinh phí, nhân lực nên sự ra đời của các công cụ kiểm thử tự động như Quick Test Professional, NUnit, JUnit, LoadRunner, Selenium… là điều tất yếu [3]. Trong các công cụ kiểm thử tự động hiện có, Cypress được biết đến là một công cụ kiểm thử tự động mới, hiện đại, sử dụng để kiểm thử các ứng dụng Web. Cypress sử dụng ngôn ngữ lập trình JavaScript và là mã nguồn mở, do đó các tổ chức không cần tốn kinh phí mua bản quyền [2].

Với mong muốn có cái nhìn xác thực, rõ ràng hơn về quy trình kiểm thử phần mềm, đảm bảo chất lượng phần mềm và tiếp cận với công cụ kiểm thử tự động Cypress, đồng thời rèn luyện kỹ năng làm việc tạo tiền đề cho định hướng tương lai sau khi ra trường nên em chọn đề tài “Tự động hóa các ca kiểm thử cơ bản của ứng dụng Web với Cypress”

## Mục tiêu của đề tài

Đề tài được thực hiện nhằm mục tiêu tìm hiểu về kiểm thử phần mềm, kiểm thử tự động phần mềm, đặc biệt là kiểm thử tự động các ứng dụng Web. Tập trung nghiên cứu vai trò, phương pháp cũng như ứng dụng Cypress trong việc kiểm thử ứng dụng Web. Từ đó, thực hiện phân tích, xây dựng các trường hợp kiểm thử cho các chức năng của một Website, sử dụng ngôn ngữ lập trình JavaScript tạo các mã lệnh từ các ca kiểm tra cơ bản để kiểm tra khả năng thực hiện đúng và đủ của từng chức năng. Từ đó đánh giá được chất lượng của Website đó.

## Phạm vi và cấu trúc đề tài

Để thực hiện được các mục tiêu của đề tài, các công việc cần được quan tâm là:

* Tìm hiểu tổng quan về kiểm thử phần mềm nói chung và kiểm thử tự động ứng dụng Web nói riêng. Bên cạnh đó cũng tìm hiểu vai trò của kiểm thử tự động trong kiểm thử phần mềm. Các công cụ hỗ trợ kiểm thử phần mềm.
* Tìm hiểu về Cypress, tính năng, hoạt động, ưu nhược điểm và các sử dụng Cypress.
* Thực hiện phân tích xây dựng bộ các ca kiểm thử cơ bản từ đó tạo các mã lệnh để thực hiện kiểm thử phát hiện lỗi, xây dựng báo cáo kiểm thử.

## Phương pháp nghiên cứu

**Phương pháp nghiên cứu lý thuyết:**

* Tiến hành thu thập và nghiên cứu các tài liệu có liên quan đến đề tài.
* Tổng hợp các tài liệu.
* Chọn lọc các tài liệu nghiên cứu để báo cáo đề tài khóa luận tốt nghiệp.

**Phương pháp nghiên cứu thực nghiệm:**

* Tìm hiểu cách thức hoạt động, cách sử dụng công cụ kiểm thử tự động Cypress.
* Ứng dụng công cụ Cypress vào kiểm thử phần mềm.
* Kiểm tra và đánh giá kết quả đạt được.

Với mục tiêu và phạm vi nêu trên, bố cục của khóa luận gồm 5 chương bao gồm:

**Chương 1: Mở đầu**

Chương 1 giới thiệu tổng quan về khóa luận tốt nghiệp bao gồm tên đề tài, lý do chọn đề tài, phương pháp thực hiện.

**Chương 2: Tổng quan về kiểm thử phần mềm**

Chương 2 giới thiệu các kiến thức tổng quan về phần mềm, kiểm thử phần mềm, kiểm thử tự động và các công cụ hỗ trợ.

**Chương 3: Công cụ kiểm thử tự động Cypress**

Chương 3 giới thiệu công cụ kiểm thử tự động Cypress: các tính năng, đặc điểm, ưu, nhược điểm và cách vận hành của Cypress.

**Chương 4: Ứng dụng Cypress vào hệ thống quản lý điều trị Methadone**

Chương 4 giới thiệu cách vận hành Cypress vào một dự án thực cụ thể và cho biết hiệu quả của nó trong thực tế.

**Chương 5: Kết luận**

Trình bày những kết quả đạt được, những thiếu sót chưa thực hiện được. Từ đó đưa ra hướng giải quyết và phát triển tiếp đề tài trong tương lai.

# Chương 2: Tổng quan về kiểm thử phần mềm

## Định nghĩa về kiểm thử phần mềm

Kiểm thử phần mềm là quá trình thẩm định phần mềm để xác định nó có được xây dựng đúng theo tài liệu yêu cầu và đáp ứng được nhu cầu của người sử dụng hay chưa [3].

## Mục đích của kiểm thử phần mềm

Kiểm thử không chỉ giới hạn ở việc thực hiện một chương trình hoặc ứng dụng với mục đích đi tìm các lỗi phần mềm (bao gồm các lỗi và các thiếu sót) mà còn là một quá trình phê chuẩn và xác minh một chương trình máy tính/ứng dụng/sản phẩm nhằm:

* Tìm ra được càng nhiều lỗi càng tốt trong điều kiện về thời gian đã định và nguồn lực sẵn có.
* Chứng minh rằng sản phẩm phần mềm phù hợp với các đặc tả của nó.
* Xác thực chất lượng kiểm thử phần mềm đã dùng chi phí và nỗ lực tối thiểu.
* Thiết kế tài liệu kiểm thử một cách có hệ thống và thực hiện nó sao cho có hiệu quả, tiết kiệm được thời gian, công sức [3].

## Các kỹ thuật kiểm thử phần mềm

Có 3 kỹ thuật kiểm thử phần mềm chính là: Kiểm thử hộp đen. kiểm thử hộp trắng, kiểm thử hộp xám [1].

### Kiểm thử hộp đen

Kỹ thuật kiểm thử hộp đen xem chương trình như là một “hộp đen”, trong đó người kiểm thử không quan tâm đến cấu trúc bên trong của chương trình mà chỉ quan tâm tới dữ liệu đầu vào và đầu ra sau khi được xử lý.

Ưu, nhược điểm: kiểm thử hộp đen có ưu điểm là có thể đánh giá phần mềm một cách khách quan, người kiểm thử có thể không hiểu biết về mã lệnh và có thể tìm ra các lỗi mà nhân viên phát triển không tìm ra. Song kiểm thử hộp đen lại có nhược điểm là thăm dò mù, do nhân viên kiểm thử không biết các chương trình thực sự được xây dựng như thế nào, dẫn đến trường hợp nếu kiểm thử hộp đen phải viết rất nhiều trường hợp kiểm thử trong khi chỉ cần viết một ca kiểm thử duy nhất để có thể kiểm tra được [3].

### Kiểm thử hộp trắng

Kỹ thuật kiểm thử hộp trắng hay còn gọi là “kiểm thử cấu trúc” là kỹ thuật kiểm thử cho phép khảo sát kiến trúc bên trong của chương trình. Kiểm thử hộp trắng là chiến lược được thực hiện trên ba trong sáu loại kiểm thử cơ bản trong các giai đoạn kiểm thử phần mềm là: kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử hồi quy. Mục tiêu của kiểm thử hộp trắng là kiểm thử bao phủ nhiều nhất các câu lệnh, điểm quyết định và các rẽ nhánh trong mã nguồn nếu có thể [1].

### Kiểm thử hộp xám

Kiểm thử hộp xám là kỹ thuật kiểm thử có sự kết hợp giữa kiểm thử hộp đen và kiểm thử hộp trắng. Trong đó ta cũng quan tâm đến dữ liệu đầu vào và đầu ra giống như trong kiểm thử hộp đen, song lại đòi hỏi có sự truy cập đến cấu trúc dữ liệu và giải thuật để thiết kế các trường hợp kiểm thử [1].

## Các mức kiểm thử

Một trong các khái niệm then chốt của việc kiểm thử là các mức của việc kiểm thử. Các mức của việc kiểm thử phản ánh mức độ trừu tượng được thấy trong mô hình thác nước của vòng đời của việc phát triển phần mềm [1].

Các mức kiểm thử có thể được mô tả sơ bộ như sau:

* Kiểm thử đơn vị (Unit Test): nhóm phát triển có trách nhiệm thực hiện trước khi bàn giao chương trình cho nhóm kiểm thử.
* Kiểm thử chức năng (Functional Test), kiểm thử phi chức năng: nhóm kiểm thử có trách nhiệm thực hiện.
* Kiểm thử tích hợp (Integration Test): áp dụng thử nghiệm với các dự án CMMI. Điều kiện thực hiện kiểm thử tích hợp là phải có báo cáo kiểm thử chức năng với kết luận đạt tiêu chuẩn để tích hợp.
* Kiểu kiểm thử nghiệm thu (Acceptance Test): sẽ được thực hiện bởi phía khách hàng.

## Quy trình kiểm thử

Quy trình kiểm thử phần mềm tổng quát [1]:

*Bước 1.* Lập kế hoạch kiểm thử (Test Planning)

*Bước 2.* Phân tích và thiết kế các ca kiểm thử (Test analysis and Design).

*Bước 3.* Thực thi kiểm thử (Test Executing).

*Bước 4.* Báo cáo kiểm thử, đánh giá (Test Report and Evaluation) [4].

## Kiểm thử tự động và các công cụ hỗ trợ

### Tổng quát về kiểm thử tự động

Kiểm thử tự động là quá trình thực hiện một cách tự động các bước trong một kịch bản kiểm thử. 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 [3].

Quy trình kiểm thử tự động gồm 4 bước:

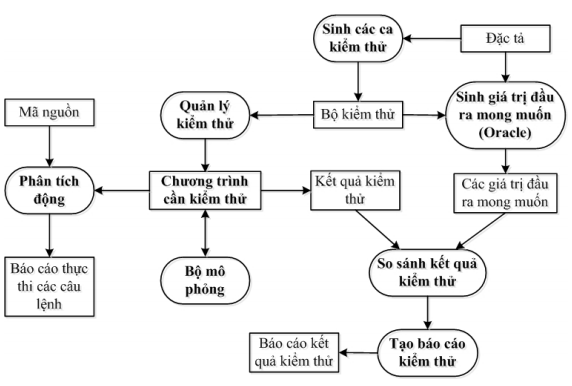
*Bước 1.* Viết kịch bản kiểm thử, dùng công cụ kiểm thử để ghi lại các thao tác lên phần mềm cần kiểm tra và tự động sinh ra một nhóm mã lệnh để đặc tả kịch bản (testscript).

*Bước 2.* Chỉnh sửa để kịch bản kiểm thử thực hiện kiểm tra theo đúng yêu cầu đặt ra, làm theo trường hợp kiểm thử cần thực hiện.

*Bước 3.* Chạy kịch bản kiểm thử, giám sát hoạt động kiểm tra phần mềm của kịch bản kiểm thử.

*Bước 4.* Kiểm tra kết quả thông báo sau khi thực hiện kiểm thử tự động. Sau đó bổ sung, chỉnh sửa những sai sót.

Trong thực tế, có rất nhiều bộ công cụ hỗ trợ kiểm thử tự động được phát triển nhằm góp phần giải quyết các vấn đề khó khăn của quy trình kiểm thử. Hình 1.1 mô tả kiến trúc chung nhất của một bộ kiểm thử tự động [1]. Trong kiến trúc này, các công cụ kiểm thử được tích hợp trong một quy trình thống nhất nhằm hỗ trợ đầy đủ các hoạt động kiểm thử trong quy trình kiểm thử các sản phẩm phần mềm.

**

Hình 2.1 Kiến trúc chung của một bộ kiểm thử tự động [1].

