**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**



**TIỂU LUẬN TỐT NGHIỆP**

XÂY DỰNG ỨNG DỤNG WEB CHAT

TÍCH HỢP WEBSITE THƯƠNG MẠI ĐIỆN TỬ

Giảng viên:

**TS. ĐÀM QUANG HỒNG HẢI**

Sinh viên thực hiện:

**Phạm Kim Chấn Nguyên** **12520944**

**Võ Đức Hòa 12520147**

Lớp: **MMT2012**

Khóa: **2012-2017**

**MỞ ĐẦU**

Trong thời đại hiện nay khi mà công nghệ thông tin đóng vai trò quan trọng góp phần vào sự phát triển nhiều mặt của đất nước, đi cùng với sự phát triển kinh tế, thì việc để tạo ra một môi trường thuận lợi trong việc giới thiệu mặt hàng, thông tin của công ty doanh nghiệp, thì tất nhiên website được xem là một phần không thể thiếu. Ngày nay, trên thị trường online có vô số website kinh doanh, và tất nhiên, sự cạnh tranh là không hề nhỏ. Do đó, nếu muốn kinh doanh hiệu quả, bạn cần phải có những biện pháp để trang web của mình trở nên hoàn thiện hơn, nâng cao chất lượng trải nghiệm cho người dùng. Và để đáp ứng được nhu cầu thông tin, thái độ, quan điển của khách hàng một cách nhanh chóng nhất khi khách hàng lướt trên website. Trước đây, hầu như tất cả các website (kể cả những website bán hàng) thì việc tương tác với người dùng hầu như là không có. Website chỉ đơn thuần là một “quầy trưng bày” online mà ở đó, người dùng, cũng chính là khách hàng chỉ có quyền “xem” mà không có quyền “hỏi” . Dần dần, trên web được tích hợp thêm tính năng bình luận. Điều này là khá hữu ích và được người dùng đánh giá cao. Theo đó, khách hàng khi vào xem trang web, có thể để lại bình luận cá nhân, đó có thể là những lời khen, lời chê hoặc những lời góp ý,… điều này thực sự có ý nghĩa đối với những người xây dựng web, bán hàng trên web, và ý nghĩa cả với những người sử dụng. Bằng cách này, khách hàng có thể bày tỏ quan điểm cá nhân về sản phẩm, dịch vụ trên web, nơi mà họ đang có như cầu tìm hiểu thông tin, tìm sản phẩm, đồng thời có thể đưa ra những thắc mắc, ý kiến về sản phẩm, về dịch vụ để được giải đáp. Nhờ đó, mà những người làm kinh doanh trên website sẽ hiểu thêm được phần nào về nhu cầu của khách hàng, kịp thời đưa ra những giải đáp cho thắc mắc của khách hàng, tiếp thu những lời khen ngợi đồng thời lắng nghe những góp ý để từng bước cải thiện chất lượng dịnh vụ kinh doanh tốt hơn. Đó là bước đà quan trọng để tăng tỉ lệ chốt đơn hàng, tăng doanh thu, phát triển kinh doanh. Việc để khách hàng bình luận trên website là khá hiệu quả, đến nay, hình thức này vẫn còn được dùng phổ biến trên nhiều trang web, tuy nhiên nó cũng có một số những nhược điểm. Người dùng khi để lại bình luận thì tất cả những ai khi truy cập vào website sẽ đều nhìn thấy những bình luận đó do vậy sẽ mất đi sự riêng tư. Nhất là những sản phẩm tế nhị, thì chắc khó có ai muốn để lại bình luận.

Vì vậy để có một môi trường mà khách hàng có bất kì thắc mắc hay đề nghị, yêu cầu gì đối với đơn vị chủ quản website đều có thể dễ dàng truyền tải. Và đặc biệt là đảm bảo được tính riêng tư cho khách hàng, vì là cửa sổ chat cho nên chỉ có người quản lý website mới đọc được, còn những người truy cập khác sẽ không thể thấy.

Qua các vấn đề được nêu trên thì nhóm chúng em chọn đề tài “Xây dựng ứng dụng chat hỗ trợ trực tuyến”.

Trong quá trình thực hiện đề tài, do khó khăn về ngôn ngữ, kiến thức nền tảng, thời gian thực hiện nên sẽ không tránh khỏi sai sót, rất mong quý thầy cô thông cảm và đóng góp ý kiến để đề tài hoàn chỉnh hơn.

Chúng em xin chân thành cảm ơn!

Tp Hồ Chí Minh, ngày 01 tháng 01 năm 2018

Nhóm sinh viên

**Phạm Kim Chấn Nguyên – Võ Đức Hòa**

LỜI CẢM ƠN

-----🙚✯🙘-----

Trước tiên, em muốn gửi lời cảm ơn sâu sắc nhất đến thầy TS. Đàm Quang Hồng Hải, đã tận tình hướng dẫn chúng em trong quá trình thực hiện đề tài này.

Tiếp theo, chúng em xin bày tỏ lời cảm ơn sâu sắc đến những thầy cô trong Khoa Mạng Máy Tính và Truyền Thông cũng như các thầy cô của trường Đại học Công Nghệ Thông Tin – Đại học Quốc Gia thành phố Hồ Chí Minh, đã giảng dạy chúng em trong bốn năm qua, những kiến thức mà chúng em nhận được trên giảng đường đại học sẽ là hành trang giúp chúng em vững bước trong tương lai.

Cuối cùng, em xin kính chúc quý thầy cô dồi dào sức khỏe, đạt nhiều thành công trong sự nghiệp giảng dạy.

Chúng em xin chân thành cảm ơn !

