**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

****

**BÀI KIỂM TRA THƯỜNG XUYÊN 2**

**HỌC PHẦN: KIỂM THỬ PHẦN MỀM**

Giáo viên hướng dẫn: Thầy Hoàng Quang Huy

Người thực hiện: Nguyễn Văn Huấn

Mã sinh viên: 2020600491

Lớp: **20222IT6013003**

**Câu 1: Trình bày và cho ví dụ về kiểm thử hộp trắng?**

* 1. **Kiểm thử hộp trắng là gì**

**Kiểm thử Hộp Trắng** là một phương pháp kiểm thử phần mềm trong đó tester biết về cấu trúc nội bộ / thiết kế. Người kiểm tra chọn đầu vào để thực hiện các đường dẫn thông qua mã và xác định đầu ra thích hợp. Kiến thức lập trình và kiến thức thực hiện là rất cần thiết trong kiểm thử hộp trắng.

Kiểm thử hộp trắng bao gồm phân tích dòng dữ liệu, điều khiển dòng, dòng thông tin, mã thực hành, ngoại lệ và những lỗi trình bày trong hệ thống để kiểm tra những hành động của phần mềm không được định hướng trước.

* 1. **Kỹ thuật kiểm thử hộp trắng**

Một số kỹ thuật Coverage Testing mà người kiểm thử có thể sử dụng:

* **Bao phủ câu lệnh (Statement Coverage)**: Kỹ thuật này yêu cầu mọi câu lệnh phải được thực thi ít nhất 1 lần trong quá trình kiểm tra kỹ thuật phần mềm. Statement Coverage cung cấp các chi tiết của cả hai khối mã được thực thi và thất bại trong tổng số các khối mã.
* **Phạm vi chi nhánh (Branch Coverage)**: Kỹ thuật này kiểm tra mọi đường dẫn if-else và các vòng điều kiện khác của một ứng dụng phần mềm. Branch coverage có thể được tính bằng cách tìm số đường dẫn tối thiểu để đảm bảo rằng tất cả các cạnh đã được che phủ.
* **Bao phủ nhánh (Path Coverage)**: Bao phủ nhánh là một phương pháp kiểm tra cấu trúc liên quan đến việc sử dụng mã nguồn của chương trình để tìm mọi đường dẫn thực thi có thể. Path Coverage đảm bảo phạm vi của tất cả các đường dẫn từ đầu đến cuối.
  1. **Các loại kiểm thử hộp trắng**

White box testing có 2 loại kiểm thử chính**: Kiểm thử đơn vị** (Unit Testing) và **Kiểm tra rò rỉ bộ nhớ** (Testing for Memory Leaks)

a**. Kiểm thử đơn vị** (Unit Testing)

Kiểm thử đơn vị là quá trình test từng module nhỏ trong hệ thống để xác nhận rằng mỗi thành phần của phần mềm thực hiện đúng với thiết kế. Unit Testing được coi là loại kiểm thử đầu tiên được thực hiện trên một ứng dụng. Các lỗi được tìm thấy trong giai đoạn này sẽ dễ dàng sửa chữa cũng như không làm phát sinh chi phí cho dự án. Người thực hiện kiểm thử đơn vị phần lớn là developers hoặc tester có kinh nghiệm về lập trình. Developers tiến hành code, phát triển các chức năng đơn lẻ và tiến hành kiểm tra lại nhằm đảm bảo lập trình hoạt động được trước khi sang giai đoạn khác.

b**. Kiểm tra rò rỉ bộ nhớ** (Testing for Memory Leaks)

Rò rỉ bộ nhớ là nguyên nhân hàng đầu khiến các ứng dụng chạy chậm hơn. Developers sẽ phải cần đến một chuyên gia QA (Quality Assurance - Đảm bảo chất lượng) có kinh nghiệm trong việc phát hiện rò rỉ bộ nhớ tư vấn về trường hợp kiểm thử. Đây là điều cần thiết để tránh ứng dụng phần mềm chạy chậm gây ảnh hưởng đến chất lượng sản phẩm cũng như trải nghiệm người dùng.

* 1. **Ưu và nhược điểm kiểm thử hộp trắng**

Ngoài những ưu điểm nổi bật, White box testing vẫn có những nhược điểm cần được cải thiện trong tương lai.

a. **Ưu điểm của kiểm thử hộp trắng**

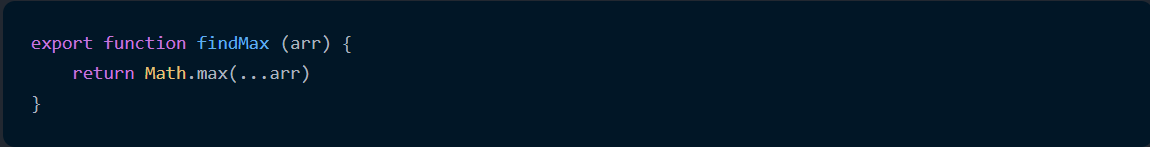
* Tối ưu hóa mã bằng cách tìm lỗi ẩn.
* Tự động hóa dễ dàng các trường hợp kiểm thử
* Kiểm tra kỹ lưỡng hơn vì tất cả các đường dẫn mã thường được bao phủ.
* Kiểm thử có thể được thực hiện sớm trong quy trình phát triển phần mềm ngay cả khi GUI (Graphical User Interface - Giao diện đồ họa người dùng) không khả dụng.

b. **Nhược điểm của kiểm thử hộp trắng**

* Đòi hỏi nguồn lực chuyên nghiệp có kiến thức và tay nghề cao về lập trình.
* Tốn khá nhiều thời gian để kiểm tra được chi tiết cấu trúc, thuật toán bên trong sản phẩm. Các ứng dụng có cấu trúc hệ thống càng lớn càng cần rất nhiều thời gian để kiểm tra đầy đủ.
* Kiểm thử diễn ra khi sản phẩm chưa được hoàn thiện nên các công cụ phục vụ cho mọi loại triển khai/nền tảng thường không sẵn có.
* Kiểm tra hộp trắng được đánh giá khá phức tạp và tốn kém. Developers không kiểm tra hộp trắng chi tiết có thể dẫn đến lỗi sản xuất.
  1. **Ví dụ về kiểm thử hộp trắng**

