ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

BÁO CÁO NGHIÊN CỨU KHOA HỌC

ĐỀ TÀI: **KIỂM THỬ ỨNG DỤNG DI ĐỘNG BẰNG CALABASH**

Sinh viên thực hiện: Nguyễn Ngọc Thoại

Giáo viên hướng dẫn: TS. Trần Thị Minh Châu

1
Giới thiệu

1.1 Phát triển hướng hành vi (BDD)

BDD là việc phát triển ứng dụng bằng cách dựa trên mô tả hành vi của ứng dụng từ khách hàng. Nó được xây dựng dựa trên TDD và kế thừa những điểm mạnh của TDD. Có thể nói BDD là một sự phát triển mới của TDD với một số đặc điểm nổi bật như:

Sử dụng ngôn ngữ tự nhiên

Đặc điểm nổi bật nhất của BDD là diễn tả các hành vi của ứng dụng bằng ngôn ngữ tự nhiên không phải ngôn ngữ kĩ thuật. Chính vì lý do này, khách hàng có thể hiểu và tham gia làm các ca kiểm thử cho ứng dụng. Điều này khiến cho khách hàng, bên phát triển có sự gắn kết chặt chẽ hơn và cũng đảm bảo được phần mềm thực hiện đúng theo yêu cầu, sớm phát hiện được lỗi để đưa ra các phương án xử lý kịp thời. Sự tương tác chặt chẽ này làm giảm đáng kể thời gian phát triển của phần mềm.

Không phụ thuộc vào môi trường phát triển ứng dụng

Với TDD thì chúng ta phải chọn được môi trường lập trình ứng dụng trước rồi cài đặt ca kiểm thử trên môi trường đó. Với BDD thì môi trường không còn quan trọng. BDD hoàn toàn độc lập với ứng dụng, điều này làm cho các ca kiểm thử trở lên khách quan hơn và ít bị phụ thuộc vào môi trường phát triển phần mềm. Điều này cũng làm cho BDD trở lên linh hoạt, có thể áp dụng với các phần mềm trên nhiều môi trường khác nhau.

Nghiệm thu tự động

Một ưu điểm khác của BDD là nghiệm thu tự động. Với TDD khi phát triển xong phần mềm cần chuyển cho bên khách hàng để nghiệm thu. Nếu nghiệm thu không thành công, bên phát triển sẽ phải sửa lại, điều này làm cho thời gian phát triển phần mềm kéo dài hơn. Với BDD thì khi phát triển xong phần mềm thì bên khách hàng cũng chấp nhận luôn bởi vì trong quá trình phát triển khách hàng đã trực tiếp tham gia kiểm định nên phần mềm đã được cài đặt theo đúng yêu cầu của khách hàng.

Kết luận

Với những ưu điểm trên có thể nói BDD phù hợp với mô hình phát triển phần mềm nhanh và hiện đại (Agile).

1.2 Vai trò, tầm quan trọng của kiểm thử tự động

Để đảm bảo chất lượng tất cả các phần mềm được kiểm thử trước khi đưa ra sử dụng. Nếu quá trình kiểm thử không tốt, khi đưa phần mềm còn nhiều lỗi vào sử dụng sẽ gây thiệt hại lớn cho khách hàng điều này làm giảm sự tín nhiệm của khách hàng với nhà phát triển. Điều này cho thấy kiểm thử giữ một vai trò quan trọng trong phát triển phần mềm. Nó đảm bảo uy tín và sự phát triển hợp tác lâu dài của nhà phát triển với khách hàng.

