Kiểm thử đơn vị (unit testing), còn được gọi là kiểm thử thành phần (component testing) tập trung vào việc kiểm tra các phần nhỏ của source code như là class mà người phát triển (developer) viết nên. Những test này có ý nghĩa rất quan trọng trong việc giúp đỡ người phát triển đảm bảo những phần nhỏ trong code chạy đúng theo mong đợi và hoạt động một cách chính xác khi được kết hợp với những phần khác của ứng dụng. Kiểm thử như vậy giúp cho việc quản lí ứng dụng theo thời gian bằng cách đảm bảo những thay đổi do người lập trình tạo ra sẽ không vô tình ảnh hưởng đến những phần khác của hệ thống.

Việc kiểm thử đối với Javascript đang ngày càng trở nên quan trọng đối với người lập trình. Ngày nay, có rất nhiều framework để chọn lựa cho việc kiểm thử. Và C9js đã sử dụng những công cụ sau để thực hiện việc kiểm thử:

1. Jasmine
   1. Các khái niệm liên quan
      1. Test-driven development (TDD)

* Là một phương pháp tiên tiến để phát triển phần mềm bắt đầu với việc phát triển các test cho mỗi một tính năng. Các test có thể chạy không đúng như mong muốn bởi vì nó được phát triển thậm chí trước cả khi phát triển phần mềm. Người lập trình sau đó sẽ phát triển và cấu trúc lại mã để vượt qua được những test này.
* Mục tiêu của TDD là khuyến khích các thiết kế đơn giản và tạo ra sự tự tin (Kent Beck), viết mã một cách gọn gàng và chạy được (Ron Jeffries).
* TDD liên quan đến *test-first programming*, tức viết test trước khi viết vừa đủ mã để thực hiện test và cấu trúc lại.
  + 1. Behavior-driven development (BDD)
* Là một quá trình phát triển phần mềm được mở rộng từ TDD. BDD kết hợp những kĩ thuật chung và nguyên tắc của TDD để cung cấp cho việc phát triển phần mềm cũng như quản lý các đội (team) với các công cụ được chia sẽ và một quá trình chia sẽ để hợp tác với nhau trong quá trình phát triển phần mềm.
* Mục tiêu của BDD là tập trung vào việc lấy được hiểu biết rõ ràng về hành vi của phần mềm thông qua thảo luận với các bên liên quan. BDD viết các test bằng ngôn ngữ tự nhiên giúp cho người không phải là lập trình viên cũng có thể đọc được. Các lập trình viên sẽ sử dụng ngôn ngữ thuần túy của họ để mô tả mục đích mã (code) mà họ viết. Điều này sẽ giúp cho các lập trình viên tập trung vào vấn đề tại sao mã này nên được viết ra, chứ không phải về vấn đề kĩ thuật, và giảm thiểu tối đa vấn đề dịch thuật giữa các ngôn ngữ.
  1. Giới thiệu về Jasmine

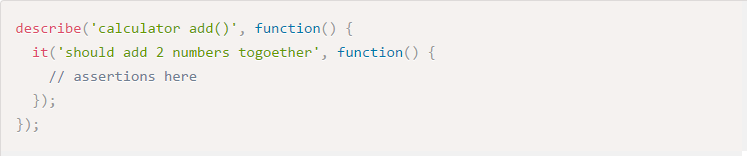
Jasmine là một framework theo mô hình BDD (behavior-driven framework), tức là sự phát triển hướng theo hành vi cho Javascript, dùng cho việc kiểm thử mã Javascript. Jasmine không phụ thuộc vào bất kì framework Javascript nào, không yêu cầu DOM, và cú pháp của Jasmine rất gọn gàng, rõ ràng, dễ dàng viết các test.

* 1. Đặc điểm của Jasmine

Để thấy được đặc điểm, cũng như các điểm mạnh, điểm yếu của Jasmine, chúng ta sẽ so sánh nó với một framework phổ biến khác dùng cho việc kiểm thử, đó là Mocha.

* + 1. API

API của Jasmine và Mocha tương tự như nhau. Người lập trình viết bộ mô tả kiểm thử (test suite) với những khối (block) *describe* và với mỗi test, hay còn gọi là một đặc tả (spec), thì sử dụng hàm *it*.



Sự khác nhau bắt đầu đến từ các hàm khẳng định hoặc kỳ vọng (assertion or expectation). Mocha không có thư viện khẳng định (assertion library) được xây dựng sẵn (có thể dùng một số thư viện hỗ trợ thêm như: Chai, should.js, expect.js,…), trong khi đó Jasmine thì hầu như có mọi thứ được xây dựng sẵn.