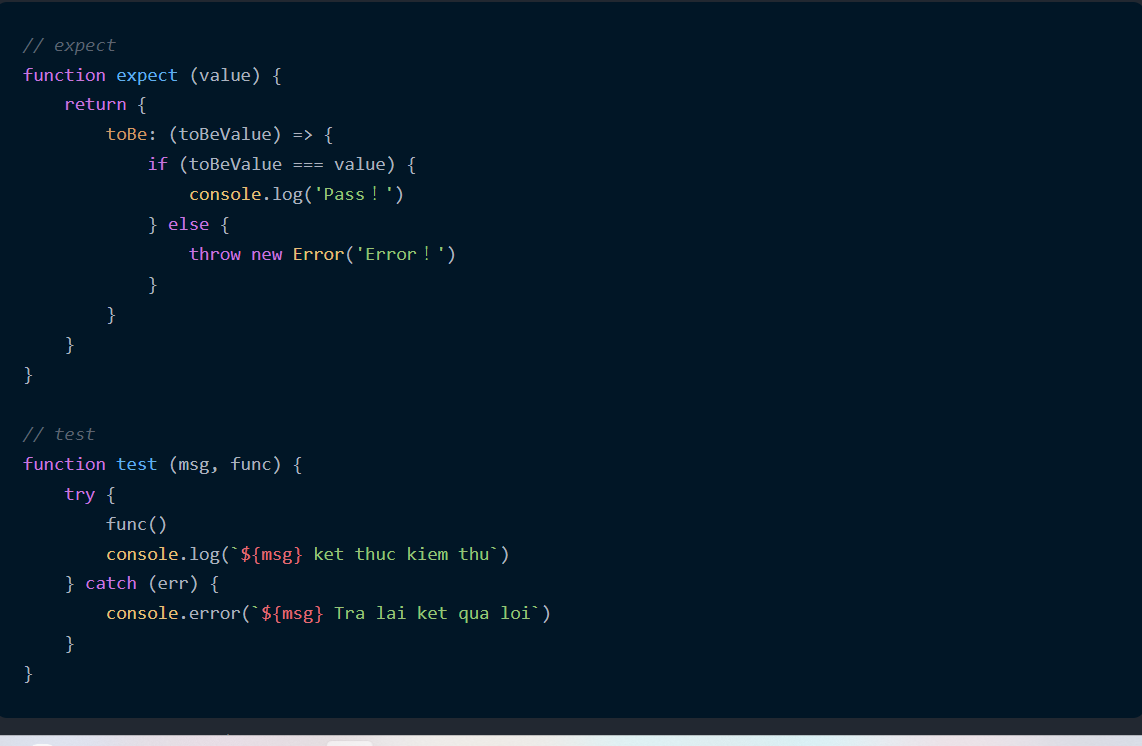
**Trường hợp kiểm thử: Unit test với Jest**

* Jest là một trình chạy thử nghiệm JavaScript, nghĩa là một thư viện javascript để tạo, và để kiểm tra các thục tục hay hàm trong js. Jest là một package có sẵn trong NPM, bạn có thể cài đặt nó trong bất kỳ dự án JavaScript nào. Jest là một trong những trình chạy thử nghiệm phổ biến nhất hiện nay và đang là lựa chọn mặc định cho các dự án React.

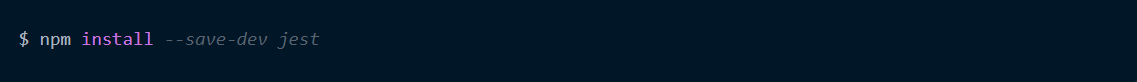


Trên kia là một hàm check xem giá trị lớn nhất trong array là bao nhiêu?

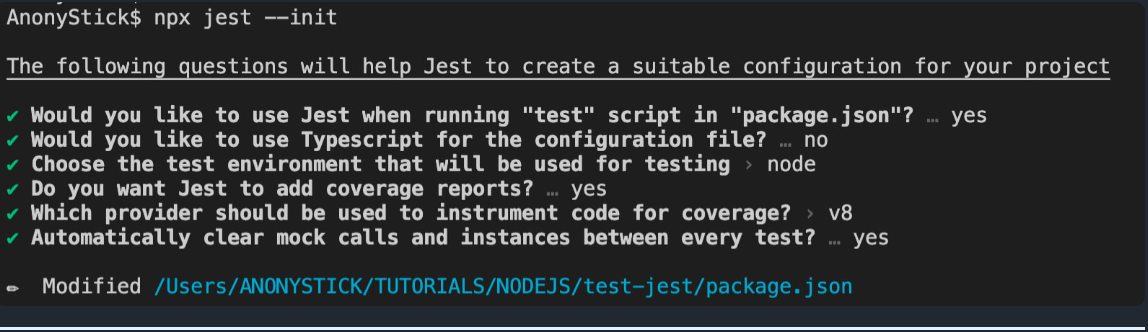
Viết hai hàm, một là test và hai là expect.



Giờ ta thử nghiệm với jest xem thế nào? Đầu tiên install jest:

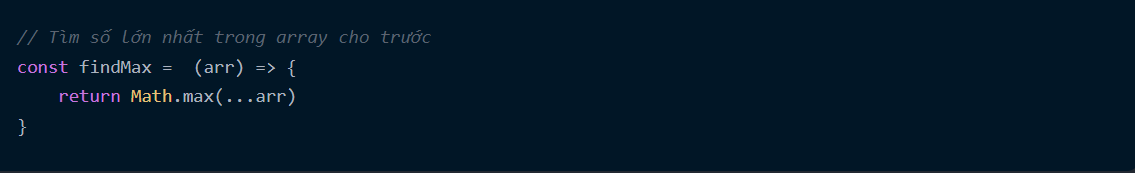


Bước tiếp theo là chạy lệnh npx jest --init để tạo tệp cấu hình jest là jest.config.js

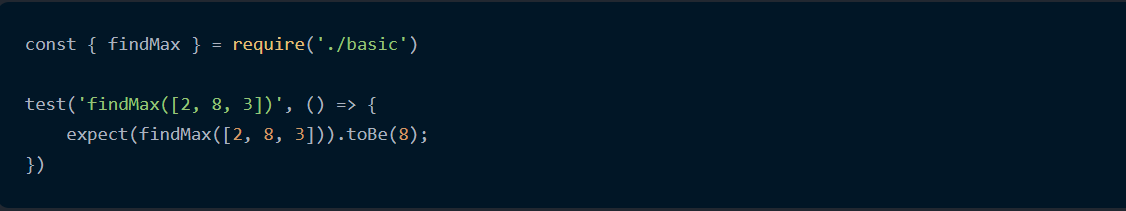


Giờ thực hành tạo 2 file basic.js và basic.test.js trong folder src:

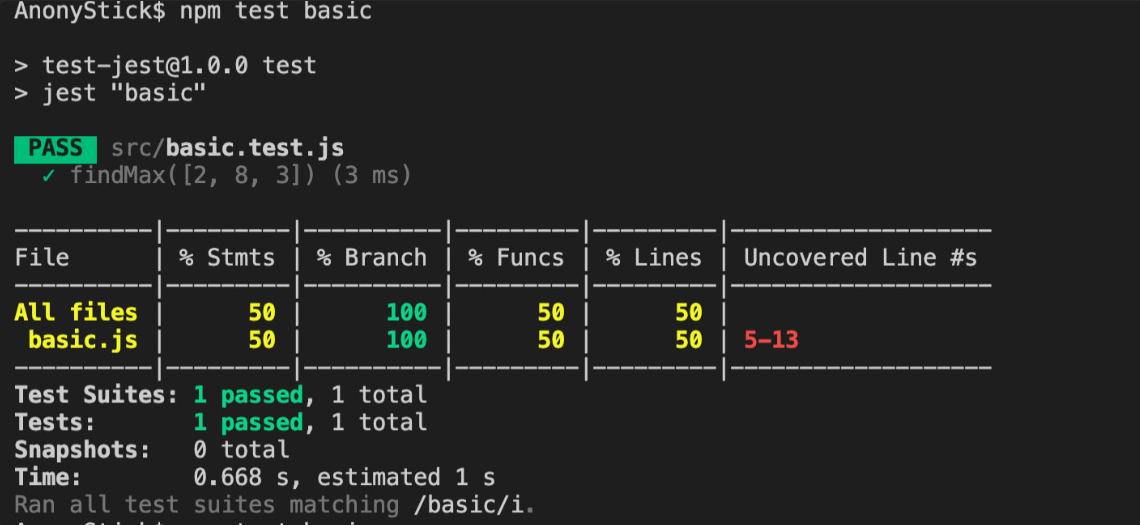
**basic.js:** Để viết code logic, như sau:



**basic.test.js**: Để viết unit test như sau:



Kết quả pass, xem hình ảnh dưới đây:



**Câu 2: Trình bày kiểm thử đường dẫn căn bản, ví dụ minh họa**

**2.1 Kiểm thử đường dẫn là gì?**

**Kiểm tra đường dẫn** là một phương pháp kiểm tra cấu trúc liên quan đến việc sử dụng mã nguồn của chương trình để tìm mọi đường dẫn thực thi có thể. Nó giúp xác định tất cả các lỗi nằm trong một đoạn mã. Phương pháp này được thiết kế để thực hiện tất cả hoặc đường dẫn đã chọn thông qua một chương trình máy tính.