Các công cụ cơ bản trong kiến trúc này bao gồm [1]:

* Quản lý kiểm thử: công cụ này cho phép quản lý việc thực hiện/thực thi các ca kiểm thử. Nó giám sát việc thực hiện từng ca kiểm thử ứng với bộ giá trị đầu vào, giá trị đầu ra mong muốn và giá trị đầu ra thực tế. JUnit là một ví dụ điển hình về công cụ này.
* Sinh các ca kiểm thử: Đây là một trong những công cụ quan trọng nhất của các bộ kiểm thử tự động. Tùy thuộc vào các kỹ thuật kiểm thử được áp dụng, công cụ này sẽ sinh ra tập các ca kiểm thử (chưa gồm giá trị đầu ra mong muốn) cho chương trình/đơn vị chương trình cần kiểm thử. Các ca kiểm thử được sinh ra chỉ chứa giá trị đầu vào để thực hiện nó. Các giá trị này có thể được lựa chọn trong cơ sở dữ liệu hoặc được sinh một cách ngẫu nhiên.
* Sinh giá trị đầu ra mong muốn: Các ca kiểm thử được sinh ra bởi công cụ trên chỉ chứa các giá trị đầu vào. Công cụ này cho phép sinh ra giá trị đầu ra mong muốn ứng với mỗi bộ dữ liệu đầu vào của mỗi ca kiểm thử. Giá trị đầu ra mong muốn này sẽ được so sánh với giá trị đầu ra thực tế khi thực hiện ca kiểm thử này nhằm phát hiện ra các lỗi/khiếm khuyết của sản phẩm.
* So sánh kết quả kiểm thử: Công cụ này so sánh giá trị đầu ra thực tế và giá trị đầu ra mong muốn của mỗi ca kiểm thử khi nó được thực hiện trên chương trình/đơn vị chương trình cần kiểm thử.
* Tạo báo cáo kiểm thử: Một trong những ưu điểm của các bộ công cụ kiểm thử tự động là nó có cơ chế sinh báo cáo kiểm thử một cách chính xác và nhất quán. Dựa vào kết quả của công cụ so sánh kết quả kiểm thử, công cụ này sẽ tự động sinh ra báo cáo kết quả kiểm thử theo định dạng mong muốn của đơn vị phát triển.
* Phân tích động: Công cụ này cung cấp một cơ chế nhằm kiểm tra việc thực hiện của các câu lệnh của chương trình cần kiểm thử nhằm phát hiện ra các lỗi và phát hiện các câu lệnh/đoạn lệnh không được thực hiện bới một tập các ca kiểm thử cho trước. Công cụ này cũng rất hiệu quả trong việc đánh giá tính hiệu quả của một bộ kiểm thử cho trước.
* Bộ mô phỏng: Có nhiều loại mình mô phỏng được cung cấp trong các bộ kiểm thử tự động. Mục đích của các công cụ này là mô phỏng quá trình thực hiện của chương trình cần kiểm thử. Ví dụ, các công cụ mô phỏng giao diện người dùng cho phép thực hiện tự động các tương tác giữa người dùng và sản phẩm. Cypress là một ví dụ về một công cụ mô phỏng giao diện người dùng cho các ứng dụng Web.

Trong thực tế, các bộ công cụ kiểm thử tự động có thể có thêm một số công cụ khác như cho phép đặc tả các tính chất của hệ thống cần kiểm thử…. Một số bộ công cụ chỉ hỗ trợ một số công cụ trong các công cụ đã liệt kê ở trên [1].

### Ưu điểm, nhược điểm của kiểm thử tự động

#### Ưu điểm

Kiểm thử tự động mang lại những ưu điểm sau cho dự án phần mềm [3]:

* Độ tin cậy cao (Reliability): Công cụ kiểm thử tự động có sự ổn định cao hơn sơ với con người, đặc biệt trong trường hợp có nhiều ca kiểm thử, nên độ tin cậy cao hơn so với kiểm thử thủ công.
* Khả năng lặp (Repeatability): Công cụ kiểm thử tự động ra đời là để giúp cho các kiểm thử viên không phải lặp đi lặp lại các thao tác (ví dụ: nhập dữ liệu, click, check kết quả, …) một cách nhàm chán với độ tin cậy và ổn định cao.
* Khả năng tái sử dụng (Reusability): Với một bộ kiểm thử tự động, người ta có thể sử dụng cho nhiều phiên bản ứng dụng khác nhau, đây chính là tính tái sử dụng.
* Tốc độ cao (High speed): Do thực thi bởi công cụ nên tốc độ của kiểm thử tự động nhanh hơn nhiều so với tốc độ của con người.
* Chi phí thấp (Cost Reduction): Nếu áp dụng kiểm thử tự động đúng cách, người ta có thể tiết kiệm được nhiều chi phí, thời gian và nhân lực, do kiểm thử tự động nhanh hơn nhiều so với kiểm thử thủ công, đồng thời nhân lực cần để thực thi và bảo trì không nhiều.

#### Nhược điểm:

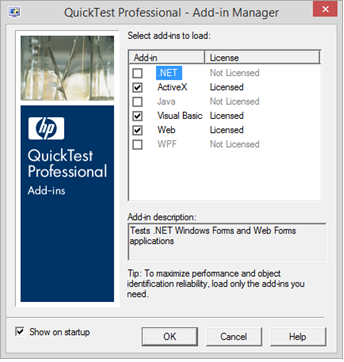
Bên cạnh đó, kiểm thử tự động cũng vẫn tồn tại những nhược điểm sau [3]:

* Khó mở rộng, khó bảo trì (Poor scalability and maintainability): Trong cùng một dự án, để mở rộng phạm vi cho kiểm thử tự động khó hơn nhiều so với kiểm thử thủ công vì cập nhật hay chỉnh sửa yêu cầu nhiều công việc như debug, thay đổi dữ liệu đầu vào và cập nhật code mới.
* Khả năng bao phủ thấp (Low coverage): Do khó mở rộng và đòi hỏi nhiều kỹ năng lập trình nên độ bao phủ của kiểm thử tự động thấp xét trên góc nhìn toàn dự án.
* Vấn đề công cụ và nhân lực (Technology and people issues): Hiện nay cũng có nhiều công cụ hỗ trợ kiểm thử tự động khá tốt nhưng chúng vẫn còn nhiều hạn chế. Ngoài ra nhân lực đạt yêu cầu (có thể sử dụng thành thạo các công cụ này) cũng không nhiều.

### Một số công cụ kiểm thử tự động

#### Quick Test Professional (QTP)

Quick Test Professional là phần mềm kiểm soát việc kiểm thử tự động các chức năng của các sản phẩm phần mềm cần kiểm thử. Sản phẩm này bao gồm một tập các mô-đun có thể tương tác với nhau nhằm quản lý toàn bộ quy trình kiểm thử phần mềm. Quick Test Professional là một công cụ hỗ trợ kiểm thử hàm (kiểm thử chức năng) và cho phép tiến hành kiểm thử hồi quy một cách tự động [1].

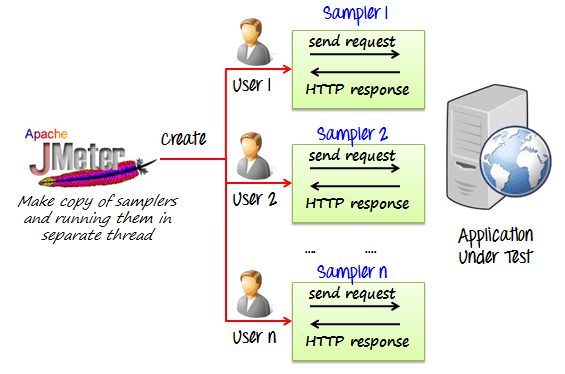


Hình 2.2 Công cụ kiểm thử tự động QuickTest Professional [11]

#### Jmeter

Jmeter là một mã nguồn mở được viết bằng Java. Công cụ để đo độ tải và performance của đối tượng, có thể sử dụng để kiểm thử hiệu năng trên cả nguồn tĩnh và nguồn động, có thể kiểm tra độ tải và hiệu năng trên nhiều loại server khác nhau như: Web – HTTP, HTTPS, SOAP, Database thông qua JDBC, LDAP, JMS, Mail – SMTP(S), POP3(S) and IMAP(S)…

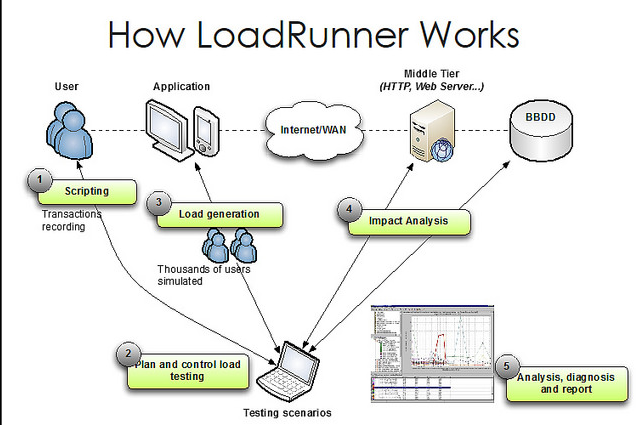
Cách thức hoạt động của Jmeter: Giả lập một nhóm người dùng gửi các yêu cầu tới một máy chủ, nhận và xử lý các response từ máy chủ và trình diễn các kết quả đó cho người dùng dưới dạng bảng biểu, đồ thị, cây… [1].



Hình 2.3 Mô tả công cụ kiểm thử hiệu năng Jmeter [5]

#### Load Runner

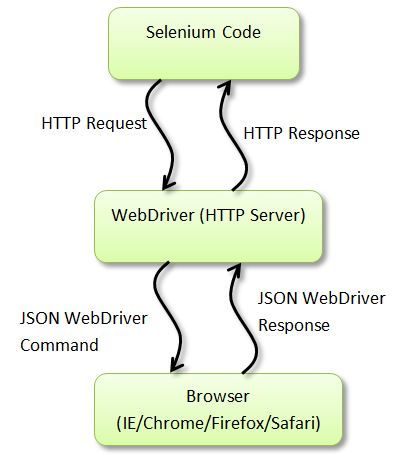
Load Runner giả lập một môi trường ảo gồm nhiều người dùng thực hiện các giao dịch cùng một lúc nhằm giám sát các thông số xử lý của phần mềm cần kiểm thử. Kết quả thống kê sẽ được lưu lại và cho phép kiểm thử viên thực hiện phân tích nhằm kiểm thử khả năng chịu tải và các yêu cầu phi chức năng khác của sản phẩm. Trong quá trình kiểm thử, Load Runner tự động tạo ra các kịch bản kiểm thử để lưu lại các thao tác người dùng tương tác lên phần mềm. Mỗi kịch bản này còn được xem là hoạt động của một người dùng ảo mà Load Runner giả lập. Ngoài ra, công cụ này còn cho phép tổ chức, điều chỉnh, quản lý và giám sát hoạt động kiểm tra khả năng chịu tải [1].



Hình 2.4 Công cụ kiểm thử tự động LoadRunner [4]

#### Selenium

Selenium là một trong những công cụ kiểm thử phần mềm tự động mã nguồn mở (open source test automation tool) mạnh mẽ nhất hiện nay cho việc kiểm thử ứng dụng Web. Selenium script có thể chạy được trên hầu hết các trình duyệt như IE, Mozilla FireFox, Chrome, Safari, Opera; và hầu hết các hệ điều hành như Windows, Mac, Linux. Selenium hỗ trợ nhiều ngôn ngữ lập trình. Selenium có 4 phần: Selenium IDE, Selenium RC, Selenium Grid và Selenium WebDriver [1].



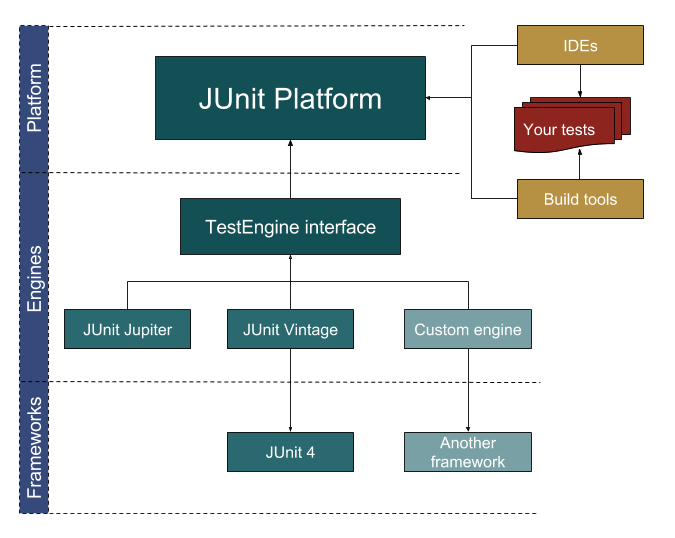
Hình 2.5 Công cụ kiểm thử tự động Selenium [6]

#### NUnit

NUnit là một khuôn khổ kiểm thử mã nguồn mở được phát triển bởi một nhóm lập trình viên (Charlie Poole, Rob Prouse and Simone Busoli), tương tác trực tiếp với Visual Studio, đồng thời có thể chạy độc lập mà không phụ thuộc vào Visual Studio [1].

