BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NGUYỄN TẤT THÀNH

VIỆN ĐÀO TẠO QUỐC TẾ NTT

---ଌ◌ଌ---

ĐỒ ÁN MÔN HỌC KIỂM THỬ PHẦN MỀM

ĐỀ TÀI
TÌM HIỂU VÀ PHÂN TÍCH CÔNG CỤ KIỂM THỬ CYPRESS

Giảng viên hướng dẫn	: TS. Trần Sơn Hải
Nhóm SV thực hiện	: Quality
Mã lớp học	: 22BITV02

TP. HCM, tháng 05 năm 2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NGUYỄN TẤT THÀNH

VIỆN ĐÀO TẠO QUỐC TẾ NTT (NIIE)

---ꕥ---

ĐỒ ÁN MÔN HỌC KIỂM THỬ PHẦN MỀM

ĐỀ TÀI
TÌM HIỂU VÀ PHÂN TÍCH CÔNG CỤ KIỂM THỬ CYPRESS

Giảng viên hướng dẫn : TS. Trần Sơn Hải

Nhóm SV thực hiện : Quality

Mã lớp học : 22BITV02

TP. HCM, tháng 05 năm 2025

NGUYEN TAT THANH UNIVERSITY
NTT INSTITUTE OF INTERNATIONAL EDUCATION
GLOBAL LEARNING, GLOBAL SUCCESS

Sinh viên thực hiện:

MSSV	Họ tên
2200004202	Phạm Nguyễn Phúc Ân
2200002135	Trần Nguyễn Quốc Anh
2200010487	Nguyễn Thị Thanh Tâm
2200005824	Ngô thị Thùy Trang

LỜI CẢM ƠN

Nhóm chúng em xin gửi lời cảm ơn chân thành đến Bộ Giáo dục và Đào tạo Trường Đại học Nguyễn Tất Thành, cùng Viện Đào tạo Quốc tế NIIE đã tạo điều kiện thuận lợi để chúng em được học tập và nghiên cứu môn Kiểm thử phần mềm – một môn học thiết thực và bổ ích đối với sinh viên ngành công nghệ thông tin.

Nhóm cũng xin bày tỏ lòng biết ơn sâu sắc đến thầy Trần Sơn Hải, giảng viên phụ trách môn học, đã tận tình giảng dạy, cung cấp nhiều tài liệu quý báu và luôn sẵn sàng hỗ trợ, giải đáp thắc mắc trong suốt quá trình học tập và thực hiện bài báo cáo.

Trong khuôn khổ môn học này, nhóm chúng em lựa chọn công cụ Cypress để tìm hiểu và phân tích các mẫu kiểm thử đã được xây dựng sẵn. Qua đó, nhóm mong muốn nâng cao khả năng áp dụng kiểm thử tự động vào thực tiễn phát triển phần mềm.

Chúng em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

1.1 Tình hình chung

Trong những năm gần đây, lập trình ứng dụng web đã trở thành một xu hướng phổ biến và không thể thiếu trong hầu hết các lĩnh vực như thương mại điện tử, giáo dục, y tế và giải trí. Các doanh nghiệp và tổ chức ngày càng đầu tư nhiều vào việc phát triển các nền tảng web để phục vụ người dùng một cách tiện lợi và hiệu quả hơn. Song song với sự phát triển nhanh chóng của công nghệ web, yêu cầu về chất lượng và độ ổn định của phần mềm cũng ngày càng cao. Vì vậy, việc kiểm thử phần mềm trở thành một bước quan trọng trong quy trình phát triển. Đặc biệt, việc áp dụng các công cụ kiểm thử tự động như Cypress đã và đang đóng vai trò then chốt trong việc đảm bảo chất lượng sản phẩm. Cypress cho phép kiểm thử giao diện người dùng một cách chính xác, nhanh chóng và dễ dàng tích hợp vào quy trình CI/CD. Nhờ vào khả năng phát hiện lỗi sớm và giảm thiểu rủi ro, các công cụ này giúp tiết kiệm thời gian, công sức cho đội ngũ phát triển, đồng thời nâng cao trải nghiệm người dùng cuối. Trong bối cảnh công nghệ liên tục thay đổi, việc sử dụng kiểm thử tự động là yếu tố không thể thiếu để tạo nên các ứng dụng web chất lượng cao.

1.2 Mục đích và nội dung kiểm thử

Trong nội dung của bài báo cáo này, trước tiên nhóm chúng em sẽ tập trung phân tích các khái niệm lý thuyết và các kỹ thuật được sử dụng trong kiểm thử phần mềm. Tiếp theo, nhóm sẽ tìm hiểu tổng quan về API và nghiên cứu các tính năng nổi bật của công cụ kiểm thử phần mềm Cypress. Dựa trên nền tảng lý thuyết đã nghiên cứu, nhóm sẽ trình bày quy trình ứng dụng kiểm thử tự động bằng Cypress để kiểm thử trang web của nhà trường.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Kiểm thử phần mềm

Kiểm thử phần mềm (Software Testing) là quá trình đánh giá một hệ thống phần mềm hoặc các thành phần của nó với mục tiêu tìm ra lỗi hoặc xác minh rằng phần mềm hoạt động đúng với các yêu cầu đã đề ra. Đây là một bước không thể thiếu trong quá trình phát triển phần mềm, nhằm đảm bảo chất lượng, độ ổn định và độ tin cậy của hệ thống trước khi được triển khai đến người dùng cuối.

2.1.1 Mục tiêu của kiểm thử phần mềm

Phát hiện lỗi (Bug Detection): Đây là mục tiêu quan trọng nhất. Kiểm thử giúp phát hiện các lỗi trong phần mềm trước khi sản phẩm được phát hành, giúp tiết kiệm chi phí và thời gian so với việc sửa lỗi sau khi triển khai.

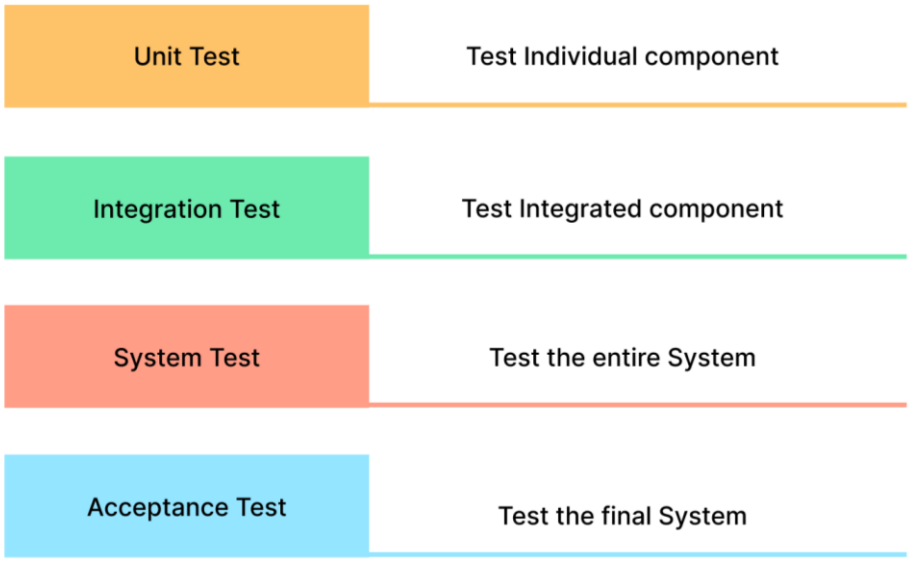
Đảm bảo chất lượng (Quality Assurance): Kiểm thử giúp đảm bảo rằng phần mềm đáp ứng các tiêu chuẩn chất lượng về mặt chức năng, hiệu suất, giao diện người dùng và độ bảo mật.

Xác nhận và thẩm định (Verification and Validation): Đảm bảo rằng phần mềm "được xây dựng đúng yêu cầu" và "được xây dựng đúng cách".

Tăng cường sự tin cậy: Khi phần mềm vượt qua các bài kiểm thử nghiêm ngặt, người dùng và khách hàng sẽ tin tưởng vào độ ổn định và hiệu quả của sản phẩm.

2.1.2 Các loại kiểm thử phần mềm

Kiểm thử phần mềm được chia thành nhiều loại tùy theo mục tiêu, kỹ thuật áp dụng hoặc giai đoạn phát triển phần mềm. Dưới đây là các cách phân loại phổ biến.



Hình 2-1. Phân loại theo mức độ kiểm thử

Kiểm thử đơn vị (Unit Testing): Kiểm tra các đơn vị nhỏ nhất của mã (thường là hàm hoặc class) để đảm bảo hoạt động đúng.

Kiểm thử tích hợp (Integration Testing): Kiểm tra các module khi tích hợp với nhau để phát hiện lỗi trong giao tiếp giữa các phần.

Kiểm thử hệ thống (System Testing): Kiểm tra toàn bộ hệ thống như một khối thống nhất để đảm bảo phần mềm đáp ứng yêu cầu.

Kiểm thử chấp nhận (Acceptance Testing): Kiểm thử cuối cùng trước khi sản phẩm được phát hành để xác nhận rằng nó đáp ứng các tiêu chí của người dùng.

2.1.3 Phân loại theo mục tiêu kiểm thử

Kiểm thử chức năng (Functional Testing): Được thực hiện để xác minh rằng phần mềm thực hiện đúng các chức năng được yêu cầu. Ví dụ như kiểm tra đăng nhập, đăng ký, xử lý thanh toán,...

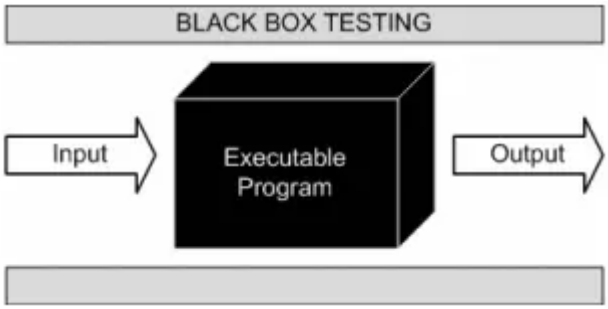
Kiểm thử phi chức năng (Non-functional Testing): Kiểm tra các thuộc tính như hiệu năng, bảo mật, khả năng mở rộng, tính tương thích,...

Kiểm thử hồi quy (Regression Testing): Kiểm tra lại các chức năng đã hoạt động đúng sau khi có thay đổi (fix bug, cập nhật tính năng mới).

Kiểm thử thăm dò (Exploratory Testing): Người kiểm thử tự do khám phá phần mềm mà không tuân theo test case cố định.

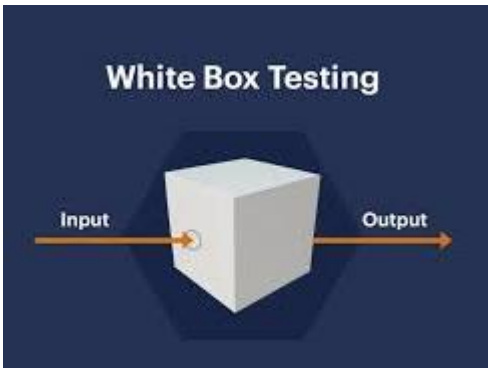
2.1.4 Phân loại theo kỹ thuật kiểm thử

Kiểm thử hộp đen (Black-box Testing): Người kiểm thử không cần biết về cấu trúc bên trong hoặc mã nguồn, chỉ quan tâm đến đầu vào và đầu ra.



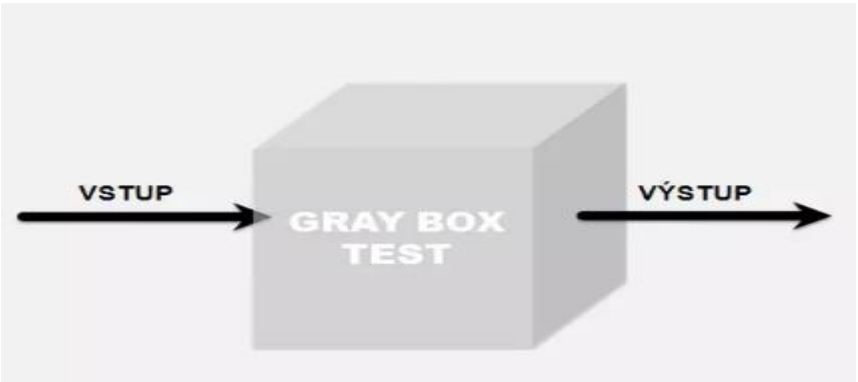
Hình 2-2. Kiểm thử hộp đen

Kiểm thử hộp trắng (White-box Testing): Người kiểm thử cần biết về mã nguồn và kiểm tra luồng logic, cấu trúc điều kiện, vòng lặp,...