Nhóm sinh viên

**Phạm Kim Chấn Nguyên – Võ Đức Hòa**

**NHẬN XÉT  
(Của giảng viên hướng dẫn)**

**-----🙚✯🙘-----**

TP.HCM, Ngày …… Tháng ……. Năm 2018

Giáo viên hướng dẫn

**NHẬN XÉT  
(Của giảng viên phản biện)**

**-----🙚✯🙘-----**

TP.HCM, Ngày …… Tháng ……. Năm 2018

Giáo viên phản biện

**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI 1](#_Toc503616575)

[**1.1.** **Tên Đề Tài** 1](#_Toc503616576)

[**1.2.** **Từ Khóa** 1](#_Toc503616577)

[**1.3.** **Mục Tiêu Đề Tài** 1](#_Toc503616578)

[**1.4.** **Đối Tượng Và Phạm Vi Nghiên Cứu** 1](#_Toc503616579)

[**1.5.1.** **Đối Tượng** 1](#_Toc503616580)

[**1.5.2.** **Phạm Vi Nghiên Cứu** 1](#_Toc503616581)

[**1.6.** **Cấu Trúc Tiểu Luận** 2](#_Toc503616582)

[CHƯƠNG 2: GIỚI THIỆU CÔNG NGHỆ VÀ LÝ THUYẾT 3](#_Toc503616583)

[**2.1.** **Giới thiệu Node.js** 3](#_Toc503616584)

[**2.2.** **Đặc điểm của Node.js** 4](#_Toc503616585)

[**2.2.1.** **Các thành phần quan trọng trong Node.js** 8](#_Toc503616586)

[**2.3.** **Các Object cơ bản** 9](#_Toc503616587)

[**2.3.1.** **Global Object (đối tượng toàn cục)** 9](#_Toc503616588)

[**2.3.2** **Sự kiện (Event)** 14](#_Toc503616589)

[**2.3.3** **Luồng ( Streams )** 16](#_Toc503616590)

[**2.3.4** **HTTP** 18](#_Toc503616591)

[**2.4.** **Socket.io** 24](#_Toc503616592)

[**2.4.1.** **Các đặc điểm của Socket.io** 25](#_Toc503616593)

[**2.4.2.** **Phương thức hoạt động của socket.io trong Node.js** 26](#_Toc503616594)

[**2.2** **Hệ Cơ Sở Dữ Liệu MONGODB** 27](#_Toc503616595)

[**2.2.1.** **Thiết kế lược đồ** 29](#_Toc503616596)

[**2.2.2.** **Chỉ mục** 32](#_Toc503616597)

[**2.2.3.** **Sao chép** 36](#_Toc503616598)

[**2.2.4.** **Truy vấn** 38](#_Toc503616599)

[**2.3.** **GIT – hệ thống quản lý phiên bản phân tán** 41](#_Toc503616600)

[**2.3.1.** **Giới thiệu về GIT** 41](#_Toc503616601)

[**2.3.2** **Đặc điểm của GIT** 42](#_Toc503616602)

[**2.3.3.** **Các thành phần quan trọng của Git** 45](#_Toc503616603)

[**2.4.** **Heroku** 51](#_Toc503616604)

[**2.4.1.** **Giới thiệu heroku** 51](#_Toc503616605)

[**2.4.2.** **Cấu trúc của horuku.** 52](#_Toc503616606)

[**2.4.3.** **Một số tính năng của horuku.** 52](#_Toc503616607)

[CHƯƠNG 3. PHÂN TÍCH THÀNH PHẦN DỮ LIỆU 53](#_Toc503616608)

[**3.1.** **Mô tả chương trình.** 53](#_Toc503616609)

[**3.2.** **Chức năng hệ thông** 53](#_Toc503616610)

[**3.3.** **Các thành phần dữ liệu** 53](#_Toc503616611)

[CHƯƠNG 4: ỨNG DỤNG WEB CHAT TÍCH HỢP WEBSITE 56](#_Toc503616612)

[**4.1.** **Giới thiệu.** 56](#_Toc503616613)

[**4.1.1.** **Các chức năng chính.** 56](#_Toc503616614)

[**4.1.2.** **Chi tiết chương trình.** 56](#_Toc503616615)

[CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 66](#_Toc503616616)

[**5.1.** **Kết luận.** 66](#_Toc503616617)

[**5.2.** **Hướng phát triển.** 66](#_Toc503616618)

**DANH MỤC HÌNH**

[Hình 2.1giới thiệu Node.js 3](#_Toc503617825)

[Hình 2.2Lược đồ mô tả các thành phần quan trọng trong Node.js 9](#_Toc503617826)

[Hình 2.3Quá trình kết nối của client tới socket.io 26](#_Toc503617827)

[Hình 2.4 Minh họa bộ sưu tập 30](#_Toc503617828)

[Hình 2.5 Mô hình Master – Slave hai nút 37](#_Toc503617829)

[Hình 2.6 Mô hình Master – Slave bốn nút 37](#_Toc503617830)

[Hình 2.7Mô hình Replica Sets hai nút 38](#_Toc503617831)

[Hình 2.8Replica Sets – Bầu chọn master mới 38](#_Toc503617832)

[Hình 2.9 Server chính trở thành server cấp 2 38](#_Toc503617833)

[Hình 2.10 giới thiệu GIT 41](#_Toc503617834)

[Hình 2.11 hoạt động của GIT tại khu vực tôt chức 44](#_Toc503617835)

[Hình 2.12 thao tác vận hành trong GIT 45](#_Toc503617836)

[Hình 2.13 các thành phần trong GIT 46](#_Toc503617837)

[Hình 2.14 kho repo 47](#_Toc503617838)

[Hình 2.15 sơ đồ Remote repository và Local repository 48](#_Toc503617839)

[Hình 2.16 sơ đồ nhánh trong GIT 49](#_Toc503617840)

[Hình 2.17 sơ đồ vận hành của commit trong GIT 49](#_Toc503617841)

[Hình 2.18 cách thức vận hành của working tree và Index 51](#_Toc503617842)

