

ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÁO CÁO LUẬN VĂN TỐT NGHIỆP

Trực quan dữ liệu với thư viện C9js

Hội đồng:

GVHD: TS. Lương Thế Nhân

GVHD: TS. Huỳnh Tường Nguyên

GVPB: ThS. Võ Thanh Hùng

Sinh viên thực hiện:

Phạm Thành Công - 51200399

Đỗ Đăng Thanh Huy - 51201337

Thành phố Hồ Chí Minh, 11/2016

Mục lục

Danh sách hình vẽ	iv
Danh sách bảng	vi
Danh sách mã nguồn	vii
1 Giới thiệu đề tài	1
1.1 Bối cảnh	1
1.1.1 Trực quan hoá dữ liệu	1
1.1.2 Ứng dụng trên nền tảng Web	2
1.2 Lí do chọn đề tài	3
1.3 Mục tiêu đề tài	4
2 Kiến thức nền tảng và Công nghệ	6
2.1 Công trình liên quan	6
2.1.1 Công cụ hỗ trợ trực quan dữ liệu phân tán Gapminder	6
2.1.2 Global Health Atlas - WHO	8
2.1.3 Thư viện đồ thị C3.js	11
2.1.4 Ứng dụng tạo bản đồ online Mapme	14
2.2 Kiến thức nền tảng	18
2.2.1 JavaScript	18
2.2.2 D3.js	20
2.2.3 OpenLayers	21
2.2.4 Webpack	23
2.2.5 Unit Testing	25
2.2.5.1 Jasmine	25
2.2.5.2 Karma	29
2.2.5.3 PhantomJS	30
2.2.5.4 Istanbul	33
2.2.6 Code Coverage	34

2.2.7	Semantic Versioning	37
2.2.8	CI-CD	38
2.2.8.1	Tích hợp liên tục (Continuous Integration)	39
2.2.8.2	Chuyển giao liên tục (Continuous Delivery)	40
2.2.8.3	Travis CI	41
3	Thư viện C9js	44
3.1	Mô tả bài toán	44
3.2	Hướng dẫn sử dụng	45
3.2.1	Cài đặt (Setup)	46
3.2.2	Tạo biểu đồ và bản đồ với C9js	48
3.2.3	Tùy biến định dạng dữ liệu đầu vào	52
3.2.4	Tùy biến kiểu đồ thị	53
3.2.5	Cập nhật đồ thị theo dữ liệu	56
3.3	Các ví dụ minh họa	57
3.4	Các hàm chức năng (API)	61
4	Hiện thực thư viện	64
4.1	Kiến trúc xây dựng	64
4.2	Hiện thực thư viện	70
4.2.1	Hiện thực (Coding)	70
4.2.2	Kiểm thử (Testing)	71
4.2.3	Triển khai (Release)	74
5	Kết luận (7)	76
5.1	Đánh giá (Unit Testing vs So sánh với các CTLQ) (5)	76
5.2	Hướng phát triển (1)	76
5.3	Kết luận (1)	76
	Tài liệu tham khảo	77

Danh sách hình vẽ

Hình 1.1:	Thông số đánh giá nhận được từ GitHub	5
Hình 2.1:	Đồ thị trực quan dữ liệu Gapminder World	7
Hình 2.2:	Biểu đồ tương tác về mật độ bác sĩ của WHO	9
Hình 2.3:	Tương tác khi click vào một hàng trong bảng dữ liệu	10
Hình 2.4:	Tương tác khi hover lên Map	11
Hình 2.5:	Thư viện đồ thị C3js	12
Hình 2.6:	Thư viện đồ thị C3js	12
Hình 2.7:	Thư viện đồ thị C3js	13
Hình 2.8:	Thư viện đồ thị C3js	14
Hình 2.9:	Thư viện đồ thị C3js	15
Hình 2.10:	Ứng dụng tạo bản đồ online Mapme	15
Hình 2.11:	Ứng dụng tạo bản đồ online Mapme	16
Hình 2.12:	Ứng dụng tạo bản đồ online Mapme	17
Hình 2.13:	Theo dõi thay đổi mã nguồn với Webpack	25
Hình 2.14:	Kết quả Unit Testing của C9js được Istanbul tổng quan	33
Hình 2.15:	Trạng thái các dòng lệnh C9js được cover trên GitHub, báo cáo bởi Istanbul	34
Hình 2.16:	Huy hiệu coverage được thêm vào repository C9js	35
Hình 2.17:	Triển khai trên <i>GitHub</i> với <i>Semantic Release</i>	38
Hình 2.18:	Triển khai trên <i>npm</i> với <i>Semantic Release</i>	39
Hình 2.19:	Sự kết hợp giữa tích hợp liên tục (CI) và phát triển theo hướng kiểm thử (TDD)	42
Hình 2.20:	Chạy kiểm thử đơn vị thư viện C9js bởi Travis CI	43
Hình 3.1:	Các tính năng chính của C9js trên trang chủ c9js.me	45
Hình 3.2:	Menu hướng dẫn trên trang chủ c9js.me	45
Hình 3.3:	Trang hướng dẫn trên Website c9js.me	46
Hình 3.4:	Đồ thị cột từ C9js	49
Hình 3.5:	Bản đồ từ C9js	50

Hình 3.6:	Thêm dữ liệu vào bản đồ với C9js	51
Hình 3.7:	Một ví dụ về thay đổi <i>style</i> với C9js	55
Hình 3.8:	Các ví dụ được phân loại theo kiểu đồ thị	58
Hình 3.9:	Cấu trúc một trang ví dụ trên trang chủ C9js	59
Hình 3.10:	Ví dụ mở rộng tái hiện ứng dụng GHA-WHO	60
Hình 4.1:	Kiến trúc các thành phần của Chart trong C9js	64
Hình 4.2:	Luồng thực thi một đối tượng trong Chart của C9js	66
Hình 4.3:	Mô hình chuẩn hoá dữ liệu bởi <i>Data Adapter</i> trong C9js	68
Hình 4.4:	Quá trình nhận dữ liệu được chuẩn hoá trong C9js	69
Hình 4.5:	Ba giai đoạn hiện thực thư viện C9js	70
Hình 4.6:	Khởi chạy môi trường phát triển với Webpack	71
Hình 4.7:	Kiểm soát Unit Testing với Karma	73

Danh sách bảng

Bảng 2.1:	So sánh PhantomJS và jsdom	32
Bảng 4.1:	Các trường trong một tập dữ liệu đã chuẩn hoá	68

Danh sách mã nguồn

Mã nguồn 2.1: Cấu trúc của một test-suite trong Jasmine	27
Mã nguồn 2.2: Ví dụ sử dụng expect trong Jasmine	27
Mã nguồn 2.3: Ví dụ sử dụng Jasmine để kiểm thử bất đồng bộ	28
Mã nguồn 2.4: Khởi chạy trình kiểm thử testcase với Mocha	29
Mã nguồn 3.1: Tải mã nguồn thông qua npmcdn	47
Mã nguồn 3.2: Tải mã nguồn thông qua unpkg	47
Mã nguồn 3.3: Tải mã nguồn <i>d3.js</i> trước tiên	47
Mã nguồn 3.4: Tải mã nguồn <i>OpenLayers 3</i> nếu muốn sử dụng chức năng Bản đồ	47
Mã nguồn 3.5: Tạo <i>container</i> để chứa biểu đồ	48
Mã nguồn 3.6: Khởi tạo biểu đồ với C9js	48
Mã nguồn 3.7: Tạo <i>container</i> để chứa bản đồ	49
Mã nguồn 3.8: Khởi tạo bản đồ với C9js	49
Mã nguồn 3.9: Thêm dữ liệu vào Bản đồ với C9js	50
Mã nguồn 3.10: Một dữ liệu mẫu với các thuộc tính lồng nhiều cấp	52
Mã nguồn 3.11: Định nghĩa định dạng dữ liệu đầu vào với C9js	52
Mã nguồn 3.12: Thay đổi <i>style</i> tại thời điểm khởi tạo Chart	53
Mã nguồn 3.13: Thay đổi <i>style</i> bằng CSS Selector	54
Mã nguồn 3.14: Thay đổi <i>style</i> với <i>setOption</i>	56
Mã nguồn 3.15: Cập nhật lại biểu đồ với <i>updateData</i>	56
Mã nguồn 3.16: Định nghĩa lại khoá-thuộc tính trong lúc cập nhật	57
Mã nguồn 4.1: Một testcase trong C9js được viết bằng Jasmine	71
Mã nguồn 4.2: Cấu hình TravisCI kiểm soát cả 3 giai đoạn	74

1 Giới thiệu đề tài

1.1 Bối cảnh

1.1.1 Trực quan hoá dữ liệu

Khái niệm

Trực quan dữ liệu (Data visualization), được xem như hướng đi trong tương lai của việc giao tiếp và truyền thông thông qua hình ảnh. Nó liên quan đến các công trình và nghiên cứu nhằm biểu diễn dữ liệu một cách trực quan, hay nói cách khác, được hiểu là "thông tin được trực quan hoá dưới dạng các lược đồ, bao gồm các thuộc tính và thông số đại diện cho các đơn vị thông tin^[?]]".

Lịch sử hình thành

Bắt đầu từ thế kỷ thứ 2 với việc sắp xếp dữ liệu vào các cột và các hàng, phát triển thành biểu diễn dưới dạng các đơn vị định lượng ban đầu trong thế kỷ 17^[?]]. Theo Hiệp hội Thiết kế Tương tác (The Interaction Design Foundation), nhà triết học và toán học người Pháp René Descartes đã đưa ra những nghiên cứu nền tảng đầu tiên làm bước đệm cho một kỹ sư người Scotland tên là William Playfair. Descartes đã phát triển một hệ trục tọa độ hai chiều để hiển thị các giá trị, vào những năm cuối thế kỷ 18, Playfair đã nhìn thấy tiềm năng trong việc thể hiện đồ họa của dữ liệu định lượng^[?]].

Trong nửa sau của thế kỷ 20, Jacques Bertin sử dụng đồ thị định lượng để đại diện cho thông tin "trực giác, rõ ràng, chính xác và hiệu quả".^[?]] John Tukey và đáng chú ý hơn là Edward Tufte đã thúc đẩy các giới hạn của trực quan dữ liệu. Tukey đưa ra cách tiếp cận mới của ông theo hướng thống kê: Phân tích dữ liệu thăm dò, và Tufte đã xuất bản cuốn sách "The Visual Display of Quantitative Information". Các đóng góp trên đã mở ra con đường rõ ràng hơn cho các công nghệ trực quan.

Tiềm năng của Trực quan dữ liệu

Ngày nay trong lĩnh vực kinh doanh hiệu quả (Business Intelligence) không có gì có thể mang chúng ta lại gần hơn với những tương tác thông minh hơn là Trực quan dữ liệu. Nhưng điều này chỉ xảy ra khi chúng ta thực sự hiểu và dùng nó một cách đúng đắn. Để đạt được điều đó, chúng ta phải thực sự hành động và vứt bỏ những quan niệm chưa đúng về Trực quan dữ liệu.

Trực quan dữ liệu ngày càng đóng vai trò quan trọng trong mọi lĩnh vực đời sống. Như sử dụng trong nghiên cứu, trong những công việc liên quan tới xử lý dữ liệu, giao dịch bởi người dùng phổ thông, và được áp dụng với tỷ lệ tăng cao trong lực lượng lao động trí óc, đặc biệt đối với các nhà phân tích.

Với sự phát triển của công nghệ cùng với sự tiến triển của dữ liệu trực quan; bắt đầu với hình ảnh vẽ tay và chuyển biến thành các ứng dụng kỹ thuật - bao gồm thiết kế tương tác cho đến các phần mềm.^[?] Các chương trình như SAS, SOFA, R, Minitab, và nhiều hơn nữa khởi đầu cho việc trực quan dữ liệu trong lĩnh vực thống kê. Các ứng dụng khác, tập trung nhiều hơn vào cá nhân hoá, các ngôn ngữ lập trình như D3, Python và JavaScript giúp cho việc hiện thực trực quan khả thi và dễ dàng hơn.

1.1.2 Ứng dụng trên nền tảng Web

Sự tăng trưởng trong khối lượng dữ liệu trong các dự án với nhu cầu chia sẻ mang tính toàn cầu ngày càng gia tăng, đòi hỏi sự đáp ứng bởi các công nghệ trên nền tảng Web để phân tích, xử lý, tương tác và hiển thị dữ liệu ngày càng lớn.

Các công nghệ trên nền Web hiện đại như HTML5, CSS3 và các thư viện JavaScript mạnh mẽ mang đến những triển vọng về trực quan trên các trình duyệt hiện tại. Hiện tại đã có hàng chục, thậm chí hàng trăm các loại thư viện, framework và ứng dụng phục vụ cho nhu cầu trực quan tùy từng mục đích cụ thể của cá nhân, doanh nghiệp, các công trình nghiên cứu,...

1.2 Lí do chọn đề tài

Đam mê trong các công nghệ mới trên nền tảng Web

Ngày nay,^{[?]1} Web ngày càng phát triển và trở thành một thế lực riêng, một đế chế riêng bao gồm các *Web pages* và *Web apps*, xây dựng nên từ các video, hình ảnh hay các nội dung tương tác. Điều mà người dùng phổ thông không để ý đến đó là các công nghệ Web và các trình duyệt đã đóng góp rất lớn trong quá trình trên.

Qua thời gian, các công nghệ Web tiến hoá và ngày càng đem lại nhiều trải nghiệm mới lạ cho các lập trình viên, nhằm đáp ứng khả năng sáng tạo và tạo mọi điều kiện để họ thể hiện khả năng thông qua nó.

Nhóm đề tài gồm các thành viên từng làm việc với nhiều dự án liên quan đến Web. Đặc biệt đều có hứng thú và đam mê với khả năng của các xu hướng phát triển hiện tại, cũng như muốn khám phá khả năng bản thân trong thế giới luôn luôn đổi mới không ngừng của Web, đó là một trong những lí do nhóm chọn đề tài này.

Đóng góp vào sự phát triển của trực quan dữ liệu

^{[?]1} Có thể nói, với sự phát triển khoa học và công nghệ như vũ bão vào những năm cuối thập niên này, những sản phẩm công nghệ mới phát triển rầm rộ đã đem lại nhiều tiện ích cho cuộc sống, nó được ứng dụng rộng rãi ở nhiều lĩnh vực; đặc biệt là lĩnh vực thông tin truyền thông. Hệ thống thông tin điện tử, trực tuyến, các Website của các tổ chức, đơn vị, doanh nghiệp được phát triển mạnh mẽ, góp phần tăng cường mối quan hệ, giao lưu, hợp tác phát triển ở nhiều lĩnh vực, nhất là lĩnh vực kinh tế, văn hóa xã hội, khoa học công nghệ, y tế, giáo dục, giải trí, con người có trong tay nhiều công cụ để chia sẻ thông tin của mình qua blog, website, diễn đàn, qua các trang mạng xã hội như Facebook, Youtube, Twitter.

Trong thời đại bùng nổ thông tin hiện nay, được tiếp sức bởi các loại công nghệ tiên các phương tiện thông tin hiện đại đã giúp cho thế giới xích lại gần nhau một cách nhanh chóng hơn. Đặc biệt, công nghệ truyền thông Internet đang phát triển như vũ bão, tạo nên những siêu lộ thông tin có dung lượng lớn và tốc độ cao, chuyển tải mọi thông tin một cách nhanh

chóng.

Trong thời đại mà thông tin được coi như một loại tài sản, việc kết hợp trực quan dữ liệu với Web có thể được xem như một xu thế mới, một hướng đi đầy tiềm năng trong tương lai. Với mong muốn đóng góp vào quá trình này, đề tài *Trực quan dữ liệu với C9js* có thể xem như một viên gạch trong bức tường đang xây dựng của trực quan dữ liệu.

1.3 Mục tiêu đề tài

Công cụ hỗ trợ trực quan giúp rút ngắn thời gian cho các lập trình viên

Mang trực quan dữ liệu trở thành một ứng dụng trên nền Web vừa dễ vừa khó. Dễ ở việc các công nghệ Web rất đa dạng, phong phú, thậm chí là bao la khi đáp ứng các nhu cầu của trực quan dữ liệu. Nhưng đó cũng là điểm hạn chế, khi có quá nhiều thư viện, nền tảng, công nghệ lập trình viên cần nắm rõ và kết hợp nếu muốn hiện thực ý tưởng thành một ứng dụng mang tính thực tế trên Web.

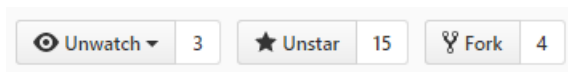
Bản thân nhóm cũng là các lập trình viên thích khám phá và nghiên cứu các công nghệ mới và phong phú trong thế giới Web. Mục tiêu thư viện C9js đề ra là giúp các lập trình viên khác khi sử dụng C9js sẽ cảm thấy dễ dàng nắm bắt, không đòi hỏi quá nhiều nỗ lực trong việc tìm hiểu hay xây dựng một ví dụ mẫu. Mặt khác, giảm thiểu tối đa thời gian cần thiết để đưa trực quan dữ liệu lên Web của mình.

Hướng đến cộng đồng mã nguồn mở

C9js hiện tại được triển khai là một mã nguồn mở (open-source) tại GitHub^[?]. Các công nghệ được sử dụng trong C9js hoàn toàn là mã nguồn mở, nên nhóm quyết định đóng góp vào cộng đồng một thư viện với khả năng mang trực quan dữ liệu trở thành một công cụ dễ dàng triển khai trên Web.

Về kiến trúc, cũng như tài liệu (documentation), C9js được thể hiện rất rõ ràng với mong muốn ngay từ ban đầu là nhận được sự đóng góp cũng như phát triển lâu dài từ các lập trình viên khác trong cộng đồng mã nguồn mở GitHub. Hiện tại, repository^[?] của C9js trên

GitHub đã nhận được sự quan tâm và đánh giá từ cộng đồng. Các thông số cụ thể (tính đến thời điểm bài báo cáo này) như hình 1.



Hình 1.1: Thông số đánh giá nhận được từ GitHub

2 Kiến thức nền tảng và Công nghệ

2.1 Công trình liên quan

2.1.1 Công cụ hỗ trợ trực quan dữ liệu phân tán Gapminder

Giới thiệu

Gapminder^[?] được thành lập tại Stockholm bởi Ola Rosling, Anna Rosling Rönnlund và Hans Rosling vào ngày 25 tháng 2 năm 2005.

Gapminder là một tổ chức phi lợi nhuận nhằm thúc đẩy sự phát triển bền vững toàn cầu, và là một thành tựu đánh dấu mục tiêu phát triển mang tầm thế kỷ của tổ chức Liên Hiệp Quốc (United Nations). Với mục tiêu đề ra nhằm gia tăng khả năng tìm hiểu, phân tích và đánh giá các thống kê, dữ liệu liên quan đến các vấn đề xã hội, tài chính và phát triển môi trường ở địa phương, quốc gia, và tầm thế giới.

Gapminder hoạt động dựa trên nền tảng được xác định trước, xây dựng các dịch vụ bởi ban quản trị, đôi khi hợp tác với các dự án tại các trường đại học, các tổ chức quốc tế, cơ quan Nhà nước cũng như các tổ chức phi chính phủ.

Mục tiêu

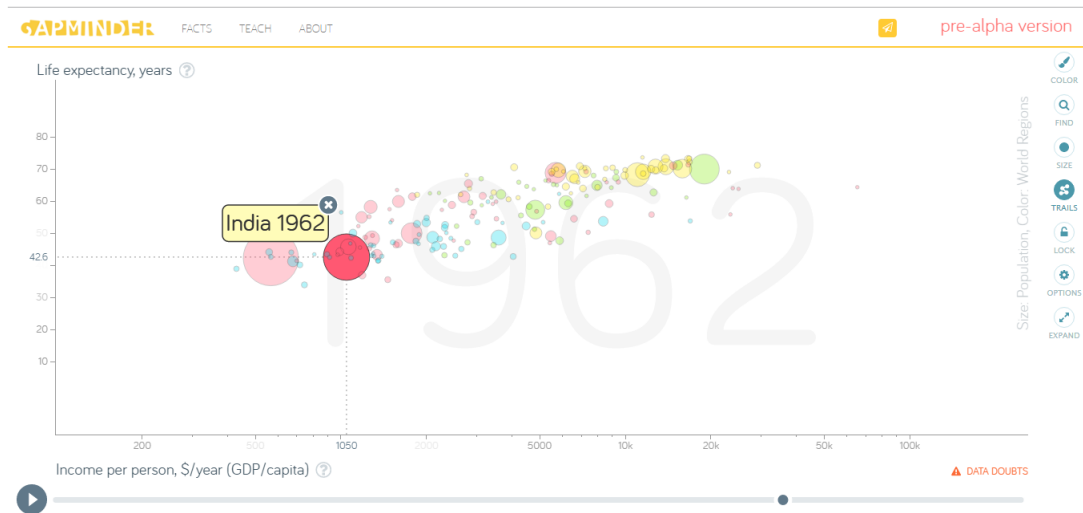
Các hoạt động ban đầu là theo đuổi sự phát triển của ứng dụng Trendalyzer (tháng 3 năm 2006). Trendalyzer là một phần mềm biểu diễn chuỗi thời gian thống kê bằng cách chuyển đổi những con số khô khan, nhàm chán thành các đồ thị chuyển động thú vị, và đặc biệt là khả năng tương tác. Trendalyzer còn được biết đến dưới tên gọi Gapminder World, xây dựng trên nền tảng web biểu diễn dữ liệu thống kê theo dòng thời gian cho tất cả các quốc gia trên thế giới.