Bất kỳ chương trình phần mềm nào cũng bao gồm, nhiều điểm vào và ra. Kiểm tra từng điểm này là một thách thức cũng như tốn thời gian. Để giảm các bài kiểm tra dư thừa và để đạt được phạm vi kiểm tra tối đa, kiểm tra đường dẫn cơ sở được sử dụng.

**2.2 Các bước kiểm thử đường dẫn cơ bản**

Các bước cơ bản liên quan đến thử nghiệm đường dẫn cơ sở bao gồm:

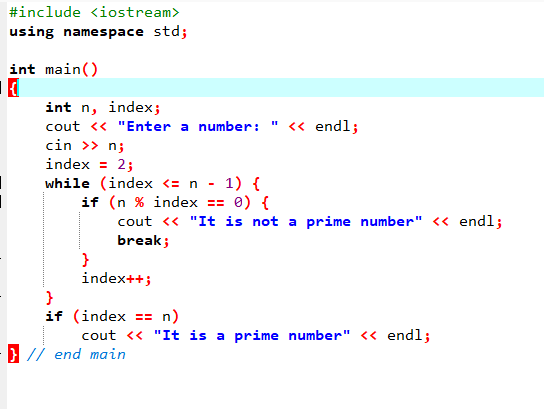
* Vẽ biểu đồ điều khiển (để xác định các đường dẫn chương trình khác nhau)
* Tính toán độ phức tạp theo chu kỳ (số liệu để xác định số lượng đường dẫn độc lập)
* Tìm một tập cơ sở của đường dẫn
* Tạo các trường hợp thử nghiệm để thực hiện từng đường dẫn

**2.3 Ưu điểm của kiểm tra đường dẫn cơ bản**

* Giúp giảm các bài kiểm tra dư thừa
* Tập trung sự chú ý vào logic chương trình
* Giúp tạo điều kiện thuận lợi cho thiết kế trường hợp phân tích so với tùy ý
* Các trường hợp thử nghiệm mà tập cơ sở bài tập sẽ thực hiện mọi câu lệnh trong chương trình ít nhất một lần

**2.4 Ví dụ về kiểm thử đường dẫn**

Xét chương trình kiểm tra một số có phải là số nguyên tố hay không

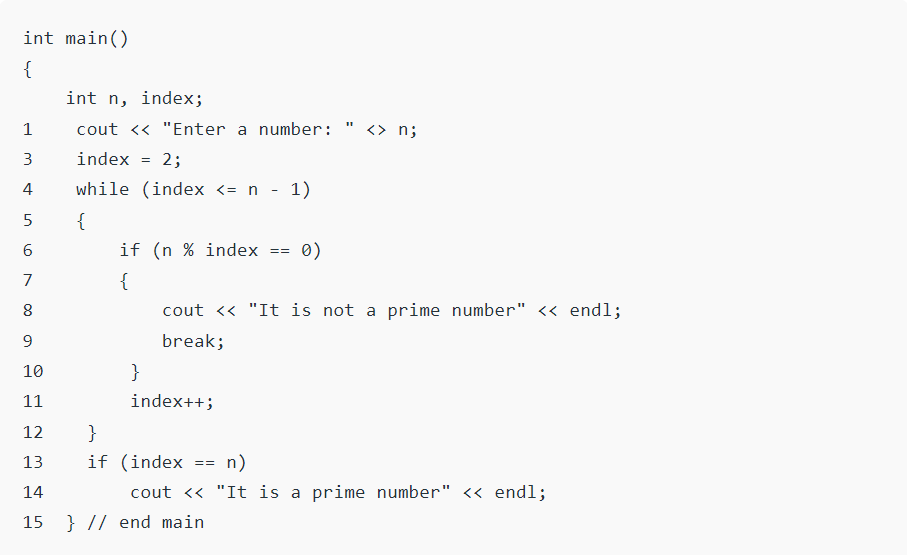


**1. Vẽ biểu đồ luồng điều khiển**

Bước 1:

Bắt đầu đánh số các câu lệnh sau khi khai báo biến (nếu chưa có biến nào được khởi tạo trong câu lệnh đó). Tuy nhiên, nếu một biến đã được khởi tạo và khai báo trên cùng một dòng, thì việc đánh số sẽ bắt đầu từ chính dòng đó.

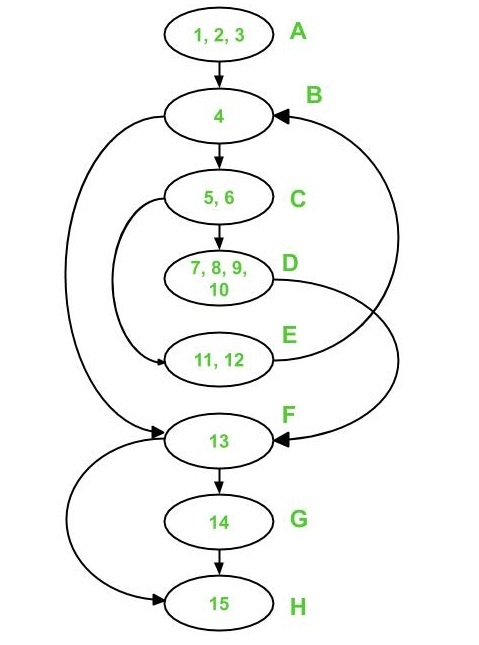
Đối với chương trình đã cho, đây là cách đánh số sẽ được thực hiện:



Bước 2:

Đặt các câu lệnh tuần tự vào một nút duy nhất. Ví dụ: các câu lệnh 1, 2 và 3 đều là các câu lệnh tuần tự và do đó nên được kết hợp thành một nút duy nhất.

Đồ thị thu được sẽ như sau:



**2. Tính độ phức tạp Cyclomatic :**

Phương pháp:

V(G) = e - n + 2\*p

Trong biểu đồ luồng điều khiển ở trên,

trong đó, e = 10, n = 8 và p = 1

Do đó,

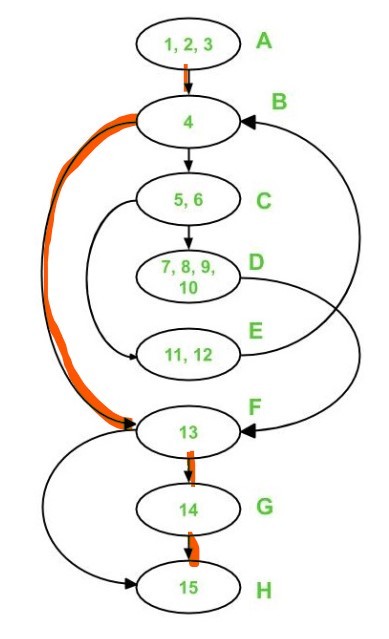
Độ phức tạp theo chu kỳ V(G) = 10 - 8 + 2 \* 1 = 4

**3. Đường đi độc lập :**

Vì độ phức tạp chu trình V(G) của đồ thị là 4, do đó có 4 đường đi độc lập.