[Hình 2.19 giới thiệu heroku 51](#_Toc503617843)

[Hình 4.1 mô hình tương tác của ứng dụng………………………………………………………56](#_Toc503617844)

[Hình 4. 2 giao diện đăng nhập 57](#_Toc503617845)

[Hình 4. 3 giao diện đăng ký 58](#_Toc503617846)

[Hình 4.4 giao diện quản lý, chat của quản lý 59](#_Toc503617847)

[Hình 4.5 giao diện hiển thị thông tin, trạng thái của tài khoản 60](#_Toc503617848)

[Hình 4.6 giao diện tìm kiếm, tạo kênh 60](#_Toc503617849)

[Hình 4.7 giao diện tạo kênh 61](#_Toc503617850)

[Hình 4.8 các tầm vực của kênh 61](#_Toc503617851)

[Hình 4.9 giao diện tầm vực của kênh 62](#_Toc503617852)

[Hình 4.10 giao diện nhập thông tin để chat trực tuyến trên website khác 63](#_Toc503617853)

[Hình 4.11 giao diện chat trực tuyến gắn trên website khác 64](#_Toc503617854)

[Hình 4.12 giao diện chat trực tuyến 65](#_Toc503617855)

**DANH MỤC BẢNG**

[**Bảng 2.1.** Bảng chấm điểm cho 2 đoạn code chạy Node.js và PHP 16](#_Toc503604856)

# CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

* 1. **Tên Đề Tài**

Xây Dựng Ứng Dụng Web Chat

Tích Hợp Website Thương Mại Điện Tử

* 1. **Từ Khóa**

Hệ cở sở dữ liệu mã nguồn mở NoSQL – MongoDB, Mã nguồn mở được xây dựng trên nền tảng JavaScript V8 engine – NodeJS, Hệ thống quản lý phiên bản phân tán – Git, Ngôn ngữ lập trình kịch bản – Javascript, Ngôn ngữ lập trình siêu văn bản – PHP,

* 1. **Mục Tiêu Đề Tài**

Hiện nay nhu cầu thông tin của khách hàng mua sắm online của khách hàng càng ngày càng tăng từ đó nhu cầu để nắm bắt về tình hình, thông tin của sản phẩm mà người mua cần quan tâm cũng tăng lên. Vậy cũng từ phía hộ kinh doanh, doanh nghiệp cũng muốn nắm bắt được nhu cầu, ý kiến của khách hàng đối với sản phẩm của mình như thế nào.

Vì vậy nhiều trang website được tích hợp các mục đánh giá, bình luận và đặc biệt có thêm mục chat trực tuyến online được xây dựng nhằm giúp cho sự tương tác giữa 2 bên bán và mua 1 cách nhanh chóng.

Từ xu hứng trên, nhóm chúng em cũng quyết định xây dựng ứng dụng web chat tích hợp website.

* 1. **Đối Tượng Và Phạm Vi Nghiên Cứu**
     1. **Đối Tượng**
* Nghiên cứu, chat realtime sử dụng socket.io trong NodeJS
* Tìm hiểu cách thức, kỹ thuật lập trình bằng NodeJS
* Nghiên cứu, tìm hiểu và triển khai cơ sở dữ liệu bằng MongoDB
* Nghiên cứu và sử dụng Git
  + 1. **Phạm Vi Nghiên Cứu**

Do thời gian nghiên cứu có hạn, nên chúng em chỉ nghiên cứu và triển khai ứng dụng chat. Ứng dụng cho phép phía client có thể tạo tài khoản, truy cập kênh

để trao đổi với phía quản trị và phía quản trị cũng có thể tương tác, quản lý các tài khoản, truy cập phía client.

* 1. **Cấu Trúc Tiểu Luận**

Báo cáo tiểu luận gồm 5 chương:

Chương 1: Tổng quan về đề tài, mục tiêu đề tài, đối tượng nghiên cứu, phạm vi nghiên cứu.

Chương 2: giới thiệu các công nghệ và lý thuyết được sử dụng để hoàn thành được tiểu luận.

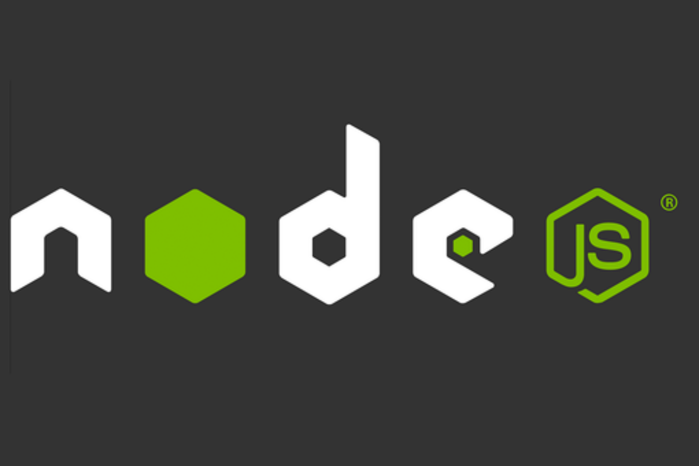
Chương 3: phân tích thành phần dữ liệu và hệ thống vận hành của ứng dụng

Chương 4: Mô tả ứng dụng và các chức năng đạt được của ứng dụng.

Chương 5: tổng kết các kết quả đạt được, nêu ra những hạn chế, khó khăn trong việc thực hiện đề tài, các hướng nghiên cứu và phát triển trong giai đoạn tiếp theo.

# CHƯƠNG 2: GIỚI THIỆU CÔNG NGHỆ VÀ LÝ THUYẾT

* 1. **Giới thiệu Node.js**



**Hình 2.1** giới thiệu Node.js

**Node.js** là một hệ thống phần mềm được thiết kế để viết các ứng dụng internet có khả năng mở rộng, đặc biệt là máy chủ web. Chương trình được viết bằng JavaScript, sử dụng kỹ thật điều khển theo sự kiện, nhập/xuất không đồng bộ để tối tiểu tổng chi phí và tối đại khả năng mở rộng. Node.js bao gồm có V8 JavaScript engine của Google, libUV, và vài thư viện khác.