#### JUnit

JUnit là một framwork đơn giản dùng cho việc tạo các ca kiểm thử đơn vị tự động, và có thể lặp đi lặp lại. Nó chỉ là một phần của kiến trúc xUnit cho việc tạo các ca kiểm thử đơn vị. Junit là một chuẩn trên thực tế cho kiểm thử đơn vị trong Java. JUnit tránh cho người lập trình phải làm đi làm lại những việc kiểm thử nhàm chán bằng cách tách biệt mã kiểm thử ra khỏi mã chương trình, đồng thời tự động hóa việc tổ chức và thi hành các bô số kiểm thử [1].



Hình 2.6 Công cụ kiểm thử tự động Junit [9]

Cypress là công cụ kiểm tra đầu cuối thế hệ tiếp theo được xây dựng cho các trang web hiện đại. Cypress là một công cụ kiểm tra miễn phí, mã nguồn mở, được cài đặt cục bộ và một dịch vụ ghi lại các bài kiểm tra của bạn. Nó sử dụng ngôn ngữ lập trình JavaScript và cho phép người dùng kiểm tra bất cứ thứ gì chạy trên trình duyệt [2].

# Chương 3: Công cụ kiểm thử tự động Cypress

## Tìm hiểu công cụ kiểm thử tự động Cypress

Cypress là một công cụ kiểm thử phần mềm tự động được sáng lập bởi Brian Man và Randall Kent. Đây là công cụ kiểm thử giao diện theo trải nghiệm người dùng tiên tiến nhất hiện nay. Sau một thời gian phát triển và chỉ tập trung vào một nhóm người dùng nhất định, Cypress đã chính thức phát hành bản beta thử nghiệm vào năm 2015 và hoàn toàn là mã nguồn mở. Bất cứ ai muốn dử dụng Cypress đều có thể tải về và sử dụng ứng dụng trên máy tính cá nhân mà không cần phải liên hệ trước hay đăng ký bất cứ thứ gì với đội ngũ sản xuất.

Cypress là một mã nguồn mở ngoại trừ ứng dụng bảng điều khiển và là một công cụ kiểm thử phần mềm tự động hóa để thử nghiệm các ứng dụng Web. Nó có khả năng hoạt động trên nhiều các trình duyệt và hệ điều hành khác nhau. Cypress có tính năng phong phú và gần gũi hơn với các công cụ phát triển hiện tại và thực tiễn hơn Selenium. Các bài kiểm tra Cypress được viết bằng Mocha và Chai, vì vậy cú pháp và định dạng sẽ quen thuộc với hầu hết các nhà phát triển Javascript [2].

## Các hệ điều hành và trình duyệt hỗ trợ

### Các hệ điều hành hỗ trợ ứng dụng

Cypress là một ứng dụng dành cho máy tính cá nhân được cài đặt trực tiếp trên máy tính. Ứng dụng này hiện tại đang hỗ trợ các hệ điều hành

* Mac OS 10.9+ (Mavericks +), chỉ có các gói nhị phân 64 bit được cung cấp cho macOS.
* Linux Ubuntu 12.04+, Fedora 21, Debian 8.
* Window 7+, chỉ có các gói nhị phân 32 bit được cung cấp cho Windows [2].

### Các trình duyệt được hỗ trợ

Cypress tự động phát hiện các trình duyệt Web hiện có trên hệ điều hành của bạn. Hiện tại, Cypress đang hỗ trợ chạy thử nghiệm trên các trình duyệt sau: Canary, Chrome, Choromium, Electron.

Nhiều trình duyệt như Firefox, Safari, Internet Explorer hiện không được hỗ trợ. Nhà sản xuất cũng nói rằng việc hỗ trợ chạy ứng dụng trên các trình duyệt này là công việc họ cần làm trong tương lai [2].

## Đặc điểm của Cypress

Những đặc điểm của công cụ Cypress được thể hiện như sau:

* Cypress là một công cụ mã nguồn mở: vì là mã nguồn mở nên chúng ta có thể sử dụng mà không phải lo lắng về phí bản quyền hay thời hạn sử dụng.
* Cypress hỗ trợ chạy trên nhiều nền tảng hệ điều hành khác nhau như Windows, Linux và Mac.
* Cypress giúp đơn giản hơn những công việc: thiết lập thử nghiệm (set up tests), viết bài kiểm tra (write tests), chạy thử nghiệm (run tests) và kiểm tra gỡ lỗi (debug tests).
* Cypress cho phép nhà phát triển dễ dàng ghi lại (recording), biên tập lại (editting), và gỡ lỗi (debugging) cho các bài kiểm tra.
* Cypress cho phép viết tất cả các loại thử nghiệm: kiểm tra kết thúc (end to end test), kiểm tra tích hợp (integration tests), kiểm thử đơn vị (unit tests).
* Cypress có thể hoạt động ở lớp mạng bằng cách đọc và thay đổi lưu lượng Web khi đang di chuyển. Điều này cho phép Cypress không chỉ thay đổi mọi thứ trong và ngoài trình duyệt, mà còn để thay đổi mã điều này có thể ảnh hưởng đến khả năng tự động hoá trình duyệt của nó.
* Cypress kiểm soát quá trình tự động hóa toàn bộ từ trên xuống dưới, đặt nó ở vị trí duy nhất để có thể hiểu mọi thứ xảy ra bên trong và bên ngoài của trình duyệt. Điều này có nghĩa Cypress có khả năng cung cấp kết quả nhất quán hơn bất kỳ công cụ kiểm tra khác.
* Cypress có quyền truy cập vào mọi thứ: vì Cypress hoạt động trong ứng dụng của nhà phát triển, nghĩa là nó có quyền truy cập vào từng đối tượng. Cho dù đó là các cửa sổ, tài liệu, một phần tử DOM, một chức năng, một bộ đếm thời gian hoặc bất cứ điều gì khác - bạn có quyền truy cập vào nó trong Cypress. Không có sự tuần tự hóa cho các đối tượng, không có giao thức over-the-wire, bạn có quyền truy cập vào tất cả mọi thứ trong tầm tay của bạn.
* Cypress không sử dụng Selenium: hầu hết các công cụ kiểm tra đầu cuối đều dựa trên Selenium, đó là lý do tại sao tất cả chúng đều có những vấn đề tương tự. Cypress đã tạo ra sự khác biệt bằng cách xây dựng một kiến trúc mới từ dưới lên. Trong khi Selenium thực hiện các lệnh từ xa thông qua mạng, Cypress chạy trong cùng một vòng lặp chạy như ứng dụng của bạn.
* Cypress tập trung vào làm thế nào để kết thúc thử nghiệm một cách tốt nhất: Cypress không phải là một khuôn khổ tự động hóa chung, cũng không phải là một khuôn khổ thử nghiệm đơn vị cho các dịch vụ cuối cùng. Đã có những công cụ khác làm điều đó. Thay vào đó, Cypress chuyên về một điều - tạo ra trải nghiệm tuyệt vời trong khi bạn viết bài kiểm tra cuối cùng cho các ứng dụng Web của mình.
* Cypress hoạt động trên bất kỳ khuôn khổ phía giao diện người dùng hoặc trang web: Cypress thử nghiệm bất cứ thứ gì chạy trên một trình duyệt Web. Tất cả kiến trúc xung quanh Cypress được xây dựng để xử lý các khuôn khổ JavaScript hiện đại đặc biệt tốt. Cypress cũng hoạt động tốt như nhau trên các trang hoặc các ứng dụng được hiển thị trên máy chủ cũ hơn thậm chí trong các dự án sử dụng khuôn khổ React, Angular, Vue, Elm, ...mới nhất.
* Các bài kiểm tra Cypress chỉ được viết bằng JavaScript: mặc dù bạn có thể biết nhiều ngôn ngữ lập trình khác nhưng Cypress chỉ hỗ trợ viết mã lệnh bằng JavaScript, đây có lẽ là một hạn chế của Cypress.
* Cypress là công cụ bao gồm tất cả: để viết các bài kiểm tra với Cypress, nhà phát triển không cần phải cài đặt nhiều công cụ riêng biệt với các thư viện phức tạp, Cypress đã làm cho tất cả chúng làm việc cùng nhau liền mạch và đơn giản hơn rất nhiều.
* Cypress dành cho các nhà phát triển và kỹ sư QA.
* Cypress chạy nhanh hơn, nhiều hơn trong cùng một khoảng thời gian so với các công cụ khác: những cải tiến kiến trúc này mở ra khả năng thực hiện TDD với các bài kiểm tra đầu cuối hoàn chỉnh ngay lần đầu tiên. Cypress đã được xây dựng để thử nghiệm và phát triển có thể xảy ra đồng thời [2].

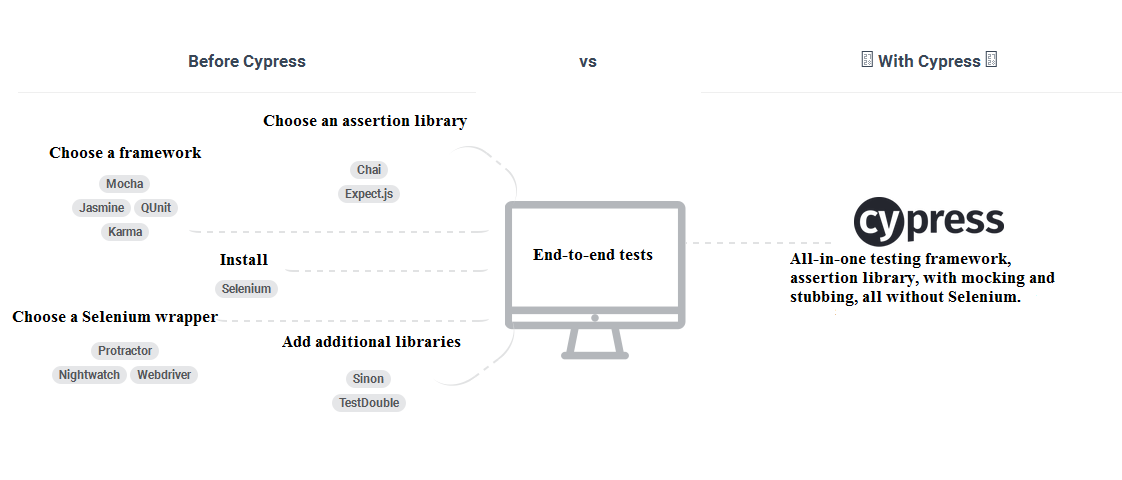
## Ngôn ngữ sử dụng viết mã lệnh

Cypress sử dụng ngôn ngữ JavaScript để viết mã lệnh [2]. JavaScript, theo phiên bản hiện hành, là một ngôn ngữ lập trình kịch bản dựa trên đối tượng được phát triển từ các ý niệm nguyên mẫu. Ngôn ngữ này được dùng rộng rãi cho các trang Web, nhưng cũng được dùng để viết mã lệnh sử dụng các đối tượng nằm sẵn trong các ứng dụng.

JavaScript là một ngôn ngữ lập trình được nhúng được trong các trang HTML. JavaScript nâng cao tính động và khả năng tương tác cho Web-site bằng cách sử dụng các hiệu ứng của nó như thực hiện các phép tính, kiểm tra biểu mẫu, viết các trò chơi, bổ sung các hiệu ứng đặc biệt, tuỳ biến các chọn lựa đồ hoạ, tạo ra các mật khẩu bảo mật và hơn thế nữa.