Hình 2-3. Kiểm thử hộp trắng

Kiểm thử hộp xám (Gray-box Testing): Kết hợp giữa hai loại trên: người kiểm thử có một phần hiểu biết về bên trong hệ thống.



Hình 2-4. Kiểm thử hộp xám

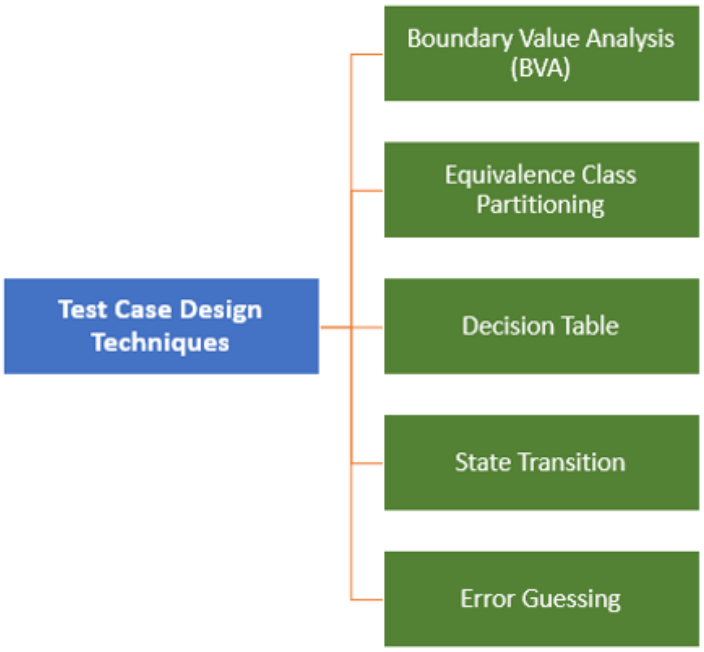
2.1.5 Quy trình kiểm thử phần mềm

Hình 2-5. Quy trình kiểm thử

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

1. Phân tích yêu cầu: Hiểu rõ hệ thống, nghiệp vụ, và yêu cầu để biết nên kiểm thử gì.
2. Lập kế hoạch kiểm thử: Xác định phạm vi, mục tiêu, chiến lược và tài nguyên cần thiết.
3. Thiết kế test case: Tạo ra các kịch bản kiểm thử mô tả đầu vào, hành động, và kết quả mong đợi.
4. Thiết lập môi trường kiểm thử: Cấu hình hệ thống, dữ liệu, công cụ phục vụ cho việc kiểm thử.
5. Thực hiện kiểm thử: Chạy test case và ghi lại kết quả thực tế.
6. Báo cáo lỗi: Nếu có lỗi, báo cáo chi tiết để lập trình viên sửa chữa.
7. Kiểm thử lại và hồi quy: Sau khi sửa lỗi, kiểm thử lại để đảm bảo không phát sinh lỗi mới.

2.1.6 Kỹ thuật thiết kế test case



Hình 2-6. Kỹ thuật thiết kế test case

Phân vùng tương đương (Equivalence Partitioning): Chia dữ liệu đầu vào thành các nhóm tương đương và kiểm thử một đại diện cho mỗi nhóm.

Phân tích giá trị biên (Boundary Value Analysis): Tập trung kiểm tra các giá trị tại ranh giới, thường là nơi dễ xảy ra lỗi nhất.

Bảng quyết định (Decision Table): Dùng bảng để biểu diễn logic phức tạp theo điều kiện và hành động.

Biểu đồ chuyển trạng thái (State Transition): Kiểm thử các hành vi của hệ thống khi chuyển giữa các trạng thái khác nhau.

Kỹ thuật phỏng đoán lỗi (Error Guessing): Kỹ thuật này dựa trên kinh nghiệm của người kiểm thử. Tester sử dụng kiến thức về phần mềm, các trường hợp biên và tình huống ngoại lệ để dự đoán lỗi có thể xảy ra. Từ đó, họ thiết kế các ca kiểm thử dựa trên những lỗi tiềm ẩn đã gặp trong quá khứ.

2.2 Công cụ Cypress

2.2.1 Giới thiệu về Cypress

Hình 2-7. Ảnh công cụ Cypress

Cypress là một công cụ kiểm thử tự động mã nguồn mở hiện đại, được xây dựng dành riêng cho việc kiểm thử ứng dụng web. Công cụ này được phát triển để giải quyết những khó khăn khi sử dụng các framework kiểm thử truyền thống như Selenium. Cypress hoạt động trực tiếp trong trình duyệt, có nghĩa là nó có thể tiếp cận DOM (Document Object Model) và các sự kiện của trình duyệt một cách tự nhiên, nhanh chóng và đồng bộ.

Điểm nổi bật của Cypress là chạy trên cùng một vòng lặp sự kiện với ứng dụng web, giúp kiểm thử diễn ra mượt mà và dễ theo dõi hơn. Điều này tạo ra sự khác biệt lớn với các công cụ kiểm thử như Selenium – vốn hoạt động như một phần mềm điều khiển trình duyệt từ bên ngoài, dẫn đến độ trễ và khó debug.

Cypress phù hợp với cả kiểm thử đơn vị (unit testing), kiểm thử tích hợp (integration testing), kiểm thử end-to-end (E2E testing), và đặc biệt là các ứng dụng SPA (Single Page Application) sử dụng React, Vue hoặc Angular.

2.2.2 Giới thiệu về End-to-End (E2E)

End-to-End (E2E) testing – hay kiểm thử đầu-cuối – là một kỹ thuật kiểm thử phần mềm dùng để kiểm tra toàn bộ luồng hoạt động của ứng dụng, từ đầu vào của người dùng đến kết quả đầu ra cuối cùng, nhằm đảm bảo rằng hệ thống hoạt động đúng như mong đợi trong môi trường thực tế.

Mục tiêu chính: Mô phỏng hành vi thực tế của người dùng để kiểm tra toàn bộ hệ thống (giao diện, logic, cơ sở dữ liệu, v.v.).

2.2.3 Kiểm thử end-to-end (E2E)

Kiểm thử đầu cuối đóng vai trò là điểm kiểm tra cuối cùng trước khi phát hành sản phẩm, đảm bảo rằng mọi thành phần của ứng dụng tương tác chính xác với các thành phần khác và toàn bộ hệ thống đáp ứng các yêu cầu về thiết kế và người dùng. Sau đây là lý do tại sao nó rất cần thiết:

a) Xác thực toàn diện

Kiểm tra E2E đánh giá toàn bộ ứng dụng, đảm bảo mọi bộ phận hoạt động ăn khớp với nhau.

- Kiểm thử E2E không chỉ kiểm tra từng chức năng đơn lẻ mà còn kiểm tra sự phối hợp giữa các thành phần trong toàn bộ hệ thống.
- Ví dụ: Khi người dùng đặt hàng trên một website thương mại điện tử, hệ thống cần đảm bảo rằng:
  + Thêm sản phẩm vào giỏ hàng hoạt động đúng.
  + Thông tin giỏ hàng được truyền đến bước thanh toán.
  + Thanh toán xử lý đúng và cập nhật tồn kho.
  ⇨ Kiểm thử E2E đảm bảo tất cả các bước này hoạt động liền mạch với nhau.

b) Trải nghiệm người dùng

Xác minh hệ thống từ góc nhìn của người dùng, nâng cao trải nghiệm tổng thể của người dùng.

- Kiểm thử E2E mô phỏng hành vi của người dùng thực tế – từ việc đăng nhập, điều hướng trang, đến thao tác click, nhập dữ liệu...
- Nó giúp đánh giá:
  + Tính logic của luồng hoạt động.
  + Sự tiện lợi và rõ ràng trong giao diện.
  + Các lỗi có thể xảy ra trong quá trình sử dụng thực tế.
  ⇨ Góp phần nâng cao trải nghiệm tổng thể cho người dùng cuối.

c) Phát hiện lỗi sớm

Xác định vấn đề sớm, giảm chi phí và công sức sửa chữa sau này.

- Thông qua kiểm thử E2E, ta có thể phát hiện sớm những lỗi phát sinh do:
  + Tích hợp không đúng giữa các module.
  + Thay đổi nhỏ gây ảnh hưởng đến phần khác.
  + Lỗi logic chỉ xảy ra khi các thành phần kết hợp với nhau.
  ⇨ Việc phát hiện lỗi sớm sẽ tiết kiệm chi phí sửa chữa, đặc biệt ở giai đoạn cuối của dự án.

d) Đảm bảo hiệu suất

Xác thực hiệu suất trong điều kiện thực tế, đảm bảo ứng dụng có thể xử lý lưu lượng và tải dữ liệu dự kiến.

- Kiểm thử E2E giúp mô phỏng môi trường và tình huống thực tế:
  + Người dùng truy cập cùng lúc.
  + Tải dữ liệu lớn.
  + Thực hiện thao tác liên tục.
- Qua đó, có thể đo lường:
  + Tốc độ phản hồi.
  + Khả năng xử lý đồng thời.
  + Độ ổn định dưới tải nặng.
  ⇨ Đảm bảo rằng hệ thống hoạt động mượt mà khi được sử dụng thực tế.

e) Kiểm tra bảo mật

Xác nhận các giao thức bảo mật có hiệu quả trên mọi phần của ứng dụng.

- Kiểm thử E2E còn có thể tích hợp kiểm tra các vấn đề bảo mật, chẳng hạn:
  + Xác minh quyền truy cập: Người dùng thường không thể truy cập tính năng của admin.
  + Kiểm tra thông tin nhạy cảm không bị lộ.
  + Xác nhận các cơ chế bảo vệ như mã hóa, xác thực, timeout phiên hoạt động đúng cách.
  ⇨ Giúp đảm bảo toàn bộ hệ thống an toàn trước các rủi ro.

f) Độ tin cậy

Thiết lập sự tin tưởng vào tính ổn định và độ tin cậy của ứng dụng trước khi đưa vào sử dụng.

- Nếu các bài kiểm thử E2E liên tục chạy thành công, ta có thể tự tin rằng hệ thống:
  + Đáp ứng yêu cầu kỹ thuật.
  + Không phát sinh lỗi trong quá trình sử dụng bình thường.
  + Hoạt động ổn định sau nhiều lần thay đổi mã nguồn (refactor, cập nhật...).
  ⇨ Tăng độ tin cậy và giảm rủi ro khi phát hành sản phẩm ra thị trường.

g) Tuân thủ yêu cầu

Đảm bảo sản phẩm cuối cùng đáp ứng cả yêu cầu kỹ thuật và kinh doanh.

- Một trong những mục tiêu lớn của kiểm thử E2E là xác minh rằng:

+ Sản phẩm cuối cùng đáp ứng đúng yêu cầu kỹ thuật được mô tả trong tài liệu.
+ Đồng thời cũng phù hợp với yêu cầu kinh doanh, như quy trình mua hàng, quy tắc chiết khấu, điều kiện giao hàng,...

⇨ Đảm bảo rằng phần mềm cuối cùng đáp ứng mục tiêu của cả doanh nghiệp lẫn người dùng.

2.2.4 Kiến trúc Cypress

Cypress có một kiến trúc đặc biệt. Thay vì chạy kiểm thử từ bên ngoài trình duyệt (remote control), Cypress chạy trong cùng một quy trình (process) với ứng dụng web – ngay bên trong trình duyệt.