**Node.js** được tạo bởi Ryan Dahl từ năm 2009, và được phát triển dưới sự bảo trợ của Joyent.

Mục tiêu ban đầu của Dahl là làm cho trang web có khả năng push như trong một số ứng dụng web như Gmail. Sau khi thử với vào ngôn ngữ Dahl chon javascript vì một API Nhập/Xuất không đầy đủ. Điều này cho phép anh có thể định nghĩa một quy ước Nhập/Xuất điều khiển theo sự kiện, non-blocking.

Vài môi trường tương tự được viết trong các ngôn ngữ khác gồm Twisted cho Python, Perl Object Environment cho Perl, libevent cho C và EventMachine cho Ruby. Khác với hầu hết các chương trình Javascript, Nodejs không chạy trên một trình duyệt mà chạy trên Server. Node.js sử dụng nhiều chi tiết kỹ thuật của CommonJS. Nó cung cấp một môi trường REPL cho phép kiểm thử tương tác.

Ngắn gọn hơn thì Node.js là một nền tảng dựa vào Chrome Javascript runtime để xây dựng các ứng dụng nhanh, có độ lớn. **Node.js** sử dụng các phần phát dinh các sự kiện (event-driven), mô hình non-blocking I/O để tạo ra các ứng dụng nhẹ và hiệu quả cho các ứng dụng về dữ liệu thời gian thực chạy trên các thiết bị phân tán.

**Node.js** cũng cung cấp cho chúng ta các Module Javascript đa dạng, có thể đơn giản hóa sự phát triển của các ứng dụng web sử dụng Node.js với các phần mở rộng.

**Node.js = Môi trường Runtime + Các thư viện Javascript**

**Node.js** được InfoWord bình chọn là “Công nghệ của năm” năm 2012.

Một khảo sát được tiến hành vào đầu năm 2017 bởi Node.js Foundation thì hiện nay Node.js đang nổi lên như một Framework phát triển dành cho việc chuyển đổi kỹ thuật số với nhiều ứng dụng đa dạng.

* 1. **Đặc điểm của Node.js**
* **Lập trình không đồng bộ và Phát sinh sự kiện (Event Driven):** Là lợi thế nếu ta đã quen thuộc với các phương pháp lập trình không đồng bộ. Tất cả các APIs của thư viện Node.js đều không đồng bộ, nghĩa là không blocking (khóa). Nó rất cần thiết vì Node.js không bao giờ đợi một API trả về dữ liệu. Server chuyển sang một API sau khi gọi nó và có cơ chế thông báo về sự kiện của Node.js giúp server nhận được phản hồi từ các API gọi trước đó. Ví dụ, nếu ta đang đọc một tập tin trên hệ thống tập tin, ta phải chỉ định một chức năng gọi lại đó là thực hiện khi đã hoàn thành các hoạt động đọc.
* **Các tiến trình đơn giản nhưng hiệu quả cao:** Node.js sử dụng một mô hình luồng đơn (single thread) với các sự kiện lặp. các cơ chế sự kiện giúp Server trả lại các phản hồi với một cách không khóa và tạo cho Server hiệu quả cao ngược lại với các cách truyền thống tạo ra một số lượng luồng hữu hạn để quản lý request. Nodejs sử dụng các chương trình đơn luồng và các chương trình này cung cấp các dịch vụ cho số lượng request nhiều hơn so với các Server truyền thống như Apache HTTP Server.
* **Chạy rất nhanh:** Dựa trên V8 Javascript Engine của Google Chrome, thư viện Node.js rất nhanh trong các quá trình thực hiện code.
* **Không đệm**: Ứng dụng Node.js không lưu trữ các dữ liệu buffer.
* **Với Node.js, Ta phải làm mọi thứ:** Node.js chỉ là môt trường – điều đó có nghĩa là ta phải tự làm tất cả. Đó không phải là một server http mặc định hoặc là bất cứ server nào khác. Điều này có thể là hơi khó hiểu với người mới, nhưng thành công thực sự của nó là đưa lại một hiệu năng đáng kinh ngạc. Một đoạn script có thể xử lý tất cả các kết nối với client. Điều này làm giảm đáng kể số lượng tài nguyên được sử dụng trong ứng dụng. Ví dụ, một ứng dụng Node.js.

*var i, a, b, c, max;*

*max = 1000000000;*

*var d = Date.now();*

*for (i = 0; i < max; i++)*

*{*

*a = 1234 + 5678 + i;*

*b = 1234 \* 5678 + i;*

*c = 1234 / 2 + i;*

*}*

*console.log(Date.now() - d);*

dưới đây là đoạn mã tương đương PHP:

*$a = null;*

*$b = null;*

*$c = null;*

*$i = null;*

*$max = 1000000000;*

*$start = microtime(true);*

for ($i = 0; $i < $max; $i++) {

$a = 1234 + 5678 + $i;

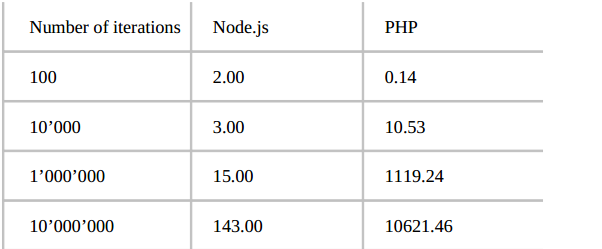
$b = 1234 \* 5678 + $i;

$c = 1234 / 2 + $i;

}

var\_dump(microtime(true) - $start);

giờ ta xem chấm điểm benchmark cho hai đoạn code trên chạy trên hai môi trường khác nhau:



***Bảng 2. 1.*** *Bảng chấm điểm cho 2 đoạn code chạy Node.js và PHP*

Khi thực hiện chạy hai đoạn code trên từ command line (console command) nên không có trễ thao tác thực thi. Nhưng khi chạy từng thử nghiệm 10 lần và lấy kết quả trung bình. PHP nhanh hơn trong các lần chạy với số lượng nhỏ vòng lặp. Nhưng vấn đề thay đổi khi số lượng vòng lặp tăng lên, số lần xử lý tăng lên thì PHP chạy chậm hơn rất nhiều trong kho Node.js có tốc độ đáng kinh ngạc. Sau tất cả thao tác, PHP chậm hơn 93% so với Node.js.