* + 1. Kiểm tra đôi (test doubles)

Test doubles thường được so sánh với đôi đóng thế (stunt doubles), tức là khi thay thế một object bằng một object khác dùng cho mục đích kiểm thử, tương tự như cách các diễn viên được thay thế bởi diễn viên đóng thế trong các cảnh hành động nguy hiểm.

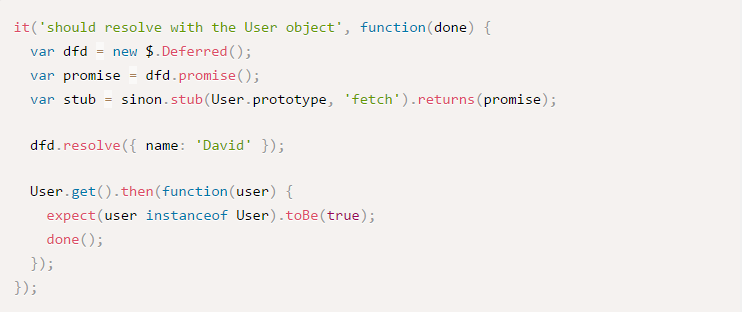
Trong Jasmine, test doubles có hình thái của *spy*. Một spy là một hàm thay thế một hàm cụ thể mà người lập trình muốn điều khiển hành vi của nó trong một test và ghi lại cách mà hàm đó được sử dụng trong suốt quá trình thực thi của test đó.

Ngược lại đối với Mocha, nó không có thư viện cho test double. Thay vào đó người lập trình cần dùng thêm *Sinon*.

Spy trong Jasmine hầu như làm được mọi thứ cần thiết cho test doubles, do đó người lập trình có thể không cần sử dụng Sinon nếu đang dùng Jasmine, nhưng Jasmine và Sinon có thể được sử dụng chung với nhau nếu người lập trình cần.

* + 1. Kiểm thử bất đồng bộ

Kiểm thử bất đồng bộ trong Jasmine phiên bản 2.x và Mocha là giống nhau.



Ở ví dụ trên, *User* là một constructor có hàm tĩnh *get*. Hàm *get* sử dụng *fetch* để thực hiện gửi yêu cầu XHR (bất đồng bộ). Ở đây người lập trình muốn kiểm tra khi hàm *get* thực thi xong, thì giá trị trả về là một instance của *User*.

* + 1. Server giả Sinon (Fake Server)

Một tính năng có trong Sinon nhưng không có trong Jasmine đó là fake server. Điều này cho phép người lập trình có thể thiết lập những phản hồi giả cho những yêu cầu AJAX.

* + 1. Chạy các test

Trong Mocha, người lập trình có thể chạy các test bằng cách viết dòng lệnh sau trong command line:



Trong Jasmine, người lập trình không có tiện ích giống như Mocha. Thay vào đó, người lập trình phải sử dụng những trình chạy test (test runner), và một test runner rất phổ biến đó là Karma, sẽ được trình bày kĩ ở phần tiếp theo.

Tóm lại, Jasmine có hầu hết mọi thứ được xây dựng sẵn bao gồm các thư viện kỳ vọng và các tiện ích test double. Tuy nhiên, Jasmine không có một test runner vì vậy người lập trình cần một công cụ như Karma để làm việc này.

1. Karma
   1. Giới thiệu

Karma là một môi trường để chạy test (test runner) cho Javascript chạy trên nền Node.js. Karma rất thích hợp để kiểm thử AngularJS hoặc bất kì các dự án Javascript khác. Sử dụng Karma để chạy các test thì người lập trình sẽ dùng một trong những các bộ mô tả kiểm thử Javascript phổ biến như Jasmine, Mocha, Qunit, … và những test đó được thực thi không chỉ trong những trình duyệt khác nhau do người lập trình chọn (Chrome, PhantomJS, Safari,…), mà còn có thể là trên các thiết bị (platform) họ chọn (máy tính để bàn, điện thoại, máy tính bảng). Ở Karma, người lập trình dễ dàng tùy chỉnh, tích hợp với các dịch vụ tích hợp liên tục phổ biến (continuous intergration packages) như Travis CI, Jenkins, Semaphore và có sự hỗ trợ mạnh mẽ về các plugin.

Nhìn chung, Karma có các tính năng tương tự như Webpack: preprocessor, plugin, loader,… Sức mạnh của Karma nằm ở khả năng chạy các test trên các trình duyệt khác nhau.