Trong phát triển ứng dụng web trước kia người ta thường kiểm thử bằng cách thủ công. Người kiểm thử chạy phần mềm và nhập các ca kiểm thử một cách thủ công. Thực tế, trong quá trình phát triển phần mềm khi cập nhật phiên bản mới ta thường phải kiểm tra phần mới thêm vào có gây lỗi cho các phần trước đó không vì vậy người kiểm thử lại phải thực hiện các ca kiểm thử trước đó. Việc lặp đi lặp lại một cách thủ công này gây ra một sự nhàm chán cho người kiểm thử. Hơn nữa việc kiểm thử thủ công là chậm, không đảm bảo được thời gian phát triển phần mềm. Việc kiểm thử thủ công cũng khó đảm bảo được độ chính xác của các ca kiểm thử, việc nhập dữ liệu nhầm, thực hiện không đúng các bước … có thể làm giảm độ tin cậy và chất lượng của các ca kiểm thử. Kiểm thử thủ công khó có thể bao phủ được hầu hết các trường hợp có thể xảy ra. Hơn nữa, kiểm thử thủ công chỉ được thực hiện khi đã có phần mềm hoàn chỉnh. Những nhược điểm trên cho thấy kiểm thử thủ công không còn phù hợp với xu hướng phát triển hiện tại.

Kiểm thử tự động được đưa ra để giải quyết các vấn đề mà kiểm thử thủ công gặp phải. Có thể kể ra một vài ưu điểm nổi bật của kiểm thử tự động như sau:

Tiết kiệm thời gian và kinh phí

Để đảm bảo chất lượng của phần mềm thì trong quá trình phát triển các ca kiểm thử được lặp đi lặp lại nhiều lần. Với kiểm thử tự động ta chỉ viết các đoạn mã kiểm thử một lần để chạy cho các lần kiểm thử khác nhau. Điều này làm giảm công sức của người kiểm thử và giảm kinh phí dành cho kiểm thử.

Đảm bảo sự chính xác của các ca kiểm thử

Các ca kiểm thử được viết dưới dạng các đoạn mã, nếu đoạn mã là chính xác thì các ca kiểm thử chạy đoạn mã này sẽ đảm bảo được chất lượng.

Kiểm tra mức độ chịu tải

Kiểm thử tự động có thể kiểm tra mức độ chịu tải của một trang web bằng cách giả lập gửi nhiều yêu cầu đến một trang web cùng một lúc. Điều này khó có thể thực hiện được nếu dùng phương pháp kiểm thử thủ công.

Viết ca kiểm thử trước khi viết chương trình

Kiểm thử tự động có thể viết các ca kiểm thử từ khi bắt đầu phát triển phần mềm. Điều này rất phù hợp với mô hình phát triển nhanh (Agile).

2

Cucumber-Capybara

2.1 Giới thiệu về Cucumber

Cucumber là một BDD framework. Cucumber sử dụng ngôn ngữ Gherkin, nó là một ngôn ngữ tự nhiên tất cả mọi người đều có thể dễ dàng đọc được. Cucumber dùng ngôn ngữ này để mô tả các hành vi của ứng dụng mà không nêu rõ chi tiết phần cài đặt của hành vi này. Cucumber mô tả các ca kiểm thử trong file .feature. Cucumber sử dụng các từ khóa như feature, scenario, given, when, then dùng cho việc mô tả một hành vi nào đó của ứng dụng. Để cài đặt các ca kiểm thử Cucumber hỗ trợ nhiều ngôn ngữ khác nhau như Ruby, Java, .NET, Python, Perl, Earlang, PHP. Tuy nhiên ngôn ngữ cài đặt được khuyến khích và là mặc định của Cucumber là ruby.

Feature

Mỗi một file .feature thường tương ứng với một feature. Bắt đầu file .feature là từ khóa Feature. Từ khóa này dùng để đưa ra một mô tả tổng quan về kịch bản của ca kiểm thử. Theo sau một từ khóa Feature thường có một hoặc nhiều Scenario.

Scenario

Mỗi Scenario mô tả các hành vi của phần mềm muốn kiểm thử. Để mô tả một Scenario Cucumber thường sử dụng các từ khóa Given, When, Then để mô tả hoàn cảnh, các hành động và kết quả của ca kiểm thử.

Given

Từ khóa Given Given dùng để đưa ra hoàn cảnh kiểm thử, tiền điều kiện để thực hiện ca kiểm thử này.

When

Từ khóa When dùng để mô tả các hành vi khi kiểm thử. Ví dụ là click vào một button, link hay điền vào form.