* **Module:** Node.js sử dụng một kiến trúc module để đơn giản hóa việc tạo ra các ứng dụng phức tạp. Module giống như các thư viện trong C, hoặc các đon vị trong Pascal. Mỗi module có chứa một tập hợp các chức năng liên quan đến “đối tượng” của các module. Ví dụ, các module http chứa các chức năng cụ thể cho HTTP. Node.js cung cấp một vài module cơ bản để giúp ta truy cập các tập tin trên hệ thống tập tin, tạo ra trình điều khiển server HTTP và TCP / UDP và thực hiện các chức năng hữu ích khác.

Để gọi một module thật dễ dàng, chỉ cần gọi hàm ***require()*** như sau:

*Var http = require(‘http’);*

Hàm require() trả về tham chiếu đến các module quy định. Trong trường hợp của mã này một tham chiếu đến các module http được lưu trữ trong biến http. Trong đoạn code trên, ta đã truyền tên của module vào trong hàm require(). Việc này chỉ định cho Node.js tìm trong thư mục nodu\_modules module tương ứng để thực hiện. Nếu node không thấy module tương ứng trong thư mục thì nó sẽ tìm trên **global module.** Ta cũng có thể chỉ định một module qua một file vật lý qua đường dẫn tương đối hay tuyệt đối như sau:

var myModule = require('./myModule.js');

Module được đóng gói từng phần mã. Đoạn mã nằm trong một module chủ yếu là private – có nghĩa là chức năng và biến được định nghĩa trong họ chỉ có thể truy cập từ bên trong của các module. Tuy nhiên, bạn có thể tiếp xúc với chức năng và / hoặc các biến được sử dụng từ biên ngoài của module. Để làm như vậy, phải sử dụng các đối tượng export với các thuộc tính và phương thức của nó với từng phần mã mà bạn muốn gọi từ bên ngoài. Hãy xem xét các module ví dụ sau:

*var PI = Math.PI;*

*exports.area = function (r) {*

*return PI \* r \* r;*

*};*

*exports.**circumference = function (r) {*

*return 2 \* PI \* r*

*};*

Trong ví dụ trên PI là biến private và chỉ được sử dụng bên trong đoạn mã, trong đó có hai hàm area() và circumference() được từ khóa export chỉ định thì sẽ có thể truy cập được từ bên ngoài.

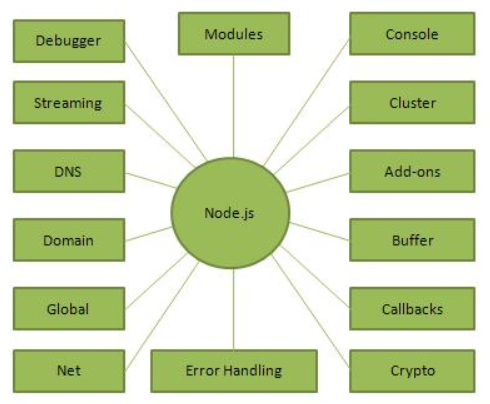
* **Global Scope:** Node.js là một môi trường chạy javascript với google V8 engine do đó hộ trợ chạy được ở server side. Do đó ta cũng nên tuân thủ các kinh nghiệm mà đã có lập trình với các ứng dụng client-side. Ví dụ, khi tạo các biến global trong node không phải lúc nào cũng có thể tạo được. Nhưng ta có thể tạo dễ dang các biến hoặc hàm global với cách bỏ từ khóa var trước các biến như sau:

*globalVariable = 1;*

*globalFunction = function(){…};*

Nhưng các biến global nên tránh sử dụng, và cẩn thận khi khai báo biến thì dùng từ khóa var để thực hiện.

* **Cộng đồng:** Cộng đồng phát triển Node.js chủ yếu tập trung ở hai nhóm google: Nodejs và Nodejs-dev, một kênh IRC là #Node.js trên mạng freenode. Có hội thao về Node.js là NodeConf được tổ chức thường niên. Hiện hay Node.js được sử dụng bởi nhiều công ty trong đó có Microsoft, eBay, GE, GoDaddy, Paypal, Uber....
  + 1. **Các thành phần quan trọng trong Node.js**



**Hình 2.2**Lược đồ mô tả các thành phần quan trọng trong Node.js

* 1. **Các Object cơ bản**
     1. **Global Object (đối tượng toàn cục)**

Như chúng ta đã biết, hệ thống module của node không khiến khích việc sử dụng niến toàn cục, tuy nhiên node cung cấp một globals quan trọng để sử dụng. Việc đầu tiên và quan trọng nhất là tiến trình global, cho thấy nhiều thao tác như quá trình truyền tín hiệu, xuất cảnh, process id (pid), và nhiều hơn nữa. Globals khác, chẳng hạn như console objects được cung cấp cho những người sử dụng để viết JavaScript cho trình duyệt web.

* **Console:**

Các console objects sử dụng một số lệnh được sử dụng để xuất thông tin để *stdout* hoặc *stderr*. Đó là các lệnh như:

**Console.log([data], […])**

Phương pháp console objects được sử dụng thương xuyên nhất là *console.log(),* mà chỉ đơn giản là viết cho stdout và gắn vào một nguồn cung cấp dữ liệu dòng.