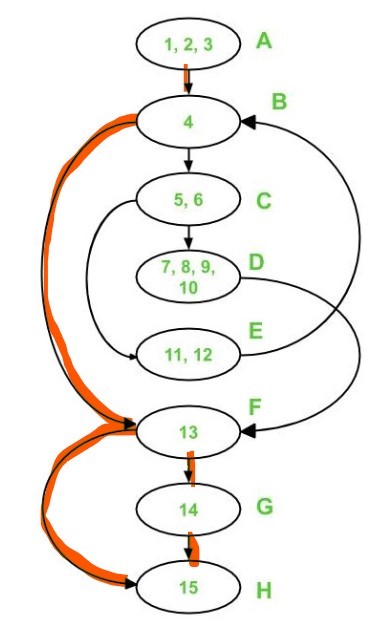
Các cạnh được bao phủ (được đánh dấu bằng màu đỏ) bởi Đường dẫn 1 là:

Đường dẫn 1 : A - B - F - G - H



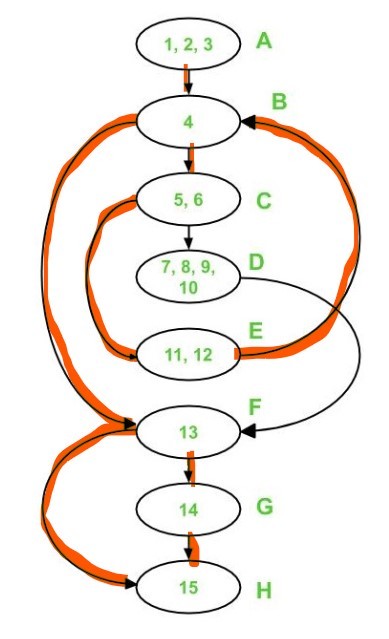
Các cạnh được bao phủ bởi Đường dẫn 1 và Đường dẫn 2 được hiển thị bên dưới:

Đường dẫn 2 : A - B - F - H



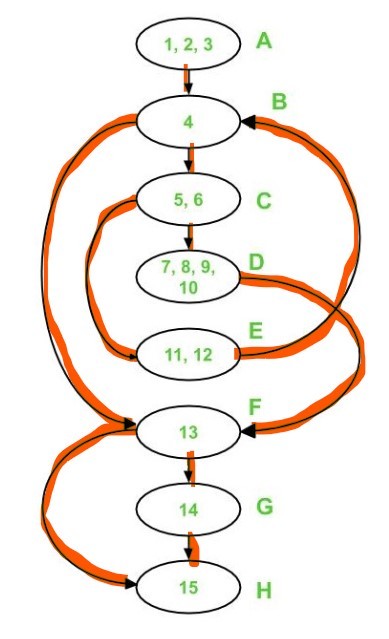
Các cạnh được bao phủ bởi Đường dẫn 1, Đường dẫn 2 và Đường dẫn 3 là:

Đường dẫn 3 : A - B - C - E - B - F - G - H



Bây giờ chỉ còn lại 2 cạnh là cạnh CD và cạnh DF. Do đó, Đường dẫn 4 phải bao gồm hai cạnh này.

Đường dẫn 4 : A - B - C - D - F - H



**4. Các trường hợp kiểm tra:**

Để rút ra các trường hợp kiểm tra, chúng ta phải sử dụng các đường dẫn độc lập có được trước đó. Để thiết kế một trường hợp thử nghiệm, hãy cung cấp đầu vào cho chương trình sao cho mỗi đường dẫn độc lập được thực thi.

Đối với chương trình đã cho, sẽ thu được các trường hợp thử nghiệm sau:

|  |  |  |  |
| --- | --- | --- | --- |
| Mã trường hợp thử nghiệm | Số đầu vào | đầu ra | Con đường độc lập được bảo hiểm |
| 1 | 1 | không có đầu ra | ABFH |
| 2 | 2 | Nó là một số nguyên tố | ABFGH |
| 3 | 3 | Nó là một số nguyên tố | ABCEBFGH |
| 4 | 4 | Nó không phải là một số nguyên tố | ABCDFH |

**Câu 3: Kiểm thử Backend và ví dụ**

**3.1 Kiểm thử Backend là gì?**

Kiểm tra Backend là phương pháp kiểm tra kiểm tra phía máy chủ hoặc cơ sở dữ liệu của ứng dụng web hoặc phần mềm. Kiểm thử backend nhằm mục đích kiểm tra lớp ứng dụng hoặc lớp cơ sở dữ liệu để đảm bảo rằng ứng dụng web hoặc phần mềm không có lỗi cơ sở dữ liệu như bế tắc, hỏng dữ liệu hoặc mất dữ liệu.

Kiểm tra Backend rất quan trọng vì nếu không được thực hiện đúng cách, nó có thể gây ra một số biến chứng nghiêm trọng như bế tắc, hỏng dữ liệu, mất dữ liệu, v.v.

**3.2 Cách thực hiện kiểm thử Backend**

Trong thử nghiệm back-end, bạn có thể trực tiếp chuyển yêu cầu thông qua một số trình duyệt với các tham số cần thiết cho chức năng và nhận phản hồi ở một số định dạng mặc định. Ví dụ, XML hoặc JSON. Bạn cũng cần kết nối trực tiếp với cơ sở dữ liệu và xác minh dữ liệu bằng các truy vấn SQL. Thông qua các tệp nhật ký, việc gỡ lỗi có thể được thực hiện.

Có nhiều giai đoạn khác nhau trong thử nghiệm back-end. Bước đầu tiên là lấy thông số kỹ thuật thiết kế cho máy chủ cơ sở dữ liệu. Bước tiếp theo là kiểm tra thiết kế đặc tả, tiếp theo là thực hiện các kiểm tra trong thiết kế này bằng mã SQL.

Các loại thử nghiệm cơ sở dữ liệu bao gồm:

* Kiểm tra kết cấu
* Thử nghiệm chức năng
* Kiểm thử phi chức năng

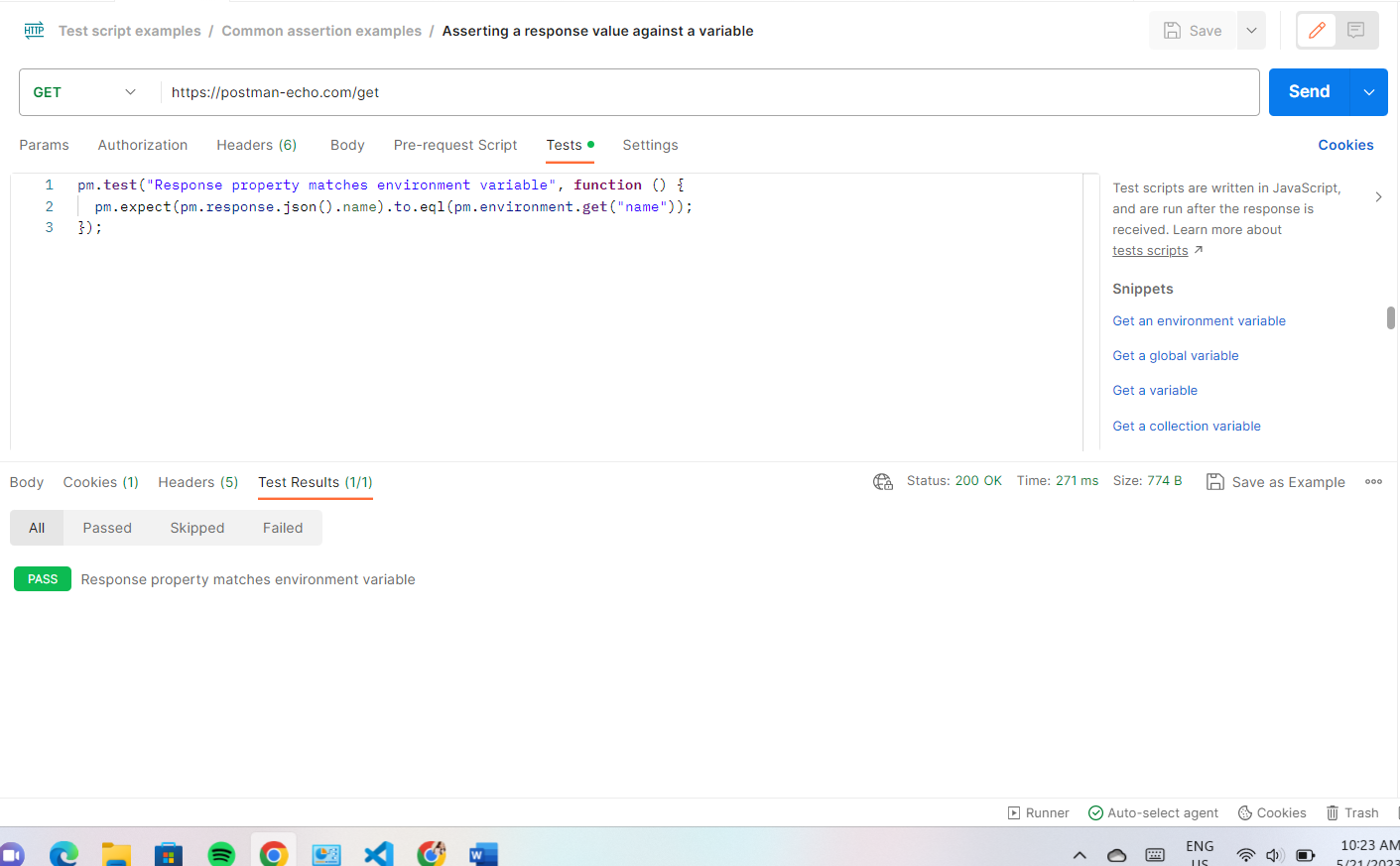
**3.3 Ví dụ về kiểm thử Backend**

**Công cụ sử dụng: Postman**

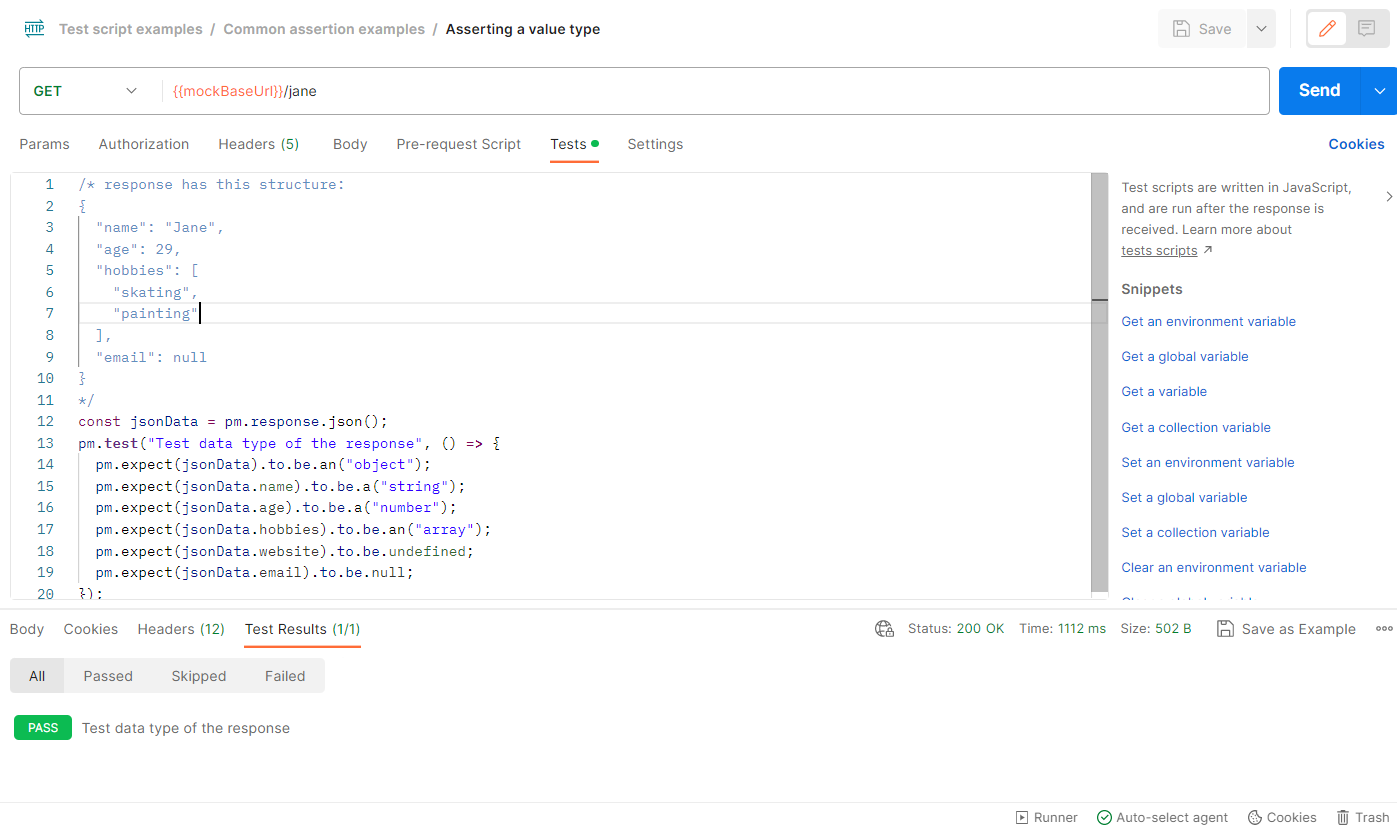
**Trường hợp thử nghiệm: Test Script trong Postman**

* Test script trong Postman được sử dụng để xác nhận các kết quả và kiểm tra tính đúng đắn của các request và response được gửi và nhận về từ API. Khi bạn tạo một test script trong Postman, bạn có thể thực hiện các kiểm tra tự động trên response data, header, status và thực hiện các yêu cầu xác thực.
* Test script trong Postman có thể giúp bạn tiết kiệm thời gian kiểm thử và đạt được các kết quả kiểm tra nhanh chóng và chính xác. Làm việc với các tự động hóa và scripts cũng giúp cho các lập trình viên tạo ra các ứng dụng tốt hơn, giảm thiểu lỗi và giảm thời gian phát triển.