Then

Từ khóa Then dùng để mô tả kết quả cần có mà phần mềm phải đáp ứng khi chạy ca

kiểm thử này.

Cài đặt ca kiểm thử

Khi Cucumber chạy nó sẽ tìm các từ khóa ở dưới Scenario như Given, When, Then rồi ghép phần cài đặt ở trong phần step_definitions để chạy cả kiểm thử.

Ví dụ:

```
Feature: Kiểm tra số nguyên tố
Scenario: Số nguyên tố
Given Tôi đang ở trang “http://localhost/nguyenTo”
When Tôi nhập số “5” vào ô text box
And Tôi click vào nút kiểm tra
Then Tôi sẽ nhìn thấy “Nguyên tố” trên màn hình

Scenario: Không phải nguyên tố
Given Tôi đang ở trang “http://localhost/nguyenTo”
When Tôi nhập số “4” vào ô text box
And Tôi click vào nút kiểm tra
Then Tôi sẽ nhìn thấy “Không phải nguyên tố” trên màn hình
```

2.2 Capybara

Capybara cung cấp các API giao tiếp với rất nhiều driver như Selenium WebDriver, Phantomjs... Sử dụng Capybara làm cho công việc kiểm thử trở nên đơn giản hơn rất nhiều so với việc dùng trực tiếp các webdriver như Selenium, Phantomjs.

Giao diện hoạt động của Capybara (Capybara API)

Điều hướng

Capybara cung cấp hàm visit để điều hướng trang web. Tham số của hàm visit là link của trang web muốn đến

Ví dụ

```
Given(/^Tôi đang ở link google.com/) do
  visit “http://google.com”
End
```

Click link hay button

Capybara cung cấp hàm sau để click vào link hay button trên trang web:

*click_link_or_button: dùng để click vào link hoặc button

* click_on: dùng để click vào link hoặc button

* click_link: dùng để click vào một đường link

* click_button: dùng để click vào một button

Tham số theo sau các hàm trên là id, title hay giá trị nhìn thấy của link trên trang web

Ví dụ

Trên trang web có một link như sau:

```
<a id=”my_link” title=”my_link_title”>Click link</a>
```

Có thể dùng một trong các cách sau để click vào link này:

```
click_link_or_button “my_link”
click_on “my_link_title”
click_link “Click link”
```

Với button cũng tương tự:

```
<button id=”my_button” title=”my_button_title”>Click button</button>
```

Để click vào button này có thể dùng các cách sau

```
click_link_or_button “my_button”
click_on “my_button_title”
click_button “Click button”
```

Thao tác với Checkbox và Radiobutton

Capybara sử dung hàm check cho checkbox và choose cho radiobutton. Theo sau các hàm này là id của checkbox, radiobutton.

Ví dụ:

Với radiobutton:

```
<input type="radio" name="overage" value="over" id="over_16">Trên 16
<input type="radio" name="overage" value="under" id="under_16">Dưới 16
```

Để chọn radiobutton này thì dùng:

```
choose “under_16”
```

Với checkbox:

```
<input type="checkbox" value="yes" name="consent_checkbox" id="consent"/>
```

Để check vào checkbox này thì dùng:

```
check “consent”
```

Up load một file lên web

Để up load file Capybara cung cấp hàm attach_file

Ví dụ:

```
<label for="form_image">Image</label>
<input type="file" name="image" id="form_image"/>
```

Để upload một file có thể dùng như sau:

```
attach_file “Image”,  “/Users/picture/foo.png”
```

hoặc `attach_file “form_image”,  “/Users/picture/foo.png”`

Kiểm tra các thành phần trong trang web

Chọn các thành phần trên trang web

* chọn theo css: Capybara cung cấp hàm find <css>

Ví dụ:

```
<div id = 'my_id'>My Text</div>
```

Để chọn thẻ này thì dùng:

```
find “my_id”
```

* Chọn theo thẻ html: Capybara cung cấp hàm have_selector

```
<h1>My page</h1>
```

Để kiểm tra xem trong thẻ h1 có phải là “My page” thì dùng:

```
have_selector “h1”, text: “My page”
```