## Sự khác biệt giữa Cypress và Selenium WebDriver

### So sánh giữa Cypress và Selenium WebDriver



Hình 3.1 Sự khác biệt giữa Cypress với các công cụ kiểm thử tự động khác [2]

Cụ thể những điểm khác biệt được thể hiện như sau [8]:

* Cài đặt Cypress với cài đặt Selenium WebDriver

Sự khác biệt đầu tiên là không có cấu hình cần thiết với Cypress: với Cypress chỉ cần cài đặt tệp tin có đuôi .exe, khi đó tất cả các trình điều khiển, phụ thuộc được tự động cài đặt và cấu hình cho bạn. Việc cài đặt chỉ tiến hành trong vài phút. Một trong những nguyên tắc thiết kế của Cypress là có thể gói và gói mọi thứ lại với nhau để làm cho trải nghiệm cuối cùng hoàn toàn dễ chịu và đơn giản cho các nhà phát triển. Với Selenium, bạn thường phải cài đặt ngôn ngữ ràng buộc và cấu hình các trình điều khiển để có thể làm việc, nó là một việc làm không hề đơn giản.

* Các trình duyệt hỗ trợ chạy kiểm thử

Với Selenium người phát triển có thể chạy thử nghiệm của mình đối với bất kỳ trình duyệt nào. Với Cypress chỉ hỗ trợ một số lượng trình duyệt hạn chế

* Cypress Runner là mã nguồn mở

Selenium và Cypress đều là mã nguồn mở, nhưng Cypress cũng có một tính năng trả tiền, đó chính là bảng điều khiển.

* Kiến trúc của Cypress so với Selenium

Selenium hoạt động bằng cách chạy bên ngoài trình duyệt và thực hiện các lệnh từ xa qua mạng. Những gì bạn có là kịch bản thử nghiệm chạy bên ngoài của trình duyệt, thực hiện các lệnh từ xa vào trình duyệt. Cypress thì khác, nó chạy bên trong trình duyệt. Cypress thực hiện trong cùng một vòng lặp chạy trên ứng dụng của bạn. Cypress cũng thúc đẩy một máy chủ Node.js để xử lý bất kỳ công việc nào cần phải xảy ra bên ngoài của trình duyệt. Do đó, Cypress có thể cho kết quả phù hợp hơn vì nó có thể hiểu mọi thứ xảy ra bên trong và bên ngoài trình duyệt. Điều này cũng cho phép bạn truy cập vào từng đối tượng mà không phải đối phó với việc tuần tự đối tượng hoặc các giao thức qua mạng. Vì vậy, nhà phát triển có quyền truy cập vào từng phần tử DOM.

* Cypress Khác với Selenium IRC như thế nào?

Cypress sử dụng rất nhiều API tự động hóa trình duyệt mà Selenium WebDriver tận dụng, nhưng không thực sự sử dụng Selenium WebDriver.Về cơ bản khi kết thúc làm cùng một điều, nhưng chỉ trên cơ sở giới hạn cho một số loại lệnh.

* Tự động hóa cho QA hoặc Nhà phát triển

Selenium là mục tiêu đối với các kỹ sư QA phát triển /kiểm thử, Cypress lại là trung tâm phát triển hơn. Nó nhằm mục đích hướng tới việc phát triển TDD trở thành hiện thực với các nhà phát triển. Vì vậy nếu nhà phát triển đang tìm kiếm một công cụ phù hợp với bài kiểm tra của mình, Selenium có thể là sự lựa chọn tốt nhất.

* Tại sao lại tạo một công cụ mới thay vì sử dụng Selenium?

Khi Cypress đang trong giai đoạn phát triển ban đầu, một trong những điều đầu tiên mà người sáng lập, Brian, đã làm là khảo sát một loạt các nhà phát triển. Một số câu hỏi mà anh ấy hỏi họ là "Bạn muốn giải quyết vấn đề gì?" Và "Bạn thực sự đang phải vật lộn với điều gì?”. Trải qua nhiều lần, ông phát hiện ra rằng các nhà phát triển chủ yếu phải vật lộn với việc viết bài kiểm tra tự động. Họ cũng phàn nàn rằng các bài kiểm tra tự động của họ không đáng tin cậy và quá chậm để sử dụng như TDD. Nhiều người cũng đã đề cập rằng đó là một phần tồi tệ nhất trong ngày của nhà phát triển. Dựa trên phản hồi đó, nhóm của ông đã xây dựng Cypress để nhắm mục tiêu cụ thể vào những vấn đề đó và cuối cùng giải quyết chúng. Tóm lại, Cypress là một công cụ tập trung vào các nhà phát triển giúp các nhà phát triển khác viết bài kiểm tra cho Web [8].

### Ưu và nhược điểm của công cụ Cypress

#### Ưu điểm của công cụ Cypress

Công cụ kiểm thử tự động Cypress có những ưu điểm như sau [8]:

* Cypress không sử dụng Selenium: hầu hết các công cụ kiểm thử kết thúc hiện tại đều đang sử dụng Selenium, đó là lý do tại sao chúng có cùng một vấn đề.
* Cypress hỗ trợ bất kỳ khuôn khổ hoặc trang Web khá tốt: có hàng trăm dự án sử dụng các khuôn khổ React, Angular, Vue, Elm, … mới nhất. Cypress cũng hoạt động tốt như nhau trên các trang hoặc các ứng dụng được hiển thị trên máy chủ cũ hơn.
* Cypress chạy nhiều và nhanh hơn hầu hết các công cụ kết thúc của Selenium
* Có màn hình chụp cho mỗi bước của kịch bản, điều này khá hữu ích nếu có bất ký lỗi sai hoặc thất bại trong thử nghiệm và cũng tốt cho việc gỡ lỗi.
* Cypress có một cú pháp rõ ràng.

#### Nhược điểm của công cụ Cypress

Bên cạnh đó, nó cũng tồn tại một số nhược điểm như là [8]:

* Cấu trúc Cypress khá khác so với các công cụ kết thúc của Selenium khác, vì vậy để bắt đầu việc đầu tiên nhà phát triển cần phải dành nhiều thời gian để hiểu cấu trúc và tìm ra cách tốt nhất để tạo ra các kịch bản của mình.
* Về tính năng, đặc điểm: Cypress đang chỉ hỗ trợ một số trình duyệt, không hỗ trợ tải tệp lên, không có trình duyệt chéo chạy thử. Cypress hỗ trợ nhiều cách tiếp cận đối tượng trang khác nhau nhưng chính điều này lại gây tranh cãi. Cypress chỉ sử dụng JavaScript điều này gây khó khăn cho người không có kỹ năng tốt về JavaScript.

## Những lệnh cơ bản trong Cypress

Một số câu lệnh chính được sử dụng trong Cypress [2]:

.should() // khẳng định rằng < tiêu đề > là chính xác.

cy.get() // truy vấn các phần tử DOM.

cy.contains() // truy vấn các phần tử DOM có nội dung phù hợp.

.find() // nhận các phần tử DOM gốc của bộ chọn.

.first() // nhận phần tử DOM đầu tiên.

.last() // nhận phần tử DOM cuối cùng.

.next() // nhận phần tử DOM kế tiếp.

.not() // loại bỏ các phần tử DOM khỏi bộ các phần tử DOM.

.parent() // nhận cha của phần tử DOM từ bộ phần tử DOM.

.prevAll() // nhận tất cả phần tử DOM trước đó.

.type() // nhấn vào một phần tử DOM.

,focus() // tập trung vào một phần tử DOM.

.clear() // xóa một phần tử đầu vào hoặc một vùng nhập dữ liệu.

.submit() // gửi một biểu mầu.

.click() // nháy chuột vào một phần tử DOM.

.dbclick() // nháy đúp chuột vào một phần tử DOM.

.select() // chọn một lựa chọn trong một dãy phần tử chọn.

cy.title() // lấy một tiêu đề.

cy.url() // nhận địa chỉ URL hiện tại.

cy.go() // quay lại hoặc chuyển tiếp trong lịch sử của trình duyệt.

cy.reload() // tải lại một trang.

cy.visit() // thăm một địa chỉ url.

## Tạo một kịch bản kiểm thử

Sau đây là ví dụ về một kịch bản kiểm thử bằng Cypress

* Tạo 1 tệp tin js trong thư mục: cypress/integration
* Đoạn mã lệnh tương ứng như sau:

describe(**'Kiểm thử trang đăng nhập'**, **function**() {  
  
1. context(**'Form login'**, **function**(){  
2. beforeEach(**function**(){  
3. cy.visit(**'http://methadone4.cloudapp.net'**);  
4. });  
  
5. it(**'Không điền email'**, **function**(){  
6. cy.get(**'input[name=password]'**).type(**'12345678{enter}'**);  
  
7. cy.get(**'.login-form'**)  
8. .find(**"div[ng-message]"**)  
9. .should(**"contain"**, **'Trường này không được để trống.'**);  
  
10. cy.url().should(**'include'**, **'/signin'**);  
11. })

Giải thích:

1-> 4: Tạo 1 function có tên Form Login sau đó thăm một trang có địa chỉ URL là <http://methadone4.cloudapp.net>.

5 -> 6: Tạo một function tên là Không điền email và tìm một đầu ra có tên là password và nhập 12345678 vào nó sau đó nhấn phím Enter.

7: Tìm một lớp có tên là login-form. 8: Tìm một khu có chứa ng-message.

9: Xác nhận nó bao gồm chuỗi ký tự:” Trường này không được để trống.”.

10: Xác nhận địa chỉ URL lúc này bao gồm “/signin”.

12. it(**'Không điền mật khẩu'**, **function**(){  
13. cy.get(**'input[name=email]'**).type(**'quanly@gmail.com'**);  
14. cy.get(**'button'**).click();  
15. cy.get(**'.login-form'**)  
16. .find(**"div[ng-message]"**)  
17. .should(**"contain"**, **'Trường này không được để trống.'**);  
  
18. cy.url().should(**'include'**, **'/signin'**);  
19. })

Giải thích:

12 -> 19: Tạo một function tên là “Không điền mật khẩu” và chỉ thực hiện nhập trường email, sau đó nhấn enter, trường mật khẩu lúc này cần hiển thị thông báo:” Trường này không được để trống.”

20. it(**'Không điền mật khẩu và email'**, **function**(){  
21. cy.get(**'button'**).click();  
  
22. cy.get(**'.login-form'**)  
23. .find(**"div[ng-message]"**).first()  
24. .should(**"contain"**, **'Trường này không được để trống.'**);  
  
25. cy.get(**'.login-form'**)  
26. .find(**"div[ng-message]"**).last()  
27. .should(**"contain"**, **'Trường này không được để trống.'**);  
  
28. cy.url().should(**'include'**, **'/signin'**);  
29. })

Giải thích:

20 -> 29: Tạo một function tên là “Không điền mật khẩu và email” và chỉ thực hiện nhấn enter, không nhập dữ liệu cho trường nào, lúc này cả 2 trường ở first và last đều cần hiển thị thông báo:” Trường này không được để trống.”

30. it(**'Email sai định dạng'**, **function**(){  
31. cy.get(**'input[name=email]'**).type(**'email'**);  
32. cy.get(**'input[name=password]'**).type(**'12345678{enter}'**);  
33.  
34. cy.get(**'.login-form'**)  
35. .find(**"div[ng-message]"**).first()  
36. .should(**"contain"**, **'Email không đúng định dạng.'**);  
  
37. cy.url().should(**'include'**, **'/signin'**);  
38. })

Giải thích:

30 -> 38: Tạo một function tên là “Email sai định dạng” và nhập email khác định dạng hợp lệ, nhập mật khẩu hợp lệ, nhấn Enter. Lúc này trường email hiển thị thông báo:” Email không đúng định dạng.”