*Console.log (wahoo ‘’);*

*// => wahoo*

*Console.log ({foo: ‘bar’});*

*// => [object object]*

Còn một lệnh có chức năng như *console.log()* là *console.info().*

**Console.error([data], [])**

Giống như *console.log()* nhưng nó được viết cho *stderr.*

*Console.error (‘kết nối cơ sở dữ kiệu không thành công’);*

Còn một lệnh có chức năng tương tự đó là *console.warn().*

**Console.dir(obj)**

Sử dụng phương pháp *inspect()* của module *sys* các đối tượng *stdout*.

*Console.dir ({foo: ‘bar’});*

*// => {foo: ‘bar’}*

**Console.assert (expression, [message])**

Nếu *expression* có giá trị là false thì *AssertionError* sẽ đưa ra *message* được cho là.

*Console.assert (connected, ‘cơ sở dữ liệu kết nối không thành công’) ;*

**Console.time(label)**

Dánh dấu thời gian bắt đầu.

**Console.timeEnd(label)**

Thời gian kết thúc, được ghi vào đầu ra. Ví dụ :

*Console.time (‘100-element’) ;*

*For (var i=0 ; i < 100 ; i++){*

*;*

*}*

*Console.timeEnd(‘100-element’) ;*

**Console.trace(label)**

In ra một tập stack các dấu vết stderr tại vị trí hiện tại.

* **Process**

Các process object gắn liền với goodies, trước tiên, chúng ta sẽ có một cái nhìn tại một số thuộc tính cung cấp thông tin về node process đó :

**Process.version**

Chuỗi phiên bản, ví dụ :

*Console.log (‘ version : ’ + process.version) ; // thông tin về version hiện tại sử dụng.*

**Process.execPath**

Đường dẫn đến thu mục thực thi chính của chương trình “/usr / local / bin / node”.

**Process.platform**

Các nền tảng ta đang sử dụng.

**Process.pid**

Các processID.

**Process.stdout()**

Một luồng có thể ghi được đến *stdout*.

Ví dụ, định nghĩa về *console.log().*

*Console.log = function (d){*

*Process.stdout.write (d + ‘\n’);*

*}*

**Process.stderr()**

Tương tự như *Process.stdout()* nhưng ở đây là ghi đến *stderr*

*Process.stderr()* và *process.stdout()* là không giống như luồng khác trong Node, khi viết chúng thường bị blocking. Chúng bị blocking trong trường hợp mà chúng liên quan đến các tập tin thường xuyên hoặc mô tả tập tin TTY. Trong trường hợp chúng liên quan đến các pipes, chúng không bị blocking như những luồng khác.

**Process.stdin()**

Một luồng có thể đọc được cho stdin. Các dòng stdin bị tạm dừng theo mặc định. Do đó, người ta phải họi *process.stdin.resume()* để đọc từ nó.

Ví dụ, mở đầu vào chuẩn và lắng nghe các sự kiện:

*Process.stdin.resume();*

*Process.stdin.setEncoding (‘utf8’);*

*Process.stdin.on (‘data’, function (chunk){*

*Process.stdout.write (‘data: ’ + chunk);*

*});*

*Process.stdin.on (‘data’, function (){*

*Process.stdout.write (‘ end’);*

*});*

**Process.ewd()**

Trả về thư mục làm việc hiện tại, ví dụ:

*cd ~ && node*

*Node > process.cwd() “/users/tj”*

**Process.getuid()**

Trả về số user ID của process đang chạy.

**Process.setuid()**

Thiết lập user ID có hiệu lục cho quá trình đang chạy. Phương pháp này chấp nhận cả một số ID, cũng như một chuỗi. ví dụ cả hai *process.setuid(501),* và *process.setuid(‘tj’)* đều hợp lệ.

**Process.getgid()**

Trả về số group ID của process đang chạy.

**Process.setgid()**

Tương tự như *process.setuid()* tuy nhiên được sử dụng trong group, cũng chấp nhận một số giá trị hoặc chuỗi đại diện. ví dụ, *process.setgid(20)* hoặc *process.setgid(‘www’).*

**Process.chdir(directory)**

Thay đổi thư mục làm việc hiện tại của process hoặc đưa một ngoại lệ nếu thất bại.

*Console.log (‘starting directory: ’ + process.cwd ());*

*Try{*

*Process.chdir (‘/ tmp’);*

*Console.log (‘new directory: ’ + process.cwd ());*

*}*

*Catch.(err){*

*Console.log (‘chrrid: ’ + err);*

*}*

* **Buffers**

Cơ bản JavaScript là Unicode thân thiện, nhưng không phải với dữ kiệu nhị phân. Khi giao tiếp với luồng TCP hoặc hệ thống tập tin, dữ liệu nhị phân cần thiết để xử lý các luồng octec.

Node cung cấp một số phương pháp cho việc khai thác, tạo và sử dụng luồng octec.

Để xử lý các dữ liệu nhị phân, node cung cấp cho chúng ta với các đối tượng toàn cục. Buffer là tương tự như một mảng các số nguyên, nhưng tương ứng với việc cấp phát bộ nhớ thô bên ngoài V8 heap. Buffer không thể được thay đổi kích cỡ. Có một số cách để xây dựng một trường hợp bộ đệm, và nhiều cách ta có thể thao tác dữ liệu của nó.

Chuyển đỗi giữa buffers và các đối tượng chuỗi JavaScript đòi hỏi một phương pháp mã hóa rõ ràng. Dưới đây là chuỗi các bảng mã khác nhau.