* 1. Hoạt động và tính năng

Karma sẽ tạo ra một server giả, và sau đó chạy các test ở nhiều loại trình duyệt khác nhau do người dùng chỉ định sử dụng dữ liệu từ server giả đó. Do Karma chỉ là môi trường nên nó cần một framework như Jasmine, Mocha để chạy test. Kết quả của mỗi test trên mỗi trình duyệt sẽ được xem xét và hiển thị thông qua command line để giúp người lập trình có thể nhìn thấy trình duyệt nào và test nào thành công hay thất bại.

Karma có thể làm những việc sau:

* Tạo ra web server
* Thực thi lại các test đối với mỗi trình duyệt
* Hiển thị kết quả của mỗi test trong console

1. PhantomJS
   1. Giới thiệu

PhantomJS là một trình duyệt không có giao diện người dùng nhưng lại cung cấp Javascript API (headless webkit scriptable), làm cho trình duyệt trở nên hữu dụng. Nó có sự hỗ trợ nhanh chóng và thuần túy cho nhiều chuẩn web khác nhau như: xử lí DOM, bộ chọn CSS (CSS selector), JSON, Canvas, và SVG.

* 1. Tính năng

PhantomJS có các tính năng sau:

* Chụp ảnh màn hình (screen capture): vì PhantomJS sử dụng WebKit, một bố cục thực (real layout) và phần mềm dựng hình (rendering engine), do đó nó có thể chụp hình một trang web. Bởi vì PhantomJS có thể vẽ mọi thứ trên trang web, nên nó có thể được sử dụng để chuyển đổi nội dung không chỉ trong HTML và CSS, mà còn SVG và Canvas
* Kiểm thử (testing): đây là một tính năng phổ biến của PhantomJS. Nó phù hợp cho việc kiểm thử dựa trên command line, và là một phần của tích hợp liên tục. PhantomJS được xem như là công cụ phổ biến để chạy các unit test. Những công cụ như Mocha, Casper rất thích hợp khi kết hợp với PhantomJS, vì những công cụ này dựa trên nó.
* Giám sát mạng (network monitoring): bởi vì PhantomJS cho phép kiểm tra lưu lượng mạng, nên nó phù hợp để xây dựng nên các bản phân tích về hành vi và hiệu suất mạng.
* Giả lập trang (page automation): do PhantomJS có thể tải và thao tác một trang web, nên nó rất phù hợp để giả lập nên nhiều trang khác nhau.
* Giao tiếp giữa các quá trình (inter process communication): hỗ trợ việc giao tiếp giữa PhantomJS và các quá trình khác (I/O, HTTP,…)
  1. Ưu và nhược điểm

Chúng ta sẽ so sánh với một trình duyệt cũng rất phổ biến trong việc hỗ trợ kiểm thử, đó là jsdom, để tìm ra điểm mạnh cũng như điểm yếu của PhantomJS:

* Như đã giới thiệu ở trên thì PhantomJS mặc dù không hỗ trợ giao diện người dùng, nhưng đây là một trình duyệt hoàn chỉnh (mặc dù nó dùng kĩ thuật dựng hình (rendering engine) rất cũ và hiếm). jsdom thì không phải là một trình duyệt hoàn chỉnh, nó không thể bố cục và dựng hình, và nó cũng không hỗ trợ điều hướng trang. Nó chỉ hỗ trợ DOM, HTML, canvas, nhiều web API khác (*không hỗ trợ SVG*), và chạy kịch bản (script).
* Người lập trình cần một trình thực thi (excutable) để chạy PhantomJS, mã của nó được viết thuần túy (native), và phải được biên dịch qua mỗi nền tảng. mã jsdom hoàn toàn là Javascript, nó chạy mỗi khi Node.js chạy.
* Chi phí (overhead) để tạo trang trong PhantomJS là cao, tốn nhiều thời gian, do đó việc chạy ít test thì mất vài phút trong PhantomJS, trong khi jsdom chỉ mất vài giây.
* Nếu tạo một kịch bản (script) để kiểm thử trong PhantomJS, thì kịch bản đó sẽ chạy ở một quá trình khác chứ không phải là ở ứng dụng web. Vì vậy khi thực hiện nhiều bước trong test và các bước này phụ thuộc lẫn nhau, khi đó ứng dụng có thể thay đổi DOM trong những bước này. Đối với jsdom thì ngược lại, các test sẽ nằm trong cùng luồng (thread) với ứng dụng web, do đó nếu test đang thực thi mã Javascript, thì ứng dụng web không thể chạy mã của nó cho tới khi test đó được thực thi xong.
  1. Ứng dụng