Kiểm tra có nội dung text không

Capybara cung cấp hàm have_content <content> để kiểm tra nội dung content có xuất hiện trên web không

Ví dụ:

```
<p>text</p>
```

Để kiểm tra xâu kí tự “text” có xuất hiện không thì dùng:

```
have_content “text”
```

Kiểm tra có button không

Capybara cung cấp hàm have_button <button> điểm kiểm tra có xuất hiện button trên web không

Ví dụ:

```
<button id=”my_button”>My Button</button>
```

Để kiểm tra có button này không thì dùng:

`have_button “my_button” hoặc have_button “My Button”`

Kiểm tra có link không

Capybara cung cấp hàm have_link <link> để kiểm tra có xuất hiện link trên web không

Ví du:

```
<a id=”my_link” title=”my_link_title”>Click link</a>
```

Để kiểm tra có link này không thì dùng:

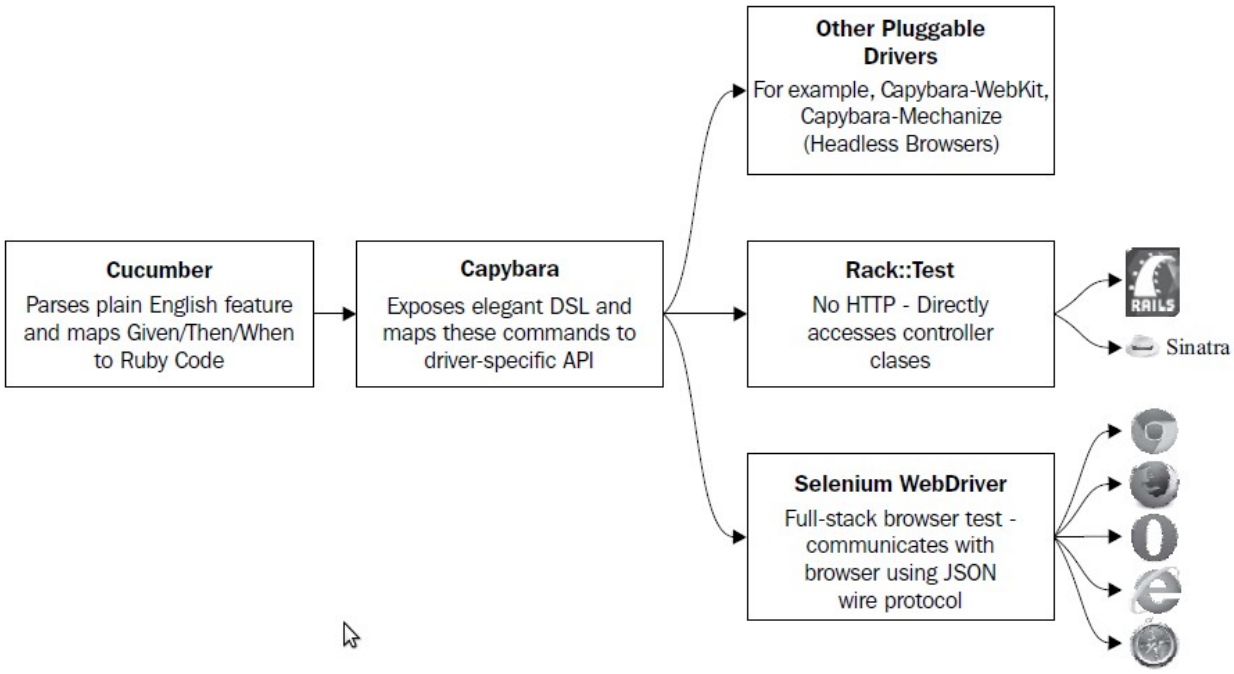
```
have_link “my_link” hoặc have_link “Click link”
```

Note: Để kiểm tra có đúng nội dung cần kiểm tra không xuất hiện trên web thì cũng làm tương tự như trên thay have.. = have_no...

```
have_content -> have_no_content
```

2.3 Mô hình hoạt động của Cucumber-Capybara

Khi Cucumber chạy nó sẽ tìm các đoạn mô tả ca kiểm thử ở trong Scenario (Given, When, Then) ghép với phần cài đặt ở trong step_definitions. Trong phần cài đặt bằng mã ruby chẳng hạn, mã ruby sẽ sử dụng đến các hàm mà Capybara cung cấp để thực hiện các ca kiểm thử. Capybara có nhiệm vụ liên kết với các web driver như Selenium WebDriver, Phantomjs ... để thực thi ca kiểm thử.