* + ‘ascii’ – 7 bit dữ liệu ASCII duy nhất. Phương Pháp mã hóa này rất nhanh chóng, và sẽ loại bỏ các bít cao nếu thiết lập. Lưu ý rằng việc mã hóa này chuyển đổi một ký tự null (‘\ 0’ hoặc ‘\ u0000’) vào 0x20 (mã ký tự của một không gian). Nếu ta muốn chuyển đổi một ký tự null vào 0x00, ra nên sử dụng ‘utf8’.
  + ‘utf8’ - Nhiều byte mã hóa ký tự Unicode. Nhiều trang web và các đinh dạng tài liệu khác sử dụng UTF-8.
  + ‘utf16le’- 2 hoặc 4 byte, ký tự Unicode mã hóa ít về cuối. Các cặp đại diện (U 10.000 FFF 10 U) được hỗ trợ.
  + ‘USC2’ – Tương tự ‘utf16le’.
  + ‘báe64’ – Mã hóa chuỗi Base64.
  + ‘Binary’ – Một cách mã hóa dữ liệu nhị phân thành chuỗi bằng cách sử dụng 8bit đầu tiên của mỗi ký tự. phương pháp mã hóa này bị phản đối và nên tránh sử dụng các đối tượng bộ đện nếu có thể. Mã hóa này sẽ được loại bỏ trong các phiên bản tương lai của Node.
  + ‘hex’ – Mã hóa mỗi byte là hai ký tự thập lục phân.
    1. **Sự kiện (Event)**

Là một khái niệm khá quan trọng trong node, và được sử dụng rất nhiều trong suốt module chính của chương trình và module của bên thứ 3. Module sự kiện chính của Node cung cấp cho chúng ta một hàm tạo, *EventEmitter.*

* **EventEmitter**

Thông thường một đối tượng kế thừa từ *EventEmitter*, ví dụ minh họa API. Đầu tiên chúng ta tạo ra một emitter, sau đó chúng ta có thể xác định bất ký số lượng callbacks sử dụng *emitter.on()* phương pháp, mà chấp nhận tên của các sự kiện và các đối tượng tùy ý thông qua như là dữ liệu. khi *emitter.emit()* được gọi, chúng ta chỉ requred để truyền các tên sự kiện, theo sau bơi bất kỳ số lượng tham số (trong trường hợp nay các chuỗi tên đầu tiên và cuối cùng).

*Var EventEmitter = require (‘event’).Eventemitter;*

*Var emitter = new EventEmitter;*

*Emitter.on (‘name’, function (first, last){*

*Console.log (first ‘,’ last);*

*})*

*Emitter.emit (‘name’, ‘tj’ , ‘holowaychuk’);*

*Emitter.emit (‘name’, ‘simon’ , ‘holowaychuk’);*

* **Kế thừa từ EventEmitter ( Inheriting From EventEmitter ).**

Được sử dụng phổ biến và thiết thực của *EventEmitter* là tính kế thừa từ nó. Điều này có nghĩa là chúng ta có thể giữ nguyên *EventEmitter* nguyên mẫu mà không bị ảnh hưởng trong khi sử dung API của nó đối với phương tiện riêng của chúng ta.

Để làm được như vậy, chúng ta bắt đầu bằng cách xác định các hàm khởi tạo Dog, trong đó tất nhiên sẽ bark từ thời gian đến thời gian ( còn được đến như một sự kiện ).

*Var EventEmitter = require (‘events’).EventEmitter;*

*Function Dog(name){*

*This.name = name;*

*}*

Ở đây chúng ta kế thừa từ EventEmitter vì vậy chúng ta có thể sử dụng các phương thức mà nó cung cấp, chẳng hạn như *EventEmitter#on()* và *EventEmitter#emit().* Nếu thuộc tính proto bị loại trừ, đừng lo lắng, chúng sẽ được trở lại sau.

*Dog.prototype.proto = EventEmitter.prototype;*

Bây giờ chúng ta có Dog được thành lập, ta có thể tạo ra … Simon! Khi Simon bark, ta có thể cho *stdout* biết bằng cách gọi *console.log()* với callback. Callback chính nó được gọi là trong ngữ cảnh của đối tượng.

*Var simon = new Dog(‘simon’);*

*Simon.on(‘bark’, function(){*

*Console.log(this.name ‘barker’);*

*});*

Bark hai lần mỗi giây.

*setInterval (function(){*

*simon.emit(‘bark’);*

*}, 500);*

* **Loại bỏ các sự kiện lắng nghe ( Removing Event Listeners )**

Như chúng ta đã biết, thì lắng nghe sự kiện chỉ đơn giản là hàm đó được gọi khi chúng ta *emit()* một sự kiện. chúng ra có thể loại bỏ nhưng người nghe bằng cách dùng lệnh removeListener (type, callback), mặc dù điều này không được dùng thường xuyên.

* + 1. **Luồng ( Streams )**

Streams là một khái niệm quan trong trong Node. Các luồng API là một cách duy nhất để xử lý luồng giống như dữ liệu. ví dụ, dự liệu có thể được xem trực tiếp một tập tin, trực tiếp vào một socket để đáp ứng một HTTP request, hoặc trực tiếp từ một nguồn chỉ cho đọc như stdin.

* **Readble Streams**

Readable streams được xem như một HTTP request kế thừa từ *EventEmitter* để lộ dữ liệu đến qua các sự kiện. Việc đầu tiên của những sự kiện này là sự kiện dữ liệu, là một đoạn tùy ý của các dữ liệu được truyền đi để xử lý sự kiện là mọt trường hợp đệm ( *Buffer instance* )

*Req.on (‘data’, function(buf)){*

*//làm gì đó với buffer*

*}*

Một sự kiện quan trọng nữa là kết thúc, đặc trưng là sự kết thúc của dữ kiệu Event. Ví dụ, đây là 1 HTTP echo server, chỉ đơn giản là “simply” các request body data thông qua các response. Vì vậy, nếu chúng ta POST “hello word”, response của ta sẽ là “hello world”.