39. it(**'Đăng nhập không thành công'**, **function**(){  
40. cy.get(**'input[name=email]'**).type(**'quanly@gmail.com'**);  
41. cy.get(**'input[name=password]'**).type(**'123{enter}'**);  
  
42. cy.url().should(**'include'**, **'/signin'**);  
43. })  
  
44. it(**'Ðăng nhập thành công'**, **function**(){  
45. cy.get(**'input[name=email]'**).type(**'quanly@gmail.com'**);  
46. cy.get(**'input[name=password]'**).type(**'12345678{enter}'**);  
  
47. cy.url().should(**'include'**, **'/main/dashboard1'**);  
48. })  
  
49. })  
50.})

Giải thích:

39 -> 50: 2function “Đăng nhập không thành công” và “Đăng nhập thành công” lần lượt là nhập dữ liệu khác cặp, cùng cặp trong cơ sở dữ liệu. Nếu đăng nhập không thành công địa chỉ URL không thay đổi. Nếu đăng nhập thành công, địa chỉ URL lúc này có đuôi là “/main/dashboard1”.

# Chương 4: Ứng dụng Cypress cho một số chức năng trên hệ thống Quản lý điều trị thuốc Methadone

## Giới thiệu bài toán

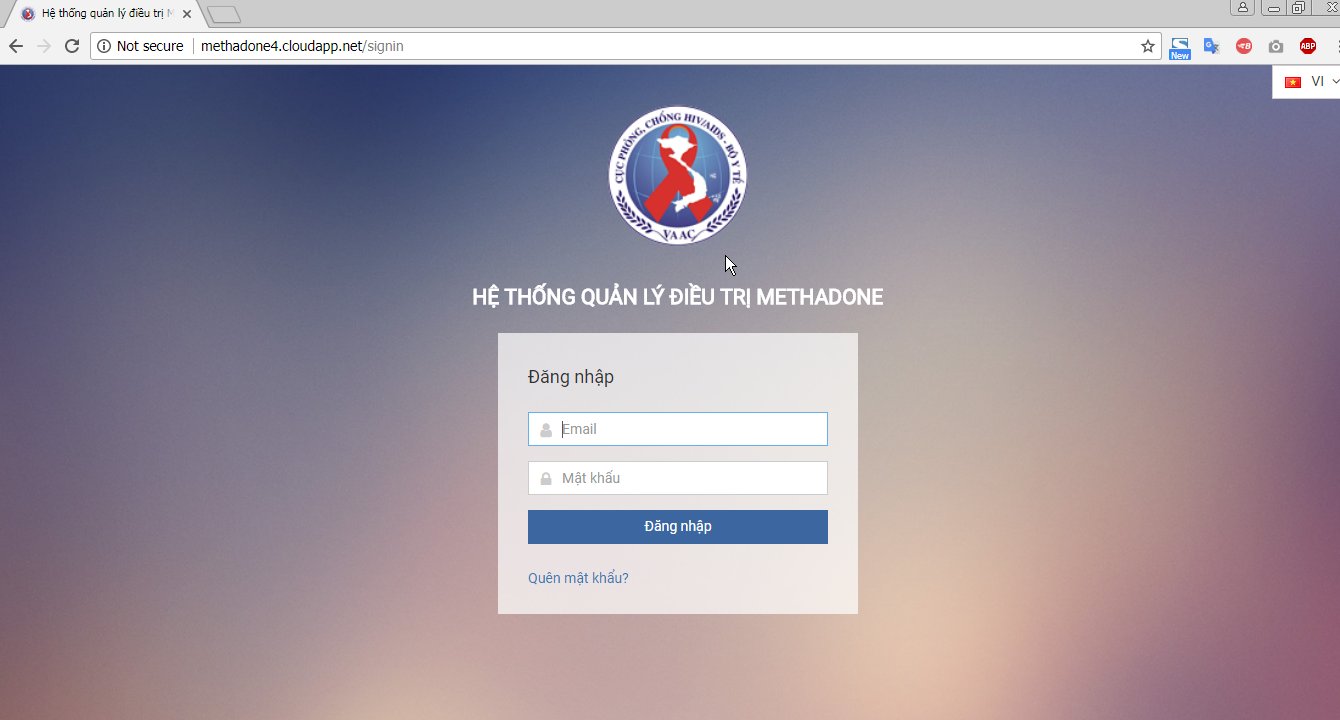
### Mục đích bài toán

Kiểm thử tự động cho các ca kiểm thử cơ bản với hệ thống quản lý điều trị thuốc Methadone.

### Giới thiệu chung về dự án

Xây dựng dự án Website quản lý điều trị thuốc Methadone nhằm đáp ứng nhu cầu hỗ trợ bệnh nhân điều trị Methadone, giúp họ dễ dàng kết nối và uống thuốc ở các cơ sở điều trị Methadone khác nhau, đồng thời để tăng cường hiệu quả, hiệu lực quản lý trên lĩnh vực này, ứng dụng công nghệ thông tin dựa trên nền tảng trình duyệt Internet là một giải pháp có tính đột phá và mang lại sự thay đổi vượt bậc trong quản lý điều trị Methadone ở nước ta.

Hệ thống quản lý thông tin điều trị Methadone mỗi một bệnh nhân được cấp 1 mã nhận diện (ID) duy nhất để quản lý toàn bộ quá trình điều trị, hệ thống quản lý bệnh nhân điều trị Methadone. Hệ thống trở thành công cụ quản lý hữu hiệu đối với ngành y tế trong hoạt động điều trị, hỗ trợ điều trị cho bệnh nhân, đồng thời nâng cao hiệu suất lao động và tạo môi trường làm việc hiện đại cho hệ thống các các cơ sở điều trị thông qua việc giảm thiểu về cơ bản các thủ tục hành chính, hồ sơ bệnh án, sổ sách, thống kê báo cáo vốn rất tốn rất nhiều thời gian và công sức để thực hiện.



Hình 4.1 Giao diện chính của hệ thống Methadone

## Hướng giải quyết

### Tổng quát

Để thực hiện áp dụng Cypress vào kiểm thử hệ thống Web Methadone, các công việc cần thực hiện như sau:

*Bước 1*. Xây dựng các ca kiểm thử cơ bản cho các ứng dụng Web.

Các ca kiểm thử cơ bản được xây dựng để dùng chung cho các ứng dụng Web, nó giúp đảm bảo rằng phần mềm được kiểm tra với mức bao phủ cần thiết, giảm bớt bỏ quên lỗi cho nhân viên kiểm thử, giúp công việc kiểm thử đảm bảo mức đúng đắn/ chính xác cho phần mềm, bao quát được vùng kiểm thử [3]. Các ca kiểm thử cơ bản cũng chia thành 2 loại: các ca kiểm thử giao diện và các ca kiểm thử chức năng. Các ca kiểm thử giao diện bao gồm những yếu tố: hiển thị, màu sắc, kiểu hiển thị, cỡ chữ, phông chữ, các nhãn, các tiêu đề, các trường dữ liệu, các nút,... Ngược lại, các ca kiểm thử chức năng chỉ tập trung thực hiện các chức năng của hệ thống như tìm kiếm, xóa dữ liệu, thêm mới,… Một ca kiểm thử thông thường sẽ bao gồm: ID, mục đích kiểm thử, dữ liệu kiểm thử, điều kiện đầu vào, các bước thực hiện, điều kiện đầu ra, kết quả mong muốn, kết quả kiểm thử. Trong khóa luận này, em sẽ xây dựng bộ các ca kiểm thử bao gồm: ID, mục đích kiểm thử, các bước thực hiện, kết quả mong muốn, kết quả kiểm thử.

*Bước 2*. Viết mã lệnh bằng JavaScript cho các chức năng của hệ thống Methadone theo các ca kiểm thử đã tạo.

Sau khi việc xây dựng các ca kiểm thử được hoàn thành, ta bắt tay vào viết các mã lệnh cho các ca kiểm thử đã được tạo. Các mã lệnh được viết bởi JavaScript, khi viết mã lênh phải đảm bảo các mã lệnh mô tả đúng các trường hợp kiểm tra, các bước thực hiện rõ ràng và có ràng buộc điều kiện kết quả.

Sau đây là ví dụ cách xây dựng mã lệnh dựa trên các ca kiểm thử có sẵn:

Bảng 4.1 Mô phỏng cách đưa các trường hợp kiểm thử thành mã lệnh

|  |  |  |
| --- | --- | --- |
|  | **Mã lệnh** | **Mô tả** |
| **Mục đích** | it(**'Không điền email'**, **function**(){ | Mục đích kiểm thử được thể hiện rõ trong tên hàm. |
| **Các bước thực hiện** | cy.visit(**'http://methadone4.cloudapp.net'**); | Truy cập vào trang Web có địa chỉ URL như bên. |
| cy.get(**'input[name=password]'**)  .type(**'12345678{enter}'**) | Nhập 12345678 vào trường password. |
| cy.get(**'button'**).click(); | Click vào botton Đăng nhập |
| **Kết quả mong muốn** | cy.get(**'.login-form'**).find(**"div[ng-message]"**).should(**"contain"**, **'Trường này không được để trống.'**); | Thông báo “Trường này không được để trống.”. |
| cy.url().should(**'include'**,**'/signin'**); | Địa chỉ URL không thay đổi (tức là không thực hiện được thao tác, vẫn ở màn hình đăng nhập) |

*Bước 3.* Thực hiện chạy các mã lệnh, ghi lỗi và lập báo cáo kiểm thử.

Sau khi thiết kế các ca kiểm thử và hiện thực hóa các ca kiểm thử thành các mã lệnh như ví dụ trên, ta thực hiện chạy các mã lệnh. Mã lệnh chạy thành công cũng không có nghĩa là chức năng đó chắc chắn đúng 100%, có thể mã lệnh tạo và chức năng đều sai giống nhau nên kết quả chạy vẫn đúng. Đây cũng là hạn chế chung của kiểm thử tự động. Từ kết quả chạy các mã lệnh nhà phát triển có thể xác định lỗi nằm ở đâu: lỗi ở mã lệnh hay lỗi ở chức năng trên hệ thống.

Bên dưới đây là ví dụ khi mã lệnh chạy bị lỗi, từ mã lệnh chạy lỗi này, nhà phát triển phân tích và xác định lỗi ở đâu từ đó tìm cách xử lý.

Bảng 4.2 Phân tích mã lệnh lỗi để tìm nguyên nhân

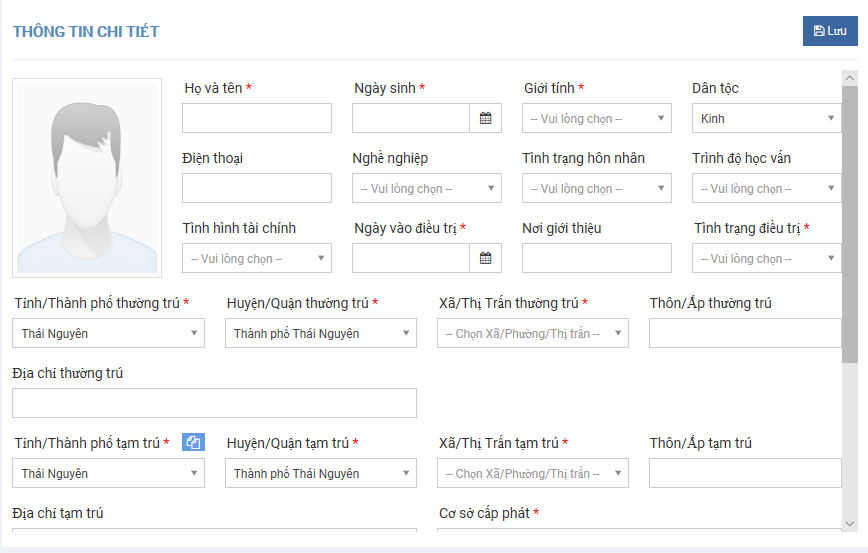
|  |  |  |
| --- | --- | --- |
| **Mã lệnh lỗi** | **Kết quả mong muốn** | **Kết quả thực tế** |
| cy.url().should(**'include'**, **'/main/dashboard'**); | Địa chỉ URL trả về có đuôi ‘main/dashboard’ | Địa chỉ URL trả về có đuôi ‘main/dashboard1’ |

### Chi tiết từng bước thực hiện

#### Xây dựng các ca kiểm thử cơ bản

Tiến hành thực hiện như các bước ở phần tổng quát, chia số trường hợp kiểm thử thành 2 phần: giao diện và chức năng. Bên dưới đây là các ví dụ cụ thể của từng trường hợp.