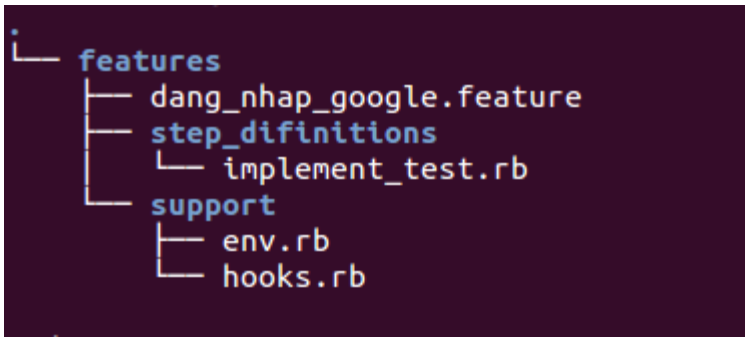


Hình 2.1 Hoạt động của Cucumber - Capybara

2.4 Giới thiệu ứng dụng minh họa

Giới thiệu ứng dụng kiểm tra sự đăng nhập tài khoản Google

Cấu trúc thư mục



Hình 2.2 Cấu trúc thư mục của project

Viết feature

features/dang_nhap_google.feature

Feature: Đăng nhập tài khoản google

Scenario: Đăng nhập với tài khoản không đúng

Given Tôi đang ở link "https://accounts.google.com/ServiceLogin?sacu=1&scc=1&continue=http%3A%2F%2Fmail.google.com%2Fmail%2F&service=mail&ltmpl=default&rm=false"

When Tôi điền "Email" là "invalidemail"

And Tôi điền "Passwd" là "invalidpasswd"

And Tôi click vào nút "signIn"

Then Tôi sẽ nhìn thấy thông báo "The email or password you entered is incorrect"

Scenario: Đăng nhập thành công

Given Tôi đang ở link "http://accounts.google.com/ServiceLogin?sacu=1&scc=1&continue=http%3A%2F%2Fmail.google.com%2Fmail%2F&service=mail&ltmpl=default&rm=false"

When Tôi điền "Email" là "ngocthoaia12"

And Tôi điền "Passwd" là "!jmmkmll8"

And Tôi click vào nút "signIn"

Then tôi nhìn thấy link "Hộp thư đến"

Khởi tạo môi trường

features/support/env.rb

```
require "capybara/cucumber"
```

features/support/hooks.rb

```
  Capybara.app_host="http://google.com"
  Capybara.default_driver = :selenium
  Capybara.default_wait_time = 15
```