Gapminder World^[?] cung cấp các video, bài thuyết trình bằng Flash và các đồ thị dưới dạng PDF thể hiện xu hướng phát triển mang tính toàn cầu.

Kể từ năm 2006 Gapminder cung cấp một lượng lớn các video thể hiện thực tế dưới góc

nhìn toàn cầu.

Từ năm 2008 Gapminder còn cung cấp các biểu đồ con (sub-graph)^[?] thể hiện xu hướng, sự khác biệt giữa các tiểu bang của Mỹ, các tỉnh thành của Trung Quốc, hay các đô thị đặc biệt so sánh số liệu thống kê từ các thành phố ở Trung Quốc, Mỹ, Ấn Độ với các quốc gia khác trên thế giới. Từ Gapminder World, chỉ bằng một cú click chuột, các dữ liệu, thống kê liên quan nằm sau các đô thị được thể hiện trên các biểu đồ con.



Hình 2.1: Đồ thị trực quan dữ liệu Gapminder World

Như ở hình 2, chúng ta có thể thấy với 1 lượng dữ liệu lớn (ở đây là độ tuổi kỳ vọng ở các quốc gia trên thế giới), có thể được thể hiện trực quan bởi đồ thị bong bóng (bubble chart). Qua kích thước các bong bóng, có thể thấy sự tương quan khác biệt giữa các quốc gia như thế nào. Hay khi trở vào 1 bong bóng, chúng ta thấy được số liệu chi tiết ở quốc gia đó, như trong hình là độ tuổi kỳ vọng ở Ấn Độ, vào năm 1962 là 42.6. Ngoài ra ở phía dưới đồ thị, có 1 thanh thể hiện dòng thời gian (timeline), khi thanh thời gian trượt đi, kích thước các quả bóng thay đổi theo như số liệu tương ứng ở các năm của các quốc gia.

Phương pháp

Gapminder sử dụng D3.js, một thư viện JavaScript phổ biến trong việc thao tác với các thành phần của một trang web (Document Object Model) dựa trên dữ liệu.

D3 cho phép thể hiện dữ liệu thông qua HTML, SVG và CSS. D3 còn là một tiêu chuẩn web cung cấp hầu hết các khả năng mà một trình duyệt hiện đại mang lại, mà không cần phải tự viết lại hay dựa trên một nền tảng khuôn khổ. Kết hợp các thành phần trực quan mạnh mẽ và hướng tiếp cận dữ liệu để thao tác trên DOM.

Đánh giá

Với mục tiêu luôn cập nhật các công cụ cũng như dữ liệu, nhằm đảm bảo tính xác thực, và đúng đắn của ứng dụng, Gapminder World đảm bảo nguồn dữ liệu chính xác nhất và cập nhật nhất. Dữ liệu đề tài sẽ sử dụng từ Gapminder để đảm bảo nguồn thông tin luôn xác thực và thể hiện đúng đắn nhất các xu hướng phát triển toàn cầu.

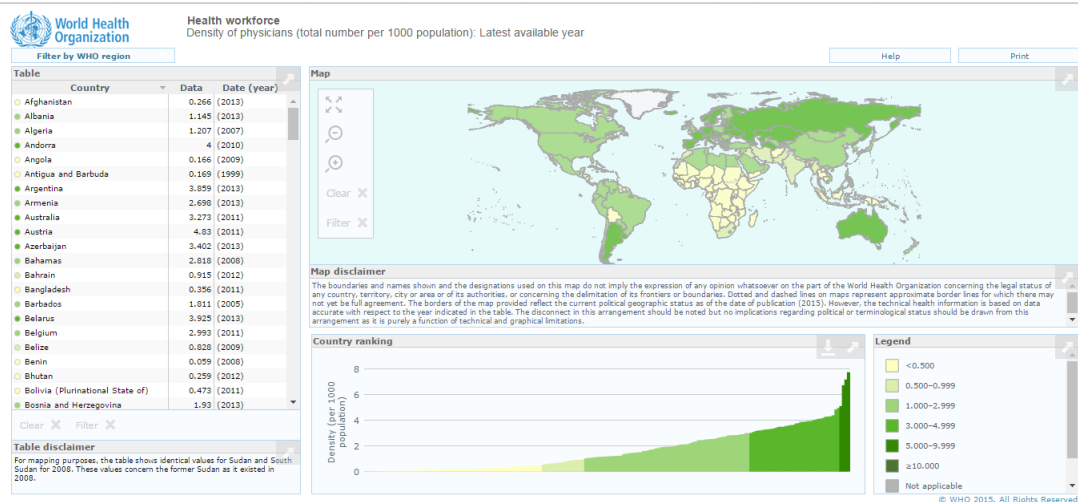
Ngoài ra, cách sử dụng các biểu đồ con để hiển thị dữ liệu liên quan của một đối tượng được tương tác giúp người dùng dễ dàng nắm bắt được chính xác các thông tin muốn truyền đạt, tăng tính thân thiện cũng như trải nghiệm người dùng. Áp dụng tính hiệu quả trên, đề tài đề xuất sử dụng cách thể hiện trải nghiệm người dùng (UI/UX) kế thừa các đặc tính của Gapminder World.

Về thư viện D3.js được sử dụng trong Gapminder World, đây là một thư viện mạnh mẽ tiếp cận dữ liệu dựa trên các thao tác với DOM. D3.js cho phép tái sử dụng các thành phần và trình nhúng (plugin) một cách đơn giản và hiệu quả. Đề tài đề xuất sử dụng D3.js làm thư viện chính để hiện thực thư viện tương tác giữa các đồ thị.

2.1.2 Global Health Atlas - WHO

Giới thiệu

Trong một môi trường điện tử, WHO's Communicable Disease Global Atlas được dùng cho việc phân tích và so sánh dữ liệu chuẩn, thống kê các bệnh truyền nhiễm ở các quốc gia, vùng lãnh thổ, và toàn thế giới. Việc phân tích và biểu diễn dữ liệu ngày càng được hỗ trợ tốt hơn thông qua các thông tin về dân số, điều kiện kinh tế xã hội, và các yếu tố môi trường. Với việc làm đó, biểu đồ Atlas này có thể cho biết được những yếu tố gây ra bệnh truyền nhiễm (2007)^[7].



Hình 2.2: Biểu đồ tương tác về mật độ bác sĩ của WHO

Mục tiêu

Ban đầu, với mục đích cung cấp những dữ liệu, báo cáo về các bệnh truyền nhiễm chính (sốt rét, HIV/AIDS,...), thì ngày nay Global Atlas của WHO cũng dùng để phân tích, biểu diễn, so sánh dữ liệu từ những vấn đề liên quan đến y tế khác như: mật độ bác sĩ, số lượng người chết do tai nạn giao thông, tuổi thọ trung bình,...

Phương pháp

Trực quan dữ liệu thành nhiều dạng biểu đồ khác nhau, các biểu đồ có thể tương tác với nhau giúp cho việc nhìn nhận, phân tích, đánh giá thông tin một cách chính xác, đầy đủ.

Ứng dụng^[?] bao gồm các thành phần:

- *Bảng dữ liệu*: Được lấy từ database của WHO.
- *Bản đồ (Map)*: Dùng thư viện Google Maps, kết hợp với bảng dữ liệu để tạo ra biểu đồ.
- *Biểu đồ cột*: Dựa vào bảng dữ liệu để tạo ra biểu đồ.
- *Chú thích*: Phân vùng dữ liệu.

Tính năng: Ứng dụng hỗ trợ tương tác giữa các biểu đồ, bảng dữ liệu

- Khi click vào một hàng trong bảng dữ liệu (hình 4), bên map sẽ phóng to đến vùng tương ứng, bên biểu đồ cột sẽ nổi bật cột tương ứng. Các tương tác sẽ được thể hiện tương tự khi click vào các biểu đồ.



Hình 2.3: Tương tác khi click vào một hàng trong bảng dữ liệu

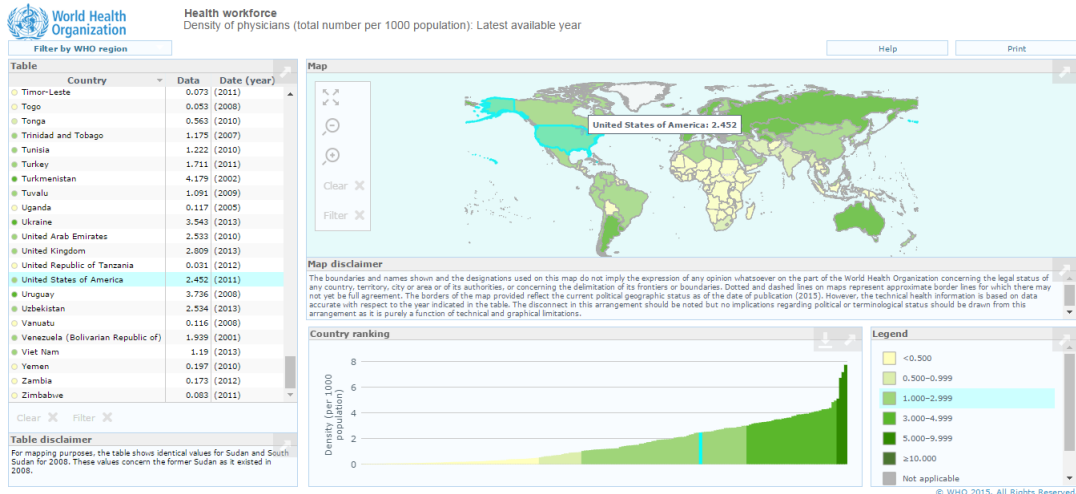
- Khi hover lên một hàng trong bảng dữ liệu (hình 5), bên các biểu đồ cũng sẽ được làm nổi bật vùng tương ứng như khi click, nhưng với 1 màu sắc khác (bên map sẽ không phóng to đến vùng tương ứng).

Đánh giá

Dữ liệu được WHO thu thập trên toàn thế giới, cập nhật theo thời gian, đảm bảo tính chính xác của ứng dụng.

Bên cạnh đó, việc trực quan thành nhiều loại biểu đồ, thể hiện được nhiều cái nhìn khác nhau của dữ liệu giúp đánh giá vấn đề 1 cách đúng đắn, đầy đủ.

Một tính năng khác đó là việc tương tác giữa các đồ thị giúp người dùng có nhiều cái nhìn từ nguồn dữ liệu, hiểu được thông tin rút trích từ các đồ thị, tăng tính thân thiện,



Hình 2.4: Tương tác khi hover lên Map

trải nghiệm của người dùng.

Do đó, đề tài đề xuất sử dụng nhiều đồ thị để biểu diễn cho nguồn dữ liệu đầu vào, và sự tương tác giữa các đồ thị.

2.1.3 Thư viện đồ thị C3.js

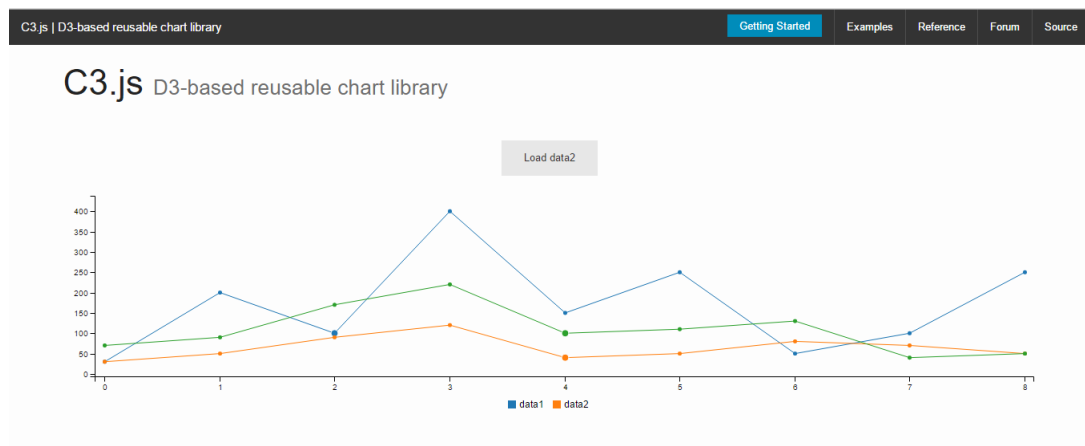
Giới thiệu

C3.js là một thư viện mã nguồn mở^[7] dựa trên thư viện D3.js, nhằm cho phép người dùng có cái nhìn sâu hơn về các ứng dụng web tích hợp đồ thị.

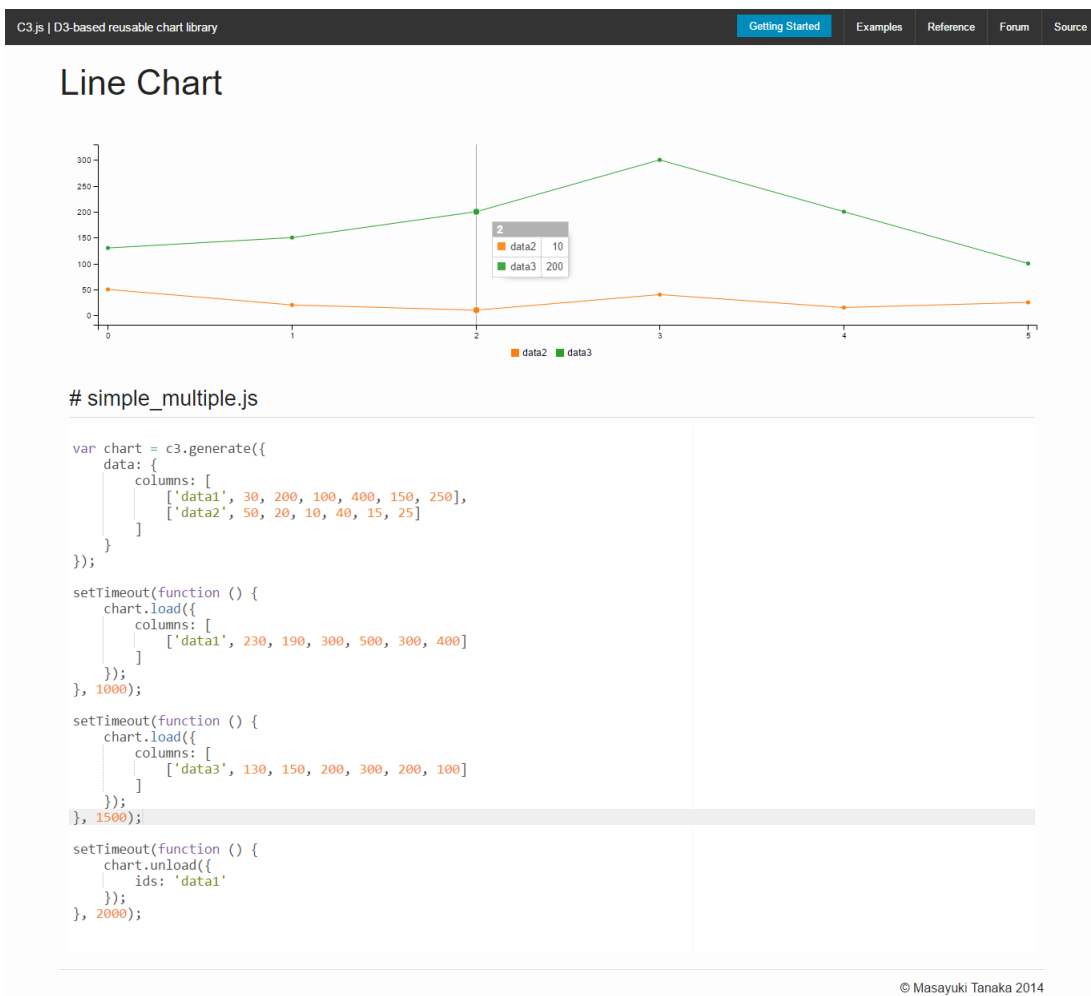
Toàn bộ mã nguồn của thư viện nằm ở: <https://github.com/masayuki0812/c3>

Mục tiêu

- *Sự thoải mái (Comfortable)*: C3 giúp người dùng có thể dễ dàng tạo một đồ thị dựa trên D3 bằng cách gộp chung các đoạn code cần thiết để cấu trúc một đồ thị hoàn chỉnh. Qua đó, người dùng không cần phải viết các đoạn code D3 rườm rà như trước nữa.



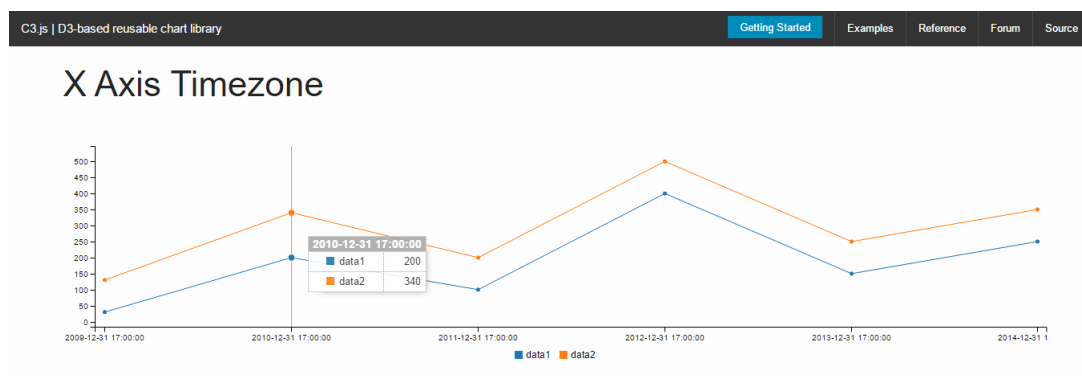
Hình 2.5: Thư viện đồ thị C3js



Hình 2.6: Thư viện đồ thị C3js

[Hình 6] Thông qua một số câu lệnh định nghĩa dữ liệu, C3 khởi tạo một đồ thị cho phép tương tác một cách trực quan thông qua hover, click,... các thành phần của đồ thị.

- *Khả năng tùy chỉnh (Customizable)*: C3 cung cấp các class tương ứng cho mỗi đối tượng trong đồ thị khi khởi tạo. Qua đó, người dùng có thể định nghĩa các style theo ý muốn theo class cũng như mở rộng cấu trúc các thành phần của đồ thị (trục dữ liệu, hình dạng hiển thị, kiểu đồ thị,..) một cách trực tiếp.



Hình 2.7: Thư viện đồ thị C3js

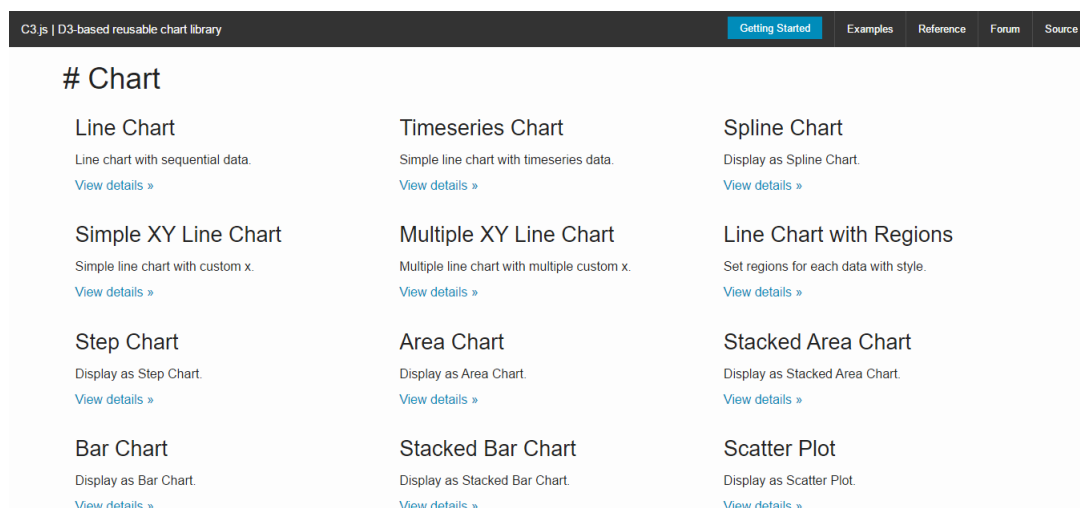
(Hình 7) Trục hoành của đồ thị được tùy chỉnh theo thời gian hiện tại với định dạng hoàn toàn tùy thuộc người sử dụng.

- *Khả năng kiểm soát (Controllable)*: C3 cung cấp đa dạng các APIs và các hàm callback để truy xuất trạng thái của đồ thị. Bằng cách đó, người dùng có thể cập nhật lại đồ thị thậm chí sau khi đồ thị đã được thể hiện. [Hình 8] Thể hiện một số thành phần được hỗ trợ bởi thư viện C3js.