Thành phần chính của kiến trúc Cypress:

Bảng 2-1. Thành phần chính của Cypress

Thành phần	Vai trò chính
Test Runner (GUI)	Giao diện trực quan để viết, chạy và theo dõi các bài kiểm thử.
Cypress Node Server	Quản lý cấu hình, ghi log, xử lý yêu cầu mạng, hỗ trợ backend trong kiểm thử.
Browser (Trình duyệt)	Nơi chạy ứng dụng web và mã kiểm thử Cypress song song (chung process).
Spec Files (Test files)	Chứa các bài kiểm thử viết bằng JavaScript.

2.2.5 Mục đích dùng Cypress

Cypress hỗ trợ nhiều loại kiểm thử khác nhau, nhằm đảm bảo ứng dụng hoạt động chính xác, ổn định và phù hợp với kỳ vọng người dùng. Dưới đây là một số loại kiểm thử phổ biến và mục đích chính của chúng khi sử dụng Cypress:

Bảng 2-2. Mục đích của các loại kiểm thử

Loại kiểm thử	Mục đích chính
E2E Testing	Mô phỏng hành vi người dùng thật
Component Testing	Kiểm thử từng phần giao diện độc lập
API Testing	Đảm bảo các API hoạt động đúng

Network Testing	Giả lập phản hồi API, kiểm thử UI với nhiều tình huống
Smoke Testing	Kiểm thử nhanh sau khi deploy
Regression Testing	Đảm bảo chức năng cũ không hỏng sau cập nhật
Negative Testing	Kiểm thử hệ thống xử lý dữ liệu sai/thất bại như thế nào

Việc sử dụng Cypress cho các mục đích kiểm thử này giúp cải thiện chất lượng phần mềm toàn diện từ giao diện người dùng đến logic xử lý nền.

2.2.6 Những tính năng chính của Cypress

a) Kiểm thử End-to-End (E2E)

Mô phỏng các hành vi thực tế của người dùng: truy cập trang, nhập liệu, click, kéo thả, v.v.

- Cypress có thể thực hiện các thao tác giống như người dùng thật:
  + Truy cập trang web (cy.visit())
  + Nhập dữ liệu vào form (cy.type())
  + Click vào nút hoặc liên kết (cy.click())
  + Thao tác kéo thả, chọn dropdown, điều hướng giữa các trang,...
  + Giúp đảm bảo luồng sử dụng thật sự hoạt động đúng.

Tự động hóa toàn bộ luồng từ đầu đến cuối.

- Kiểm thử E2E kiểm tra toàn bộ quy trình (ví dụ: từ đăng nhập → chọn sản phẩm → thanh toán).

⇨ Giúp phát hiện lỗi tích hợp giữa các module khác nhau trong hệ thống.

b) Tự động reload (Auto-reload)

Khi chỉnh sửa file test, Cypress tự động reload và chạy lại test ngay lập tức trong trình duyệt.

- Mỗi khi thay đổi mã trong file test, Cypress sẽ tự động phát hiện và chạy lại test ngay lập tức trong trình duyệt đang mở.

⇨ Không cần phải tắt/mở lại hoặc chạy lệnh thủ công.

Không cần chạy lại thủ công.

- Tăng tốc quá trình viết và kiểm thử script.

- Trải nghiệm tương tác trực tiếp, phù hợp với tư duy "test while coding".

c) *Kiểm thử component / UI riêng lẻ*

Có thể kiểm thử từng thành phần nhỏ (component) trong ứng dụng React, Vue, Angular,…

- Cypress không chỉ hỗ trợ kiểm thử toàn trang mà còn hỗ trợ kiểm thử unit và component testing.

⇨ Có thể kiểm tra từng button, form, modal,... riêng biệt mà không cần phải render toàn ứng dụng.

- Ví dụ: Test riêng 1 component LoginForm của React mà không cần kết nối backend.

d) *Time-travel debugging*

Cypress ghi lại từng bước kiểm thử → có thể quay lại từng bước và xem trạng thái DOM tại thời điểm đó.

- Mỗi hành động (click, nhập, kiểm tra) đều được ghi lại.
- Có thể xem lại trạng thái DOM, dữ liệu và lỗi tại từng bước cụ thể.

Dễ dàng debug lỗi hơn rất nhiều.

- Click vào từng bước trong Test Runner để xem DOM tại thời điểm đó.

⇨ Tăng tốc độ phát hiện và sửa lỗi.

e) *Kiểm soát mạng (Network Control)*

Intercept và giả lập các request API : cy.intercept().

- Có thể chặn, thay đổi hoặc mô phỏng các phản hồi API mà ứng dụng gọi tới backend.

Cho phép test trong trường hợp server trả lỗi, trả chậm, hoặc phản hồi giả.

- Giúp kiểm tra ứng dụng phản ứng như thế nào nếu:
  + Server trả về lỗi 500.
  + Kết nối bị trễ.
  + Phản hồi bị thay đổi nội dung.
  ⇨ Không cần phụ thuộc vào backend thật.

f) *Đợi tự động (Automatic waiting)*

Cypress tự động đợi cho các phần tử xuất hiện, tải xong, hoặc hành động hoàn tất.

- Cypress đợi tự động cho đến khi điều kiện được thỏa mãn: phần tử xuất hiện, ajax hoàn tất, animation xong,...
- Không cần wait() thủ công như trong Selenium.
- ⇨ Giảm lỗi "flaky" (test không ổn định do timing).

g) *Giao diện Test Runner trực quan*

Có GUI hiển thị kết quả từng bước kiểm thử, DOM hiện tại, lỗi cụ thể.

- Khi chạy test bằng npx cypress open, Cypress hiển thị:
  + Tên test case.
  + Kết quả từng bước.
  + Ảnh chụp DOM, lỗi (nếu có).
  + ⇨ Dễ quan sát, dễ xác định test nào fail và tại sao fail.

h) *Chụp ảnh màn hình và quay video*

Tự động chụp ảnh nếu test bị fail.

- Khi test bị lỗi, Cypress sẽ chụp ảnh màn hình tại thời điểm xảy ra lỗi.

Có thể ghi lại toàn bộ quá trình chạy test → hữu ích khi tích hợp CI/CD.

- Khi chạy test trong chế độ headless (npx cypress run), Cypress có thể ghi video toàn bộ phiên kiểm thử.

i) *Tích hợp CI/CD dễ dàng*

Hỗ trợ các hệ thống như GitHub Actions, GitLab CI, CircleCI, Jenkins,…

- Dễ cấu hình để chạy test tự động mỗi khi có cập nhật code (push/pull request).
- Kết hợp với npx cypress run để kiểm thử tự động.
- ⇨ Phù hợp với quy trình DevOps và phát triển liên tục.

j) *Dễ mở rộng & có hệ sinh thái plugin*

Nhiều plugin hữu ích từ cộng đồng :

- Cypress có plugin marketplace với nhiều công cụ hỗ trợ nâng cao:
  + Gửi cảnh báo lỗi lên Slack, Discord,...
  + Tạo báo cáo nâng cao (Allure, Mochawesome,...)

+ Visual Testing – So sánh ảnh giao diện trước/sau để phát hiện lỗi UI.

⇨ Dễ tích hợp vào hệ thống lớn và chuyên nghiệp hơn.

2.2.7 Thực hành với Cypress

Cypress là một công cụ kiểm thử tự động mã nguồn mở, hiện đại và mạnh mẽ, được thiết kế đặc biệt cho kiểm thử ứng dụng web (end-to-end testing). Cypress nổi bật nhờ sự tích hợp chặt chẽ với trình duyệt, giúp việc viết, chạy và gỡ lỗi test dễ dàng và trực quan hơn, đặc biệt phù hợp với developer và tester hiện đại trong môi trường Agile.

Tốc độ và phản hồi nhanh: Cypress chạy trực tiếp trong trình duyệt, cho phép kiểm thử xảy ra ngay trong ngữ cảnh của ứng dụng, từ đó cung cấp phản hồi nhanh chóng và chính xác.

Tích hợp DevTools mạnh mẽ: Mỗi bước test đều có thể được xem lại trong giao diện GUI, đi kèm với log chi tiết, ảnh chụp màn hình và video tự động (nếu cấu hình).

Kiến trúc đơn giản: Cypress không phụ thuộc vào Selenium, mà sử dụng kiến trúc riêng dựa trên Node.js, do đó dễ cài đặt và cấu hình. Chỉ cần một dòng lệnh là có thể khởi tạo và chạy được.

Hỗ trợ nhiều loại kiểm thử:

- End-to-End Testing
- Integration Testing
- Unit Testing

Tích hợp CI/CD dễ dàng: Cypress dễ dàng tích hợp vào pipeline CI/CD như GitHub Actions, GitLab CI, Jenkins, CircleCI, v.v… bằng cách sử dụng dòng lệnh cypress run.

Hệ sinh thái phong phú: Cung cấp nhiều plugin chính thức và cộng đồng mạnh mẽ, ví dụ như cypress-axe (accessibility testing), cypress-real-events, cypress-mochawesome-reporter.

Khả năng ghi log và debug mạnh: Mỗi hành động test có thể được debug trực tiếp trong trình duyệt với DevTools, cho phép kiểm tra trạng thái DOM, network, console tại từng thời điểm.

Giao diện thân thiện: Cypress cung cấp một Test Runner GUI hiển thị trạng thái của từng test case, các bước được thực hiện và kết quả chi tiết.

2.2.8 Điều hướng quy trình làm việc

Luồng hoạt động (Flow) cơ bản:

1. Cypress khởi động một trình duyệt thực tế (Chrome, Edge, Electron,...).
2. Nó tải ứng dụng + mã kiểm thử vào cùng một frame trong trình duyệt.
3. Các bài test trong spec.js chạy cùng lúc và có thể truy cập trực tiếp vào DOM, localStorage, sessionStorage, cookies,…
4. Bất kỳ yêu cầu mạng nào (API call, AJAX, XHR…) có thể bị chặn, sửa đổi hoặc giả lập thông qua cy.intercept().
5. Kết quả được gửi về Test Runner → nơi thấy bước kiểm thử, trạng thái, DOM, lỗi,…

2.2.9 Một số câu lệnh cơ bản trong Cypress

- cy.visit('https://example.cypress.io'): Khởi động một trang web
- cy.get('selector')
- cy.get('#submit-button').click(): Click vào phần tử
- cy.get('#email').type('user@example.com'): Nhập dữ liệu vào input
- cy.get('.message').should('contain', 'Đăng nhập thành công'): Kiểm tra nội dung (Assertions)
- cy.get('#agree').check().should('be.checked'): Kiểm tra trạng thái checkbox / radio
- cy.get('#newsletter').uncheck().should('not.be.checked'): Kiểm tra trạng thái checkbox / radio
- cy.get('select').select('Option 1'): Làm việc với dropdown
- cy.get('select').should('have.value', 'option1'): Làm việc với dropdown
- cy.get('.item').each(($el, index, $list) => {

  // thực hiện hành động với từng $el

  }): Lặp qua các phần tử

- cy.get('#username').invoke('val').then((username) => {
  cy.log('Username là: ' + username)

  }): Lưu trữ và sử dụng giá trị

a) Kỹ thuật CSS Selector

CSS Selector trong Cypress là kỹ thuật dùng để xác định và truy cập các phần tử trên trang web thông qua cú pháp CSS. Đây là phương pháp phổ biến và mạnh mẽ giúp kiểm thử tự động xác định đúng các thành phần trong DOM để thực hiện các hành động như click, nhập dữ liệu hoặc kiểm tra nội dung. Trong Cypress, CSS Selector thường được sử dụng với cú pháp cy.get('selector').

b) Các CSS Selector phổ biến

Selector theo ID

- Cú pháp: #id-name
- Ví dụ: cy.get('#login-button')

Selector theo class

- Cú pháp: .class-name
- Ví dụ: cy.get('.nav-item')

Selector theo thẻ HTML

- Cú pháp: tagname
- Ví dụ: cy.get('button')

Selector theo thuộc tính (attribute)

- Cú pháp: [attribute="value"]
- Ví dụ: cy.get('input[type="email"]')

Selector kết hợp (kết hợp nhiều điều kiện)

- Cú pháp: tag.class, tag#id, tag[attribute="value"]
- Ví dụ: cy.get('input#username[type="text"]')

Selector con và tổ tiên (descendant & child)

- Cú pháp:
  + A B: phần tử B là con cháu của A
  + A > B: phần tử B là con trực tiếp của A
- Ví dụ:
  + cy.get('.form-group input')
  + cy.get('ul > li')

Pseudo-class selector (chọn theo trạng thái đặc biệt)

- Cú pháp: :nth-child(n), :first-child, :last-child, :checked, v.v.
- Ví dụ:
  + cy.get('li:first-child')
  + cy.get('input:checked')

2.2.10 So sánh các công cụ kiểm thử tự động

Bảng 2-3. So sánh các công cụ kiểm thử tự động

Tiêu chí	Cypress	Selenium	Playwright
Ngôn ngữ	JavaScript	Java, Python, JS...	JavaScript, Python
Tốc độ	Nhanh	Chậm hơn	Nhanh
Debug	Giao diện tốt	Khó	Tốt
Giao diện đồ họa	Có	Không	Không
Cài đặt	Dễ (npm)	Phức tạp	Dễ
Hỗ trợ kiểm thử API	Có	Cần thêm tool	Có
Tích hợp CI/CD	Tốt	Tốt	Tốt
Hỗ trợ mobile	Không	Có	Có

2.3 API

2.3.1 API là gì?

API (Application Programming Interface – Giao diện lập trình ứng dụng) là một tập hợp các định nghĩa, quy tắc và công cụ cho phép các phần mềm hoặc hệ thống khác nhau có thể giao tiếp và tương tác với nhau. API đóng vai trò như một "cầu nối" giữa các phần mềm, cho phép gửi yêu cầu và nhận lại kết quả mà không cần can thiệp vào mã nguồn nội bộ hoặc logic xử lý phía sau.