* Kiểm thử giao diện cho màn hình chức năng thêm mới bệnh nhân.



Hình 4.2 Giao diện màn hình chức năng thêm mới bệnh nhân

Bảng 4.3 Các ca kiểm thử giao diện cơ bản

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Mục đích kiểm thử** | **Các bước thực hiện** | **Kết quả mong muốn** | **KQKT** |
| TC\_001 | Kiểm tra màn hình ởtrạng thái mặc định | 1. Kiểm tra title của màn hình | Màn hình chức năng được mở: - Hiển thị title của chức năng trên màn hình: <Thông tin chi tiết> | Pass |
| TC\_002 | 2. Kiểm tra focus của chuột | Focus được set vào trường đầu tiên có thể edit | Pass |
| TC\_003 | 3. Kiểm tra các giá trị mặc định của các trường | Hiển thị đầy đủ các trường thông tin | Pass |
| TC\_004 | 4. Kiểm tra header, footer | Hiển thị các giá trị mặc định của các trường đúng. | Pass |

Sau khi thực hiện viết các trường hợp kiểm thử (testcase) như trên, ta bắt đầu xây dựng mã nguồn. Mã nguồn bên dưới được xây dựng cho các trường hợp kiểm thử đã liệt kê bên trên:

describe(**'Kiểm tra màn hình ở trạng thái mặc định'**, **function**() {  
 *// Đăng nhập và chuyển tới trang thêm bệnh nhân*1. beforeEach(**function**(){  
2. cy.visit(**'http://methadone4.cloudapp.net'**);  
3. cy.get(**'input[name=email]'**).type(**'bacsi@gmail.com'**);  
4. cy.get(**'input[name=password]'**).type(**'12345678'**);  
5. cy.get(**'button[type=submit]'**).click();  
  
6. cy.wait(3000);  
7. cy.visit(**'/main/patients/new'**);  
8. cy.wait(3000);  
9. })

Giải thích:

1: Tạo một function.

2: Truy cập vào Website có địa chỉ URL: “http://methadone4.cloudapp.net”.

3: Tìm đầu vào có tên là name và nhập dữ liệu đầu vào tương ứng.

4: Tìm đầu vào có tên password và nhập dữ liệu đầu vào tương ứng.

5: Click vào botton có kiểu là Submit.

6,8: Thiết lập thời gian chờ của ứng dụng.

7: Truy cập vào chức năng Thêm mới bệnh nhân bằng cách truy cập trực tiếp vào địa chỉ URL.

10. **var** addPatientFormSelector = **'form.ng-scope.ng-invalid-required.ng-valid-pattern.ng-pending.ng-dirty.ng-valid-parse'**;

Giải thích:

10: Khởi tạo biến có tên addPatientFormSelector thực hiện chỉ thị ngModel liên kết một đầu vào.

11. **var** patientFields = [  
12. {name: **'name'**, label: **'Họ và tên'**},  
13. {name: **'birthdate'**, label: **'Ngày sinh'**},  
14. {name: **'gender'**, label: **'Giới tính'**},  
15. {name: **'ethnicity\_id'**, label: **'Dân tộc'**,  
16. **default**: {val: **'Kinh'**, selector: **'span.ng-binding'**}},  
17. {name: **'mobile\_phone'**, label: **'Điện thoại'**},  
18. {name: **'jobs'**, label: **'Nghề nghiệp'**},  
19. {name: **'marital\_status'**, label: **'Tình trạng hôn nhân'**},  
20. {name: **'education\_level'**, label: **'Trình độ học vấn'**},  
21. {name: **'financial\_status'**, label: **'Tình hình tài chính'**},  
22. {name: **'admission\_date'**, label: **'Ngày vào điều trị'**},  
23. {name: **'referral\_agency'**, label: **'Nơi giới thiệu'**},  
24. {name: **'province\_id'**, label: **'Tỉnh/Thành phố thường trú'**,  
25. **default**: {val: **'Hà Nội'**, selector: **'span.ng-binding'**}},  
26. {name: **'district\_id'**, label: **'Huyện/Quận thường trú'**,  
27. **default**: {val: **'Chương Mỹ'**, selector: **'span.ng-binding'**}},  
28. {name: **'ward\_id'**, label: **'Xã/Thị Trấn thường trú'**},  
29. {name: **'hamlet'**, label: **'Thôn/Ấp thường trú'**},  
30. {name: **'address'**, label: **'Địa chỉ thường trú'**},  
31. {name: **'resident\_province\_id'**, label: **'Tỉnh/Thành phố tạm trú'**,  
32. **default**: {val: **'Hà Nội'**, selector: **'span.ng-binding'**}},  
33. {name: **'resident\_district\_id'**, label: **'Huyện/Quận tạm trú'**,  
34. **default**: {val: **'Chương Mỹ'**, selector: **'span.ng-binding'**}},  
35. {name: **'resident\_ward\_id'**, label: **'Xã/Thị Trấn tạm trú'**},  
36. {name: **'resident\_hamlet'**, label: **'Thôn/Ấp tạm trú'**},  
37. {name: **'resident\_address'**, label: **'Địa chỉ tạm trú'**},  
38. {name: **'identification\_type'**, label: **'Loại giấy tờ'**},  
39. {name: **'identification\_number'**, label: **'Số'**},  
40. {name: **'identification\_issued\_date'**, label: **'Ngày cấp'**},  
41. {name: **'identification\_issued\_by'**, label: **'Nơi cấp'**}  
42. ];

Giải thích:

11 ->42: Khởi tạo một biến có tên patientFields chứa tất cả các trường thông tin trên màn hình gồm tên, nhãn hiển thị.

43. **var** contactFields = [  
44. {name: **'name'**, label: **'Họ và tên'**},  
45. {name: **'contact\_type'**, label: **'Mối quan hệ'**},  
46. {name: **'address'**, label: **'Địa chỉ'**},  
47. {name: **'telephone'**, label: **'Điện thoại'**},  
48. ];

Giải thích:

43 -> 48: Khởi tạo một biến có tên contactFields chứa tất cả các trường thông tin trên màn hình gồm tên, nhãn hiển thị trong mục thông tin người thân.

49. it(**'Hiên thị title của chức năng trên màn hình'**, **function** () {  
50. cy.get(addPatientFormSelector).find(**'.caption-subject'**).contains(**'Thông tin chi tiết'**);  
51. });

Giải thích:

49 -> 51: Sử dụng biến đã tạo addPatientFormSelector để tìm tiêu đề của chức năng.

52. it(**'Focus vào input đầu tiên của form'**, **function** () {  
 cy.get(addPatientFormSelector).find(**'input'**).first().focused().should(**'exist'**);  
53. });

Giải thích:

52 -> 53: Thực hiện tương tự để xác định chuột có focus vào trường đầu tiên hay không.

54. it(**'Kiểm tra form có đầy đủ các trường'**, **function** () {  
  
55. cy.get(addPatientFormSelector).within(**function**() {  
 *// kiểm tra đầy đủ các trường của bệnh nhân*56. **for** (let i = 0; i < patientFields.length; i++) {  
57. **var** field = patientFields[i];  
58. **var** selector = **'[ng-model="patient.'**+ field[**'name'**] +**'"]'**;  
59. cy.get(selector).closest(**'.form-group'**).contains(field[**'label'**]);  
60. }

Giải thích:

54 -> 60: Sử dụng vòng lặp for duyệt để kiểm tra xem đã đầy đủ các trường của bệnh nhân chưa.

*// kiểm tra đầy đủ các trường của người thân bệnh nhân*61. cy.contains(**'Thông tin người thân'**)  
62. cy.get(**'div[name="contactForm"]'**).within(**function**() {  
63. **for** (let i = 0; i < contactFields.length; i++) {  
64. **var** field = contactFields[i];  
65. **var** selector = **'[ng-model="contact.'**+ field[**'name'**] +**'"]'**;  
66. cy.get(selector).closest(**'.form-group'**).contains(field[**'label'**]);  
67. }  
68. })  
69. });  
70. });

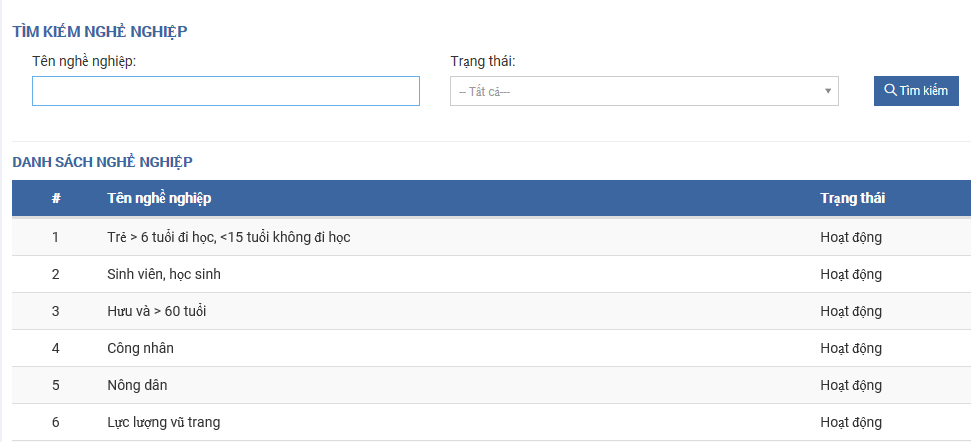
Giải thích:

61 -> 68: Sử dụng vòng lặp for duyệt để kiểm tra xem đã đầy đủ các trường của mục người thân bệnh nhân chưa.

71. it(**'Kiểm tra giá trị mặc định'**, **function** () {  
  
72. cy.get(addPatientFormSelector).within(**function**() {  
 *// kiểm tra đầy đủ các trường của bệnh nhân*73. **for** (let i = 0; i < patientFields.length; i++) {  
74. **var** field = patientFields[i];  
75. **var** defaultValue = patientFields[i].**default**;  
76. **if** (defaultValue) {  
77. **var** selector = **'[ng-model="patient.'**+ field[**'name'**] +**'"]'**;  
 78.cy.get(selector).find(defaultValue.selector).contains(defaultValue.val);  
79. }  
80. }  
81. });  
82. });  
83.});

71 - > 83: Sử dụng vòng lặp for duyệt để kiểm tra giá trị mặc định của các trường của bệnh nhân đã đúng hay chưa.

* Kiểm thử chức năng tìm kiếm, một trong số những chức năng đã có của hệ thống.



Hình 4.3 Màn hình chức năng tìm kiếm

Các ca kiểm thử chức năng được xây dựng tương ứng:

Bảng 4.4 Các ca kiểm thử chức năng tìm kiếm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Mục đích kiểm thử** | **Các bước thực hiện** | **Kết quả mong muốn** | **KQKT** |
| TC\_001 | Kiểm tra chức năng tìm kiếm với textbox tên nghề nghiệp | 1. Kiểm tra tìm kiếm với giá trị mặc định. | Hiển thị tất cả thông tin thỏa mãn điều kiện tìm kiếm. | Pass |
| TC\_002 | 2. Kiểm tra tìm kiếm không có kết quả. | Hiển thị thông báo “Không có bản ghi nào” | Fail |
| TC\_003 | 3. Kiểm tra tìm kiếm khi nhập ký tự khoảng trắng vào đầu và cuối. | Không có lỗi. Hiển thị các kết quả tìm kiếm thỏa mãn điều kiện textbox nhập vào và đã được trimspace khoảng trắng ở đầu và cuối | Pass |
| TC\_004 | 4. Kiểm tra tìm kiếm khi nhập toàn ký tự đặc biệt. | Hiển thị kết quả tìm kiếm thỏa mãn giá trị textbox đã nhập. | Pass |

Các mã lệnh kiểm tra tương ứng:

describe(**'Tim kiem'**, **function** () {  
 *// Đăng nhập và chuyển tới trang tìm kiếm nghề nghiệp*1. beforeEach(**function**(){  
2. cy.visit(**'http://methadone4.cloudapp.net'**);  
3. cy.get(**'input[name=email]'**).type(**'quanly@gmail.com'**);  
4. cy.get(**'input[name=password]'**).type(**'12345678'**);  
5. cy.get(**'button[type=submit]'**).click();  
  
6. cy.visit(**'/main/admin/administrators/employments'**);  
7. });

Giải thích:

1 -> 7: Đăng nhập vào hệ thống bằng tài khoản quản lý và truy cập vào chức năng Quản lý nghề nghiệp qua địa chỉ URL.