Phương pháp

Như đã trình bày ở trên, C3js sử dụng thư viện D3js nhằm tạo các đồ thị có khả năng tái sử dụng, cũng như giúp người dùng đào sâu hơn vào khái niệm trực quan dữ liệu thông qua đồ thị trên nền ứng dụng web.

Ngoài ra dựa theo cấu trúc mã nguồn mà C3js chia sẻ, có thể thấy cấu trúc tổ chức của C3js



Hình 2.8: Thư viện đồ thị C3js

theo hướng hướng đối tượng, được tổ chức theo các class (trục, dữ liệu, đồ thị,..) và các phương thức, thuộc tính đi kèm class.

Đánh giá

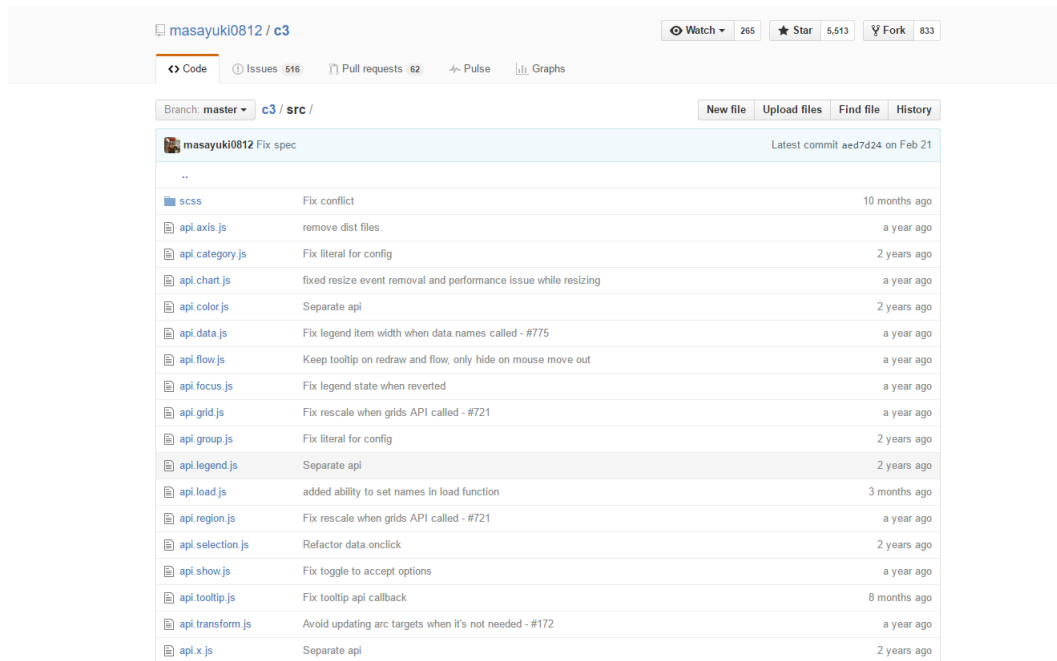
Cách tổ chức và hiện thực theo hướng hướng đối tượng như trên giúp tiết kiệm số dòng mã, tận dụng khả năng tái sử dụng, đồng thời giúp việc mở rộng về sau rất hiệu quả.

Đề tài đề xuất sử dụng cách hiện thực API theo hướng như thư viện C3js.

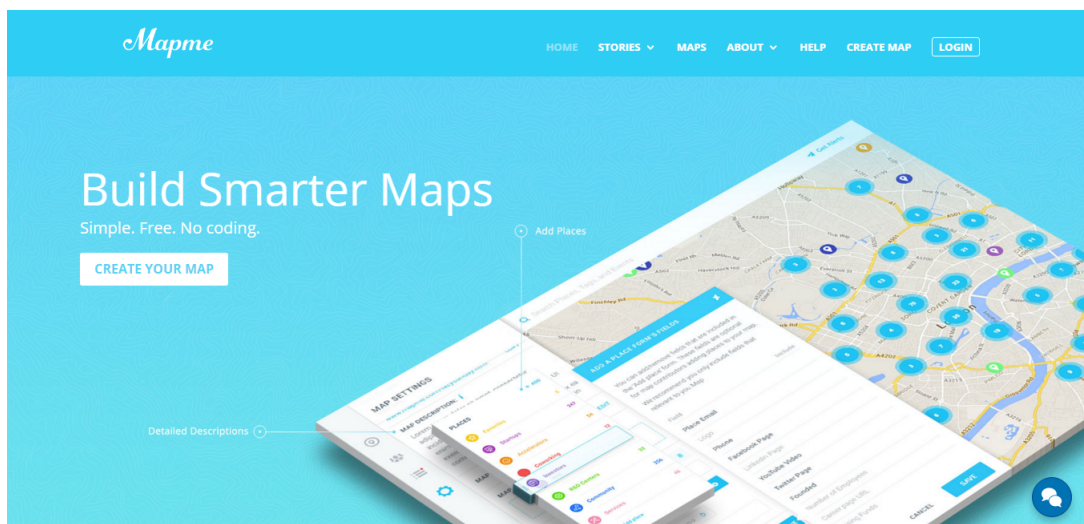
2.1.4 Ứng dụng tạo bản đồ online Mapme

Giới thiệu

Mapme^{[?]1} là một ứng dụng cho phép người dùng tạo bản đồ, dữ liệu được xử lý trực tiếp từ các file Excel đầu vào. Bản đồ tạo ra có các marker tương ứng với dữ liệu, có khả năng tương tác trực tiếp thông qua các sự kiện định nghĩa trước. Ngoài ra, bản đồ được tạo ra còn có thể được nhúng vào các trang web khác thông qua một đoạn mã nhúng. Đặc biệt toàn bộ tương tác từ những bước khởi tạo bản đồ đầu tiên, đến lúc xuất thành quả, đều không yêu cầu người dùng phải viết một dòng mã nào.



Hình 2.9: Thư viện đồ thị C3js



Hình 2.10: Ứng dụng tạo bản đồ online Mapme

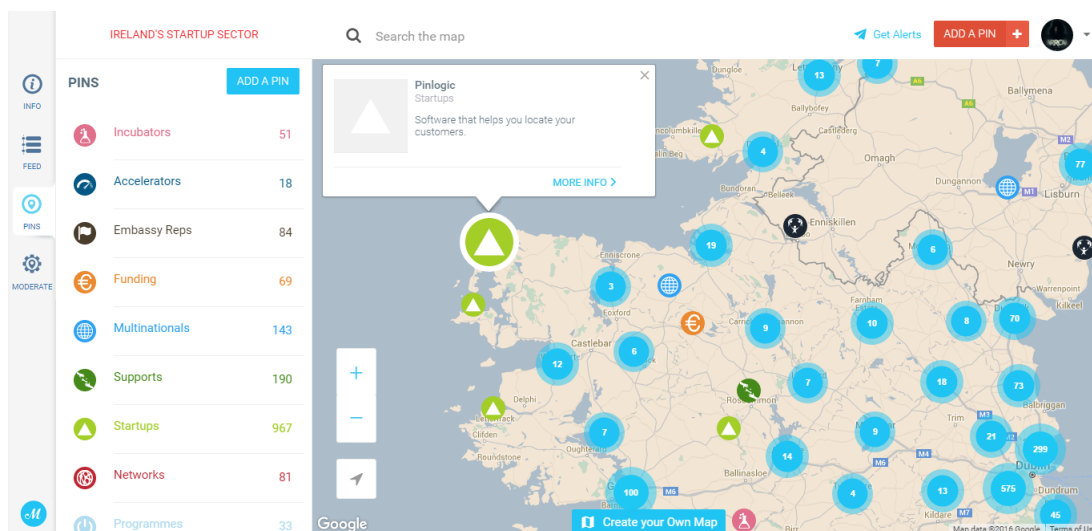
Mục tiêu

Đầu tiên nói đến các đối tượng mà Mapme hướng tới, đó là các tổ chức, chính phủ, các doanh nghiệp hay các tổ chức cộng đồng phi lợi nhuận.

Phục vụ trong các lĩnh vực khác nhau như giáo dục, các sự kiện, xuất bản hay truyền thụ cảm hứng trong đời sống hằng ngày.

Phương pháp

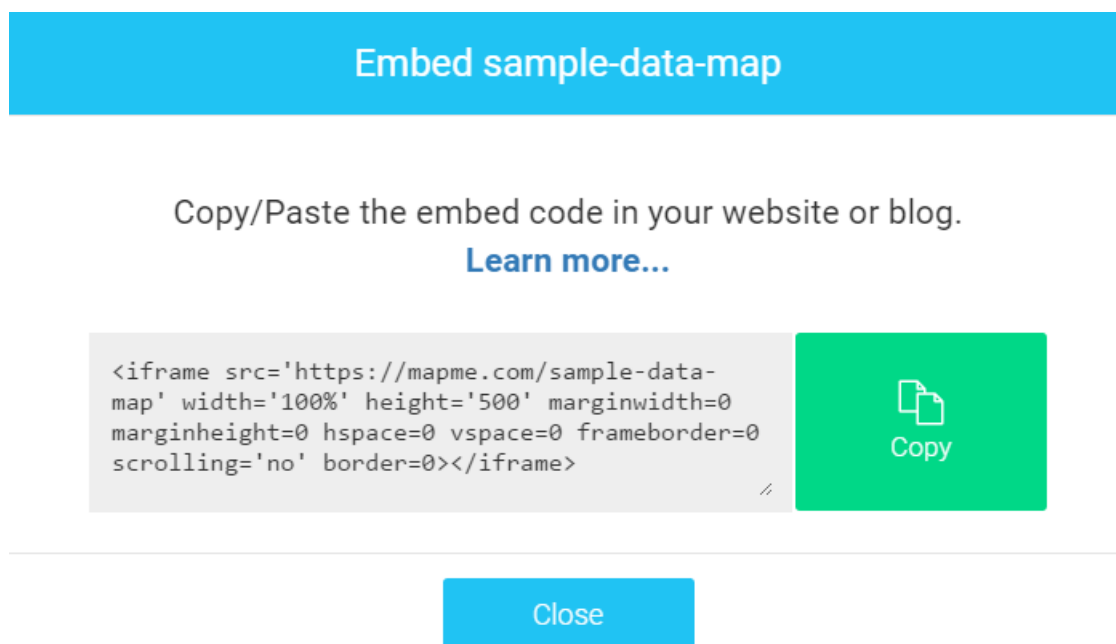
Sử dụng OpenStreetMap (viết tắt OSM)^[7], một dịch vụ bản đồ thế giới trực tuyến với nội dung mở, nhằm mục đích cung cấp dữ liệu địa lý do nhiều người cùng cộng tác với nhau trên hệ thống Wiki. Nó thường được gọi "Wikipedia của bản đồ". Dự án OSM được sáng lập năm 2004, chủ yếu để cạnh tranh với các công ty và cơ quan chính phủ cung cấp dữ liệu địa lý theo các điều khoản sử dụng được coi là quá chặt chẽ. OSM cho phép ứng dụng cập nhật dữ liệu địa lý trực tuyến một cách chính xác, luôn được cập nhật nhằm đảm bảo các dữ liệu luôn đi sát thực tế nhất.



Hình 2.11: Ứng dụng tạo bản đồ online Mapme

Như hình 11 có thể thấy, các thông tin về khởi nghiệp ở quốc gia Ireland được trực quan thông qua các biểu tượng (marker). Cùng với đó là các số liệu tương ứng được xử lý trực tiếp từ các file Excel đầu vào. Ngoài ra, người dùng còn có thể tương tác với các marker này

để thể hiện thông tin chi tiết ở các điểm chỉ định.



Hình 2.12: Ứng dụng tạo bản đồ online Mapme

Đặc biệt, Mapme còn hỗ trợ sinh mã nhúng [Hình 12], dưới dạng iframe, giúp người dùng nhúng trực tiếp bản đồ mình vừa tạo vào các website tùy mục đích.

Đánh giá

Đầu tiên là về cách sử dụng OpenStreetMap làm dịch vụ cung cấp bản đồ, đảm bảo ứng dụng luôn chính xác về các dữ liệu địa lý so với thực tế. Nhóm đề tài đề xuất sử dụng OpenStreetMap làm dịch vụ cung cấp bản đồ cho API mà nhóm đề tài thực hiện.

Ngoài ra, ứng dụng cho phép người dùng tương tác trực tiếp với các marker trên bản đồ, hiển thị các thông tin liên quan đến các địa điểm đó. Thay vì phải đọc các dữ liệu thô theo vùng miền, theo quốc gia, vùng lãnh thổ như trước, cách tương tác này giúp người dùng dễ dàng cập nhật thông tin cũng như có cái nhìn tổng quan, khả năng so sánh giữa các địa điểm khác nhau mà không phải chăm chú vào các con số nhàm chán như trước.

2.2 Kiến thức nền tảng

2.2.1 JavaScript

Giới thiệu

JavaScript là một ngôn ngữ nhẹ (lightweight), thông dịch, hướng đối tượng với first-class function, và được biết đến nhất như là một ngôn ngữ kịch bản (scripting language) cho các trang Web, nhưng **JavaScript** cũng có thể được sử dụng trong các môi trường non-browser như node.js hay Apache CouchDB. Javascript là một prototype-based, ngôn ngữ kịch bản đa mô hình lập trình (multi-paradigm), động (dynamic), hỗ trợ nhiều kiểu như hướng đối tượng, lập trình mệnh lệnh (imperative programming), và lập trình hàm (functional programming).

Chuẩn cho **JavaScript** là **ECMAScript**. Phiên bản hiện nay là **ECMAScript 7** được công bố vào tháng 6/2016. Tuy nhiên, cho đến hiện nay, các trình duyệt chỉ hỗ trợ chuẩn **ECMAScript 5**, sắp tới đây sẽ hỗ trợ cho **ECMAScript 6**. Nhóm sử dụng chuẩn **ECMAScript 6** như là tiêu chuẩn để coding thư viện. Các tính năng mạnh mẽ của **ECMAScript 6** sẽ được liệt kê ở những phần sau.

Tính năng

JavaScript nâng cao sự linh động và khả năng tương tác cho website bằng cách sử dụng các hiệu ứng của nó như thực hiện các phép tính, kiểm tra form, viết các trò chơi, bổ sung các hiệu ứng đặc biệt, tùy biến các chọn lựa đồ họa, tạo ra các mật khẩu bảo mật và hơn thế nữa.

Cụ thể, **JavaScript** có thể được dùng để:

- *Tương tác với người dùng*: Chúng ta có thể viết mã để đáp lại các sự kiện. Các sự kiện này sẽ có thể phát sinh bởi người dùng: nhấp chuột hay được phát sinh từ hệ thống, định lại kích thước của trang ...
- *Thay đổi nội dung động*: Mã JavaScript có thể dùng để thay đổi nội dung và vị trí các phần tử một cách động trên một trang nhằm đáp lại sự tương tác với người dùng.

- *Kiểm tra tính hợp lệ dữ liệu*: Chúng ta có thể viết mã nhằm kiểm tra tính hợp lệ của dữ liệu do người dùng nhập vào trước khi nó được gửi lên Web server để xử lý.

ECMAScript 6

ECMAScript 6 hay gọi tắt là ES6 hay ES2015, đây là phiên bản nâng cấp, tập hợp các kỹ năng nâng cao của Javascript. ES6 ra đời năm 2015 nên cái tên ES2015 được lấy làm tên chính thức với nhiều tính năng mới học hỏi các ngôn ngữ cấp cao khác, hy vọng dần theo thời gian Javascript trở thành một ngôn ngữ lập trình hướng đối tượng.

Một số tính năng mạnh mẽ của ES6:

- *Classes*: Với việc hỗ trợ class, các cú pháp trông đẹp đẽ, đơn giản, dễ nhìn, rõ ràng hơn rất nhiều trong việc tạo ra đối tượng và giải quyết vấn đề thừa kế.
- *Modules*: Hỗ trợ import/export các giá trị (biến, hàm, ...) từ/đến các module.
- *Block Scoped*: Khai báo biến với từ khóa let để cho biết biến này chỉ tồn tại trong phạm vi block của nó.
- *Destructuring Assignment*: Dễ dàng khởi tạo các biến trong suốt các câu lệnh gán từ một mảng.
- *Arrow function*
- *Default Parameter Values*: Cho phép gán giá trị mặc định cho tham số.
- *Rest Parameter*: Kết hợp các argument còn lại thành một tham số của hàm.

Ứng dụng

Dựa trên những đặc điểm và khả năng mạnh mẽ trong hỗ trợ lập trình API của JavaScript đã kể trên, nhóm đề xuất sử dụng JavaScript làm ngôn ngữ chính để hiện thực API.

Ngoài ra, nhóm đề xuất sử dụng chuẩn ECMAScript 6 làm tiêu chuẩn cho các ràng buộc, quy tắc (coding conventions) khi hiện thực API.

2.2.2 D3.js

Giới thiệu

D3.js được viết bởi Mike Bostock, là 1 thư viện JavaScript dùng để thao tác với document dựa trên dữ liệu (data). D3 giúp mang dữ liệu đến cuộc sống bằng cách sử dụng HTML, SVG, và CSS. Sức mạnh của D3 trên chuẩn web cung cấp đầy đủ các tính năng của những trình duyệt hiện đại mà không buộc mình vào một khuôn khổ nhất định nào, kết hợp với các thành phần trực quan mạnh mẽ và một phương pháp hướng dữ liệu để thao tác DOM (Document Object Model).

D3 không phải là framework giải quyết mỗi vấn đề riêng biệt nào, thay vào đó D3 giải quyết điểm cốt lõi của vấn đề: sự thao tác hiệu quả các document dựa trên dữ liệu. Do đó, D3 có được sự linh hoạt cao, có thể sử dụng toàn bộ các tính năng của các tiêu chuẩn web như HTML, SVG, và CSS.

D3 cho phép nhúng bất kì kiểu dữ liệu nào vào 1 phần tử DOM, và sau đó có thể áp dụng CSS3, HTML, và/hoặc SVG lên dữ liệu này. Cuối cùng, D3.js có thể giúp tạo ra dữ liệu tương tác thông qua việc sử dụng những sự biến đổi (transformation), chuyển đổi (transition) hướng dữ liệu của D3.js.

Tính năng

Một số tính năng mạnh mẽ của D3:

- *Selections*: Tập hợp các node tùy ý gọi là selections, và D3 sử dụng phương pháp declarative để biến đổi selections này. Đặc biệt D3 cũng áp dụng các selectors được định nghĩa bởi W3C Selectors API, nhờ đó các phần tử sẽ được chọn lựa ra một cách dễ dàng.
- *Dynamic Properties*: Dễ dàng thay đổi các thuộc tính, style một các động thông qua các function trong D3.
- *Enter và Exit*: Sử dụng enter và exit các selections, người dùng có thể tạo ra nhiều

node mới cho các dữ liệu mới và xóa các node mà không cần thiết. Với việc sử dụng đúng những tác vụ cần thiết trên các node sẽ làm tăng hiệu suất và mang đến những hiệu ứng chuyển đổi tuyệt vời hơn.

- *Transformation*: D3 sử dụng các tiêu chuẩn web: HTML, SVG, và CSS. Do đó nếu trình duyệt giới thiệu một phiên bản mới, người dùng có thể sử dụng D3 ngay lập tức, không yêu cầu cập nhật. Ngoài ra giúp người dùng dễ dàng debug với sự trợ giúp của trình duyệt.
- *Transitions*: Interpolator của D3 hỗ trợ cả 2 kiểu primitive, ví dụ như là số và số nhúng trong string (font-size, dữ liệu về path, ...) và các giá trị phức tạp. Người dùng thậm chí có thể mở rộng interpolator để hỗ trợ cho các thuộc tính phức tạp và cấu trúc dữ liệu.

Ứng dụng

D3 được viết bằng JavaScript, và sử dụng kiểu lập trình hàm (functional programming) do đó có thể sử dụng lại code và thêm những hàm chức năng mà người dùng muốn. Điều đó có nghĩa là nó sẽ mạnh mẽ như thế nào là dựa vào người dùng muốn làm ra nó. Cách người dùng chọn để định dạng, thao tác, tương tác với dữ liệu là tùy thuộc ở họ.

D3 rất thích hợp dùng trong đề tài, nó giúp xây dựng framework trực quan dữ liệu mà người dùng muốn, dễ dàng sử dụng, do đó giúp tạo ra API một cách dễ dàng hơn, API dễ hiểu, dễ sử dụng.

2.2.3 OpenLayers

Giới thiệu

OpenLayers được tạo ra bởi MetaCarta sau hội nghị O'Reilly Where 2.0 (29-30/06/2005) và chính thức phát hành thành mã nguồn mở trước hội nghị O'Reilly Where 2.0 (13-14/06/2006) bởi MetaCarta Labs. Hai dự án khác về công cụ xây dựng bản đồ phát hành bởi MetaCarta là FeatureServer và TileCache. Từ tháng 11/2007, OpenLayers trở thành dự án mã nguồn mở chính thức của Hội đồng Không gian địa lý.

Sơ qua về OpenLayers, đây là một thư viện JavaScript mã nguồn mở (được chứng nhận bởi BSD License) nhằm phục vụ mục đích biểu diễn thông tin trên bản đồ đối với các trình duyệt Web, không phụ thuộc phía server. Nó cung cấp API chuyên về xây dựng các ứng dụng địa lý trên nền Web tương tự như Google Maps và Bing Maps, với một điểm khác biệt duy nhất, OpenLayers hoàn toàn miễn phí, được xây dựng và phát triển nên bởi cộng đồng phát triển phần mềm mã nguồn mở. Thư viện xây dựng trên Prototype JavaScript Framework.