Ví dụ thực tế: Khi người dùng sử dụng ứng dụng đặt đồ ăn, họ không trực tiếp giao tiếp với cơ sở dữ liệu của nhà hàng. Thay vào đó, ứng dụng sẽ gọi API của hệ thống đặt hàng để lấy danh sách món ăn, gửi đơn đặt hàng và theo dõi trạng thái đơn hàng. Tất cả những thao tác đó đều thông qua API.

Một API thông thường bao gồm:

- Các endpoint (điểm cuối): là địa chỉ cụ thể trong hệ thống để thực hiện hành động.

- Các phương thức (methods): GET (lấy dữ liệu), POST (tạo mới), PUT (cập nhật), DELETE (xóa)
- Các tham số và tiêu đề (parameters and headers): chứa dữ liệu đầu vào và thông tin xác thực.

2.3.2 Các loại API phổ biến

a) REST API (Representational State Transfer)

Sử dụng các phương thức HTTP như GET, POST, PUT, DELETE.

Định dạng dữ liệu trả về thường là JSON hoặc XML.

Đơn giản, dễ sử dụng, không có trạng thái (stateless).

Được ứng dụng nhiều trong các hệ thống web, di động hiện đại.

b) SOAP API (Simple Object Access Protocol)

Là giao thức dựa trên XML với tiêu chuẩn nghiêm ngặt.

Có tính bảo mật cao, hỗ trợ giao dịch phức tạp.

Thường được sử dụng trong các hệ thống doanh nghiệp lớn như ngân hàng, bảo hiểm.

c) GraphQL

Cho phép client truy vấn chính xác những gì họ cần.

Giảm bớt dữ liệu không cần thiết so với REST.

Dễ kiểm soát và tối ưu hiệu suất ở cả phía client và server.

d) WebSocket API

Cho phép truyền dữ liệu hai chiều (bi-directional) giữa client và server theo thời gian thực.

Rất phù hợp với các ứng dụng như chat, dashboard theo dõi thời gian thực, game online.

e) Open API / Swagger

Là một chuẩn mô tả API nhằm tạo tài liệu, thử nghiệm, và mô phỏng API một cách dễ dàng.

Giúp các nhóm phát triển có thể hiểu và triển khai API đồng nhất.

2.3.3 Đặc điểm của kiểm thử API

Việc kiểm thử API có một số đặc điểm nổi bật so với kiểm thử giao diện người dùng (UI Testing), bao gồm:

- Không phụ thuộc giao diện người dùng: Kiểm thử API tập trung vào lớp logic và truyền dữ liệu, không cần có giao diện đồ họa.
- Tốc độ xử lý nhanh hơn: Vì không cần tải hay hiển thị giao diện, kiểm thử API thường được thực hiện nhanh chóng.
- Tính chính xác cao: Kiểm tra trực tiếp logic nghiệp vụ và các phản hồi từ hệ thống.
- Tự động hóa tốt: Phù hợp để tích hợp vào các quy trình CI/CD, giúp kiểm thử liên tục.
- Phát hiện lỗi sớm: Do API thường là nền tảng chính giữa các hệ thống, nên việc phát hiện lỗi sớm giúp giảm chi phí sửa lỗi sau này.
- Thử nghiệm nhiều tình huống phức tạp: Dễ dàng giả lập các trường hợp như truyền sai kiểu dữ liệu, thiếu thông tin, lỗi mạng, xác thực sai,…

2.3.4 Các phương pháp kiểm thử API

a) *Functional Testing (Kiểm thử chức năng)*

Mục tiêu: Kiểm tra API có thực hiện đúng chức năng yêu cầu không.

Ví dụ: Gửi yêu cầu GET để lấy danh sách sản phẩm và kiểm tra xem dữ liệu trả về có đúng định dạng, số lượng và thông tin cần thiết không.

b) *Performance Testing (Kiểm thử hiệu năng)*

Đo lường thời gian phản hồi, tốc độ xử lý của API khi có số lượng lớn yêu cầu đồng thời.

Dùng các công cụ như JMeter, Postman, K6 để thực hiện mô phỏng tải.

c) *Security Testing (Kiểm thử bảo mật)*

Đảm bảo API được bảo vệ khỏi các cuộc tấn công như SQL injection, XSS, và các lỗi xác thực.

Kiểm tra các cơ chế bảo vệ như OAuth2, JWT, API key,…

d) Compatibility Testing (Kiểm thử tương thích)

Đảm bảo API hoạt động trên nhiều môi trường, trình duyệt hoặc hệ điều hành khác nhau.

Đặc biệt cần thiết với các hệ thống có nhiều loại client khác nhau (web, mobile, desktop).

e) Negative Testing (Kiểm thử tiêu cực)

Gửi các yêu cầu không hợp lệ (ví dụ: bỏ trống trường bắt buộc, truyền sai kiểu dữ liệu) để xem API có xử lý lỗi hợp lý hay không.

Giúp đảm bảo hệ thống không bị "sập" khi gặp dữ liệu sai.

f) Regression Testing (Kiểm thử hồi quy)

Kiểm tra xem API cũ có bị ảnh hưởng khi có chức năng mới được cập nhật

Giúp đảm bảo hệ thống ổn định trong dài hạn.

2.3.5 Tầm quan trọng của kiểm thử API

Đảm bảo chất lượng lõi hệ thống: Vì API là "xương sống" giúp các thành phần giao tiếp, nên nếu API bị lỗi có thể ảnh hưởng đến toàn bộ hệ thống.

Tiết kiệm chi phí sửa lỗi: Phát hiện lỗi sớm ở tầng API sẽ dễ sửa hơn là sau khi đã tích hợp giao diện và các thành phần khác.

Tăng tốc độ phát triển: API được kiểm thử tốt sẽ giúp các nhóm frontend, mobile,… có thể phát triển song song mà không cần lo về dữ liệu sai.

Hỗ trợ CI/CD: Tích hợp kiểm thử API vào pipeline giúp phát hiện lỗi tự động mỗi lần deploy.

Cải thiện trải nghiệm người dùng gián tiếp: API nhanh và chính xác sẽ giúp UI hoạt động mượt mà, tăng độ tin cậy của ứng dụng.

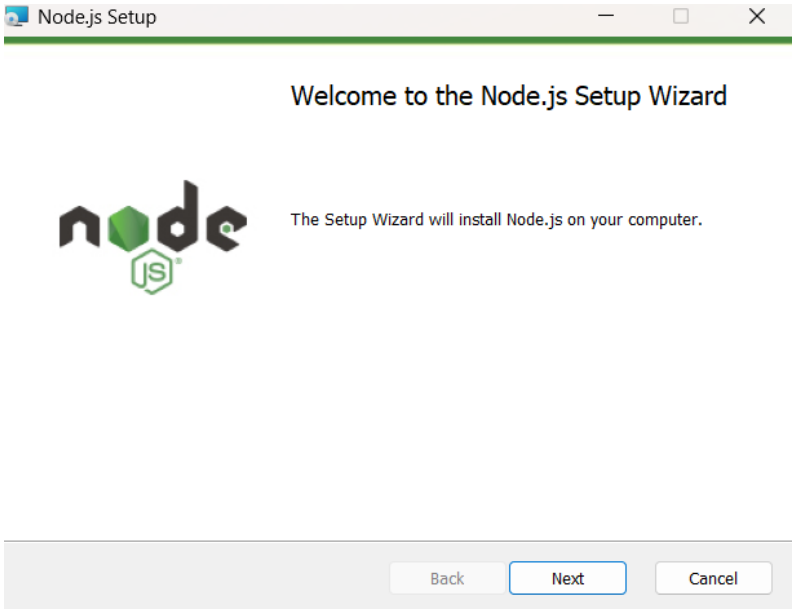
CHƯƠNG 3. THỰC HÀNH VỚI CYPRESS

3.1 Yêu cầu hệ thống

- Node.js phiên bản 12 trở lên.
- Hệ điều hành: Windows, macOS, hoặc Linux.
- Trình duyệt được hỗ trợ: Chrome, Firefox, Edge, hoặc Electron.