8. it(**'Tim kiem theo gia tri mac dinh'**, () => {  
9. **var** total\_record = **""**;  
10. cy.get(**'ul[total-items="employments.total"]'**).next();  
11. .then(total => {  
12. total\_record = total.text();  
  
13. cy.get(**'a[ng-click="search()"]'**)  
14. .click();  
  
15. cy.get(**'ul[total-items="employments.total"]'**).next()  
16. .should(**'contain'**, total\_record);  
17. });  
18. });

Giải thích:

9. Khởi tạo biến total\_record bằng rỗng.

10 -> 12: Nhập giá trị rỗng vừa tạo vào textbox tên nghề nghiệp.

13 -> 14: Thực hiện nhấn vào botton Tìm kiếm.

15 -> 16: Khẳng định kết quả hiển thị ra là tất cả các giá trị.

19. it(**'Tim kiem khong ra ket qua'**, () => {  
20. cy.get(**'form[ng-submit="search()"]'**).find(**'input'**).type(**'67\*\*\*'**);  
21. cy.get(**'a[ng-click="search()"]'**);  
22. .click();  
  
23. cy.get(**'ul[total-items="employments.total"]'**).next();  
24. .should(**'contain'**, **'Không có bản ghi nào.'**));  
25. });

Giải thích:

19 -> 25: Tương tự tìm kiếm khi nhập vào textbox chuỗi ký tự 67\*\*\* và kết quả trả ra là thông báo: “Không có bản ghi nào.”

26. it(**'Tim kiem với nhap ki tu space ở dau cuoi'**, () => {  
27. **var** total\_record = **"trang"**;  
28. **var** imSpaceStr = **' trang '**;  
29. cy.get(**'ul[total-items="employments.total"]'**).next();  
30. .then(total => {  
31. total\_record = total.text();  
32. cy.get(**'form[ng-submit="search()"]'**).find(**'input'**).type(imSpaceStr);  
33. cy.get(**'a[ng-click="search()"]'**);  
34. .click();  
  
35. cy.get(**'ul[total-items="employments.total"]'**).next()  
36. .should(**'contain'**, total\_record);  
37. });  
38. });

Giải thích:

26 -> 38: Tương tự thực hiện lệnh tìm kiếm với chuỗi không có space, sau đó tìm kiếm với chuỗi có space ở đầu và cuối, kết quả tìm kiếm 2 lần phải như nhau.

39. it(**'Tim kiem khi nhap ky tu dac biet'**, () => {  
40. **var** test = **">"**;  
41. cy.get(**'form[ng-submit="search()"]'**).find(**'input'**).type(test);  
42. cy.get(**'a[ng-click="search()"]'**)  
43. .click();  
44. cy.get(**'table'**).find(**'tr'**).should(($data) => {  
45. expect($data).to.contain(test);  
46. });  
47. });  
48.});

Giải thích:

39 -> 48: Nhập ký tự đặc biệt là dấu “>”. Kết quả tìm kiếm các giá trị phải chứa ký tự đã nhập.

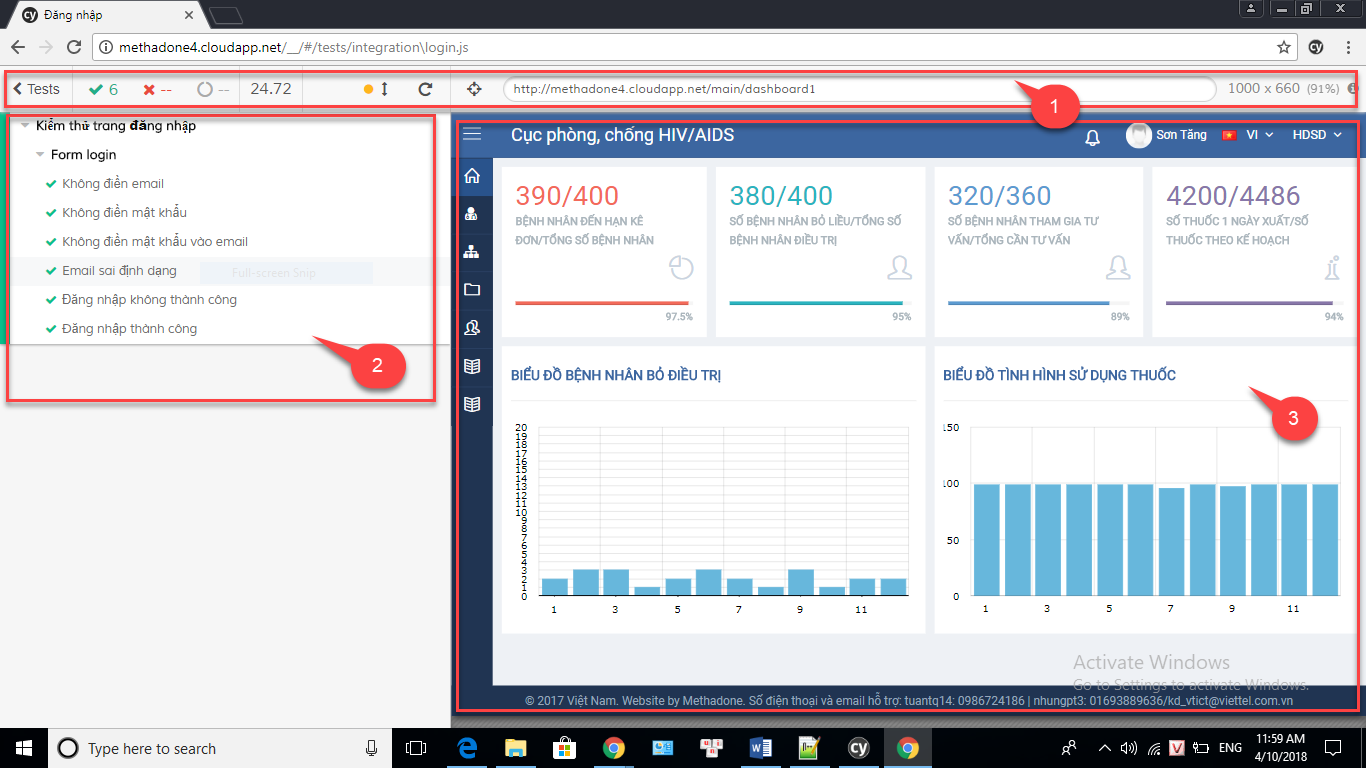
#### Cách chạy các mã lệnh trong Cypress

Sau khi mở Cypress và chọn tập tin chứa các mã lệnh, lúc này để chạy các mã lệnh, người phát triển có 2 cách để thực hiện:

* Nhấn Run All Tests để hệ thống tự động chạy lần lượt tất cả các mã lệnh hiện có.
* Nhấn trực tiếp vào từng mã lệnh nếu chỉ muốn thực hiện chạy độc lập từng mã lệnh.

Sau khi chạy mã lệnh bằng 1 trong 2 cách trên, Cypress tự động tìm trình duyệt trên máy tính cá nhân phù hợp và thực hiện chạy lệnh trên chính trình duyệt đó. Các mã lệnh được mô phỏng như thao tác của người dùng. Hình 4.7 mô tả ví dụ cụ thể về kết quả khi chạy một mã lệnh với Cypress.

Màn hình hiển thị kết quả chạy các mã lệnh đăng nhập:

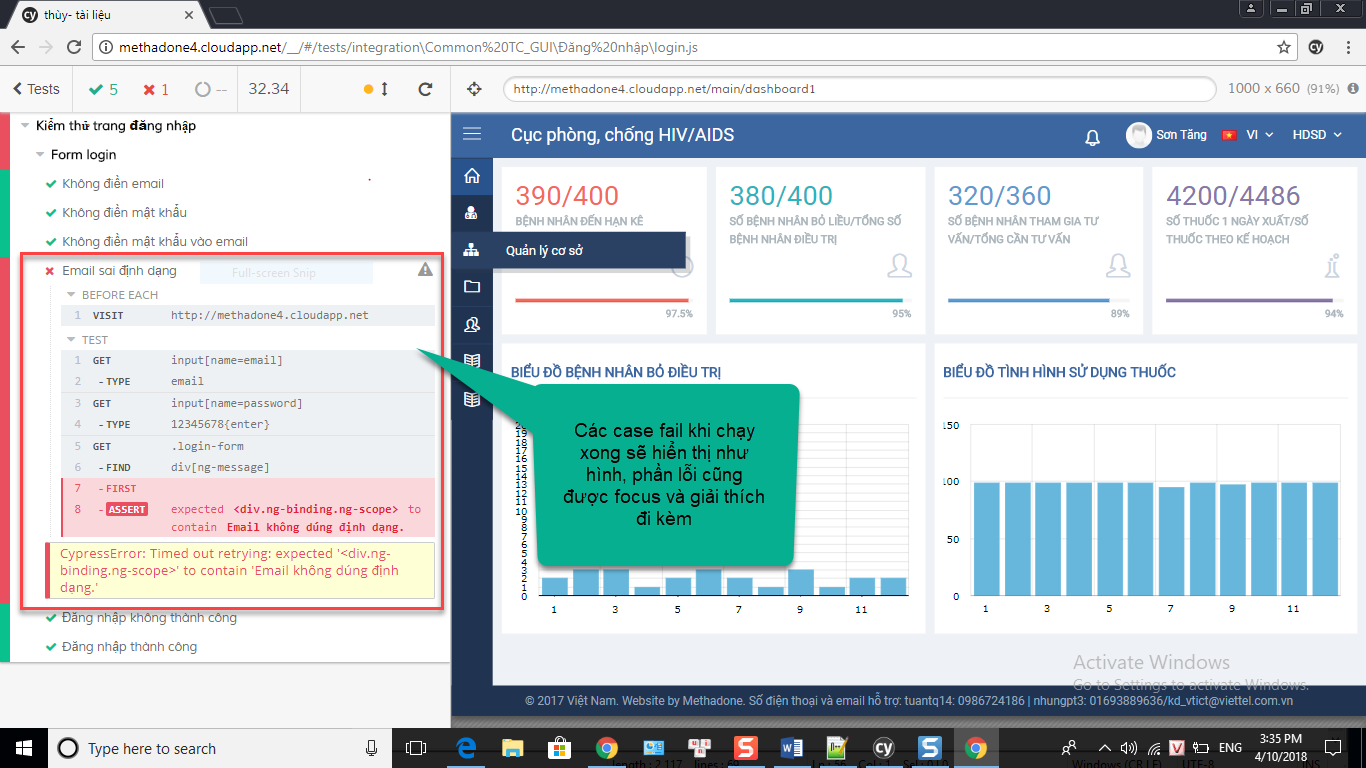


Hình 4.4 Màn hình kết quả sau khi chạy mã lệnh đăng nhập

Màn hình kết quả sau khi chạy mã lệnh chia thành 3 phân vùng:

* Vùng 1: Hiển thị các thông tin: số lượng các hàm (bao gồm cả các hàm thành công và các hàm thất bại), thời gian chạy, thời gian chờ, phóng to, truy cập lại, địa chỉ URL của trang Web đang chạy, kích thước màn hình trang Web hiển thị.
* Vùng 2: Hiển thị các hàm: khi chạy đến hàm nào, nếu thành công thì sẽ đánh dấu tích màu xanh, nếu thất bại thì đánh dấu đỏ (lỗi cũng sẽ được hiển thị cụ thể bên dưới từng hàm). Khi gặp hàm thất bại nếu chạy lần 1 kết quả thất bại thì hệ thống sẽ bỏ qua và tiếp tục chạy những hàm kế tiếp, sau đó quay lại chạy lại hàm thất bại trước đó.
* Vùng 3: Mô phỏng thao tác người dùng trên trang Web cần kiểm tra, khi các lệnh ở vùng 2 chạy thì thao tác tương ứng sẽ được thực hiện ở đây.