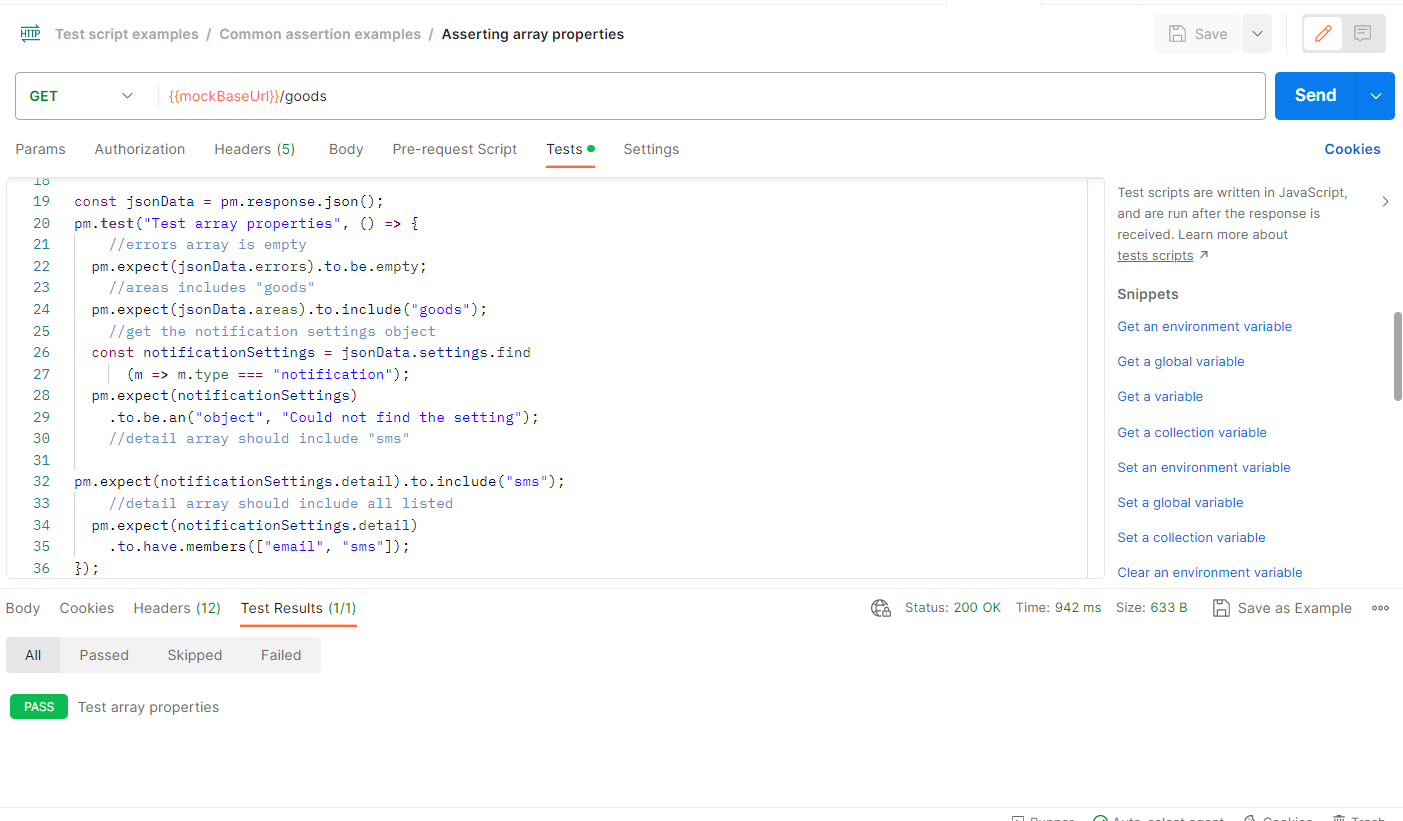
**Xác nhận giá trị phản hồi đối với một biến**