Hiện nay, OpenLayers đã ra tới phiên bản OpenLayers 3. OpenLayers 3 được viết lại nhằm dễ hiểu hơn, hướng tới các tính năng của HTML5 và CSS3. Thư viện tiếp tục hỗ trợ trình chiếu, các giao thức tiêu chuẩn và các chức năng chỉnh sửa. Phiên bản mới này tập trung vào cải thiện hiệu năng, xây dựng đơn giản hơn, các thành phần trở nên trực quan hơn, API được cải tiến và còn nhiều hơn nữa.

Tính năng

Openlayers có 4 tính năng chính:

- *Thể hiện Lớp dưới dạng ô (Tiled Layers)*: Các lớp này được kéo về từ OpenStreetMap (OSM), Bing, MapBox, Stamen, MapQuest và bất cứ nguồn nào mà bạn có thể tìm thấy. Hỗ trợ đa dạng các Layer mà bạn có thể sử dụng trong ứng dụng của mình.
- *Thể hiện Lớp dưới dạng Vector (Vector Layers)*: Sinh ra dữ liệu dạng vector từ GeoJSON, TopoJSON, KML, GML và một số định dạng khác.
- *Tính đối ứng nhanh gọn (Fast & Mobile Ready)*: Hỗ trợ tùy chỉnh các thành phần một cách nhanh gọn và thuận tiện nhất. Cho phép xây dựng một bản tùy chỉnh gọn nhẹ gồm các thành phần mà lập trình viên cần.
- *Mài dũa các góc cạnh và dễ dàng tùy chỉnh (Cutting Edge & Easy to Customize)*: Bản đồ sinh ra bởi công nghệ WebGL, Canvas 2D, và từ những tính năng ưu việt nhất của HTML5. Dễ dàng tùy chỉnh style bằng CSS.

Ứng dụng

Đề tài hướng tới xây dựng thư viện JavaScript cho phép tương tác giữa bản đồ và các đồ thị dựa trên dữ liệu đầu vào, cho nên OpenLayers, cụ thể OpenLayers 3 sẽ là công cụ đặc lực cho việc xây dựng thành phần bản đồ trong đề tài.

2.2.4 Webpack

Giới thiệu

Webpack là một mô-đun đóng gói (module bundler) rất mạnh. Một gói (bundle) là một tập tin Javascript mà kết hợp với các tài nguyên đi chung với nhau và nên được gửi trả về client trong một phản hồi cho một tập tin yêu cầu duy nhất. Một bundle có thể bao gồm các dạng tập tin như Javascript, CSS, HTML, và hầu hết bất kì loại tập tin khác.

Có 4 khái niệm quan trọng (core concept) trong Webpack:

- *Entry*: Webpack tạo ra đồ thị của tất cả các phụ thuộc (dependency) trong ứng dụng. Điểm bắt đầu của đồ thị này chính là điểm đầu vào (entry). Điểm đầu vào nói cho webpack biết được nơi bắt đầu và đi theo đồ thị này để biết cần gộp những gì.
- *Output*: Một khi đã gộp tất cả các tài nguyên lại với nhau, người lập trình cần nói cho webpack biết nơi đóng gói ứng dụng. Output mô tả cho webpack cách xử lý các mã được gộp.
- *Loaders*: Mọi tài nguyên trong dự án của người lập trình đều là mối quan tâm của webpack (điều này không có nghĩa là mọi tài nguyên đều được gộp lại cùng với nhau). Webpack xem mỗi tập tin (css, html, scss, jpg, ...) như là một mô-đun. Tuy nhiên, webpack chỉ biết Javascript. Loaders trong webpack sẽ chuyển đổi những tập tin này thành những mô-đun khi nó được thêm vào trong đồ thị.
- *Plugins*: Loaders chỉ thực thi chuyển đổi trên mỗi tập tin, do đó plugins được sử dụng rất phổ biến để thực hiện những hành động và chức năng tùy chỉnh trên các mảnh (chunk) hoặc bộ (compilation) của các mô-đun. Hệ thống plugin của webpack cực kỳ mạnh mẽ và dễ dàng tùy chỉnh.

Hoạt động và Tính năng

Webpack quét toàn bộ source code của ứng dụng, tìm kiếm câu lệnh import, xây dựng một đồ thị của những dependency, và cho ra 1 hoặc nhiều bundle. Với 1 số các plugin hỗ trợ, Webpack có thể tiền xử lý (preprocess) và thu gọn (minify) các tập tin không phải Javascript khác nhau như CoffeeScript, Typescript, SASS, và LESS,... và có thể chuyển đổi code từ ES6 sang ES5 để chạy được trên trình duyệt.

Một số tính năng đáng chú ý của webpack:

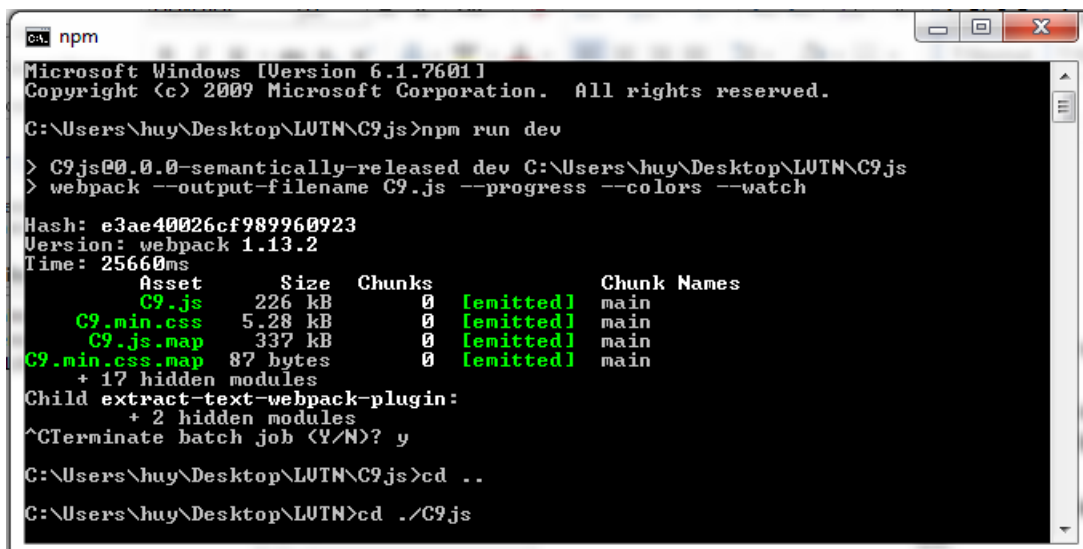
- Hỗ trợ các module format như AMD, CommonJS thông qua các loader (plug-in) như ES6.
- Hỗ trợ các package manager như bower, npm
- Các loader cho dạng không phải code như CSS, template, ...
- Có chế độ chạy trên môi trường phát triển.

Ứng dụng

Với các tính năng mạnh mẽ của Webpack, C9js sử dụng Webpack để:

- Chuyển đổi code từ ES6 sang ES5 nhờ vào loader Babel.
- Đóng gói các tập tin Javascript thành một tập tin duy nhất, dùng cho việc triển khai (release).
- Hỗ trợ chạy trên môi trường phát triển (development environment).

Một ví dụ về việc áp dụng Webpack trong quá trình phát triển thư viện: Hình 2.13 Theo dõi sự thay đổi của các tập tin Javascript, CSS trong quá trình coding để tiến hành build lại thư viện.



```

C:\Users\huy\Desktop\LUTN\C9js>npm run dev
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\huy\Desktop\LUTN\C9js>npm run dev
> C9js@0.0.0-semantic-release dev C:\Users\huy\Desktop\LUTN\C9js
> webpack --output-filename C9.js --progress --colors --watch

Hash: e3ae40026cf989960923
Version: webpack 1.13.2
Time: 25660ms
   Asset      Size  Chunks             Chunk Names
  C9.js       226 kB          0  [emitted]  main
  C9.min.css  5.28 kB          0  [emitted]  main
  C9.js.map   337 kB          0  [emitted]  main
  C9.min.css.map 87 bytes          0  [emitted]  main
+ 17 hidden modules
Child extract-text-webpack-plugin:
+ 2 hidden modules
^CTerminate batch job (Y/N)? y

C:\Users\huy\Desktop\LUTN\C9js>cd ..
C:\Users\huy\Desktop\LUTN>cd ./C9js
  
```

Hình 2.13: Theo dõi thay đổi mã nguồn với Webpack

2.2.5 Unit Testing

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 lý 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:

2.2.5.1 Jasmine

Các khái niệm liên quan

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.

Behavior-driven development (BDD):

- Là một quá trình phát triển phần mềm được mở rộng từ TDD. BDD kết hợp những kỹ 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 sẻ và một quá trình chia sẻ 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 kỹ thuật, và giảm thiểu tối đa vấn đề dịch thuật giữa các ngôn ngữ.

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.

Đặc điểm

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.

- *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 testcase, hay còn gọi là một đặc tả (spec), thì sử dụng hàm it.

```
1 describe('calculator add()', function() {  
2   it('should add 2 numbers together', function() {  
3     // assertions go here  
4   });  
5 });
```

Mã nguồn 2.1: Cấu trúc của một test-suite trong Jasmine

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 expect(calculator.add(1, 4)).toEqual(5);
```

Mã nguồn 2.2: Ví dụ sử dụng expect trong Jasmine

- *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.

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

```
1 it('should resolve with the User object', function(done) {
2   var dfd = new $.Deferred();
3   var promise = dfd.promise();
4   var stub = sinon.stub(User.prototype, 'fetch').returns(
      promise);
5
6   dfd.resolve({name: 'David'});
7
8   User.get().then(function(user) {
9     expect(user instanceof User).toBe(true);
10    done();
11  });
12});
```

Mã nguồn 2.3: Ví dụ sử dụng Jasmine để kiểm thử bất đồng bộ

Ở ví dụ Mã nguồn 2.3, 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.

- *Sinon (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.

- *Thực thi các testcase*: Trong Mocha, người lập trình có thể chạy các test bằng cách viết dòng lệnh như Mã nguồn 2.4 trong command line:

```
1 | mocha test --recursive --watch
```

Mã nguồn 2.4: Khởi chạy trình kiểm thử testcase với Mocha

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 kỹ 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.

2.2.5.2 Karma

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 kỳ 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.

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.

2.2.5.3 PhantomJS

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 lý DOM, bộ chọn CSS (CSS selector), JSON, Canvas, và SVG.

Tính năng

PhantomJS có các tính năng sau:

- *Chụp ảnh màn hình (screen capture)*: Vì PhantomJS sử dụng WebKit, một bộ 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,...).

Ưu điểm 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:

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 kỹ 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.

Tiêu chí	PhantomJS	jsdom
Định nghĩa	Là một trình duyệt hoàn chỉnh (không hỗ trợ giao diện), với kỹ thuật dựng hình (rendering engine) rất cũ và hiếm thấy	Không phải là một trình duyệt hoàn chỉnh, nó không thể bố cục và dựng hình, và cũng không hỗ trợ điều hướng trang. Chỉ hỗ trợ DOM, HTML, canvas, nhiều web API khác (không hỗ trợ SVG), và chạy kịch bản (script)
Môi trường thực thi	Cần một trình thực thi (executable) để chạy PhantomJS, là native-code nên cần được biên dịch thông qua một framework	Hoàn toàn là JavaScript, có thể chạy thông qua NodeJS
Chi phí (overhead) để tải trang	Cao, tốn nhiều thời gian, do đó việc chạy test-case có thể mất vài phút	Mất vài giây với cùng số lượng test-case
Cách thực thi mã nguồn (load script)	Nếu tạo một đoạn mã để kiểm (test) thử trong PhantomJS, thì test 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	Ngược lại với PhantomJS, 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

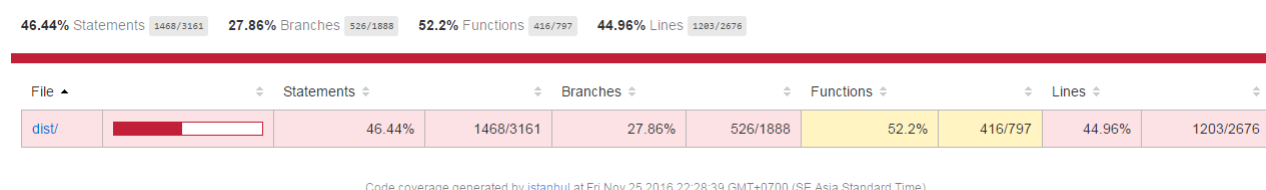
Bảng 2.1: So sánh PhantomJS và jsdom

2.2.5.4 Istanbul

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 đặc 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.



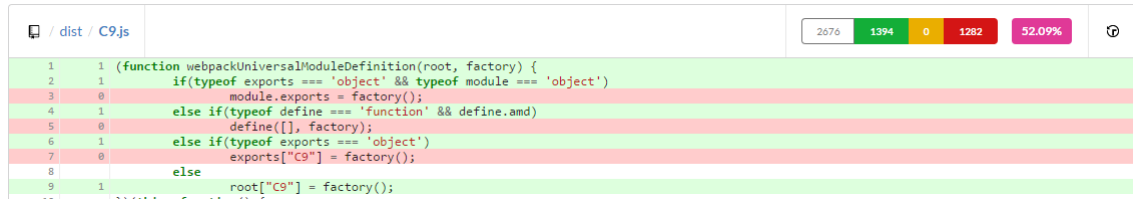
Hình 2.14: Kết quả Unit Testing của C9js được Istanbul tổng quan

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, Cobertura, ...
- 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.



Hình 2.15: Trạng thái các dòng lệnh C9js được cover trên GitHub, báo cáo bởi Istanbul

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.

2.2.6 Code Coverage

Giới thiệu

Như đã giới thiệu sơ qua về thuật ngữ code coverage ở mục 2.2.5.4 (Istanbul), ở phần này sẽ nói rõ hơn về thuật ngữ này cũng như công cụ để báo cáo mức độ bao phủ mã nguồn, đó là codecov.io.

Code coverage:

Là một sự đo lường được dùng để cho thấy những dòng nào trong mã nguồn được thực thi bởi bộ mô tả kiểm thử (test suite). Có ba thuật ngữ được dùng để mô tả sự thực thi của mỗi dòng:

- *Hit:* Cho biết đoạn mã được thực thi.
- *Partial:* Cho biết đoạn mã không hoàn toàn được thực thi và vẫn còn nhánh (branch) không được thực thi.

- *Miss*: Cho biết đoạn mã không được thực thi.

Độ bao phủ được tính bằng công thức:

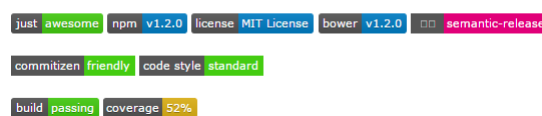
$$\frac{total\ hits}{\sum (hit, partial, miss)}$$

Theo thuật ngữ của Layman^[7] Code coverage cung cấp sự đo lường trực quan của mã nguồn được thực thi bởi bộ kiểm thử. Thông tin này cho lập trình viên biết họ cần viết những test mới ở đâu để đạt được mức độ bao phủ cao hơn. Code coverage thúc đẩy lập trình viên tăng mức độ bao phủ, và có thể tìm ra các lỗi mới, các vấn đề về cú pháp trong mã nguồn trong quá trình phát triển, để họ có thể giải quyết trước khi đưa ứng dụng ra cộng đồng.

Codecov.io:

Là công cụ hỗ trợ tạo báo cáo về mức độ bao phủ mã nguồn của codecov.io và có thêm một huy hiệu coverage trong tập tin README của mã nguồn C9js trên GitHub (Hình 2.16).

C9js



Hình 2.16: Huy hiệu coverage được thêm vào repository C9js

Codecov.io đem vấn đề bao phủ mã nguồn lên một tầm cao mới. Không giống như những công cụ coverage khác, các công cụ mã nguồn mở hay tính phí. Codecov tập trung vào vấn đề tích hợp (integration) và thúc đẩy các pull request (các yêu cầu được ghép mã vào nhánh chính – master) một cách trong sáng. Công cụ Codecov cho phép các số liệu bao phủ (coverage) được thể hiện trực tiếp trong quá trình làm việc (workflow) để thúc đẩy nhiều hơn việc tăng cao mức độ bao phủ, đặc biệt là ở các pull request, nơi mà ở trong những quy trình làm việc hiện đại ngày nay, các tính năng mới và các vấn đề sửa lỗi xảy ra phổ biến.

Tính năng

- *Browser Extension*: Hỗ trợ xuất báo cáo với các đo đạc, thông số trên trình duyệt. Dễ dàng theo dõi và có cái nhìn tổng quan.
- *Auto-Merging Builds*: Hỗ trợ hầu hết các ngôn ngữ, kết hợp với tích hợp liên tục và xây dựng trong một báo cáo.
- *Notifications*: Gửi thông báo qua nhiều kênh giao tiếp như Slack, Gitter, Hipchat, hỗ trợ tạo Webhook,...
- *Coverage Compare*: So sánh sự khác biệt trong sự thay đổi mã nguồn nhờ *git diff*.
- *Pull Request Comments*: Là một bản mô tả chi tiết của một pull request. Nó cung cấp thông tin chi tiết về các thay đổi trong sự bao phủ (coverage) của pull request để giúp tăng nhanh tốc độ xem lại mã nguồn và đảm bảo các pull request đều đã được kiểm tra.
- *Commit Status*: Trạng thái dự án trong thư mục codecov sẽ đo đạc độ bao phủ của dự án đó và so sánh nó với pull request hoặc commit cha (parent commit).

Ứng dụng

C9js sử dụng codecov.io để cho biết mức độ bao phủ của mã nguồn thư viện trực tiếp trên các kênh như GitHub và codecov.io:

- *GitHub*: <https://github.com/csethanhcong/C9js>
- *codecov.io*: <https://codecov.io/gh/csethanhcong/C9js>

Qua các báo cáo mà codecov.io đem lại như tính chính xác của mã nguồn (thông qua các test đã đạt hay không đạt), số lượng các câu lệnh, dòng, nhánh, hàm đã được bao phủ (cover),... đã góp phần làm cho mã nguồn thêm sự ổn định và tin cậy.

2.2.7 Semantic Versioning

Giới thiệu

Trong hệ thống có nhiều các phụ thuộc (*dependency*), vấn đề phát hành (*release*) những gói phiên bản mới nhanh chóng trở thành nỗi ác mộng.

- Nếu các đặc tả phụ thuộc (*dependency specification*) quá chặt chẽ, người dùng sẽ ở trong tình trạng khóa phiên bản (*version lock*), tức là gói (*package*) không có khả năng nâng cấp mà không phải phát hành ra một phiên bản mới đối với mỗi gói phụ thuộc.
- Nếu các phụ thuộc được đặc tả quá lỏng lẻo, các phiên bản sẽ trở nên hỗn tạp (*version promiscuity*).

Những nguyên nhân trên sẽ khiến dự án gặp khó khăn trong việc phát triển sau này. Và một giải pháp cho vấn đề này đó là đưa ra một tập các quy tắc và yêu cầu để chỉ ra cách các số phiên bản được gán cho và tăng lên. Và hệ thống này được gọi là *Semantic Versioning*, dưới hệ thống này, số phiên bản và cách nó thay đổi sẽ cho biết ý nghĩa về những đoạn mã và những gì đã được thay đổi từ một phiên bản đến phiên bản tiếp theo.

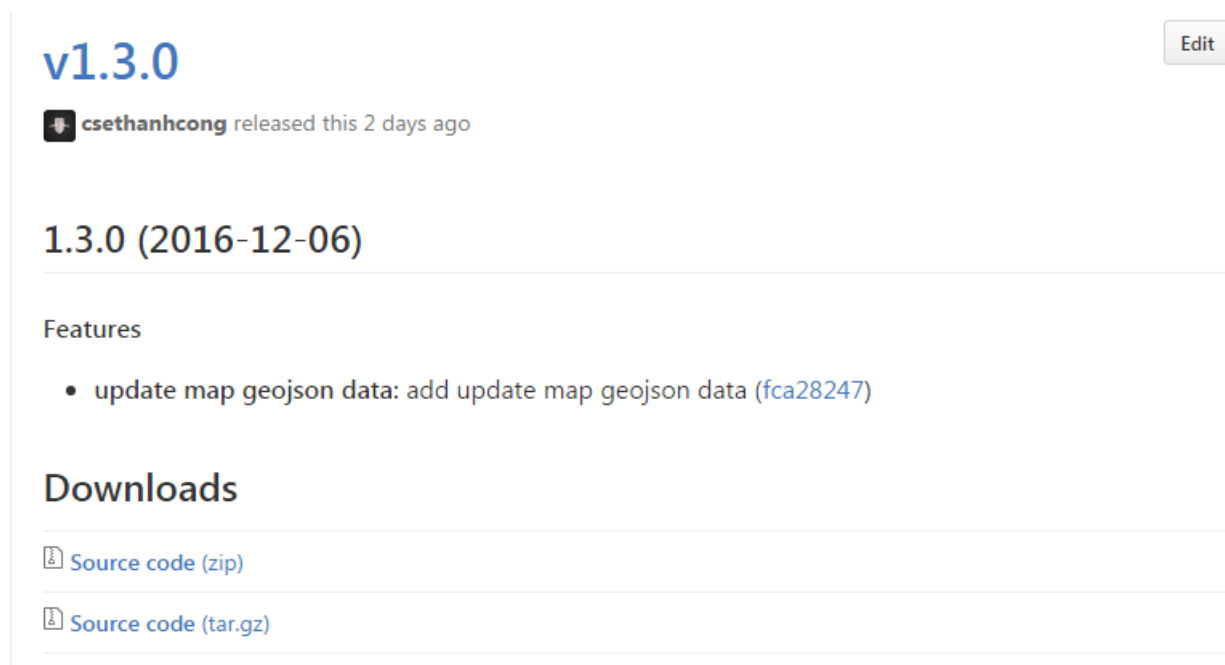
Semantic Versioning định nghĩa số phiên bản sẽ theo định dạng **x.y.z**, hay còn có thể ghi là **major.minor.patch**, mỗi con số này tăng lên sẽ có nghĩa là:

- Thay đổi *MAJOR* khi có một thay đổi lớn trong API mà không tương thích với API phiên bản trước, hay nói các khác là hoàn toàn khác với API trước trong mã nguồn, về kiến trúc cũng như cách sử dụng API.
- Thay đổi *MINOR* khi có thêm chức năng mới cho API, sự thay đổi này vẫn giữ được sự tương thích đối với API phiên bản trước.
- Thay đổi *PATCH* khi sửa chữa các lỗi (fix bug) có trong API.

Semantic Release

Semantic Release là công cụ giúp cho việc xuất bản package một cách hoàn toàn tự động:

- Xác định phiên bản mới các đúng đắn dựa vào commit của những người lập trình, số phiên bản được xác định giống như các quy tắc mà *Semantic Versioning* đã đưa ra.
- Tạo một bản phát hành trên *Github*, bao gồm tag cho phiên bản, tạo ra *changelog* (Hình 2.17).
- Phát hành qua các kênh như *npm*, *bower*, *GitHub* (Hình 2.18).



Hình 2.17: Triển khai trên *GitHub* với *Semantic Release*

2.2.8 CI-CD

Dưới đây là những phương pháp phát triển phần mềm mà C9js đã sử dụng:

★ C9js public

A JavaScript library, used for interactive map-chart based on various types of data.

just awesome npm v1.3.0 license MIT License bower v1.3.0 semantic-release

commitizen friendly code style standard

build passing coverage 51%

C9js is a d3 and OpenLayers-based JavaScript library, built on top with ES6's new features to generate interactive charts and map from arbitrary data, also brings to you strong features.

Follow the link for more information: <http://C9js.me>

Tutorial and Examples

- [Getting Started](#)
- [Examples](#)

Hình 2.18: Triển khai trên *npm* với *Semantic Release*

2.2.8.1 Tích hợp liên tục (Continuous Integration)

Khái niệm

Theo Martin Fowler^[?], tích hợp liên tục là phương pháp phát triển phần mềm đòi hỏi các thành viên trong nhóm tích hợp công việc thường xuyên. Mỗi ngày, các thành viên đều phải theo dõi và phát triển công việc của họ ít nhất một lần. Việc này sẽ được một nhóm khác kiểm tra tự động, nhóm này sẽ tiến hành kiểm thử truy hồi để phát hiện lỗi nhanh nhất có thể. Cả nhóm thấy rằng phương pháp tiếp cận này giúp giảm bớt vấn đề về tích hợp hơn và cho phép phát triển phần mềm gắn kết nhanh hơn.

1.1.2. Điều kiện áp dụng

Những dự án được hưởng lợi từ tích hợp liên tục:

- Nhóm phát triển có ít hơn 50 người, độ phức tạp của dự án không cao.

- Các sản phẩm có sử dụng phần mềm nhúng.

Lợi ích

Một số lợi ích mà tích hợp liên tục mang lại:

- Giảm thiểu rủi ro vì có thể phát hiện sớm lỗi, tiết kiệm thời gian và tiền bạc khi sửa lỗi.
- Hạn chế lỗi khi sắp đến ngày bàn giao sản phẩm.
- Nhà phát triển giảm thời gian sửa lỗi khi phát hiện lỗi lúc chạy kiểm thử đơn vị do họ thường xuyên tích hợp.
- Luôn có sản phẩm có thể chạy hoặc demo cho khách hàng.

2.2.8.2 Chuyển giao liên tục (Continuous Delivery)

Khái niệm

Theo Martin Fowler^{[?]1}, Chuyển giao liên tục là nguyên tắc phát triển phần mềm mà ở đó phần mềm có thể được phát hành ở bất kì thời gian nào.

Điều kiện áp dụng

Nên áp dụng chuyển giao liên tục khi:

- Phần mềm có thể triển khai xuyên suốt vòng đời của nó.
- Ưu tiên việc triển khai hơn là tạo ra tính năng mới.
- Mọi người có thể lấy được phản hồi nhanh chóng và tự động khi có người thay đổi hệ thống.
- Có thể triển khai bất kì phiên bản ở bất kì môi trường nào theo nhu cầu.

Lợi ích

Các lợi ích chủ yếu của chuyển giao liên tục:

- Giảm rủi ro khi triển khai: khi người phát triển triển khai những thay đổi nhỏ, khi đó khả năng sai sót sẽ ít hơn và dễ dàng sửa lỗi khi trục trặc phát sinh.
- Tiến trình trở nên tin cậy: những nhà phát triển theo dõi tiến trình bằng cách theo dõi công việc khi hoàn thành. Nếu “hoàn thành” tức là “nhà phát triển sẽ thông báo là nó được hoàn thành”, điều này có ít độ tin cậy hơn việc triển khai nó thành sản phẩm.
- Phản hồi người dùng: rủi ro lớn nhất đối với bất kì phần mềm nào đó là nhà phát triển cho ra một thứ nào đó không hữu dụng. Nếu nhà lập trình làm việc với người dùng thật sớm hơn và thường xuyên hơn, thì họ sẽ có được phản hồi nhanh chóng hơn để tìm ra giá trị thực sự của phần mềm.

2.2.8.3 Travis CI

Giới thiệu

Travis CI là hệ thống hiện thực tích hợp liên tục (CI) nhanh chóng và gọn lẹ, tích hợp với Github, GitLab, Bitbucket.

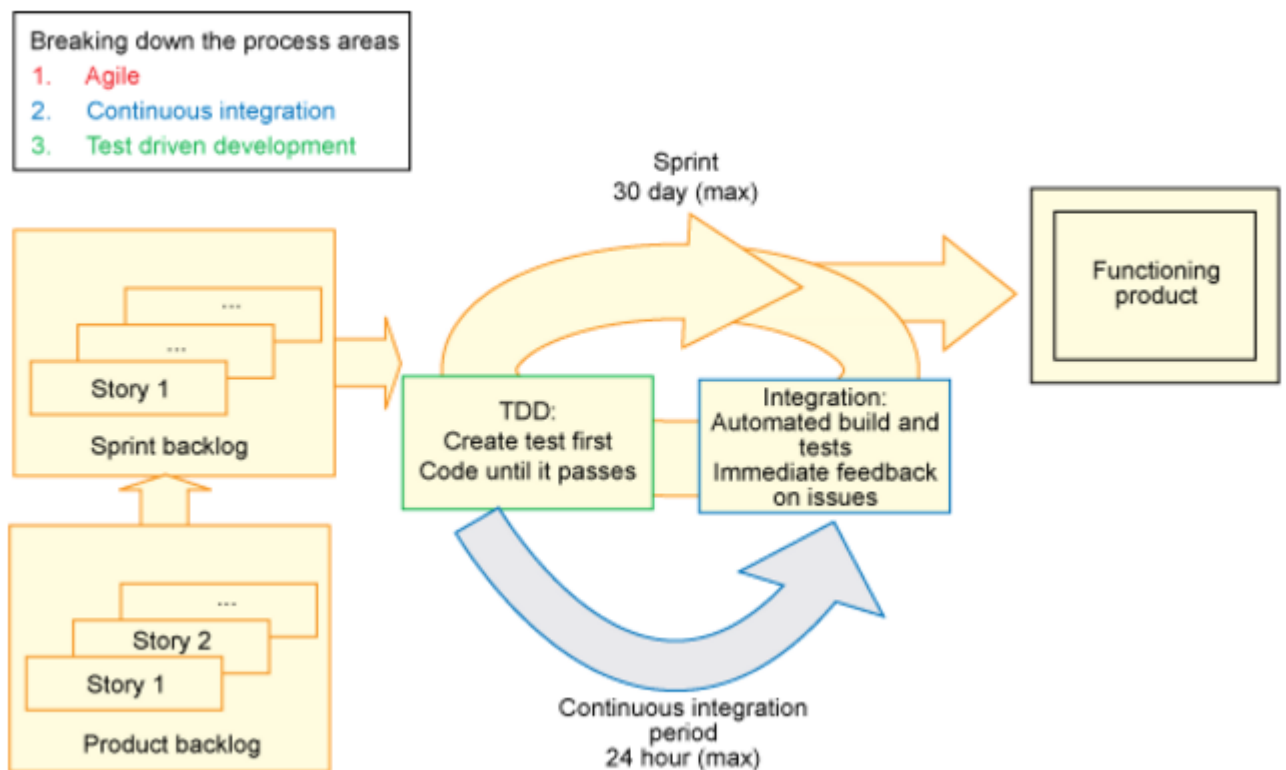
Tính năng

- Theo dõi các test khi các test này chạy.
- Giữ các cấu hình (config) cùng với mã nguồn của nhà phát triển.
- Hỗ trợ thông báo qua các kênh: Slack, HipChat, Email,...
- Một máy ảo hoàn toàn mới mỗi khi biên dịch (build).
- Chạy các test song song.

- Hỗ trợ thêm các hệ điều hành Linux, Mac (và iOS).
- Có command line và API mạnh mẽ.
- Hoàn toàn miễn phí cho các phần mềm mã nguồn mở.

Ứng dụng

Mọi việc trở nên tự động khi sử dụng Travis CI. Mỗi khi commit mã nguồn, hệ thống sẽ tự động lấy mã từ nguồn tích hợp, biên dịch, sau đó chạy các bộ kiểm thử cũng như thông báo tình trạng hiện tại của dự án qua nhiều kênh: email, Slack, ... từ đó dễ dàng tìm ra “hung thủ” làm hư hại mã, cũng như các test không vượt qua được,... Sau khi hệ thống chạy thành công, có thể yêu cầu thực hiện một vài tác vụ như báo cáo độ bao phủ mã nguồn, phát hành phần mềm,...



Hình 2.19: Sự kết hợp giữa tích hợp liên tục (CI) và phát triển theo hướng kiểm thử (TDD)

C9js áp dụng Travis CI để hiện thực tích hợp liên tục, phát triển hướng kiểm thử, cùng với chuyển giao liên tục trong quá trình phát triển thư viện (Hình 2.20). Quy trình thực hiện sẽ được nêu chi tiết trong phần Hiện thực thư viện.

```
337 C9 Unit Test - Donut Chart
338   ✓ should create DonutChart with id #chart
339 JS 2.1.1 (Linux 0.0.0): Executed 13 of 33 SUCCESS (0 secs / 0.179 secs)
340 C9 Unit Test - Donut Chart
341   ✓ should create DonutChart with id #chart
342   ✓ should create DonutChart with innerRadius: 50, outerRadius: 150
343 JS 2.1.1 (Linux 0.0.0): Executed 14 of 33 SUCCESS (0 secs / 0.185 secs)
344   ✓ should create DonutChart with innerRadius: 50, outerRadius: 150
345
346 C9 Unit Test - Event Listener
347   ✓ should show data on div#expand
348 JS 2.1.1 (Linux 0.0.0): Executed 15 of 33 SUCCESS (0 secs / 0.194 secs)
349 C9 Unit Test - Event Listener
350   ✓ should show data on div#expand
```

Hình 2.20: Chạy kiểm thử đơn vị thư viện C9js bởi Travis CI

3 Thư viện C9js

3.1 Mô tả bài toán

Thông qua các công trình đã tìm hiểu, nhóm đề tài đánh giá và trích xuất ra các điểm hạn chế, các vấn đề mà các công trình đó chưa giải quyết được hoặc đã giải quyết nhưng đã lỗi thời cần nâng cấp để phù hợp với bối cảnh hiện tại của các công nghệ Web. Phần đánh giá, so sánh với các công trình liên quan sẽ được thể hiện chi tiết trong mục **Đánh giá**.

Bài toán mà nhóm đề tài đặt ra:

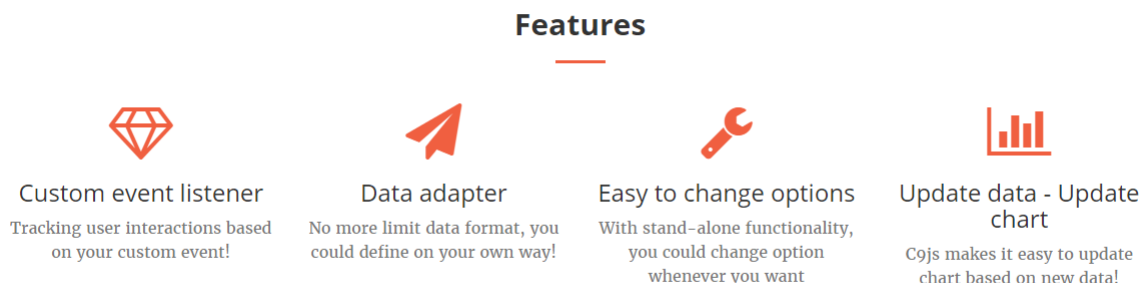
Hỗ trợ trực quan dữ liệu trên Web thông qua thư viện C9js.

Các vấn đề "thất nút" cần giải quyết:

Cũng là các mục tiêu và hướng phát triển **C9js** đã, đang và mong muốn đạt được:

- *Hoàn toàn là mã nguồn mở:* **C9js** là thư viện JavaScript mã nguồn mở, được triển khai thông qua GitHub ở địa chỉ <https://github.com/csethanhcong/C9js>.
- *Kiến trúc xây dựng theo tiêu chuẩn ES2015 (hay ES6):* Nghiên cứu cẩn trọng qua các mã nguồn mở có khả năng trực quan dữ liệu, việc chọn kiến trúc để xây dựng từ ban đầu là rất quan trọng, đặc biệt trong khả năng mở rộng nhằm đáp ứng nhu cầu tùy chỉnh hay đóng góp từ cộng đồng mã nguồn mở. Tiêu chuẩn ES6 hỗ trợ lập trình hướng đối tượng với các tính năng như *class*, *import/export*, *kế thừa* đáp ứng được yêu cầu mở rộng về sau của C9js, cũng như tính năng *getter/setter* giúp đảm bảo các thuộc tính được bảo toàn, không được truy xuất trực tiếp bởi người dùng.
- *Bao gồm các tính năng mới và mạnh mẽ:* Các tính năng riêng biệt của C9js được thể hiện rõ trên trang chủ của thư viện, tại địa chỉ <http://c9js.me/>. Ví dụ như tính năng *DataAdapter*, đây là tính năng mới hỗ trợ người dùng tự định nghĩa định dạng dữ liệu đầu vào, giảm thiểu tối đa nỗ lực trong quá trình tiền xử lý dữ liệu. Ngoài ra còn có

các tính năng khác phát triển và có nhiều đặc tính nổi bật. Phần trình bày chi tiết sẽ được nêu ở phần sau.

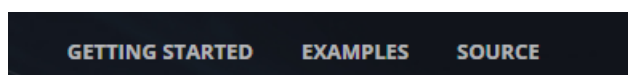


Hình 3.1: Các tính năng chính của C9js trên trang chủ c9js.me

3.2 Hướng dẫn sử dụng

Với mong muốn đem lại trải nghiệm nhanh nhất, đơn giản nhất và ít tốn công sức nhất cho người dùng (ở đây là các lập trình viên), nhóm đề tài đã triển khai Website <http://c9js.me/> gồm các hướng dẫn từ cơ bản đến chuyên sâu, và gồm cả các ví dụ cụ thể nhất.

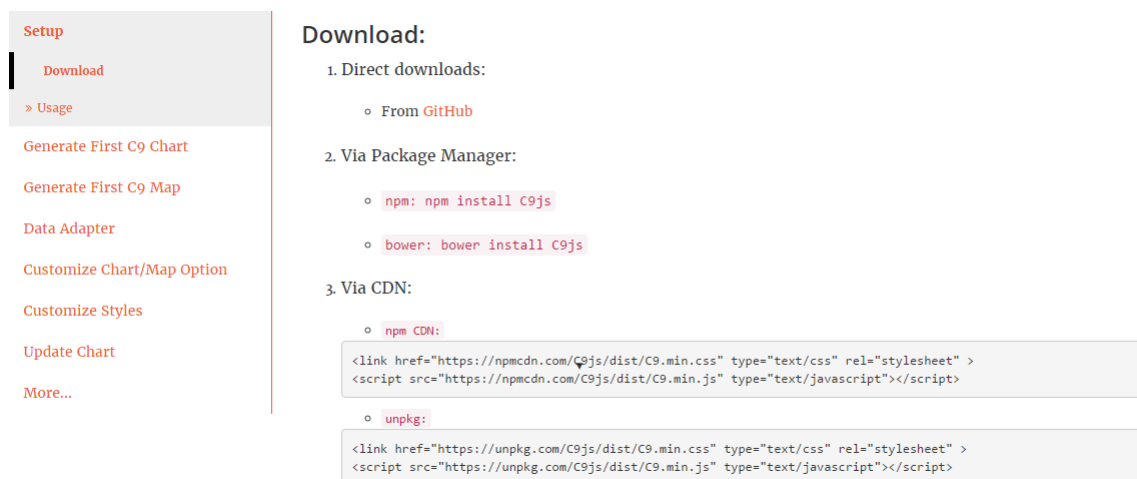
Qua Website c9js.me, có thể thấy mục **Getting Started** ở các nút ở trang chủ (landing page) hay thông qua menu ở đầu trang như hình 3.2



Hình 3.2: Menu hướng dẫn trên trang chủ c9js.me

Khi đó, màn hình hướng dẫn xuất hiện như hình 3.3

Như có thể thấy, menu giao diện bên trái thể hiện các mục hướng dẫn người dùng, bao gồm các bước cơ bản nhất để sử dụng và tùy biến các tính năng trong C9js theo ý muốn



Hình 3.3: Trang hướng dẫn trên Website c9js.me

3.2.1 Cài đặt (Setup)

Tải xuống (Download):

Hiện tại, nhằm đáp ứng tối đa lượng người dùng, C9js được triển khai qua nhiều kênh khác nhau

- *Tải trực tiếp*: Có thể tải trực tiếp ở địa chỉ <https://github.com/csethanhcong/C9js/releases/latest>.
- *Các trình quản lý tập tin (Package Manager)*: Hiện tại phổ biến nhất có 2 trình quản lý là **npm** và **bower**. Và C9js được phân phối qua cả 2 kênh này. Người dùng cài đặt thông qua một trong các lệnh:
 - **npm**: `npm install C9js`
 - **bower**: `bower install C9js`
- *Mạng phân phối nội dung (CDN)*: Ngoài ra, người dùng còn có thể truy cập các mã nguồn cần thiết từ C9js thông qua hai dịch vụ CDN phổ biến là **npmcdn** và **unpkg**
 - **npmcdn**:

```
1 <link href="https://npmcdn.com/C9js/dist/C9.min.css"
  type="text/css" rel="stylesheet" >
2 <script src="https://npmcdn.com/C9js/dist/C9.min.js"
  type="text/javascript"></script>
```

Mã nguồn 3.1: Tải mã nguồn thông qua **npmcdn**

- **unpkg**:

```
1 <link href="https://unpkg.com/C9js/dist/C9.min.css" type=
  "text/css" rel="stylesheet" >
2 <script src="https://unpkg.com/C9js/dist/C9.min.js" type=
  "text/javascript"></script>
```

Mã nguồn 3.2: Tải mã nguồn thông qua **unpkg**

Cách dùng (Usage):

Thư viện C9js hỗ trợ tạo biểu đồ dựa trên *d3.js* và bản đồ dựa trên *OpenLayers 3* nên người dùng cần tải các thư viện này trước.

Tùy theo mục đích sử dụng, nếu chỉ dùng chức năng tạo biểu đồ thì chỉ cần tải thư viện *d3.js*, tương tự đối với chức năng tạo bản đồ.