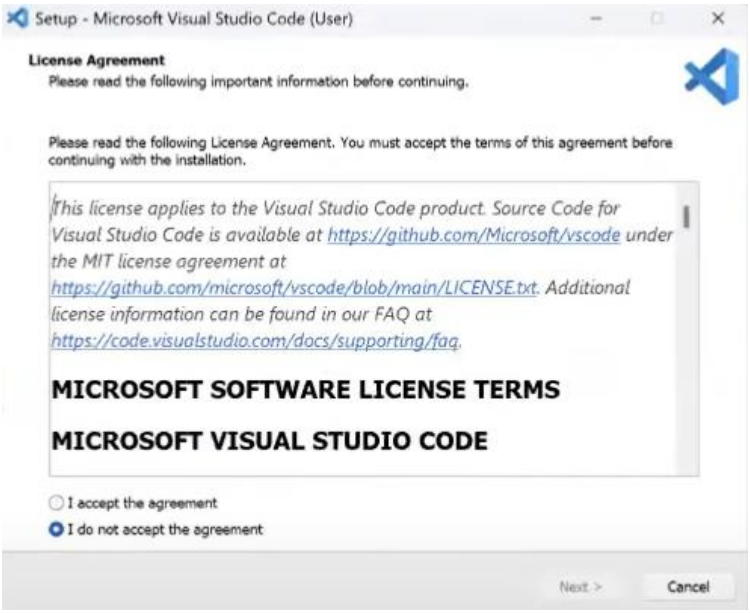
3.2 Cài đặt môi trường cho Cypress

1. Tải Node.js tại trang chủ https://nodejs.org/en
2. Cài đặt Node.js



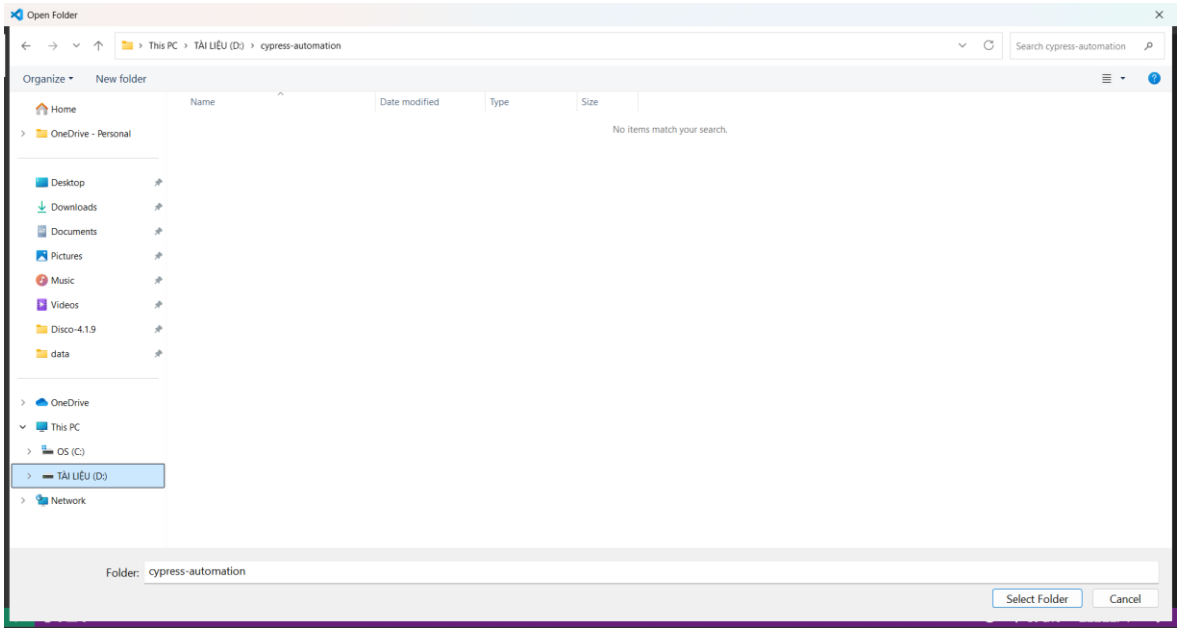
Hình 3-1. Cài đặt Node.js

3. Cài đặt VS Code



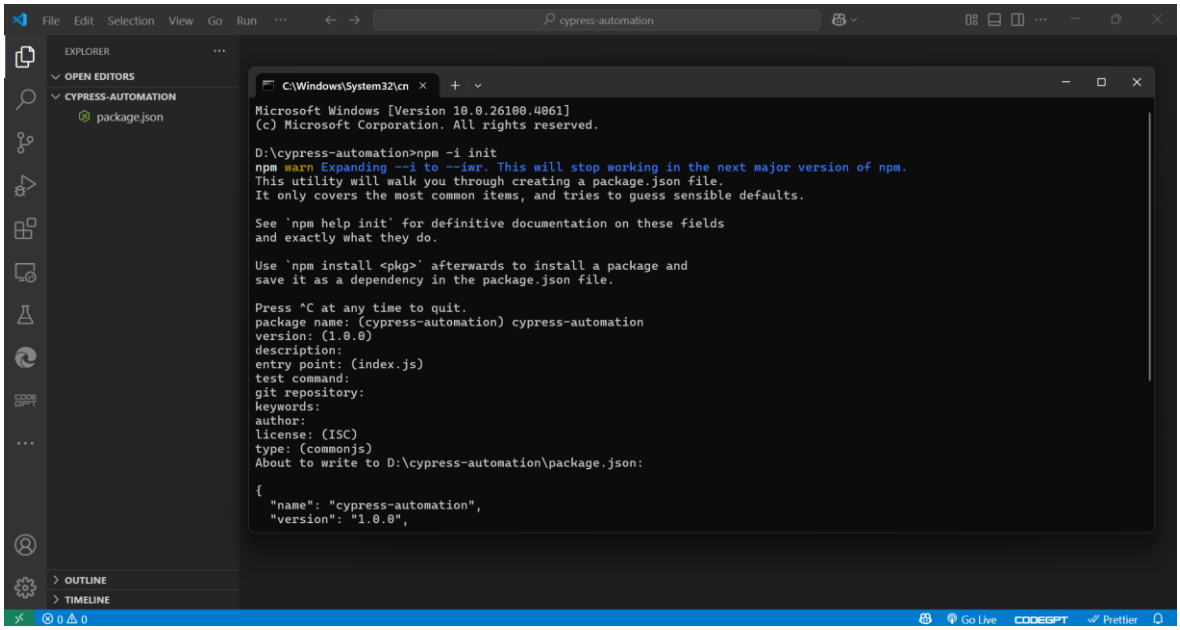
Hình 3-2. Cài đặt VS Code

4. Tạo và setup thư mục chứa Cypress



Hình 3-3. Setup thư mục chứa Cypress

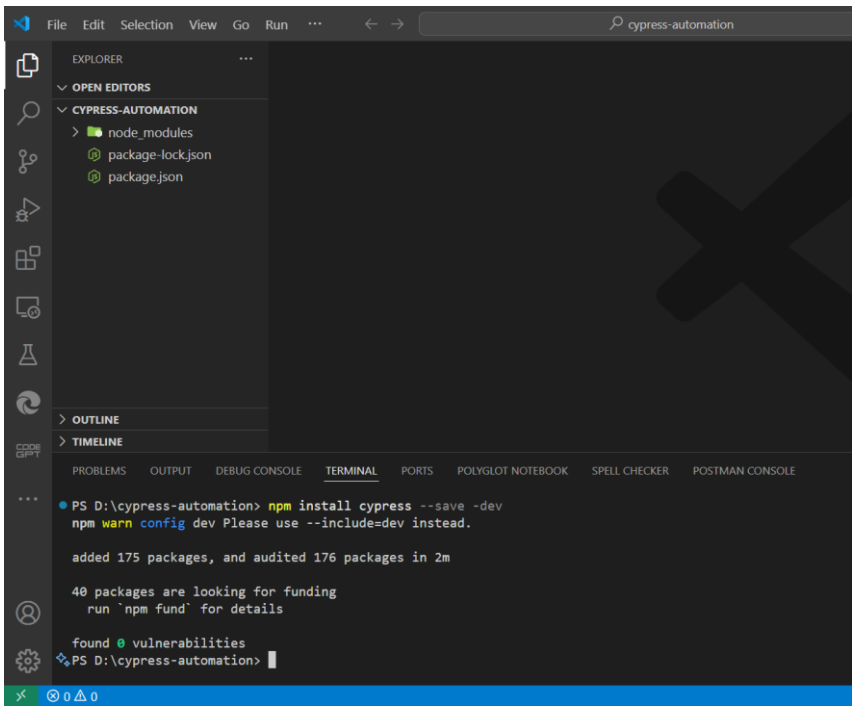
5. Cấu hình file package.json cho thư mục