****

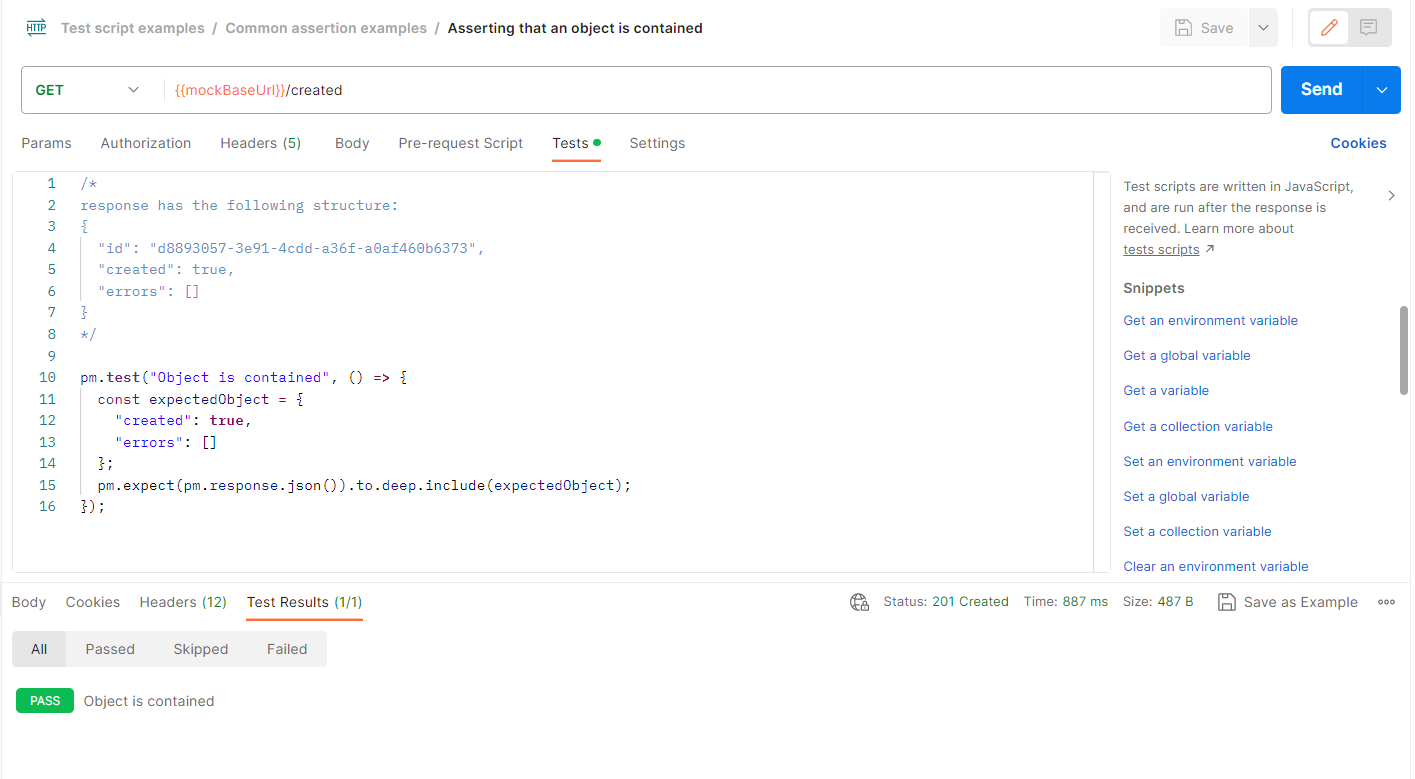
**Khẳng định một loại giá trị**

****

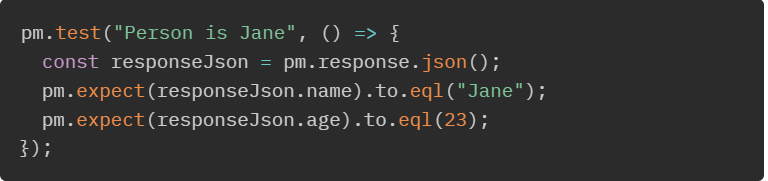
**Khẳng định thuộc tính mảng**

****

**Khẳng định một đối tượng được chứa**

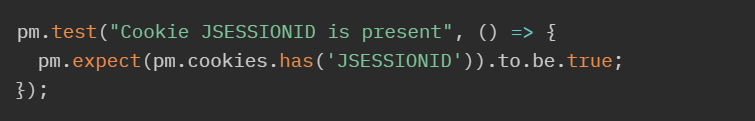
****

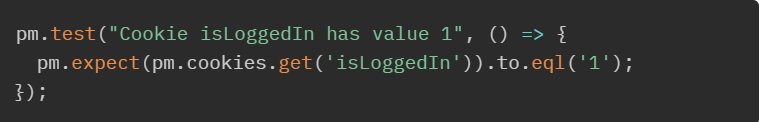
**Đưa ra các xác nhận về phản hồi HTTP**

****

**Kiểm tra cookie**

Kiểm tra xem có cookie nào trong phản hồi không:

****

****

**Câu 4: Tìm hiểu và sử dụng các kỹ thuật kiểm thử được hỗ trợ bởi công cụ kiểm thử Watir.**

**4.1 Khái niệm Watir**

**Watir** (Web Application Testing in Ruby) là một thư viện Ruby mã nguồn mở dành cho kiểm thử tự động. Watir tương tác với một trình duyệt giống như cách mọi người thường làm: nhấp vào các liên kết, điền vào biểu mẫu và xác thực văn bản.

Người dùng watir có thể xây dựng một web scraper (trích xuất nội dung trang web) để mô phỏng việc vào một trang web, đăng nhập, gửi bình luận hoặc tải dữ liệu về và nhiều thứ khác nữa.

Watir cho phép kỹ sư kiểm thử tạo ra những test case dễ đọc hiểu và maintain. Đây là một công cụ khá dễ sử dụng và linh hoạt.

**4.2 Các thành phần của Watir**

a) **Watir-Classic**

* Chỉ hỗ trợ Internet Explorer trên Windows
* Đã tích hợp sẵn các khả năng Liên kết và nhúng đối tượng
* Watir-Classic hoạt động bằng cách mô phỏng trình duyệt
* Watir-Classic trực tiếp điều khiển trình duyệt thông qua giao thức OLE, được triển khai trên cấu trúc Mô hình đối tượng thành phần(COM)

b) **Watir-Webdriver**

* Hỗ trợ trên các trình duyệt phổ biến như Chrome, Firefox, Opera,..
* Là một phiên bản hiện đại của Watir API dựa trên Selenium
* Được xây dựng từ đặc tả HTML

**4.3 So sánh với ứng dụng kiểm thử khác**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tính năng | Watir | Selenium | TestComplete | UFT | Katalon Studio |
| Nền tảng phát triển test | Đa nền tảng | Đa nền tảng | Windows | Windows | Đa nền tảng |
| Ngôn ngữ viết script | Ruby | Java, C#, Perl, Python, JavaScript, Ruby, PHP | JavaScript, Python, VBScript, JScript, Delphi, C++ and C# | VBScript | Java/Groovy |
| Hệ điều hành được hỗ trợ | Windows, Mac, Linux | Windows, Mac, Linux | Windows, Mac | Windows | Windows, Mac, Linux |
| Yêu cầu kĩ năng lập trình | Không nhiều như Selenium Webdriver / UFT.Nhưng cần kiến thức tốt về Ruby | Kỹ năng nâng cao cần thiết để tích hợp các công cụ khác nhau | Không yêu cầu, Cần kỹ năng tạo test scripts | Không yêu cầu. Cần kỹ năng tạo test scripts | Không yêu cầu. Cần kỹ năng tạo test scripts |
| Lưu trữ đối tượng và bảo trì | Ruby đã xây dựng các khả năng Liên kết và nhúng (Object) và nhúng (OLE) | XPath, UI Maps | Xây dựng dựa vào kho đối tượng để lấy ra các đối tượng thông dụng | Xây dựng dựa vào kho đối tượng để lấy ra đối tượng chính xác | xây dựng dựa vào kho đối tượng, Xpath, nhận dạng lại đối tượng |
| Tạo và lưu kết quả test | Sử dụng RSpec framework WATIR có thể tạo HTML report để giữ kết quả test | Selenium Webdriver có thể tạo kết quả test bằng việc tương tác với frameworks khác (như TestNG) | Có thể truy xuất kết quả test sử dụng định dạng như MHT HTML XML PDF tcLogX | Sau mỗi lần thực hiện test QTP/UFThiển thị kết quả test , mỗi dòng test script sẽ hiển thị pass/fail | Tạo report của test suites với những định dạng như HTML, CSV, PDF và Junit sử dụng context menu trong test explorer |
| Framework được hỗ trợ | Cucumber, RSpec, Test/Unit | C# : Nunit Java: JUnit/TestNG Python: pyunit, py.test, robot Javascript:WebdriverJS, WebdriverIO, Nightwatch JS | Data-driven testing framework | Data Driven automation framework Keyword Driven automation framework |  |

**4.4 Demo**

**4.4.1 Hướng dẫn sử dụng**

Tải thư viện Watir

*require ‘watir’*

Mở trình duyệt

*browser=Watir::Browser.new:chrome # firefox or ie*

Mở trình duyệt với URL cụ thể: (mở với trình duyệt mặc định: chrome)

*browser=Watir::Browser.start(“*[*https://www.google.com*](https://www.google.com)*/”)*

Di chuyển đến URL cụ thể:

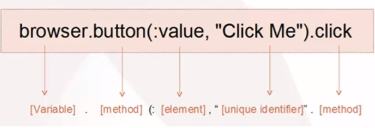
*browser.goto(“*[*https://www.facebook.com/*](https://www.facebook.com/)*”)*

Đóng trình duyệt

*Browser.close*

Truy cập tới 1 Element

Cấu trúc:



Text box hoặc text area

*t = browser.text\_field({name: "username"})*

hoặc *t = browser.text\_field name: "username"*

Button

*b = browser.button({value: "Click Here"})*

Drop down list

*d = browser.select\_list({name: "month"})*

Check box

*c = browser.checkbox({name: "enabled"})*

Radio Button

*r = browser.radio({name: "payment type"})*

Form

*f = browser.form({name: "address"})*

*f = browser.form({action: "submit"})*

Link

*l = browser.link({url: "http://google.com"})*

*l = browser.link({href: "http://google.com"})*

Table: trỏ tới một ô trong bảng (hàng 2 cột 1)

*td = browser.table({name: 'recent\_records'})[2][1]*

Kiểm tra nội dung

* Trả về html của trang hoặc element:

*browser.html*

*e.html*

* Trả về text của trang hoặc element:

*browser.text*

*e.text*

* Trả về title của tài liệu

*browser.title*

* Lấy text từ status bar

*browser.status*

* Trả về true nếu text mô tả xuất hiện trên trang

*browser.text.include? 'llama'*

* Trả về nội dung của bảng dạng array

*browser.table({id, 'recent\_records'}).to\_a*

**4.4.2 Demo ví dụ**

Ví dụ 1:

Công việc: mở trang “Google.co.in” và nhấp vào liên kết Gmail.