```
1 <link href="/path/to/C9.css" rel="stylesheet" type="text/css">
2 <!-- Load d3.js and C9.js -->
3 <script src="/path/to/d3.v3.min.js" charset="utf-8"></script>
4 <script src="/path/to/C9.min.js"></script>
```

Mã nguồn 3.3: Tải mã nguồn *d3.js* trước tiên

```
1 <link rel="stylesheet" href="https://openlayers.org/en/v3.19.1/
  css/ol.css" type="text/css">
2 <script src="https://openlayers.org/en/v3.19.1/build/ol.js" type=
  "text/javascript"></script>
3 <script src="/path/to/C9.min.js"></script>
```

Mã nguồn 3.4: Tải mã nguồn *OpenLayers 3* nếu muốn sử dụng chức năng Bản đồ

3.2.2 Tạo biểu đồ và bản đồ với C9js

Sau khi đã tải đầy đủ các tập tin như bước trên, người dùng có thể tạo biểu đồ, bản đồ tương tác với các mã nguồn dựng sẵn.

Tạo bản đồ cột (Bar Chart):

C9js được nhóm xây dựng dựa trên các tính năng mới nhất của *ES6*, ngoài ra còn mong muốn đem lại sự tiện dụng và rõ ràng cho người dùng.

Đầu tiên, cần tạo một *khung chứa* (*container*) trong cấu trúc cây **DOM** của Website.

```
1 <div id="chart"></div>
```

Mã nguồn 3.5: Tạo *container* để chứa biểu đồ

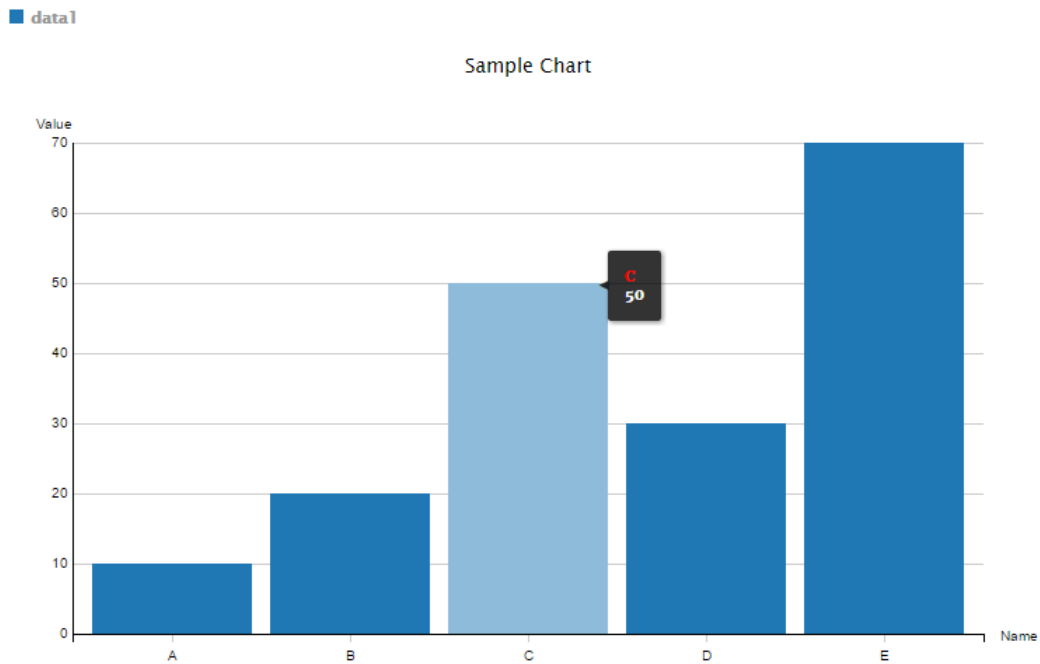
Sau đó, khởi tạo biểu đồ như Mã nguồn 3.6

```
1 var option = {
2   id: "chart",
3   data: {
4     plain: [
5       {name: "A", value: 10},
6       {name: "B", value: 20},
7       {name: "C", value: 50},
8       {name: "D", value: 30},
9       {name: "E", value: 70},
10    ],
11  },
12 };
13 var barChart = new C9.BarChart(option);
14 barChart.draw();
```

Mã nguồn 3.6: Khởi tạo biểu đồ với C9js

Khi đó, trên Website sẽ xuất hiện biểu đồ cột như hình 3.4, người dùng có thể tương tác với

các thành phần trên đồ thị như *Cột*, *Chú thích (Legend)* sẽ hiển thị các *Tooltip* chứa các dữ liệu liên quan đến thành phần đang được tương tác..



Hình 3.4: Đồ thị cột từ C9js

Ngoài ra, C9js còn hỗ trợ tạo nhiều loại đồ thị khác nhau như: *Đồ thị đường (Line Chart)*, *đồ thị thời gian (Timeline)*, *đồ thị tròn (Pie Chart)*, *đồ thị hình xuyến (Donut Chart)*.

Tạo bản đồ (Map):

Giống như tạo biểu đồ, đầu tiên người dùng cần tạo một *container* cho bản đồ.

```
1 <div id="chart"></div>
```

Mã nguồn 3.7: Tạo *container* để chứa bản đồ

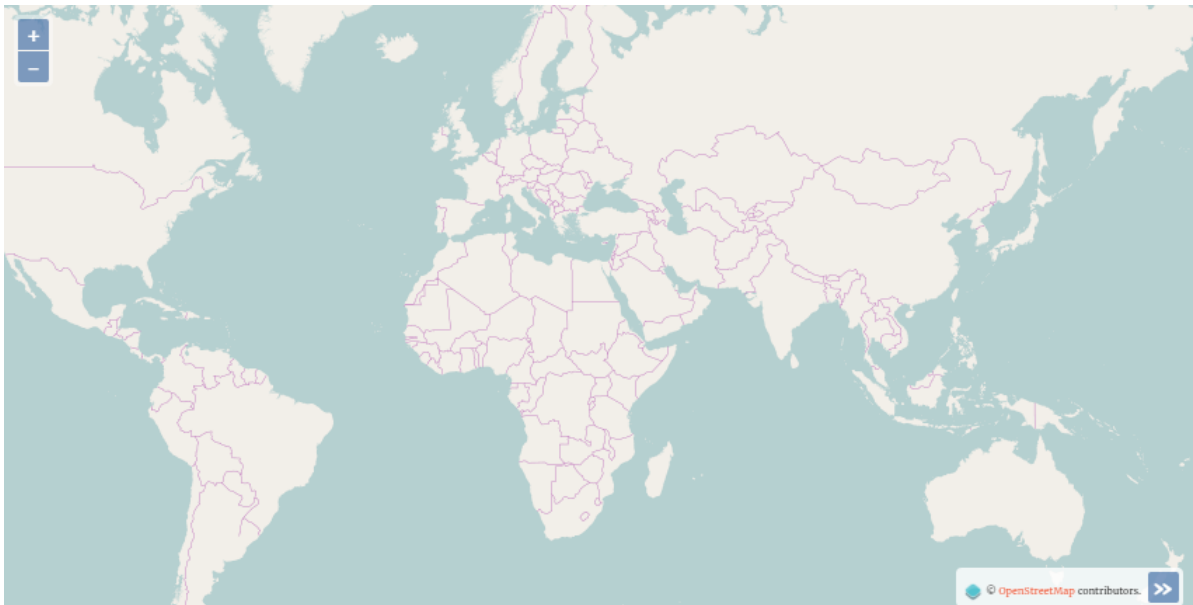
Khởi tạo bản đồ tại *container* đó.

```
1 var option = {
2   id: "map",
3 };
```

```
4 var map = new C9.Map(option);  
5 map.draw();
```

Mã nguồn 3.8: Khởi tạo bản đồ với C9js

Kết quả ta có bản đồ với **Layer** mặc định là *Tile* cùng **dữ liệu địa lý** được cung cấp bởi *OpenStreetMap* như hình 3.5. Bên cạnh đó, C9js hỗ trợ nhiều loại **Layer** khác nhau như: *Tile*, *Vector*, *Image*, *VectorTile*,... với nhiều nhà cung cấp **dữ liệu địa lý** khác nhau như: *OpenStreetMap*^[1], *Bing Map*^[2], *Stamen*^[3],



Hình 3.5: Bản đồ từ C9js

Hình 3.5 là bản đồ dạng tĩnh. C9js cho phép người dùng thêm dữ liệu tùy biến và tương tác trên bản đồ.

```
1 var options = {  
2   id: "map",  
3   layers: {  
4     type: "Image",  
5     source: {  
6       name: "ImageVector",  
7       source: {
```

```

8         name: 'Vector',
9         url: 'https://openlayers.org/en/v3.19.1/examples/
           data/geojson/countries.geojson',
10        format: 'GeoJSON'
11    },
12    },
13    },
14    };
15    var map = new C9.Map(option);
16    map.draw();

```

Mã nguồn 3.9: Thêm dữ liệu vào Bản đồ với C9js

Như Mã nguồn 3.9, với dữ liệu được truyền vào thông qua địa chỉ bên ngoài (ở đây là dữ liệu các quốc gia được lấy từ *OpenLayers*) bởi thông số *url*. Ta được kết quả như hình 3.6. , các giá trị được thể hiện khi đưa chuột (*hover*) lên các vùng trên bản đồ, ở đây là tên các quốc gia trên thế giới.



Hình 3.6: Thêm dữ liệu vào bản đồ với C9js

3.2.3 Tùy biến định dạng dữ liệu đầu vào

Với tên gọi là `DataAdapter`, đây là một trong những tính năng độc đáo và mới lạ của C9js, nhằm tạo điều kiện tốt nhất cho người dùng trong quá trình tiền xử lý dữ liệu đầu vào. Nội dung cụ thể, tính năng và mô hình xây dựng nên `DataAdapter` sẽ được trình bày chi tiết trong mục **Kiến trúc xây dựng**.

Quay lại với cách sử dụng tính năng `DataAdapter`. Trên trang chủ C9js, nhóm có trình bày về khái niệm cơ bản để người dùng dễ dàng nắm bắt, cũng như ví dụ sử dụng. `DataAdapter` hỗ trợ nhiều định dạng dữ liệu khác nhau như *csv* (*Comma-separated values*), *tsv* (*Tab-separated values*), *json*, *text*, *xml* và cả *XHR* (*XMLHttpRequest*)^{[?][?]}. Thêm nữa, `DataAdapter` còn cung cấp tính năng tự định nghĩa định dạng dữ liệu đầu vào. Ví dụ ta có một dữ liệu mẫu như Mã nguồn 3.10.

```
1  [
2    {
3      username: 'Adam',
4      property: {
5        salary: 1000,
6        age: 28,
7      },
8      ...
9    },
10   ...
11  ]
```

Mã nguồn 3.10: Một dữ liệu mẫu với các thuộc tính lồng nhiều cấp

Khi đó, người dùng chỉ việc định nghĩa các khoá tương ứng *name*, *value* tùy ý, theo bất kỳ trường nào trong dữ liệu mẫu, như ví dụ trên có thể định nghĩa theo các khoá lồng nhiều cấp bằng dấu "." (Mã nguồn 3.11).

```
1  var option = {
2    data: {
```

```
3      // Your own defined-keys go here
4      keys: {
5          'name': 'username',
6          'value': 'property.salary'
7      }
8  }
9  }
10 var chart = new C9.BarChart(option);
11 chart.draw();
```

Mã nguồn 3.11: Định nghĩa định dạng dữ liệu đầu vào với C9js

3.2.4 Tùy biến kiểu đồ thị

Việc trực quan dữ liệu có mang tính hiệu quả, hay gây ấn tượng với thị hiếu người dùng hay không hoàn toàn phụ thuộc vào *style* của loại trực quan đó. C9js cho phép lập trình viên toàn quyền tùy chỉnh *style* theo ý muốn và mục đích của mình.

Có 3 cách để thay đổi *style* trong C9js:

- *Thông qua các thiết lập (option) tại thời điểm khởi tạo đối tượng Chart/Map*

Mã nguồn 3.12 thực hiện các thay đổi về các trục tọa độ (axis), cụ thể là xoay các tick trên trục x 45°(theo chiều kim đồng hồ), không hiện lưới (grid), đối với trục y thực hiện thay đổi nội dung hiển thị trên các tick dựa vào dữ liệu trả về. Ta được kết quả như hình 3.7.

```
1 var option = {
2     axis: {
3         x: {
4             show: true,
5             grid: false,
6             tick: {
7                 rotate: 45
```

```
8         }
9     },
10    y: {
11        show: true,
12        tick: {
13            format: function(data, index) {
14                return '-' + data + '-';
15            }
16        }
17    },
18 }
19 };
20 var chart = new C9.LineChart(option);
21 chart.draw();
```

Mã nguồn 3.12: Thay đổi *style* tại thời điểm khởi tạo Chart

- *Thông qua các CSS Selector được định nghĩa sẵn*

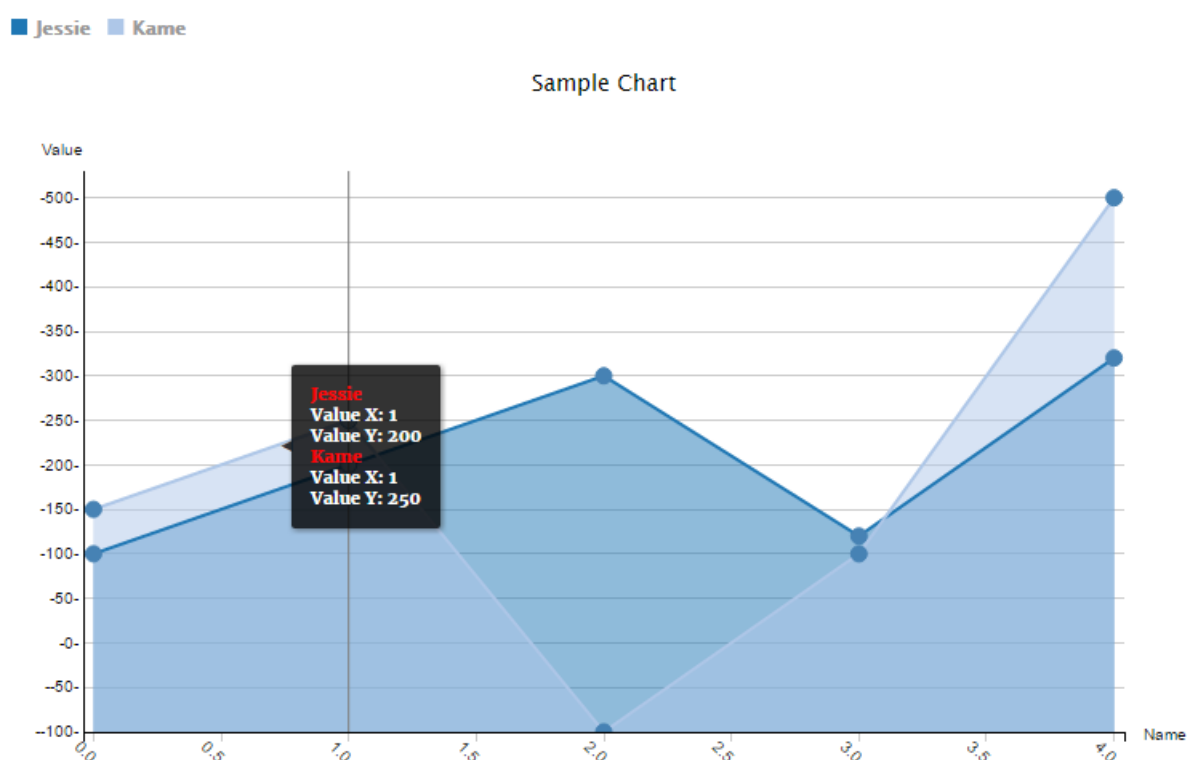
Mỗi thành phần (axis, tick, grid, bar, legend, line, ...) trong C9js đều có các CSS Class^[?] tương ứng. Lập trình viên có thể thay đổi *style* thông qua các CSS Selector này (Mã nguồn 3.13). Một điểm nhỏ thú vị của C9js nữa là các thành phần liên quan đến màu sắc (color, background-color, font-color, ...), giá trị đầu vào có thể dưới dạng tên tiếng Anh, mã RGB hay mã Hex.

```
1 .c9-chart-bar.c9-custom-rect {
2     opacity: 0.5;
3     background-color: black; // rgb(), hex-code available
4 }
```

Mã nguồn 3.13: Thay đổi *style* bằng CSS Selector

- *Thông qua hàm `setOption`*

Nhằm giảm thời gian và rắc rối khi phải đưa toàn bộ các tùy chỉnh, thuộc tính vào lúc khởi tạo, C9js có một tính năng gọi là hàm-đứng-riêng (stand-alone function) `setOption`



Hình 3.7: Một ví dụ về thay đổi *style* với C9js

giúp lập trình viên có thể thay đổi tùy chỉnh bất cứ lúc nào sau khi khởi tạo đối tượng Chart/Map. Việc thay đổi được ví dụ như Mã nguồn 3.14, có thể thay đổi các tùy chỉnh lồng nhiều cấp bằng phân cách dấu ".", tương tự như tính năng DataAdapter đã trình bày ở trên.

```
1 | chart.setOption('grid.x.show', true);
```

Mã nguồn 3.14: Thay đổi *style* với *setOption*

3.2.5 Cập nhật đồ thị theo dữ liệu

Một tính năng không thể thiếu đối với trực quan dữ liệu trên nền Web, đó là cập nhật theo thời gian thực. C9js cung cấp API với hàm *updateData* giúp lập trình viên hiện thực điều đó. Ví dụ 3.15 thể hiện cách sử dụng *updateData*, ở đây biểu đồ *DonutChart* sẽ được cập nhật lại sau 5 giây với dữ liệu mới được đưa vào, cùng định dạng với dữ liệu ban đầu.

```
1 | // ... chart drawn already
2 | var chart = new C9.DonutChart(option);
3 | chart.draw();
4 | // Then, update it
5 | setTimeout(function(){
6 |     chart.updateData([
7 |         {name: "Male", value: 45},
8 |         {name: "Female", value: 55},
9 |     ]);
10 | }, 5000);
```

Mã nguồn 3.15: Cập nhật lại biểu đồ với *updateData*

Ngoài ra, lập trình viên còn có thể tái định nghĩa lại các khoá-thuộc tính trong lúc cập nhật biểu đồ. Ví dụ ta có mẫu dữ liệu ban đầu được thể hiện dưới dạng biểu đồ tròn (PieChart) với khoá là *age*, sau đó có thể cập nhật lại biểu đồ với cùng mẫu dữ liệu đó, nhưng thể hiện theo khoá *salary* như ví dụ 3.16. Việc này có ý nghĩa giảm tối đa thao tác trên dữ liệu nếu lập trình viên muốn sử dụng lại dữ liệu cũ nhưng chỉ khác khoá mà không cần phải xử lý lại dữ liệu như ban đầu.