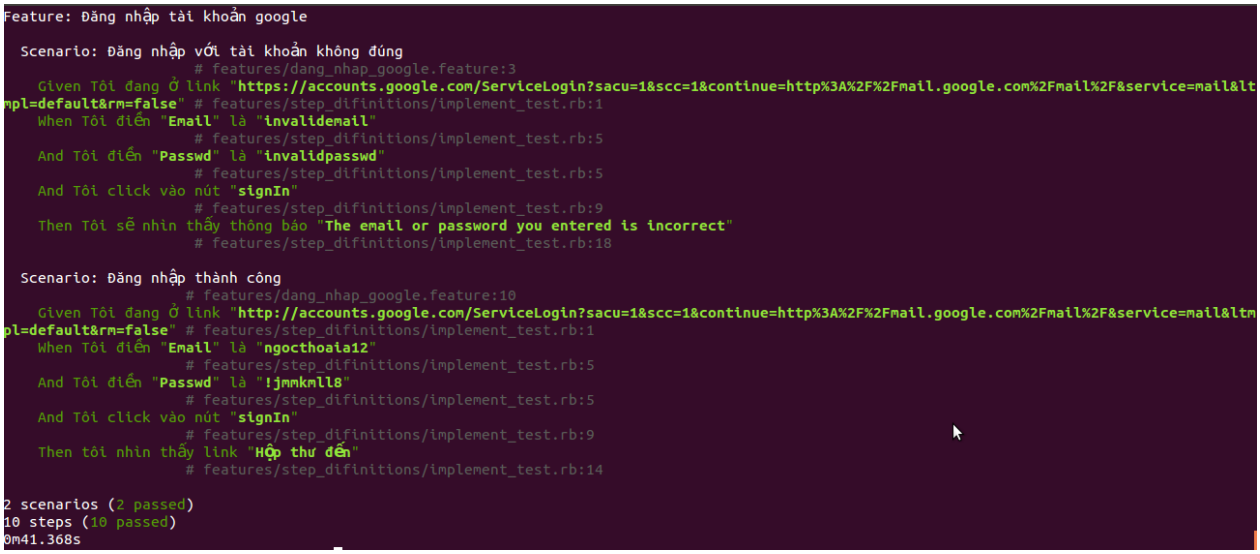
Cài đặt ca kiểm thử

features/step_difinitions/implement.rb

```
Given(/^Tôi đang ở link "(.*?)"$/) do |link|
 visit link
end
When(/^Tôi điền "(.*?)" là "(.*?)"$/) do |id, value|
 fill_in id, with: value
end
When(/^Tôi click vào nút "(.*?)"$/) do |button|
 should have_button button
 click_button button
end
Then(/^tôi nhìn thấy link "(.*?)"$/) do |link|
 should have_link(link)
end
Then(/^Tôi sẽ nhìn thấy thông báo "(.*?)"$/) do |message|
 should have_content message
End
```

Kết quả

Kiểm thử trong 2 trường hợp đăng nhập google thành công với tài khoản đúng và không thành công với tài khoản sai.



Hình 2.3 Kết quả kiểm thử

2.5 Tổng kết về kiểm thử tự động viết bằng Cucumber-Capybara

Cucumber là một framework để kiểm thử các ứng dụng web một cách tự động. Với Cucumber các ca kiểm thử được viết bằng ngôn ngữ đơn giản và dễ hiểu trên cơ sở của behavior-driven development (BDD). Cucumber cũng rất linh hoạt trong việc lựa chọn môi trường để viết các ca kiểm thử. Cucumber hỗ trợ rất nhiều ngôn ngữ tự nhiên khác nhau để viết file feature và phần cài đặt cũng hỗ trợ nhiều ngôn ngữ lập trình phổ biến hiện này. Điều này khiến cho người viết kiểm thử không bị hạn chế về ngôn ngữ và khách hàng cũng dễ dàng tham gia vào thực hiện các ca kiểm thử. Cucumber làm cho khách hàng và bên phía lập trình có thể tương tác với nhau nhiều hơn. Vì thế chất lượng của phần mềm được đảm bảo, khách hàng sớm chấp nhận phần mềm hơn, thời gian phát triển phần mềm cũng được giảm thiểu. Các ca kiểm thử của cucumber cũng rất dễ hiểu và dễ để đọc nên cũng không cần tài liệu cho các ca kiểm thử này.

Capybara là một framework cung cấp một giao diện đơn giản để kiểm thử ứng dụng web. Với Capybara việc kiểm thử ứng dụng web trở lên đơn giản hơn, các ca kiểm thử dễ đọc, dễ hiểu và thời gian để viết kiểm thử cũng giảm xuống đáng kể. Capybara cũng hỗ trợ nhiều webdriver khác nhau, như thế sẽ có nhiều sự lựa chọn các webdriver sao cho phù hợp với mục đích và yêu cầu của kiểm thử.

Sự kết hợp giữa Cucumber và Capybara tạo ra công cụ để viết kiểm thử tự động cho các ứng dụng web đơn giản và hiệu quả.