Giải thích mã:

1*) request 'watir'* - yêu cầu thư viện Watir để chạy chương trình

2) *test\_site = “*[*https://www.google.co.in/*](https://www.google.co.in/)*” -* Khai báo biến.

3) browser = Watir :: Browser.new: firefox: Mở trình duyệt Firefox.Ngoài ra, chúng ta có thể mở trình duyệt khác như sau:

*Watir :: Browser.new: internet\_explorer -* Dành cho Internet Explorer

*Watir :: Browser.new: safari -* Dành cho Safari

Nếu chúng ta không đề cập đến tên trình duyệt, thì chúng ta sẽ chỉ cần viết dòng lệnh *[Watir :: Browser.new],* Watir sẽ mở trình duyệt Chrome vì Chrome là trình duyệt mặc định cho Watir.

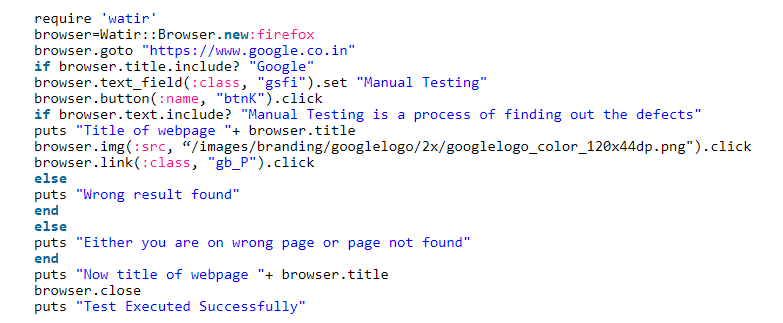
4) *browser.goto test\_site* hoặc *browser.goto “https://www.google.co.in/” -*  được sử dụng để điều hướng đến trang web được chỉ định.

5) *browser.link (: href, “https://mail.google.com/mail/?tab=wmosystem).click* - Tại đây chúng ta nhấp vào liên kết Gmail. Siêu kết nối được coi là một liên kết trong mã và chúng ta đang sử dụng thuộc tính “href” của nó để định vị siêu kết nối này.

6*) puts “Test Executed”* - Dòng này sẽ viết chuỗi “Test Executed” trên dấu nhắc lệnh.

7) *browser.close* - Dòng này sẽ đóng trình duyệt.

Ví dụ 2: Công viêc: mở trang web “www.google.co.in”.  Tìm kiếm văn bản " Manual Testing". Cuối cùng quay lại trang chủ và nhấp vào liên kết “Gmail”.



Giải thích mã:

1) *browser.goto https://www.google.co.in* - Điều hướng đến trang web được chỉ định.

2) *if browser.title.include? “Google”* - Ở đây chúng ta đang kiểm tra tiêu đề của trang web xem nó có như mong đợi hay không. Trong phần khác của phần "*if*" này, chúng tôi đưa ra tuyên bố này " *Either you are on the wrong page or page not found*”. Câu lệnh này sẽ được hiển thị trên dấu nhắc lệnh.

3*) browser.text\_field(:class, “gsfi”).set “Manual Testing”* - Ở đây chúng tôi đang nhập văn bản “Kiểm tra thủ công” vào hộp văn bản tìm kiếm.

4*) browser.button(:name, “btnK”).click* - Ở đây chúng tôi đang nhấp vào nút ““Google Search””.

5) *if browser.text.include? “Manual Testing is a process of finding out the defects”* - Bây giờ chúng tôi đang kiểm tra văn bản trên trang web để xác minh xem chúng tôi có nhận được kết quả tìm kiếm chính xác hay không. Trong phần '*else*' của câu lệnh "*if*" này, chúng tôi đặt câu lệnh "Đã tìm thấy kết quả sai".

6) đặt “Tiêu đề trang web“ + browser.title - Ở đây chúng tôi đang hiển thị tiêu đề của trang web trên dấu nhắc lệnh.

7)*browser.img(:src, “/images/branding/googlelogo/2x/googlelogo\_color\_120x44dp.png”).click* - Tại đây chúng tôi nhấp vào hình ảnh để quay lại trang chủ.

8) *browser.link(:class, “gb\_P”).click* - Tại đây chúng tôi đang nhấp vào liên kết Gmail.

9) *puts “Now the title of webpage “+ browser.title* - Bây giờ một lần nữa chúng ta đặt tiêu đề của trang web trên dấu nhắc lệnh. Nó phải là tiêu đề của trang Gmail.

10) *browser.close* - Ở đây chúng ta đang đóng trình duyệt.

11) *puts “Test Executed Successfully*” – Để đảm bảo rằng tất cả các bước được thực hiện chính xác, chúng ta đặt câu lệnh này trên dấu nhắc lệnh.

Ví dụ 3

Trong ví dụ này, chúng tôi sẽ cố gắng sử dụng các lệnh Watir tối đa. Chúng ta sẽ xem mô tả cùng với mã ở dưới đây.

|  |
| --- |
| require 'watir'  browser=Watir::Browser.new:chrome  browser.goto “http://watir.com/examples/forms\_with\_input\_elements.html”    # Ở đây chúng ta sẽ xem cách chèn dữ liệu vào trường văn bản.  t1=browser.text\_field id: 'new\_user\_first\_name'  t1.set "Testing"  t2=browser.text\_field id: 'new\_user\_last\_name'  t2.set “testdata"    # Ở đây chúng ta sẽ thấy cách chọn dữ liệu từ danh sách chọn.  sl=browser.select\_list id: 'new\_user\_languages'  sl.select 'English'  sl.selected\_options    # Ở đây chúng ta sẽ thấy cách chọn nút radio.  rd=browser.radio value: 'no'  rd.set  chk=browser.checkbox value: 'cars'  chk.set    # Ở đây chúng ta sẽ thấy cách bấm vào nút.  btn=browser.button name: 'new\_user\_button\_2'  btn.click    # Đây là một yêu cầu chờ đợi. Yêu cầu chương trình phải chờ đợi  # cho đến khi có sự kiện sảy ra  Watir::Wait.until {browser.text.include? 'submit'}    # Ở đây tôi đã sử dụng câu lệnh if này vì đây là trang web thử nghiệm  # và không có gì xảy ra sau khi nhấp vào nút. Vì vậy, chỉ để hiển thị  # rằng nút được nhấp, tôi đã sử dụng câu lệnh If  if browser.text.include? "submit"  puts "Data Submitted"  else  puts "Data not yet submitted"  end    # Đây là một tuyên bố chờ đợi ngầm.  # Nó sẽ khiến chương trình của bạn bắt buộc phải đợi trong 30 giây.  browser.driver.manage.timeouts.implicit\_wait = 30    # Bạn có thể gửi các khóa đặc biệt cho trình duyệt.  # Có một danh sách lớn các phím này.  browser.send\_keys :page\_up    # Ở đây chúng tôi đang chụp ảnh màn hình và lưu nó trong một tệp png.  browser.screenshot.save ‘screenshot1.png'    browser.send\_keys :page\_down  browser.screenshot.save ‘screenshot2.png'    puts "TestCase Executed"  browser.close |