```
1  setTimeout(function(){
2      chart.updateData([
3          {
4              name: "Male",
5              property: {
6                  age: 28,
7                  salary: 5000
8              }
9      },
10     {
11         name: "Female",
12         property: {
13             age: 30,
14             salary: 4500
15         }
16     },
17     ], {
18         value: 'property.salary'
19     });
20 }, 5000);
```

Mã nguồn 3.16: Định nghĩa lại khoá-thuộc tính trong lúc cập nhật

3.3 Các ví dụ minh hoạ

Bên cạnh các hướng dẫn từ cơ bản đến chuyên sâu như trên, nhóm đề tài còn tạo một mục gồm các mã nguồn có sẵn ứng với các tính năng, tùy chỉnh và đặc điểm của C9js. Tham khảo tại mục **Examples** trên trang chủ C9js (Hình 3.2).

Các ví dụ được phân loại theo (Hình 3.8):

- *Kiểu đồ thị, bản đồ:* BarChart, LineChart, Timeline, DonutChart, PieChart, Map.

- Các tiện ích mở rộng: Axis, Legend, Tooltip, Table.
- Các tính năng: EventListener, DataAdapter, UpdateData.
- Các ví dụ mở rộng: Advance Examples.

Bar Chart

Single Bar

Display as Single Bar Chart

Stacked Bar

Display as Stacked Bar Chart

Grouped Bar

Display as Grouped Bar Chart

Logarithmic Data

Bar Chart with Logarithmic Data

Pie Chart

Simple Pie

Display as Pie Chart

Pie Radius

Change Pie Chart Radius

Donut Chart

Simple Donut

Display as Donut Chart

Donut Radius

Change Donut Chart Inner & Outer Radius

Line Chart

Hide Point

Show/Hide point on Line Chart

Point Style

Set style for point

Hide Area

Show/Hide area on Line Chart

Line Style

Set style for line

Interpolate Line

Set interpolate style for line

Show Subchart

Show/Hide subchart on Line Chart

Multi Values

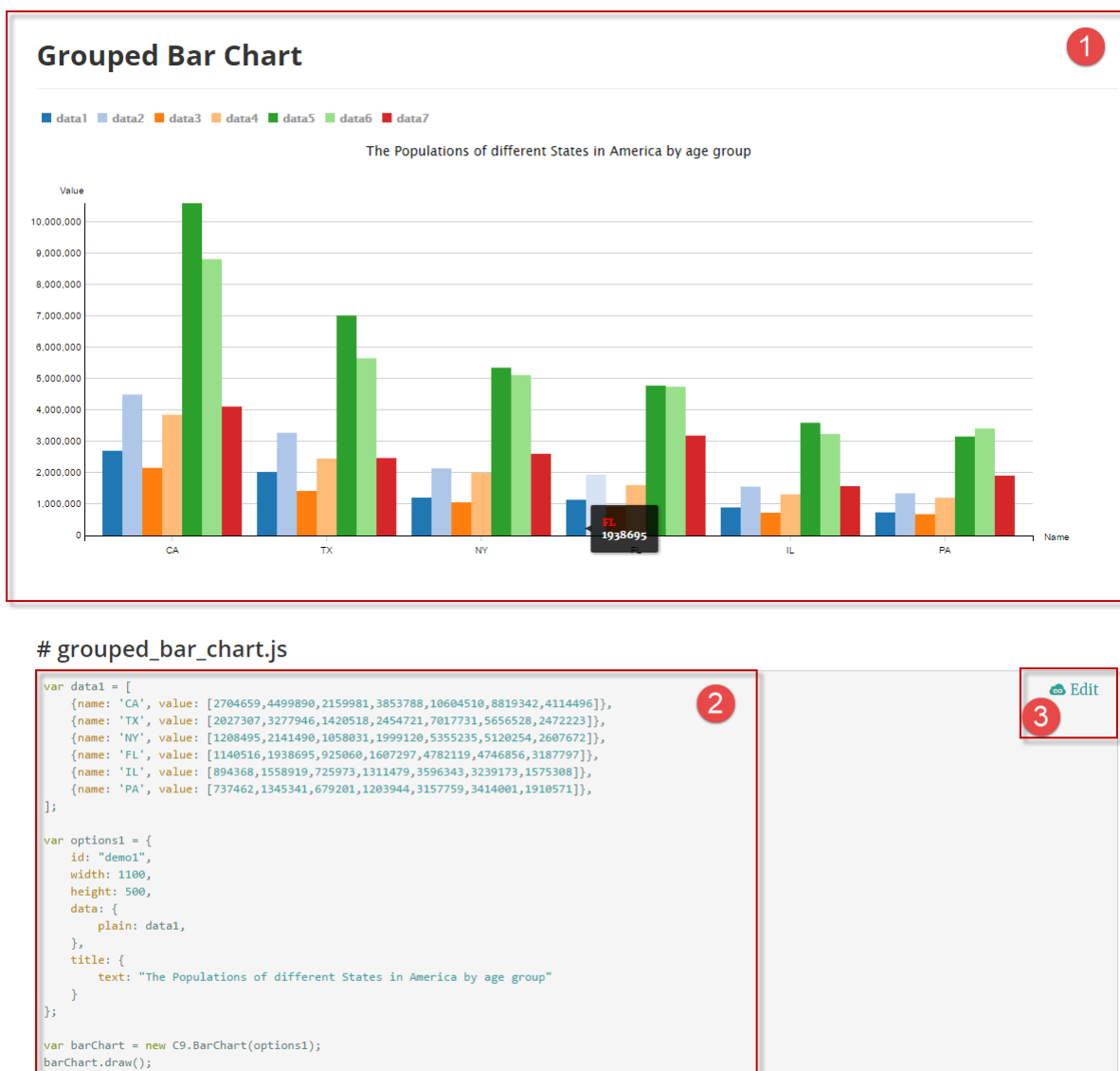
Display multi-value on Line Chart

Hình 3.8: Các ví dụ được phân loại theo kiểu đồ thị

Cấu trúc của một trang ví dụ bao gồm ba thành phần (Hình 3.9):

- 1 - Biểu đồ, bản đồ được tạo ra.
- 2 - Mã nguồn mẫu.

- 3 - Đường dẫn tới trang *jsfiddle* - đây là một trang chỉnh sửa code trực tuyến, nhóm đã để sẵn code mẫu như trên ví dụ và người dùng có thể trực tiếp chỉnh sửa ở *jsfiddle*.

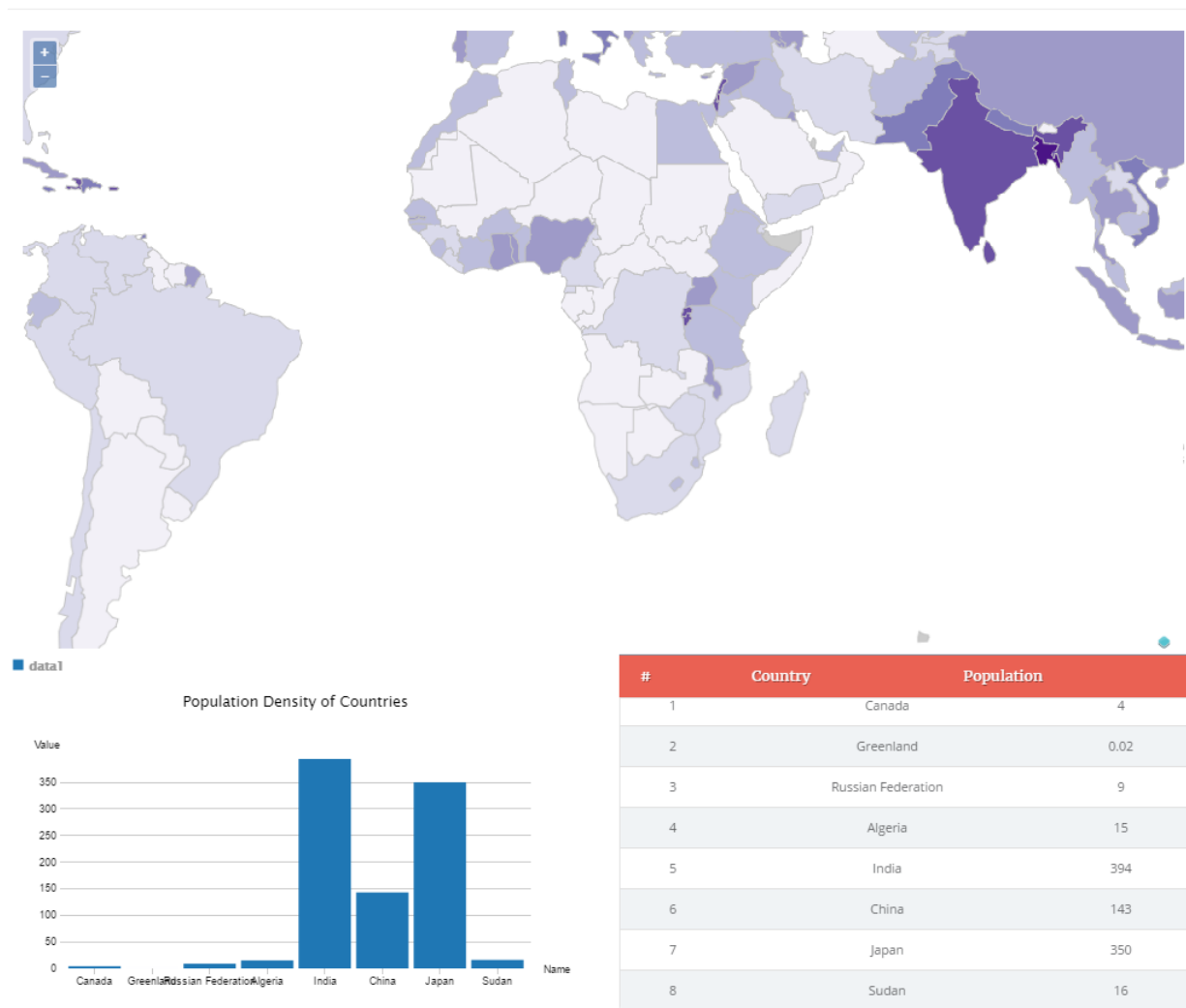


Hình 3.9: Cấu trúc một trang ví dụ trên trang chủ C9js

Bên cạnh đó, nhóm còn tạo sẵn ví dụ mở rộng: Tương tác giữa bản đồ - biểu đồ (nằm trong mục Advance Examples), một phần nhằm tái hiện ứng dụng Global Health Atlas - WHO^[1] đã trình bày ở mục 2.1, một phần muốn giúp người dùng có cái nhìn sâu sắc hơn về tính tương tác trong C9js. Qua đó, có thể tạo các ví dụ mở rộng khác dựa trên mã nguồn mẫu.

Như có thể thấy ở hình 3.10, có ba thành phần chính được tạo ra: *BarChart*, *Map*, *Table*. Khi tương tác với một điểm trên *Map*, *BarChart* sẽ được cập nhật lại dựa trên dữ liệu trả về từ sự kiện tương tác với *Map* - đây là tính năng EventListener của C9js, các tương tác (click, hover, mousemove) trên các thành phần của C9js (Chart, Map) sẽ được trả về dữ liệu tương ứng với điểm đó. Đối với thành phần *Table*, khi hover lên *BarChart*, hàng chứa giá trị tương đương sẽ được làm nổi bật giúp người dùng có cái nhìn trực quan và sinh động đối với dữ liệu.

Map and Chart



Hình 3.10: Ví dụ mở rộng tái hiện ứng dụng GHA-WHO

3.4 Các hàm chức năng (API)

C9js gồm hai thành phần chính là Chart và Map, nên API được tổ chức nhằm hỗ trợ tối đa các lập trình viên theo hai thành phần này.

Chart

1. `draw()`: Tạo Chart dựa vào đối tượng đã được khai báo và khởi tạo.
2. `setOption(key, value)`: Thay đổi một tùy chỉnh trong đối tượng khởi tạo.
 - `key <String>`: Khoá định nghĩa tùy chỉnh sẽ thay đổi.
 - `value <String, Number>`: Giá trị áp dụng cho thay đổi.
3. `updateData(newData [, newDataConfig])`: Cập nhật lại đồ thị dựa trên dữ liệu mới nhập vào.
 - `newData <Object>`: Dữ liệu mới để cập nhật.
 - `newDataConfig <Object>`: Định nghĩa lại khoá mới.
4. `on(eventType, callback)`: Lắng nghe sự kiện và trả về dữ liệu tương ứng trên đối tượng được tương tác.
 - `eventType <String>`: Loại sự kiện (hover, click, mousemove).
 - `callback <Function>`: Chứa dữ liệu trả về.

Map

Đối với Map, C9js phân biệt hai loại đối tượng (*Object*) như sau:

- Đối tượng người dùng tự định nghĩa: Thông qua các hàm `addData()` hoặc `createObject()` và sẽ gồm các trường sau khi chuẩn hoá là *coord* (*coordinate*), *name*, *value*.
- Đối tượng GeoJSON: Đối tượng được tạo ra bởi tập tin GeoJson truyền vào bởi người dùng, dữ liệu mặc định sẽ được C9js truy xuất thông qua trường *properties* (mặc định các tập tin GeoJSON có trường này) và trường *id*.

Các hàm API của Map trong C9js được liệt kê như sau:

1. `addData(plain [, keys])`: Tạo một/nhiều đối tượng trên bản đồ (Line, Polygon, Marker), có thể định nghĩa khoá đầu vào.
 - *plain* <Object>: Dữ liệu đầu vào, yêu cầu có các trường *coord*, *name*, *value*.
 - *keys* <Object>: Định nghĩa các trường dữ liệu đầu vào để C9js truy xuất.
2. `createObject(data, style)`: Tạo một đối tượng mẫu trên Map, chứa các trường theo yêu cầu.
 - *data* <Object>: Dữ liệu đầu vào, yêu cầu có các trường *coord*, *name*, *value*.
 - *style* <Object>: Tùy chỉnh style cho đối tượng được vẽ lên Map.
3. `getObjects()`: Trả về FeatureObject từ OpenLayers, người dùng có thể truy xuất dữ liệu qua hàm `get('data')`.
4. `createLayer(layerType, layerSource)`: Tạo một lớp (*layer*) trên Map.
 - *layerType* <String>: Kiểu của Layer (*Tile*, *Vector*, *Image*, *VectorTile*).
 - *layerSource* <String>: Hỗ trợ việc cấu hình đơn giản cho nguồn (*source*). C9js hỗ trợ một số nguồn như *BingMaps*, *Stamen*, *TileJSON*, *TileArcGISRest*, *Vector*, *Cluster*, *ImageVector*, *OSM*.
5. `createLayerFromGeojson(url, data, style)`: Tương tự như hàm `createObject()` nhưng ở đây có thêm tham số *url* cho phép người dùng tải tập tin GeoJSON để liên kết với dữ liệu đầu vào. Là bản mở rộng từ hàm `createLayer()`.

- *url* <String>: Đường dẫn chứa tập tin GeoJSON.
- *data* <Object>:
 - *plain* <Object>: Dữ liệu thô, chứa các trường *coord*, *name*, *value*.
 - *file* <Object>: Dữ liệu đầu vào được tải từ tập tin
 - *process* <Function>: Hàm cho phép người dùng tiền xử lý dữ liệu, như thêm/xoá/sửa.
 - *condition* <Function>: Điều kiện để liên kết dữ liệu từ tập tin GeoJSON với dữ liệu đầu vào của người dùng.
- *style* <Object>: Tùy chỉnh style cho đối tượng được vẽ lên Map.

6. `updateGeojsonData(url, data, style)`: Tương tự như hàm `createLayerFromGeojson()`, dùng để cập nhật lại dữ liệu tại một đối tượng trên Map.

- *url* <String>: Đường dẫn chứa tập tin GeoJSON.
- *data* <Object>:
 - *plain* <Object>: Dữ liệu thô, chứa các trường *coord*, *name*, *value*.
 - *file* <Object>: Dữ liệu đầu vào được tải từ tập tin
 - *process* <Function>: Hàm cho phép người dùng tiền xử lý dữ liệu, như thêm/xoá/sửa.
 - *condition* <Function>: Điều kiện để liên kết dữ liệu từ tập tin GeoJSON với dữ liệu đầu vào của người dùng.
- *style* <Object>: Tùy chỉnh style cho đối tượng được vẽ lên Map.

7. `on(eventType, callback)`: Lắng nghe sự kiện và trả về dữ liệu tương ứng trên đối tượng được tương tác, ở đây là *FeatureObject* - đối tượng cung cấp bởi OpenLayers, người dùng có thể truy xuất dữ liệu đối tượng bằng hàm `get('data')`.

- *eventType* <String>: Loại sự kiện (hover, click, mousemove).
- *callback* <Function>: Chứa dữ liệu trả về.

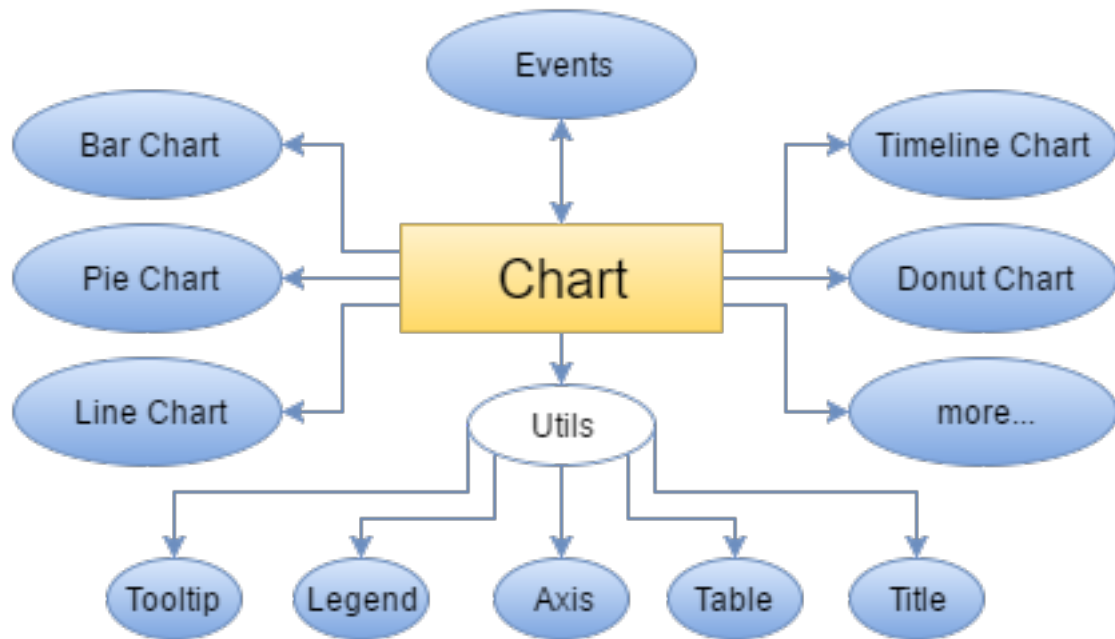
4 Hiện thực thư viện

4.1 Kiến trúc xây dựng

Thư viện C9js được nhóm xây dựng trên hai nguyên tắc chính, xuyên suốt cả quá trình phát triển: *Component-based* và *Data-driven*.

Component-based

Component-based^[?] được hiểu là xây dựng theo kiến trúc chia nhỏ thành các thành phần riêng biệt, hoạt động theo từng luồng chức năng riêng biệt, không tác động tới các thành phần khác. Việc xây dựng theo tiêu chí này giúp kiến trúc rõ ràng, mã nguồn trong sáng, việc mở rộng các thành phần về sau không ảnh hưởng tới các thành phần đã tồn tại.



Hình 4.1: Kiến trúc các thành phần của Chart trong C9js

Đối với thành phần Chart (Hình 4.1), có một *Base Class* là *Chart* sẽ bao gồm:

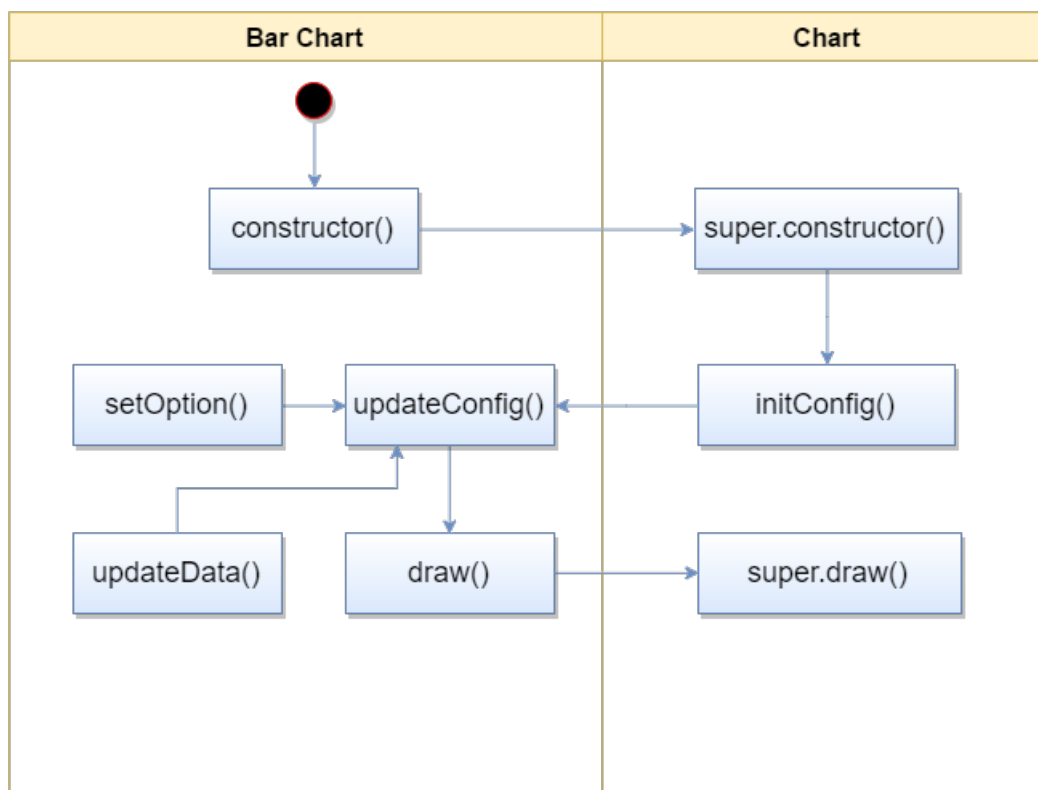
- Các thuộc tính chính:
 - *margin*: Khoảng cách tới khung chứa ngoài của Chart.

- *size*: Bao gồm chiều rộng (*width*) và chiều cao (*height*).
 - *colorRange*: Mảng chứa giá trị các màu để thể hiện lên các thành phần một cách nhất quán.
 - *id*: *id* của khung chứa Chart.
 - ...
- Các thành phần trong *Utils*:
- *Tooltip*: Bảng hiển thị khi lướt chuột qua các thành phần Chart.
 - *Legend*: Chú thích các dữ liệu.
 - *Axis*: Các trục tọa độ (*x*, *y*).
 - *Table*: Bảng tổng quan dữ liệu.
 - *Title*: Tiêu đề Chart.

Ngoài ra, còn có thành phần *Events* bao gồm các sự kiện tương tác (*click*, *hover*, *mousemove*) trên các thành phần của Chart. Khi các sự kiện này xảy ra, dữ liệu tương ứng tại vùng được tương tác sẽ được trả về cho người dùng.

Kế thừa từ *Base Class - Chart*, các đồ thị khác nhau như đồ thị cột (*Bar Chart*), đồ thị tròn (*Pie Chart*), đồ thị dòng (*Line Chart*), đồ thị thời gian (*Timeline Chart*), đồ thị hình khuyên (*Donut Chart*) sẽ là các đối tượng chính được lập trình viên sử dụng trong mã nguồn của họ. Ứng với mỗi thành phần sẽ có một tập tin mã nguồn tương ứng tổ chức theo cấu trúc thư mục giống như kiến trúc các thành phần của Chart trong C9js (Hình 4.1).

Ví dụ minh họa (Hình 4.2) mô tả một đối tượng *BarChart* được vẽ bởi C9js. Tại thời điểm khởi tạo, lớp (*Class*) sẽ gọi hàm khởi tạo của đối tượng *Base*, sau đó các thuộc tính mà phương thức mà mỗi đối tượng *Chart* đều có sẽ được tạo thông qua hàm *initConfig()*. Tiếp theo, các tùy chỉnh người dùng nhập vào lúc khởi tạo sẽ được kết hợp với các tùy chỉnh mặc định bởi hàm *updateConfig()* - vì các tùy chỉnh trong C9js thuộc kiểu *Object* nên hàm *updateConfig()* hỗ trợ tính năng trộn sâu (*Deep Merge*)^[?] nhằm đảm bảo các tùy chỉnh thống nhất và toàn vẹn. Cuối cùng, biểu đồ được hiển thị trên Web thông qua hàm *draw()*,



Hình 4.2: Luồng thực thi một đối tượng trong Chart của C9js

nhưng trước tiên các đối tượng của lớp *Base* sẽ được vẽ trước bằng *super.draw()*. Ngoài ra, C9js còn cung cấp các API như *setOption()* và *updateData()*, xét về bản chất cả hai hàm này đều cập nhật các thuộc tính trong một đối tượng của C9js nên sẽ gọi hàm *updateConfig()* để thực hiện thao tác này và cập nhật lại đồ thị bằng *draw()* .

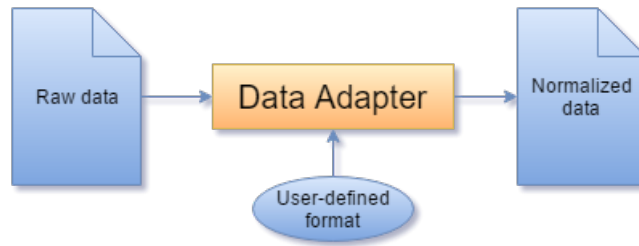
Đối với thành phần Map, chỉ có một *Class* chính là *Map* sẽ bao gồm các thành phần cần cho một bản đồ cơ bản:

- *Bản đồ* : chứa tất cả các thành phần của bản đồ (tương ứng với thành phần *ol.Map* của Openlayer 3).
- *Vị trí nhìn (View)*: C9js hỗ trợ tập trung góc nhìn (zoom) đến một vị trí kinh độ, vĩ độ (longitude, latitude) cụ thể do người dùng chỉ định (tương ứng với thành phần *ol.View* của Openlayer 3).
- *Các lớp (Layer)*: Các lớp trên bản đồ, trong mỗi lớp phải định nghĩa nguồn (source) của lớp đó như là OSM, BingMaps, Vector,... (xem mục 3.4), có thể có nhiều lớp, đè lên lẫn nhau tùy vào thứ tự tạo lớp.
- *Các đối tượng (Feature)* như đã trình bày ở mục 3.4.
- *Cửa sổ bật lên (Popup)*: Hiện lên cửa sổ hiển thị các dữ liệu với định dạng do người dùng chỉ định (thông qua định nghĩa thuộc tính format lúc truyền các thuộc tính cho Map).

Ngoài ra còn có thành phần *Events* bao gồm các sự kiện như: *click*, *pointermove* (vị trí chuột nằm trên một đối tượng) và *postrender* (sau khi bản đồ được hiển thị hoàn toàn). Sau khi sự kiện xảy ra, đối tượng tương tác sẽ được trả về cho người dùng.

Data-driven

Data-driven^[?] là mô hình lập trình tập trung vào xử lý và thao tác trên dữ liệu, dữ liệu sinh ra sẽ kiểm soát luồng thực thi chương trình.



Hình 4.3: Mô hình chuẩn hoá dữ liệu bởi *Data Adapter* trong C9js

Đối tượng	Các trường sau khi chuẩn hoá
Bar Chart	name, value
Line Chart	name, x(giá trị theo trục x), y(giá trị theo trục y)
Timeline Chart	name, value
Donut Chart	name, value
Pie Chart	name, value
Map	name, value, coor(toạ độ)

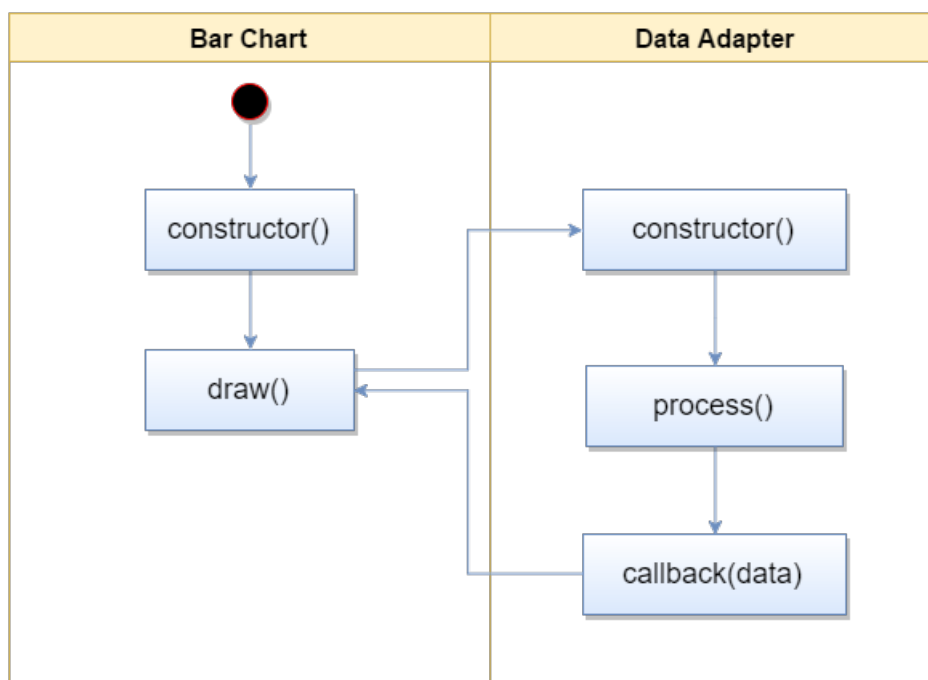
Bảng 4.1: Các trường trong một tập dữ liệu đã chuẩn hoá

Trong C9js, có một thành phần gọi là *Data Adapter*. Đây là thành phần đảm nhận nhiệm vụ xử lý toàn bộ về phần dữ liệu. Từ dữ liệu thô người dùng nhập vào hoặc có thể từ một tập tin hay một yêu cầu bên ngoài (*external request*), *Data Adapter* sẽ kết hợp với định dạng tự định nghĩa bởi người dùng (*user-defined format*) để đưa ra dữ liệu được chuẩn hoá và sử dụng trong các thành phần khác như Chart, Map (Hình 4.3). Chính thành phần này tạo nên tính năng *DataAdapter*, là một tính năng mới, giúp gạt đi những giới hạn về định nghĩa dữ liệu đầu vào mà các thư viện khác thường gặp.

Về định dạng tự định nghĩa bởi người dùng, ở ví dụ 3.11 có chỉ cách thực thi, thông qua các trường như *name*, *value* cần định nghĩa để tạo ra dữ liệu chuẩn hoá(*normalized data*). Dữ liệu chuẩn hoá trong C9js thực chất có các trường bắt buộc tùy theo từng đối tượng Chart (*Bar Chart*, *Line Chart*, ...) hay *Map* được liệt kê như bảng 4.1.

Data Adapter còn có chức năng tạo một trường gọi là *data-ref*. Vì các thành phần trong

C9js đều tương tác được với nhau thông qua thao tác của người dùng, nên cần thiết phải có một cầu nối giữa các thành phần này dựa trên một sự nhất quán về dữ liệu, hay nói cách khác, phải dựa trên một đặc điểm chung liên quan tới dữ liệu. Mỗi một giá trị (vì mỗi tập dữ liệu đầu vào đều được chuyển thành các mảng chứa các dữ liệu con) sẽ được *đánh dấu* bằng một *UUID*^[?] có định dạng `c9-xxxxxx-4xxx-yxxx`, khi đó thông qua các *selection* của *d3* ta có thể dễ dàng tham khảo đến tất cả các thành phần đang có chung *UUID* về dữ liệu đang được tương tác.



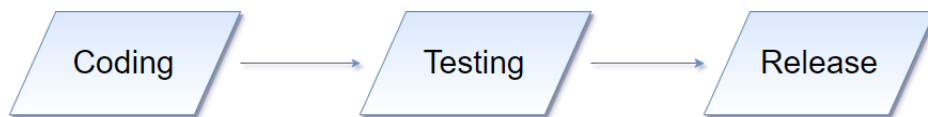
Hình 4.4: Quá trình nhận dữ liệu được chuẩn hoá trong C9js

Bây giờ, xét về luồng đi của dữ liệu trong C9js. Qua hình 4.4, khi người dùng khởi tạo một đối tượng trong C9js và gọi hàm *draw()* để hiển thị ra Web, lúc đó một đối tượng *DataAdapter* sẽ được khởi tạo và xử lý dữ liệu trên đối tượng ban đầu. Quá trình *process* bao gồm các thao tác như xử lý tập tin (nếu người dùng truyền dữ liệu thông qua tập tin), xử lý yêu cầu (nếu là *external request*) và cuối cùng là chuẩn hoá dữ liệu như đã trình bày ở trên. Để đảm bảo dữ liệu luôn sẵn sàng trước khi hiển thị, nhóm sử dụng tính năng *callback*^[?] trong JavaScript. Sau khi dữ liệu đã được xử lý xong, các thao tác để hiển thị các thành phần của Chart, Map được đưa vào trong *callback* và thực thi với *data* sau cuối.

Đối với thành phần Map, dữ liệu sẽ được móc nối với các đối tượng, thông qua phương thức `get('data')` của đối tượng, người dùng có thể lấy được dữ liệu mà họ đã nhập vào (từ dữ liệu nhập tay hoặc từ tập tin). Ngoài ra người dùng có thể lấy được kiểu của đối tượng thông qua phương thức `get('type')` trên đối tượng đó, kết quả trả về sẽ là *c9-geojson* nếu đó là đối tượng *GeoJSON*, tương tự đối với các đối tượng thông thường sẽ là: *c9-marker*, *c9-line*, *c9-polygon*, *c9-multipolygon*.

4.2 Hiện thực thư viện

Quá trình hiện thực thư viện C9js được nhóm chia làm 3 giai đoạn riêng biệt nhằm tối ưu hoá kết quả đạt được từ từng giai đoạn (Hình 4.5).

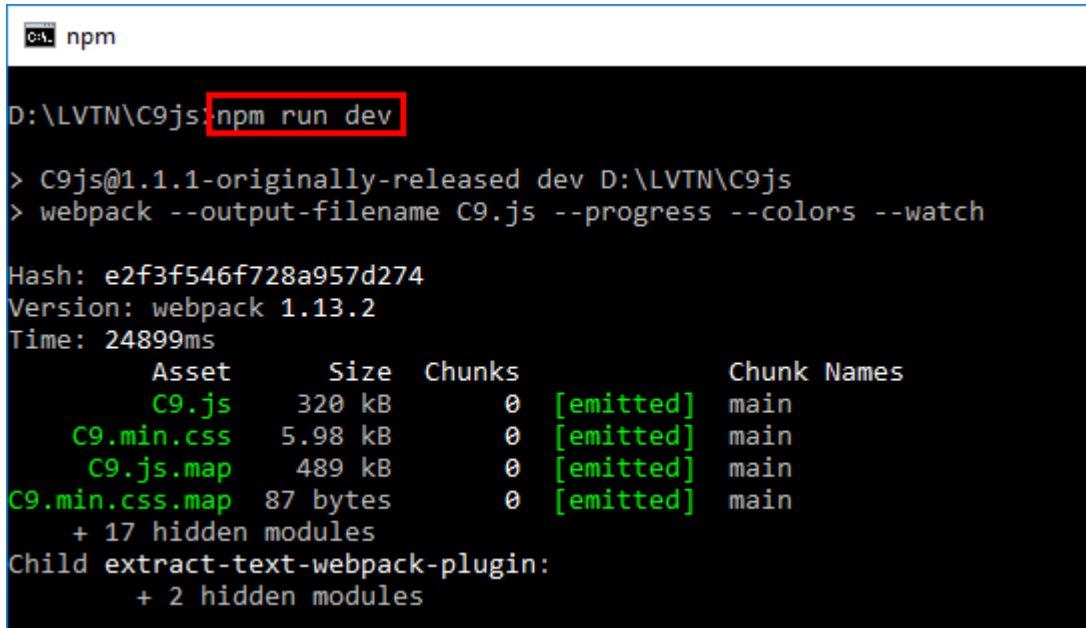


Hình 4.5: Ba giai đoạn hiện thực thư viện C9js

4.2.1 Hiện thực (Coding)

Dựa trên các kiến thức nền tảng đã trình bày và kiến trúc hệ thống đề xuất xây dựng, C9js được hiện thực dưới môi trường phát triển (*environment*) sử dụng *Webpack*, *D3.js*, *OpenLayers* (Hình 4.6).

Lệnh `npm run dev` sẽ chạy dựa trên tập tin cấu hình cho Webpack (*webpack.config.js*) đã dựng sẵn. Ở đây, Webpack sẽ gộp chung các tập tin mã nguồn JavaScript thành một tập tin có tên *C9.js*, sau đó nén và sinh tập tin *source map*^[?] nhằm giảm thời gian truy vấn dòng lệnh thực thi trên mã nguồn chính. Như hình 4.6, Webpack sẽ quan sát khi có thay đổi trong mã nguồn trong thư mục `src` nhờ vào cấu hình `-watch` và tự động sinh ra các tập tin như *C9.js*, *C9.js.map*, *C9.min.css*, *C9.min.css.map* trong thư mục `dist`. Nhóm chỉ tập trung vào hiện thực các tính năng mà không cần quan tâm tới việc gộp, nén hay sinh các tập tin cần thiết trong quá trình phát triển.