Hình 3-4. Cấu hình file package.js cho thư mục

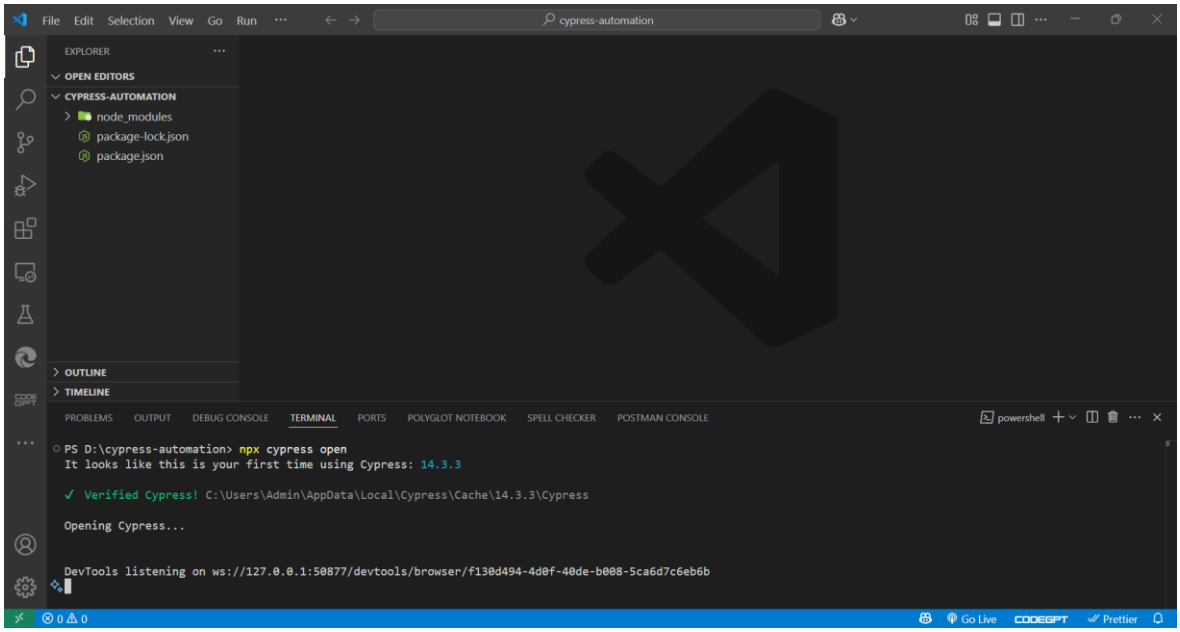
3.3 Cài đặt Cypress

Cài đặt và cấu hình Cypress



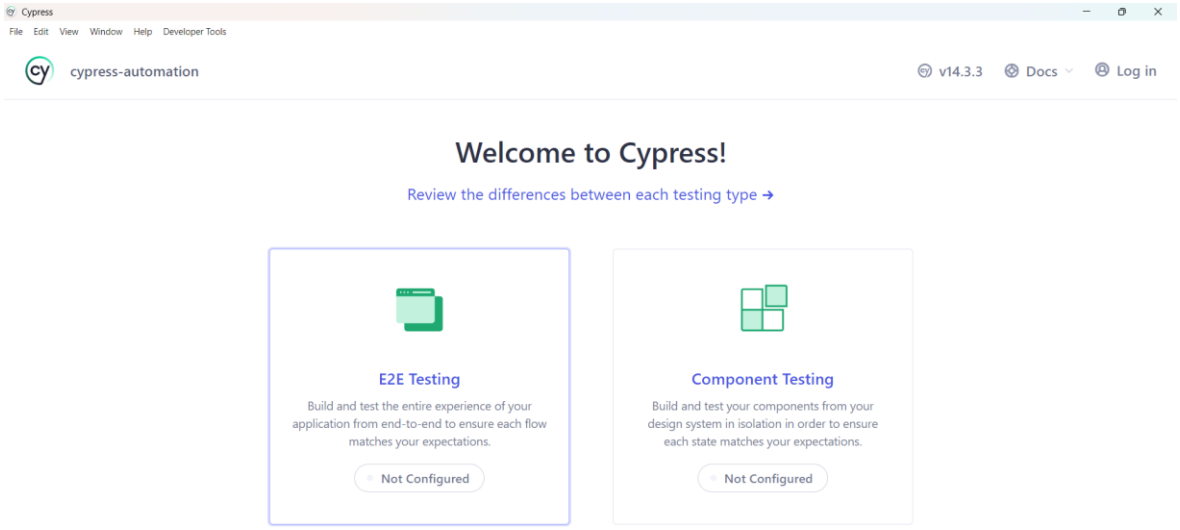
Hình 3-5. Cài đặt và cấu hình Cypress

1. Dùng câu lệnh để khởi động Cypress



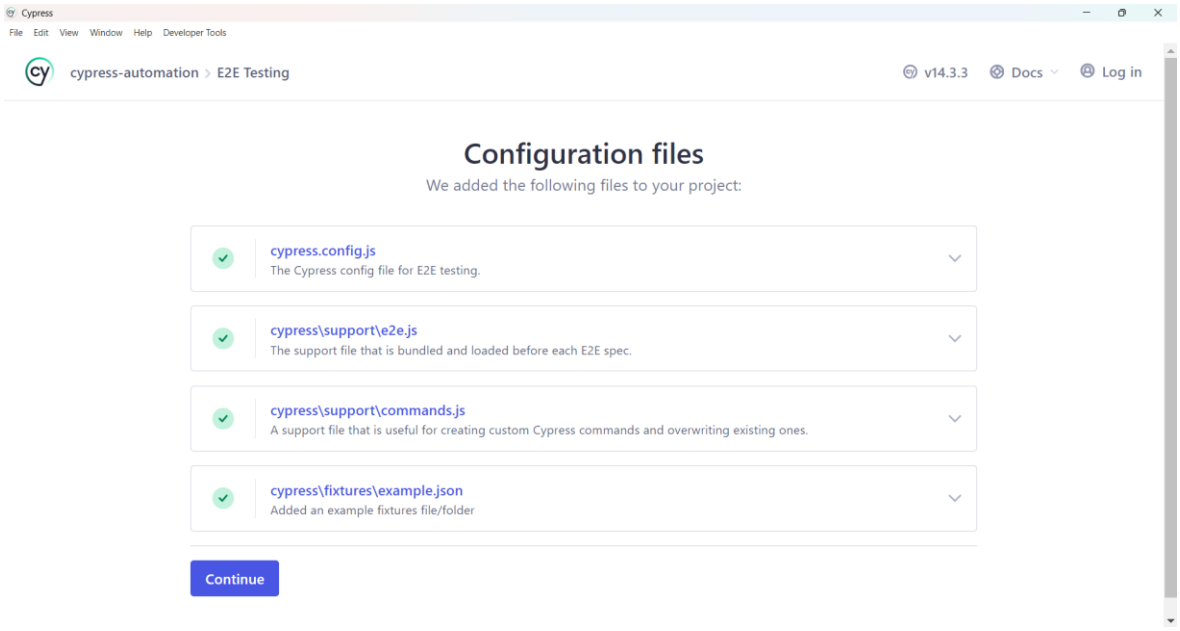
Hình 3-6. Khởi động Cypress

2. Chọn cách thức kiểm thử



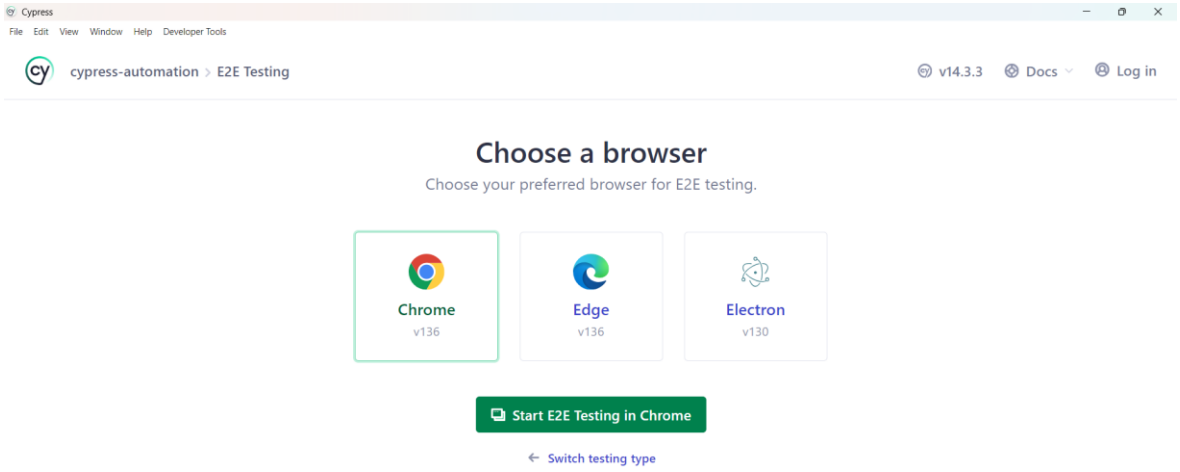
Hình 3-7. Chọn phương thức kiểm thức

3. Các thực mục sẽ được cấu hình sẵn trong Cypress



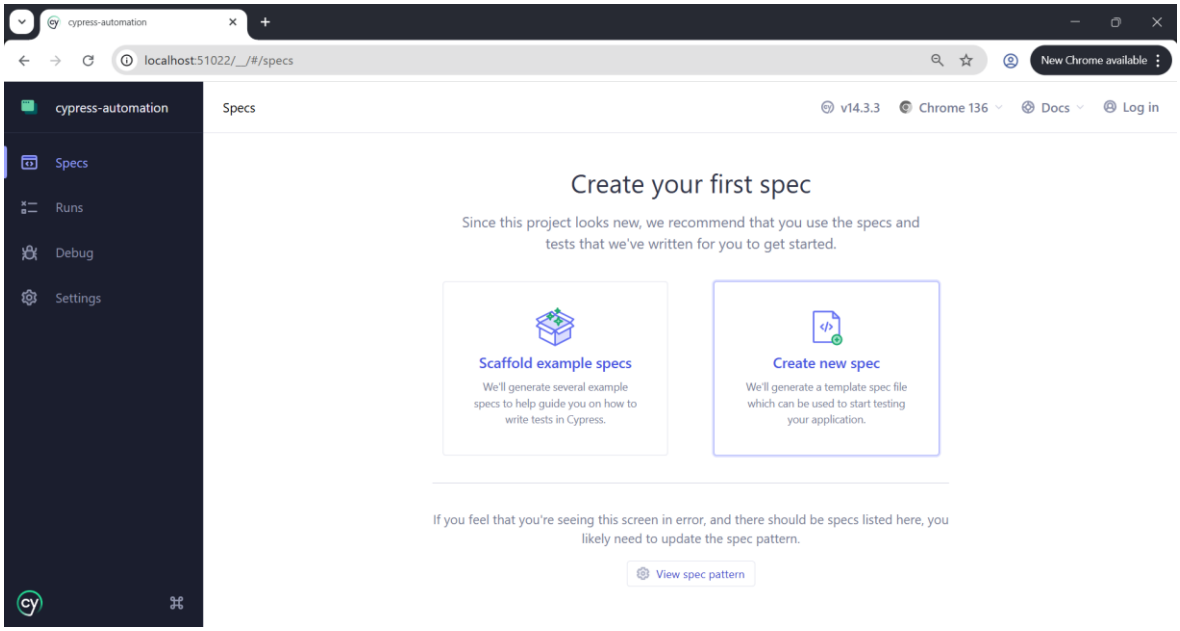
Hình 3-8. Các thư mục được cầu hình sẵn trong Cypress