Khi chạy nhóm mã lệnh không thành công (fail) thì màn hình report kết quả hiển thị như sau:



Hình 4.5 Màn hình kết quả sau khi chạy mã lệnh thất bại

#### Cấu trúc mã nguồn

Cấu trúc mã nguồn của dự án chia thành 2 phần: mã nguồn cho các ca kiểm thử giao diện và mã nguồn cho các ca kiểm thử chức năng. Chi tiết mã nguồn như hình các hình vẽ bên dưới:



Hình 4.6 Cấu trúc mã nguôn cho các ca kiểm thử chức năng



Hình 4.7 Cấu trúc mã nguồn cho các ca kiểm thử giao diện

#### Ca kiểm thử thất bại và xử lý lỗi

Khi thực hiện chạy các mã lệnh, lỗi có thể xảy ra ở 2 phía: lỗi ở mã lệnh và lỗi ở chức năng hệ thống.

* Lỗi ở mã lệnh: nếu có lỗi xảy ra trong mã lệnh thì khi chạy mã lệnh, màn hình hiển thị như sau. Cypress báo đầy đủ thông tin lỗi, lỗi ở dòng lệnh, ký tự nào được đánh dấu rõ ràng. Từ đó nhà phát triển có thể xem lại và sửa đổi mã lệnh cho chính xác.

Ví dụ đoạn mã lệnh trên bị lỗi ở dòng thứ 4, khi chạy kết quả hiển thị như sau:

beforeEach(**function**(){  
 cy.visit(**'http://methadone4.cloudapp.net'**);  
 cy.get(**'input[name=email]'**).type(**'quanly@gmail.com'**);  
 cy.get(**'input[name=password]'**).type(**'12345678'**);  
 cy.get.(**'button[type=submit]'**).click();  
  
 cy.visit(**'/main/admin/administrators/employments'**);  
 })

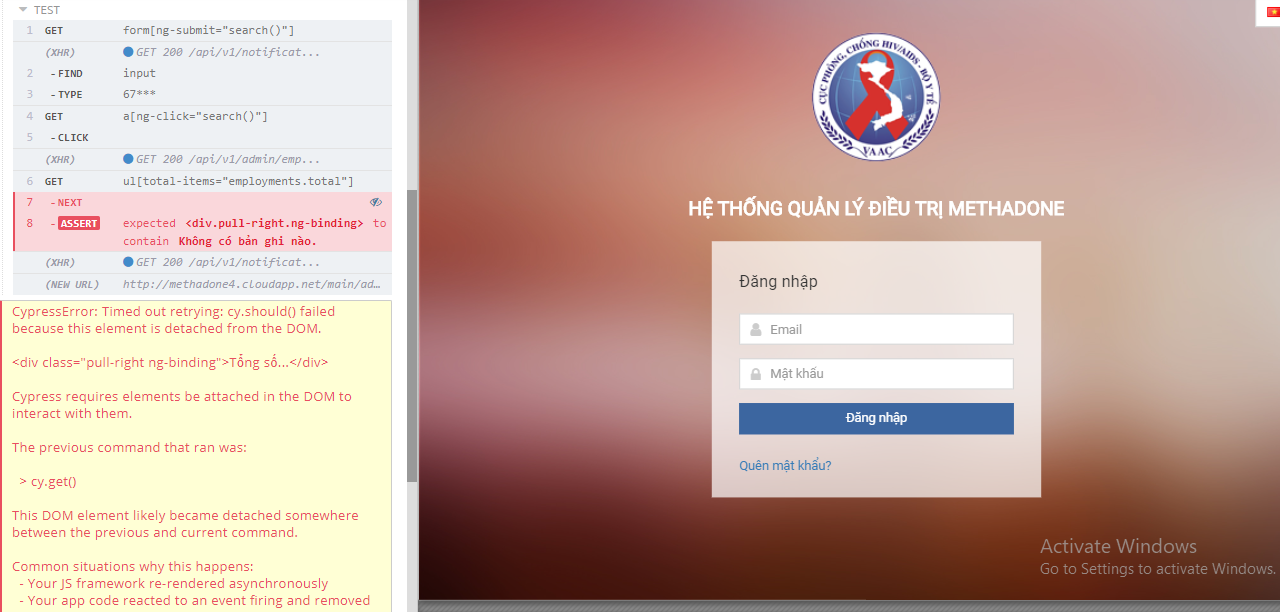


Hình 4.8 Màn hình khi mã lệnh chạy lỗi 1

* Lỗi ở chức năng hệ thống: khi mã lệnh viết đã chính xác nhưng kết quả chạy ca kiểm thử bị lỗi thì lỗi xảy ra lúc này là do hệ thống đang thực hiện sai kết quả mong muốn.

Ví dụ khi chạy đoạn lệnh kiểm tra chức năng tìm kiếm không có kết quả, kết quả đang là thất bại, nguyên nhân lỗi là hệ thống không thỏa mãn dòng lệnh đã viết. Cụ thể, ở trường hợp khi tìm kiếm không có kết quả, hệ thống phải hiển thị thông báo:” Không có bản ghi nào” nhưng trên hệ thống thật đang không có thông báo này. Kết quả hiển thị như thế này là hợp lý. Từ kết quả kiểm thử trên, nhân viên kiểm thử báo cáo lại lỗi để các nhà phát triển tiến hành xác minh và sửa lỗi.

it(**'Tim kiem khong ra ket qua'**, () => {  
 cy.get(**'form[ng-submit="search()"]'**).find(**'input'**).type(**'67\*\*\*'**);  
 cy.get(**'a[ng-click="search()"]'**)  
 .click();  
  
 cy.get(**'ul[total-items="employments.total"]'**).next();  
 .should(**'contain'**, **'Không có bản ghi nào.'**));  
 });



Hình 4.9 Màn hình khi mã lệnh chạy lỗi 2

#### Báo cáo kết quả kiểm thử

Báo cáo kết quả kiểm thử cho các ca kiểm thử giao diện và chức năng

**Báo cáo kiểm thử**

**(Test Report)**

Bảng 4.5 Bảng báo cáo kết quả kiểm thử các chức năng đã thực hiện

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Tên chức năng** | **Tổng số ca kiểm thử đã được viết** | **Số mã lệnh đã được viết** | **Thành công** | **Thất bại** |
| 1 | Đăng nhập | 6 | 6 | 6 | 0 |
| 2 | Giao diện màn hình chính | 120 | 78 | 60 | 18 |
| 3 | Giao diện màn hình thêm mới | 162 | 120 | 93 | 27 |
| 4 | Chức năng tìm kiếm, xóa | 11 | 11 | 11 | 0 |

Đánh giá kết quả đạt được: dựa trên các ca kiểm thử cơ bản đã có của ứng dụng Web, em đã xây dựng được các mã lệnh để kiểm thử những trường hợp có trong ứng dụng. Một vài trường hợp có trong các ca kiểm thử cơ bản nhưng không có trong ứng dụng cụ thể thì không được viết, vì vậy có những chức năng mà tỷ lệ số mã lệnh đã viết/ số ca kiểm thử là chênh lệch. Mặc dù vậy, trong dự án thực tế như bên trên, kiểm thử tự đông nói chung và kiểm thử tự động bằng Cypress nói riêng đã giúp rút ngắn thời gian kiểm thử, lượng lỗi tìm ra là tương đối chính xác. Từ đó góp phần nâng cao chất lượng sản phẩm.

# Chương 5: Kết luận

Kiểm thử tự động là phương pháp hữu dụng và ấn tượng được sử dụng để giúp tiến hành kiểm thử một cách hiệu quả. Tuy nhiên, không phải dự án nào cũng phù hợp với kiểm thử tự động. Nhưng không thể phủ nhận lợi ích to lớn của kiểm thử tự động mang lại như tính hiệu quả trong công việc, tính tin cậy, cải thiện chất lượng sản phẩm. Khóa luận đã tìm hiểu nghiên cứu các khái niệm, chức năng, cách hoạt động của kiểm thử tự động, cụ thể là công cụ Cypress và đã áp dụng nó trong dự án thực tế. Hiệu quả nó mang lại cho dự án thực tế là tiết kiệm thời gian cho nhân viên kiểm thử trong giai đoạn kiểm thử hồi quy và kiểm thử lặp lại khi hệ thống có thêm chức năng mới. Công cụ Cypress nhận đầu vào là các mã lệnh mô phỏng thao tác người dùng, sử dụng ngôn ngữ lập trình JavaScript và đầu ra là các ca kiểm thử đã được xác định kết quả từ đó giúp nhà phát triển tiết kiệm công sức thời gian nâng cao hiệu quả việc kiểm tra. Môi trường Cypress là một môi trường làm việc chuyên nghiệp, những hỗ trợ của nó đang dần hoàn thiện và càng ngày thì Cypress càng cải tiến thêm nhiều tính năng giúp cho những kỹ thuật viên kiểm tra thực hiện công việc dễ dàng và việc thực hiện theo nhóm cũng thống nhất hơn, đồng bộ hơn.

Tuy nhiên, trong quá trình thực hiện nghiên cứu có một số những ca kiểm thử chưa được áp dụng vào hệ thống nên việc tự động hóa hoàn toàn là không thể. Trong tương lai, em sẽ tập trung tìm các phương pháp để có thể xử lý những vấn đề hạn chế trên. Ngoài ra, phương pháp sẽ được nghiên cứu thêm để có thể áp dụng cho các ca kiểm thử phức tạp của hệ thống thực tế.

TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | Phạm Ngọc Hùng, Trương Anh Hoàng và Đặng Văn Hưng, *Giáo trình kiểm thử phần mềm*, Hà Nội, 2014. |
| [2] | "Cypress," 2018. [Online]. Available: https://www.cypress.io/. |
| [3] | "TestingVN," 2018. [Online]. Available: http://www.testingvn.com. |
| [4] | "Flickriver," 9 January 2016. [Online]. Available: http://www.flickriver.com/photos/137246140@N05/24157951942/. |
| [5] | "JMeter for beginners," 26 August 2016. [Online]. Available: http://cybosol.com/blog/jmeter-for-beginners/. |
| [6] | "Selenium Webdriver," 25 March 2016. [Online]. Available: http://selenium-interview-questions.blogspot.com/2016/03/selenium-webdriver-json-protocol.html. |
| [7] | "VIBLO," 24 Feb 2017. [Online]. Available: https://viblo.asia/p/so-sanh-giua-kiem-tra-phan-cung-va-kiem-thu-phan-mem-hardware-verification-and-software-testing-part1-924lJXLYKPM. |
| [8] | J. Colantonio, "Joecolantonio," 16 November 2017. [Online]. Available: https://www.joecolantonio.com/2017/11/16/cypress-io-vs-selenium-test-automation.. |
| [9] | DANIEL, "JAVA & WEB DEVELOPMENT," 17 May 2017. [Online]. Available: http://dolszewski.com/testing/7-reasons-why-you-should-start-using-junit-5-today/. |
| [10] | G. Dorward, "Red-badger," July 2017. [Online]. Available: https://blog.red-badger.com/blog/2017/6/16/cypress-a-genuine-alternative-to-selenium-at-last. . |
| [11] | J. Mohan, "Blogspot," 5 August 2009. [Online]. Available: http://hpqtp10.blogspot.com/2009/08/how-to-work-with-silverlight-based.html. |
| [12] | M. Utting, B. Legeard, Practical Model-Based Testing: *A Tools Approach, Morgan Kauffman*, 2007. |
| [13] | M. H. a. B. K. W. Hung Q. Nguyen, Global Software Test Automation: *A Discussion of Software Testing for Executives*, Happy About, 2006. |
| [14] | T. G. a. B. G. Elfriede Dustin, *Implementing Automated Software Testing*: How to Save Time and Lower Costs While Raising Quality, Addison-Wesley Professional, 2009. |
| [15] | C. Cherry, *Test Automation: A Story Lost in Translation*, 2012. |