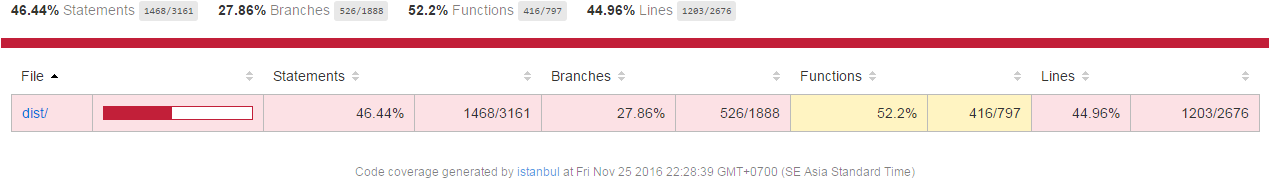
Hiện nay thì xu hướng sử dụng Mocha/Chai/jsdom ngày càng phổ biến, hiệu suất tốt hơn so với Jasmine/Karma/PhantomJS rất nhiều, đặc biệt là đối với những dự án lớn. Đã có nhiều dự án lớn đi theo mô hình TDD, bắt đầu với Jasmine/Karma/PhantomJS, tuy nhiên sau một thời gian, người ta đã nhận thấy món nợ kĩ thuật (technical debt) trở nên ngày càng lớn do tốc độ chạy của bộ ba này quá chậm khi dự án ngày một lớn hơn, khi đó Mocha/Chai/jsdom là một lựa chọn tốt hơn để giải quyết vấn đề và ngày càng được ưa chuộng.

C9js ban đầu cũng đã sử dụng bộ Mocha/Chai/jsdom để thực hiện việc kiểm thử. Tuy nhiên vì một lí do lớn nhất C9js đã không sử dụng, chính là jsdom không hỗ trợ SVG. Mà đây là thành phần chính trong các biểu đồ, do D3 dựng lên (render). Trong khi PhantomJS có hỗ trợ đầy đủ các chuẩn web, thuận lợi cho việc kiểm thử các thành phần khác nhau.

1. Istanbul
   1. Giới thiệu

Là công cụ dùng để đo lường mức độ mà mã nguồn (source code) của thư viện, ứng dụng Javascript đã được kiểm thử thông qua các bộ mô tả kiểm thử (test suite), hay còn gọi với một thuật ngữ là *code coverage*. Istanbul được viết bằng Javascript.

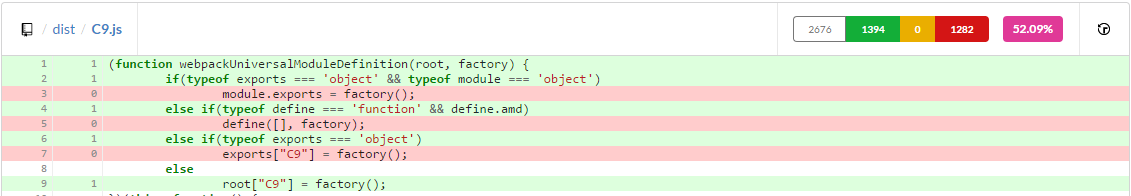
Istanbul sẽ đo đạc số lượng các câu lệnh (statement), nhánh (branch), hàm (function), dòng lệnh (line) đã chạy bằng những test mà người lập trình đã viết nên.



* 1. Tính năng

Các tính năng của Istanbul:

* Theo dõi sự bao phủ (coverage) các câu lệnh, nhánh, và hàm của thư viện Javascript
* Module loader sẽ móc nối với các đoạn mã đã được đo đạc trên các kênh (github, codecov,…)



* Chạy độc lập với trình chạy test (test runner) khi thực hiện kiểm thử đơn vị
* Có thể báo cáo kết quả đo đạc qua nhiều định dạng: HTML, LCOV, Cobertuna, …
* Có khả năng sử dụng những công cụ trung gian (middleware) để kiểm thử tập tin Javascript trên trình duyệt
* Có thể được dùng trên command line như là một thư viện (viết dòng lệnh trong command line với *istanbul* đứng đầu)
* Kiểm thử rất tốt trên node và trình duyệt
  1. Ứng dụng

Istanbul hỗ trợ *bao phủ* mã Javascript trong nhiều trường hợp bao gồm kiểm thử đơn vị và kiểm thử chức năng (functional test) ở phía server.

C9js áp dụng Istanbul để đo lường mức bao phủ của các câu lệnh, nhánh, hàm trong khi thực hiện kiểm thử đơn vị, đảm bảo tính ổn định và độ tin cậy cho mã nguồn.