4. Chọn trình duyệt để chạy Cypress



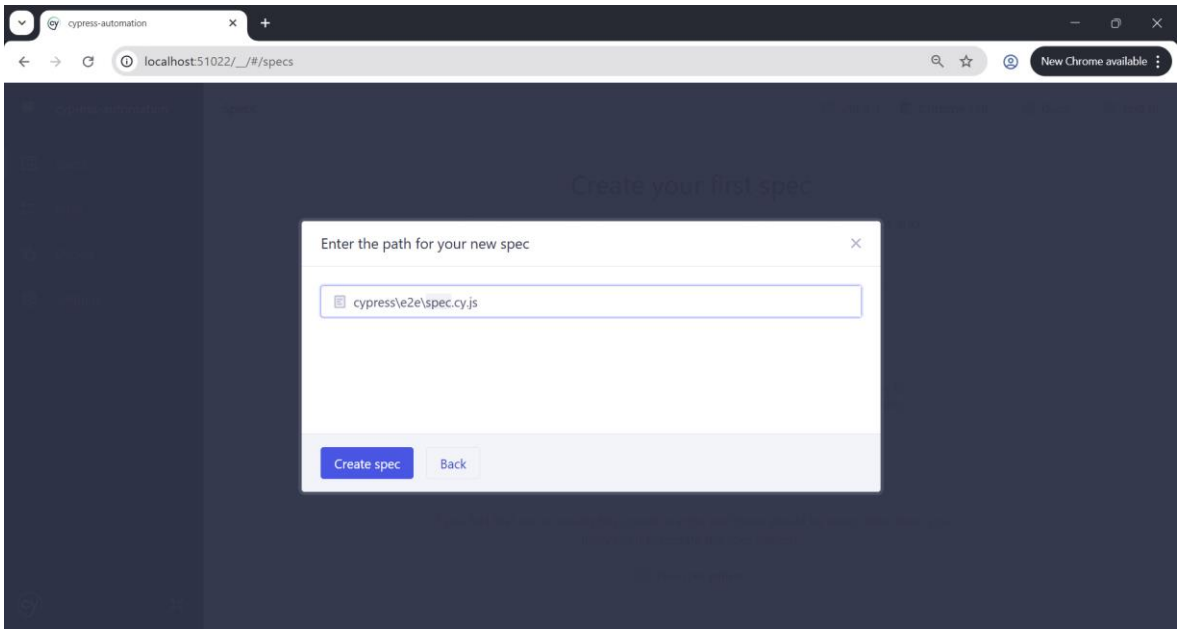
Hình 3-9. Chọn trình duyệt chạy Cypress

5. Chọn tạo thông số kỹ thuật mới



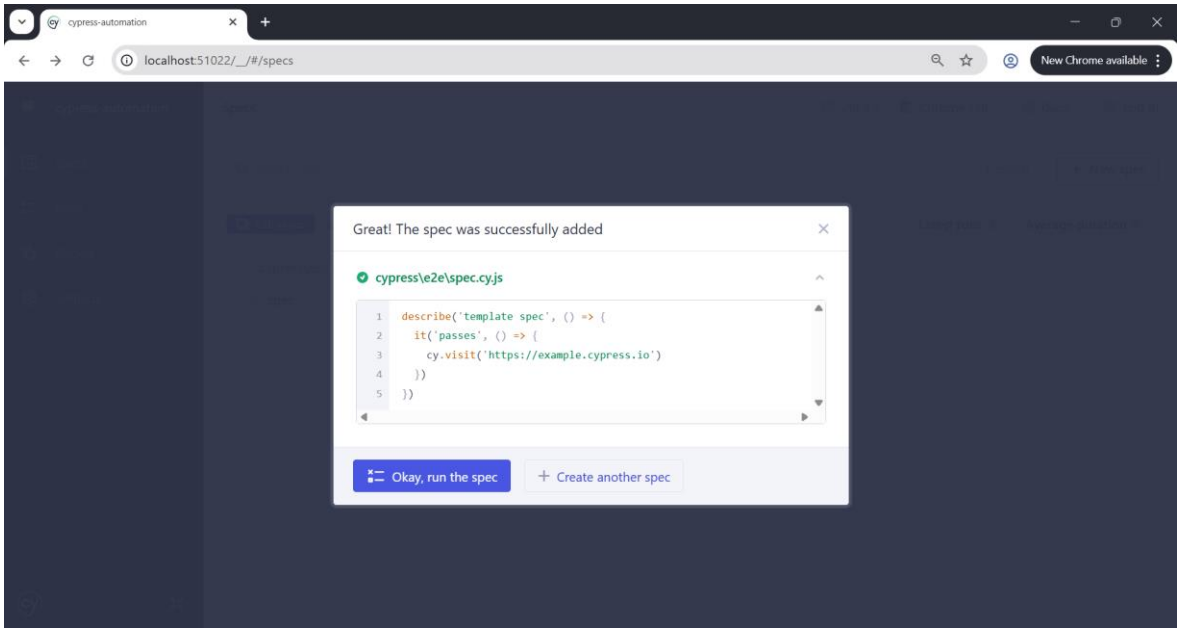
Hình 3-10. Tạo thông số kỹ thuật mới

6. Đặt tên cho file test



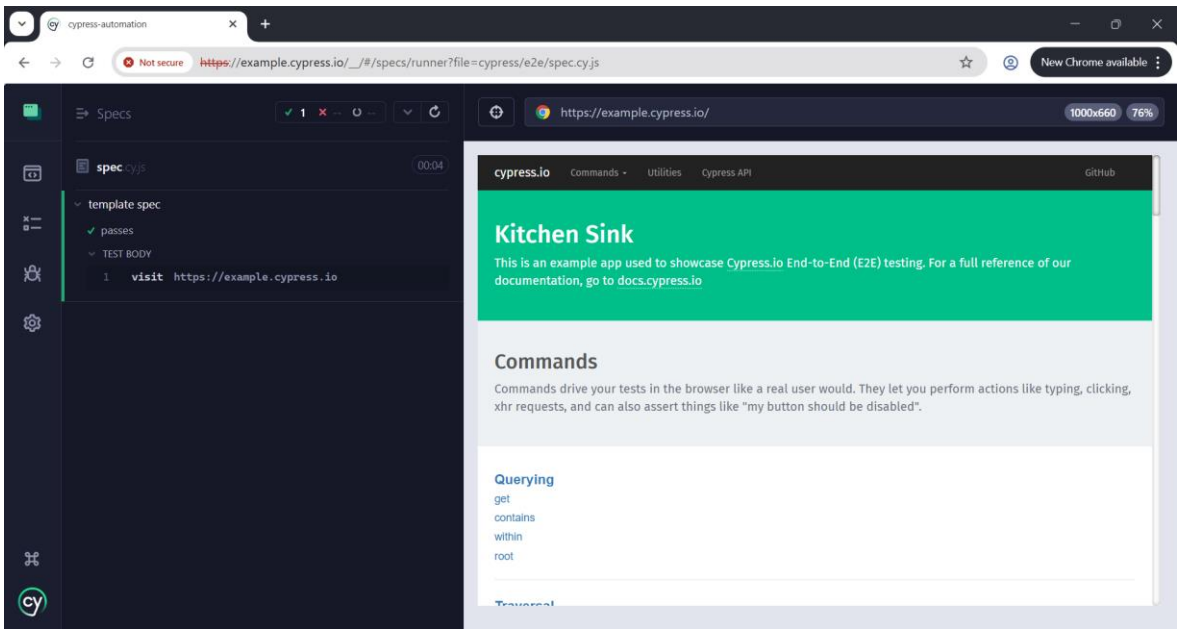
Hình 3-11. Đặt tên file kiểm thử

7. Chạy file kiểm thử mẫu



Hình 3-12. Chạy file kiểm thử mẫu

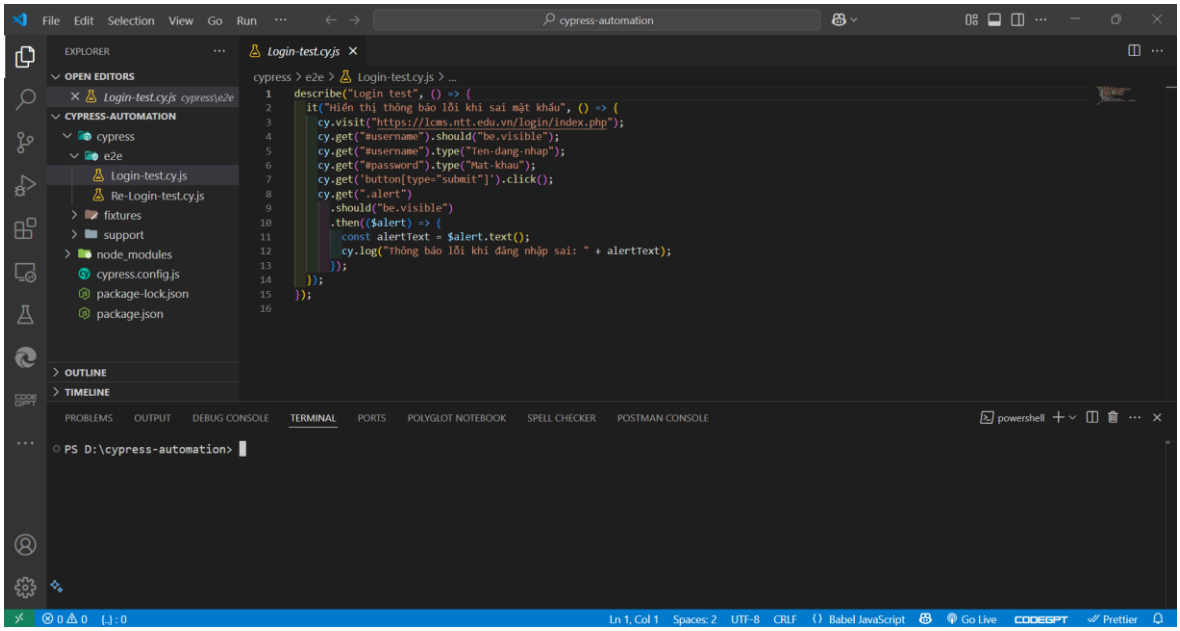
8. Nhận kết quả từ Cypress



Hình 3-13. Kết quả bài mẫu từ Cypress

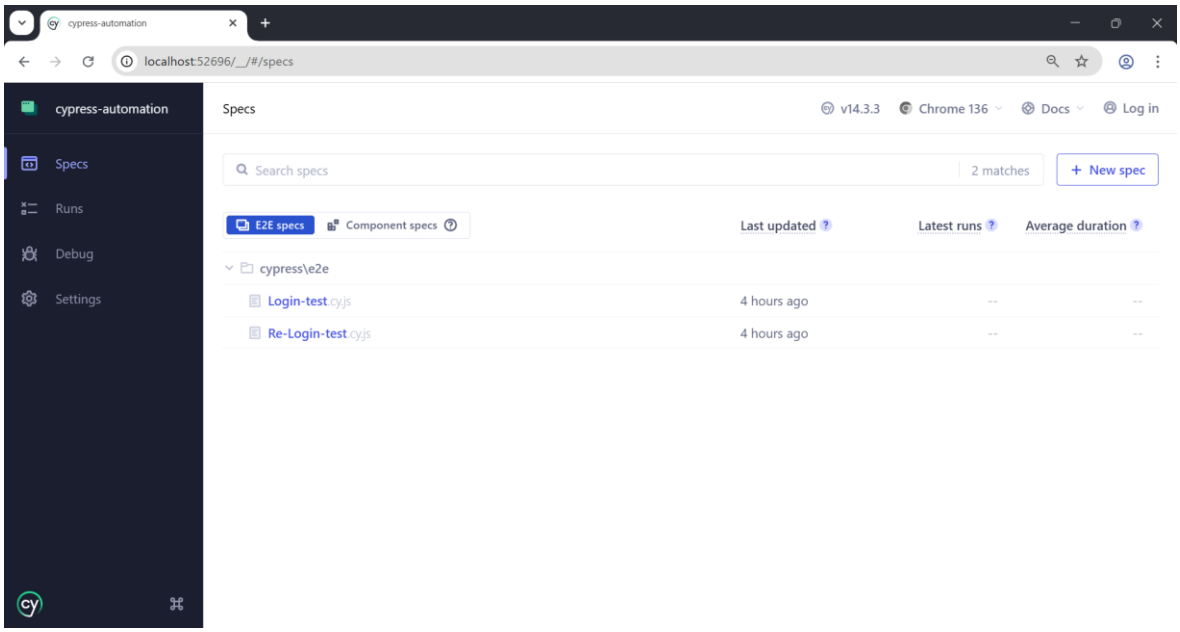
3.4 Thực hành với Cypress

1. Viết test script cho trang đăng nhập LCMS của trường



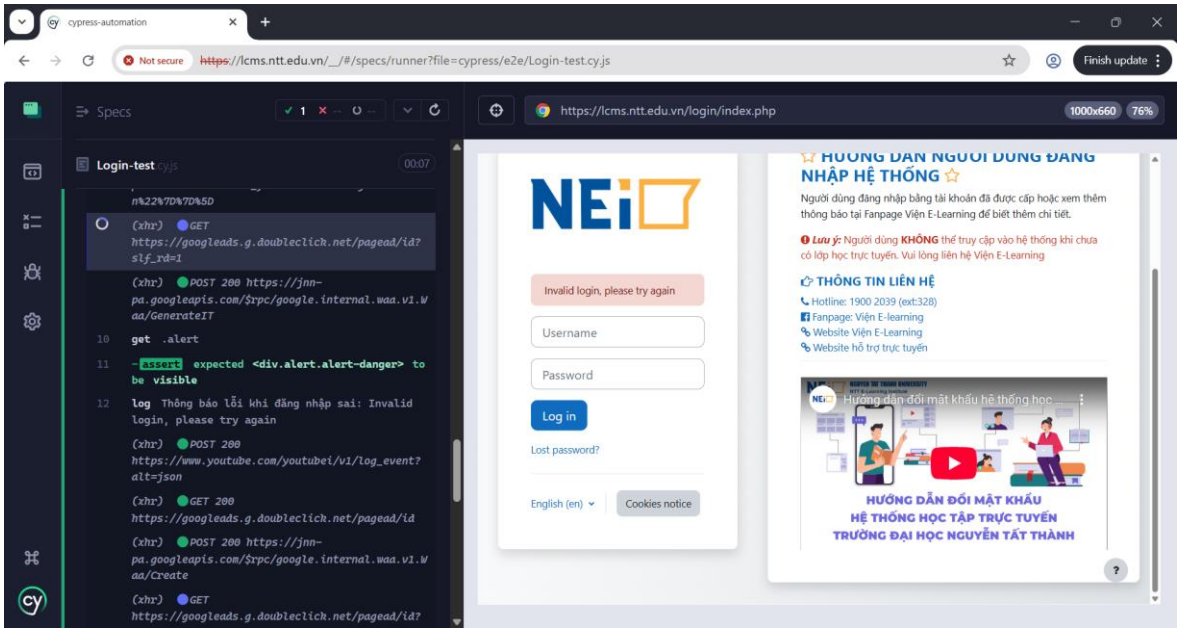
Hình 3-14. Viết test script cho cho trang đăng nhập

2. Chạy Cypress và mở nơi chứa test script



Hình 3-15. Nơi chạy test script

3. Chọn và chạy test script đăng nhập sai thông tin

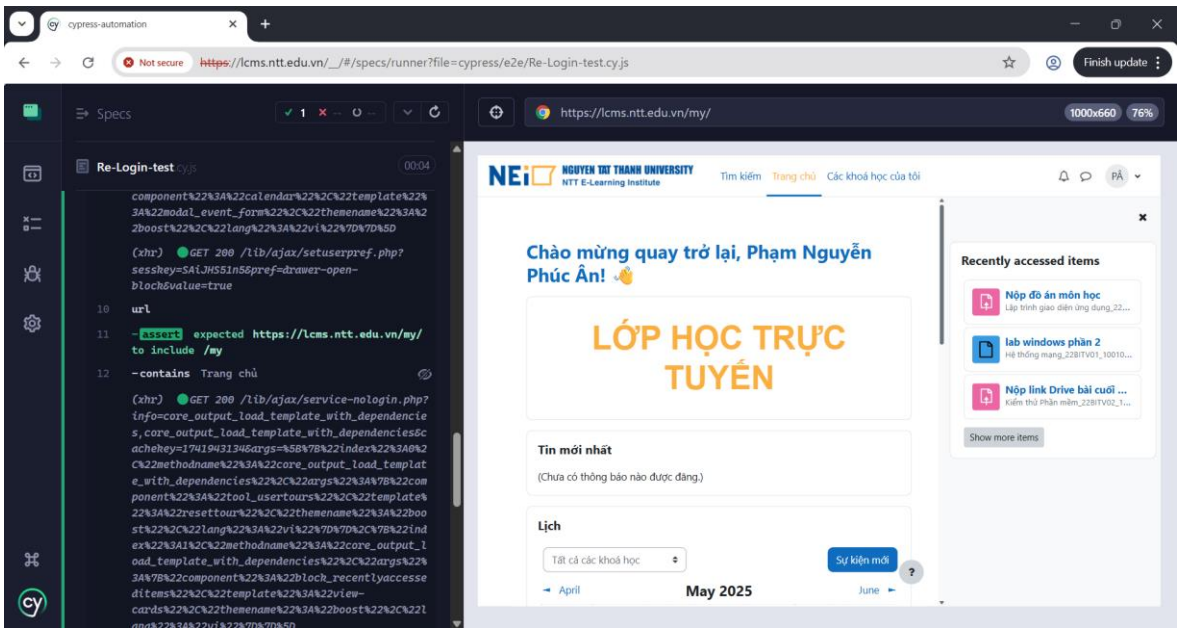


Hình 3-16. Chạy test script đăng nhập sai

Màn hình bên trái hiển thị quá trình kiểm thử và giao diện đã đạt kết quả khi đăng nhập sai thông tin thì sẽ nhận được thông báo (alert) là "Invalid login, please try again".

Màn hình bên phải hiển thị kết quả giao diện của quá trình kiểm thử.