```

C:\> npm

D:\LVTN\C9js> npm run dev

> C9js@1.1.1-originally-released dev D:\LVTN\C9js
> webpack --output-filename C9.js --progress --colors --watch

Hash: e2f3f546f728a957d274
Version: webpack 1.13.2
Time: 24899ms

   Asset      Size  Chunks             Chunk Names
   C9.js      320 kB       0  [emitted]  main
  C9.min.css   5.98 kB       0  [emitted]  main
   C9.js.map   489 kB       0  [emitted]  main
C9.min.css.map 87 bytes       0  [emitted]  main
+ 17 hidden modules
Child extract-text-webpack-plugin:
+ 2 hidden modules
  
```

Hình 4.6: Khởi chạy môi trường phát triển với Webpack

4.2.2 Kiểm thử (Testing)

Các testcase được viết bằng *Jasmine*. Như ví dụ 4.1 mô tả một testcase với mong muốn tạo một đối tượng *BarChart* với id là *#chart*.

```

1 describe('C9 Unit Test - Bar Chart', function() {
2
3     it('should create Bar Chart with id #chart', function() {
4
5         var data = [
6             {name: 'A', value: .08167},
7             {name: 'B', value: .01492},
8             {name: 'C', value: .02782}
9         ];
10
11         var option = {
12             id: "chart",
13             width: 1100,
  
```



```

14         height: 500,
15         data: {
16             plain: data,
17         }
18     };
19
20     var barChart = new C9.BarChart(option);
21     barChart.draw();
22
23     expect(barChart).toEqual(jasmine.any(C9.BarChart));
24     expect(barChart).not.toBeNull();
25     expect(barChart.id).toEqual('#chart');
26
27 });
28 });

```

Mã nguồn 4.1: Một testcase trong C9js được viết bằng Jasmine

Các testcase được nhóm hiện thực theo đúng nguyên tắc trong thiết kế hệ thống đã đề ra, tức là hướng tới đối tượng hiện thực. Ứng với mỗi thành phần trong mã nguồn sẽ có một bộ testcase (*test-suite*) mô tả các hành vi của một thành phần trong C9js, như ví dụ 4.1 là một *test-suite* về *Bar Chart*.

Các *test-suite* này được quản lý và kiểm soát trạng thái bởi *Karma*. Thông qua lệnh *karma start*, *Karma* được khởi chạy với tập tin cấu hình (*karma.conf.js*) (Hình 4.7). Như có thể thấy, *Karma* khởi động một *server* và kiểm soát trạng thái các *test-suite* trên trình duyệt mà nhóm đã chỉ định, ở đây là *PhantomJS*. Kết quả, ứng với mỗi *test-suite*, tình trạng *SUCCESS-FAILED* của mỗi *test-case* được *Karma* tổng hợp lại, đồng thời đi kèm với thời gian thực thi cho mỗi *test-case*. Đặc biệt, giống như *Webpack*, *Karma* cũng hỗ trợ chức năng *live server*, tức là không cần phải chạy lại (*re-run*) quá trình kiểm thử mỗi lần thay đổi mã nguồn, mà *Karma* sẽ tự kiểm soát và quản lý thao tác này.

```
C:\WINDOWS\system32\cmd.exe

D:\LVTN\C9js>karma start

START:
11 12 2016 16:15:23.616:INFO [karma]: Karma v1.3.0 server started at http://localhost:9
11 12 2016 16:15:23.620:INFO [launcher]: Launching browser PhantomJS with unlimited con
11 12 2016 16:15:23.933:INFO [launcher]: Starting browser PhantomJS
11 12 2016 16:15:39.877:INFO [PhantomJS 2.1.1 (Windows 8 0.0.0)]: Connected on socket /
05425
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 0 of 33 SUCCESS (0 secs / 0 secs)

  C9 Unit Test - Axis
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 1 of 33 SUCCESS (0 secs / 0.069 secs)
  C9 Unit Test - Axis
    ✓ should create x axis
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 2 of 33 SUCCESS (0 secs / 0.095 secs)
    ✓ should create y axis
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 3 of 33 SUCCESS (0 secs / 0.114 secs)
    ✓ should hide all axis
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 4 of 33 SUCCESS (0 secs / 0.129 secs)
    ✓ should show axis grid
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 5 of 33 SUCCESS (0 secs / 0.15 secs)
    ✓ should change format axis to $xxx$
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 6 of 33 SUCCESS (0 secs / 0.169 secs)
    ✓ should rotate x axis by -30 degree

  C9 Unit Test - Bar Chart
PhantomJS 2.1.1 (Windows 8 0.0.0): Executed 7 of 33 SUCCESS (0 secs / 0.188 secs)
  C9 Unit Test - Bar Chart
    ✓ should create Bar Chart with id #chart
```

Hình 4.7: Kiểm soát Unit Testing với Karma

Bước cuối cùng trong giai đoạn Testing là phát sinh tập tin báo cáo (*lcov.info*) để *codecov.io* sử dụng cho mục đích thống kê mức độ bao phủ trên mã nguồn. Chi tiết về thống kê sẽ được trình bày trong mục 5.1.

4.2.3 Triển khai (Release)

Việc triển khai thư viện C9js qua nhiều kênh khác nhau được hỗ trợ bởi *semantic-release*. *Semantic-release* chia quá trình này làm 3 bước chính:

1. *semantic-release pre*: Dựa trên các thay đổi của mã nguồn, quyết định phiên bản mới cho thư viện.
2. *npm publish*: Triển khai thư viện với phiên bản mới qua kênh *npm*.
3. *semantic-release post*: Tạo *changelog* và một bản triển khai trên *GitHub*.

Tổng quan

Với tư tưởng phát triển mã nguồn mở tích hợp và triển khai liên tục, nhóm sử dụng *TravisCI* nhằm kiểm soát toàn bộ 3 giai đoạn trên. *TravisCI* tiến hành qua 3 bước (Mã nguồn 4.2):

1. *Before*: Khi nhận được một *commit* từ *GitHub*, *TravisCI* tiến hành cài đặt môi trường và các phụ thuộc (*dependency*) cần thiết.
2. *Running*: Tiến hành gộp, nén tập tin (*npm run build*), thực hiện Unit Testing và tạo báo cáo độ bao phủ mã nguồn (*npm run ci*).
3. *After*: Gửi báo cáo cho *codecov.io* để tạo trang tổng quan và cuối cùng triển khai thông qua *semantic-release*. Thông báo đến các bên liên quan nếu cài đặt *notifications: true*.

```
1 | sudo: false
2 | language: node_js
3 | cache:
```

```
4   directories:
5     - node_modules
6 branches:
7   only:
8     - master
9 notifications:
10    email: true
11 node_js:
12   - '4'
13 before_install:
14   - npm i -g npm@~2.0.0
15 before_script:
16   - npm prune
17 script:
18   - npm run build
19   - npm run ci
20 after_success:
21   - npm run report-coverage
22   - npm run semantic-release
```

Mã nguồn 4.2: Cấu hình TravisCI kiểm soát cả 3 giai đoạn

5 Kết luận (7)

5.1 Đánh giá (Unit Testing vs So sánh với các CTLQ) (5)

- Unit Testing - So sánh với c3 => Rút ra các factor để đánh vào đáp ứng Mục tiêu => Tính năng mới => Rút ngắn time - So sánh với WIND => Lỗi thời => Ưu => Nhược -> Hướng phát triển

5.2 Hướng phát triển (1)

5.3 Kết luận (1)

Tài liệu tham khảo

- [1] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 322(10):891–921, 1905.