4. Chọn và chạy test script đăng nhập đúng thông tin



Hình 3-17. Chạy test script đăng nhập đúng

Màn hình bên trái hiển thị quá trình kiểm thử khi nhập đúng thông tin đăng nhập thì sẽ được chấp nhận chuyển đến trang https://lcms.ntt.edu.vn/my/ cũng như là trang chủ của website LCMS.

Màn hình bên phải là giao diện sau khi đăng nhập đúng thông tin sẽ được chấp nhận và chuyển đến trang chính của website LCMS.

CHƯƠNG 4. KẾT LUẬN

4.1 Ưu điểm

Mã nguồn mở, không mất phí.

- Cypress là công cụ open-source, có thể tải về và sử dụng miễn phí.
- Thích hợp với cá nhân, nhóm nhỏ hoặc startup không muốn đầu tư chi phí vào công cụ test ban đầu.

Đơn giản, dễ cài đặt và viết kịch bản test.

- Cài đặt dễ dàng chỉ với 1 lệnh npm install cypress.
- Không cần cấu hình phức tạp như Selenium.
- Cú pháp gần với tự nhiên (cy.get().click(), cy.type()) nên người mới học cũng dễ hiểu.

Document đầy đủ, chi tiết, có tích hợp sẵn các examples trong tool giúp cho người mới bắt đầu dễ tiếp cận hơn.

- Trang tài liệu chính thức của Cypress rõ ràng, có ví dụ cụ thể.
- Tool còn tích hợp sẵn test mẫu để học cách viết script nhanh chóng.

Debug dễ dàng do các thao tác đã được ghi lại trong quá trình chạy kịch bản test.

- Mỗi bước test được hiển thị trực quan trong Test Runner.
- Có thể "time-travel" quay lại từng bước và xem trạng thái DOM tại thời điểm đó

⇨ Giúp phân tích lỗi dễ hơn.

Không cần thêm thời gian wait hoặc sleeps vào scripts, Cypress tự động đợi phần tử xuất hiện trước khi tiếp tục.

- Cypress có cơ chế đợi tự động: không cần viết wait(2000) như với Selenium.
- Giảm lỗi test do đợi chưa đủ thời gian hoặc đợi quá lâu.

Dễ dàng kiểm soát và kiểm tra các trường hợp liên quan đến băng thông mạng, có thể khai báo lưu lượng mạng theo bất kì cách nào mong muốn.

- Cypress hỗ trợ cy.intercept() giúp:
  + Chặn/gửi lại các request.
  + Giả lập tình huống server bị chậm, lỗi, mất kết nối.
- Có thể mô phỏng nhiều tình huống mạng khác nhau mà không cần thay đổi backend.

Hỗ trợ chụp ảnh lỗi và quay video toàn bộ quá trình chạy kịch bản test.

- Nếu test bị fail, Cypress sẽ tự chụp ảnh màn hình điểm lỗi.
- Có thể ghi lại video toàn bộ quá trình test khi chạy ở chế độ headless.
- Hữu ích cho debug, báo cáo lỗi, hoặc review QA/Dev.

Cung cấp trang Dashboard service có thể xem tổng quan, báo cáo kết qủa sau mỗi lần chạy.

- Dashboard của Cypress Cloud (tùy chọn sử dụng) cung cấp:
  + Tổng hợp kết quả chạy test.
  + Xem lại lỗi, ảnh, video.
  + Phân tích xu hướng test qua thời gian.
  ⇨ Giúp quản lý test hiệu quả hơn trong các dự án CI/CD.

4.2 Nhược điểm

Cộng đồng sử dụng chưa nhiều.

- So với Selenium hoặc Playwright, Cypress có cộng đồng nhỏ hơn.
- Ít tài nguyên tham khảo hoặc hỗ trợ từ người dùng khác hơn.

Không hỗ trợ test trên native mobile app.

- Cypress chỉ hoạt động trên trình duyệt (web).
- Không thể kiểm thử các ứng dụng di động như Android/iOS native (ứng dụng cài trên điện thoại).

⇨ Nếu cần test app di động, phải dùng Appium hoặc Detox.

Không thể hỗ trợ tương tác với nhiều tab do chạy bên trong trình duyệt.

- Cypress chạy bên trong 1 tab duy nhất của trình duyệt.
- Không thể chuyển qua lại giữa nhiều tab hoặc cửa sổ như một số tình huống yêu cầu.

⇨ Ví dụ: xác minh email mở ở tab khác → không làm được bằng Cypress.

Nếu là bản free thì trang Dashboard service có 1 số hạn chế nhất định, nếu muốn sử dụng full dịch vụ thì phải mất phí.

- Bản miễn phí của Cypress Cloud chỉ giới hạn số lượng test run/tháng, không có một số tính năng nâng cao (team insights, parallelization nâng cao,...)
- Muốn dùng đầy đủ tính năng → cần trả phí.

Chỉ hỗ trợ Javascript.

- Cypress chỉ cho phép viết test bằng JavaScript hoặc TypeScript.
- Không hỗ trợ các ngôn ngữ như Java, Python, C#,...

⇨ Không phù hợp nếu dùng ngôn ngữ khác.

Hỗ trợ 1 số ít framwork: Mocha JS.

- Cypress sử dụng Mocha và Chai làm framework test chính.
- Không hỗ trợ các framework kiểm thử khác như Jasmine, Jest (một cách chính thức).

Trình duyệt được hỗ trợ còn hạn chế:

- Cypress hỗ trợ: Chrome, Chromium, Edge, Firefox, Electron.
- Không hỗ trợ: Safari, các trình duyệt di động thật sự (dù có thể mô phỏng trên responsive).

⇨ Với các dự án yêu cầu cross-browser test toàn diện, Cypress không đủ bao phủ.

4.3 Hướng phát triển

Mặc dù Cypress đã mang lại nhiều lợi ích trong việc kiểm thử giao diện người dùng với độ chính xác cao, thời gian thực thi nhanh, và khả năng tự động hóa mạnh mẽ, nhưng để nâng cao hơn nữa hiệu quả sử dụng công cụ này trong các dự án thực tế, nhóm chúng tôi đề xuất một số hướng phát triển sau:

4.3.1 Tích hợp Cypress vào quy trình CI/CD

Việc tích hợp Cypress vào các pipeline CI/CD (Continuous Integration / Continuous Deployment) là xu hướng tất yếu để đảm bảo chất lượng phần mềm trong suốt quá trình phát triển và triển khai. Trong tương lai, nhóm có thể kết hợp Cypress với các nền tảng như Jenkins, GitHub Actions, GitLab CI, CircleCI,... nhằm:

- Tự động chạy test sau mỗi lần commit/push.
- Phát hiện lỗi sớm và giảm rủi ro khi triển khai lên môi trường thật.
- Tăng tính nhất quán và độ tin cậy của sản phẩm.

4.3.2 Sử dụng Cypress Cloud (Dashboard Service) hiệu quả hơn

Hiện tại, Cypress Cloud cung cấp nhiều tính năng nâng cao như phân tích lịch sử test, test song song (parallel testing), tự động ghi lại video và ảnh lỗi,... Tuy nhiên, bản miễn phí vẫn còn hạn chế về số lượt chạy và các báo cáo phân tích chuyên sâu. Trong tương lai, nếu nhu cầu dự án tăng lên, nhóm có thể:

- Xem xét nâng cấp lên phiên bản trả phí để sử dụng đầy đủ các tính năng.
- Tận dụng các phân tích xu hướng lỗi để tối ưu hóa test case và cải thiện quy trình phát triển.

4.3.3 Mở rộng phạm vi test sang các loại kiểm thử khác

Hiện tại, nhóm chủ yếu sử dụng Cypress cho kiểm thử giao diện người dùng (UI Testing). Tuy nhiên, Cypress cũng có thể mở rộng để kiểm thử API, kiểm thử hiệu năng (ở mức cơ bản), hoặc mô phỏng các tình huống mạng. Nhóm có thể:

- Viết thêm các kịch bản kiểm thử API bằng cy.request() hoặc cy.intercept().
- Mô phỏng mạng yếu, server chậm, lỗi phản hồi để kiểm tra khả năng xử lý lỗi của hệ thống.
- Kết hợp với công cụ đo hiệu suất khác nếu cần kiểm thử performance chuyên sâu.

4.3.4 Xây dựng thư viện test case tái sử dụng

Một hướng phát triển quan trọng là xây dựng các hàm hoặc mô-đun kiểm thử có thể tái sử dụng, giúp giảm thời gian viết script mới, đồng thời đảm bảo chuẩn hóa trong cách kiểm thử. Cụ thể:

- Tạo các custom commands (Cypress.Commands.add) cho các hành vi lặp lại.
- Tách riêng các hàm xử lý dữ liệu, xác thực hoặc các bước chuẩn bị dữ liệu test.
- Quản lý tập trung dữ liệu test với file JSON, fixture,...

4.3.5 Kết hợp Cypress với công nghệ mới (AI, ML, Reporting tools)

Trong thời gian tới, có thể nghiên cứu thêm việc kết hợp Cypress với các công nghệ hiện đại nhằm nâng cao chất lượng kiểm thử:

- Ứng dụng AI để đề xuất test case, phân tích log lỗi hoặc tự động sửa lỗi.
- Tích hợp với các công cụ báo cáo như Allure, Mochawesome để tạo ra báo cáo chi tiết, thân thiện với người dùng không chuyên.
- Kết hợp với các công cụ test coverage để đánh giá độ bao phủ của các test case hiện có.

4.3.6 Mở rộng kỹ năng và công nghệ kiểm thử

Để đảm bảo tính linh hoạt trong việc kiểm thử trên nhiều nền tảng, nhóm có thể song song học hỏi và sử dụng thêm các công cụ khác như

- Appium để kiểm thử ứng dụng di động.
- Playwright để kiểm thử trên nhiều trình duyệt (bao gồm Safari).
- Postman/Newman để kiểm thử API một cách độc lập và kết hợp với Cypress.

CHƯƠNG 5. Tổng kết

Thông qua việc sử dụng Cypress, nhóm đã có cơ hội trải nghiệm một công cụ kiểm thử tự động hiện đại, nhanh chóng, và dễ sử dụng. Trong suốt quá trình thực hiện, nhóm đã tìm hiểu và áp dụng các tính năng của Cypress để kiểm thử giao diện, mô phỏng hành vi người dùng và kiểm soát các luồng mạng. Điều này không chỉ giúp nhóm nắm vững lý thuyết về kiểm thử phần mềm, mà còn củng cố kỹ năng thực hành trong môi trường thực tế.

Tuy nhiên, để tận dụng tối đa sức mạnh của Cypress trong quy mô dự án lớn, nhóm cần định hướng phát triển công cụ này một cách bài bản hơn: từ tích hợp với quy trình CI/CD đến mở rộng các loại kiểm thử, xây dựng hệ thống tái sử dụng, và học hỏi thêm các công nghệ bổ trợ. Đây sẽ là nền tảng quan trọng để đảm bảo chất lượng phần mềm cho những dự án thực tiễn trong tương lai.

TÀI LIỆU THAM KHẢO

Cypress.io. (n.d.). Getting started with Cypress. Retrieved May 18, 2025. Truy cập ngày 13/03/2025 từ: https://docs.cypress.io/guides/getting-started/installing-cypress

Grinberg, M. (2020). Cypress Essentials: Your guide to end-to-end testing. Leanpub.

Palani, S. (2021). Hands-On Test Automation with Cypress. Packt Publishing.

Fowler, M. (2018). Test automation best practices. Retrieved. Truy cập ngày 05/04/2025 từ: https://martinfowler.com/articles/practical-test-pyramid.html

Mozilla Developer Network. (n.d.). Understanding asynchronous JavaScript. Retrieved May 18, 2025. Truy cập ngày 05/04/2025 từ: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous

Nguyễn Văn A. (2022). Giáo trình kiểm thử phần mềm. Nhà xuất bản Đại học Bách Khoa.

Trần, T. H. (2023). Ứng dụng Cypress trong kiểm thử frontend. Luận văn cử nhân, Trường Đại học Công nghệ Thông tin.