*Var http = require (‘http’);*

*http.createServer (function (req, res){*

*res.writeHeadd(200);*

*req.on (‘data’, function (data){*

*res.write (data);*

*});*

*Req.on (‘end’, function(){*

*Res.end ();*

*});*

*}).listen(3000);*

Module *sys* thực sự có một chức năng được thiết kế đặc biệt cho hành động “simply” nay, aptly tên *sys.pump().* Nó chấp nhận một luồng đọc như là đối số đầu tiên, và viết dòng thứ hai.

*Var http = require (‘http’),*

*Sys = require (‘sys’);*

*http.createServer (function (req, res){*

*res.writeHead (200);*

*sys.pump (req, res);*

*}).listen (3000);*

**Stream.readable**

Giá trị boolean được mặc định là true, nhưng sẽ thành false sau khi xảy ra một lỗi, luồng đến một ‘ kết thúc ’, hoặc *destroy()* được gọi.

**Stream.resume ()**

Tiếp tục lại sự kiện ‘data’ sau khi pause().

**Stream.pause ()**

Vấn đề là một tín hiệu tư vấn cho các lớp giao tiếp cơ bản, yêu cầu không có thêm dữ liệu được gửi cho đến khi resume () được gọi.

Lưu ý là, do tính chất tư vấn, luồng nhất định sẽ không được tạm dừng ngay lập tức, do đó, sự kiện ‘data’ có thể được phát ra cho một khoảng thời gian không xác đinh, ngay cả sau khi pause () được gọi.

**Stream.destroy ()**

Đóng các tập tin mô tả cơ bản. stream không còn có thể ghi và đọc được.

* **Writable Stream.**

Một lớp cơ sở cho việc tạo ra các Writable Stream. Tương tự như Readable Stream, ta cũng có thể tạo ra các lớp con bằng cách ghi đè không đồng bộ khi sử dụng câu lệnh.

*\_write (chunk, cb).*

**Stream.writeable**

Giá trị boolean được mặc định đó là *true*, nhưng sẽ thành *false* sau khi lỗi xảy ra hoặc *end ()* */destroy ()* được gọi.

**Stream.Write (string, [encoding])**

Viết chuỗi với encoding cho luồng. trả về *true* nếu chuỗi đã được bỏ vào bộ đệm kernel. Trả về *false* để cho biết rằng bộ đệm kernel đã đầy, và dữ liệu sẽ được gửi đi trong tương lai. Sự kiện *‘drain’* sẽ cho biết khi nào bộ đệm kernel rỗng. việc mã hóa mặc định *‘utf8’*.

**Stream.Write (buffer)**

Tương tự như trên, ngoại trừ với bộ đệm.

**Stream.end ()**

Kết thúc dong vói EOF hoặc FIN. Cuộc gọi này cho phép hàng đợi ghi dữ liệu được gửi trước khi đong luồng.

**Stream.end (string, encoding)**

Gửi chuỗi với mã hóa nhất định và chấm dứt dòng với EOF hoặc FIN, điều này rất hữu ích để giảm số lượng các gói tin gửi đi.

**Stream.destroy ()**

Đóng mô tả tập tin cơ bản. Stream không có thể ghi và cũng không thể đọc được. các luồng sẽ không phát ra bất kỳ chi tiết ‘data’, hoặc sự kiện ‘kết thúc’. Bất kỳ hàng đợi dữ liệu ghi sẽ không được gửi đi. Các luồng được tái sự kiện ‘close’ khi tài nguyên của mình đã được xử lý.

* + 1. **HTTP**

Để sử dụng HTTP server và client ta dùng lệnh *require (‘http’).*

Các giao diện HTTP trong node được thiết kế để hỡ trợ nhiều tính năng của các giao thức truyền thông khó sử dụng. Trong đó, có thể là đoạn mã hóa, tin nhắn.

HTTP headers được biểu diễn bởi một đối tượng như thế này:

*{ ‘content-leght’: ‘123’,*

*‘content-type’: ‘text/plain’,*

*‘connection’: ‘keep-alive’,*

*‘accept’: ‘\*/\*’ }*

Key được lowercased và giá trị không được sửa đổi.

**http.STATUS\_CODES**

Một bộ sưu tập của tất cả các mã trạng thái tiêu chuẩn của HTTP response, và mô tả ngắn gọn cho từng cái, ví dụ, *http.STATUS\_CODES [404] === ‘Not Found’.*

**http.createServer ([requestListener])**

trả về một đối tượng web server mới. RequesListener là một chức năng được tự động thêm vào sự kiện ‘request’.

**http.createClient ([port], [host])**

hàm này bị phản đối sử dụng, mà sử dụng hàm http.request() để thay thế. Xây dựng một HTTP client mới. Port và host tham chiếu đến máy chủ để được kết nối.

* **Class: htttp.server**

**Server.listen(port, [hostname], [backlog], [callback])**

Bắt đầu chấp nhận các kết nối trên port chỉ định và hostname. Nếu hostname được bỏ qua, các server sẽ chấp nhận các kết nối trực tiếp đến bất kỳ địa chỉ Ipv4 (INADDR\_ANY). Để nghe một socket unix, cung cấp một tên tập tin thay vì port và hostname.

Backlog là chiều dài tối đa của hàng đợi kết nối đang chờ. Chiều dài thực tế sẽ được xác định bởi hệ điều hành thông qua các thiết lập *sysctl* như *tcp\_max\_sys\_backlog* và *somaxconn* trên Linux. Giá trị mặc định của tham số này là 511 (không phải là 512).

Hàm này là không đồng bộ. gọi lại tham số cuối cùng sẽ được thêm vào như là một listener cho sự kiện ‘*listen’*. Xem thêm *net.Server.listen (port).*

**Server.listen (path, [callback])**

Bắt đầu một máy chủ socket UNIX lắng nghe cho các kết nối trên đường dẫn nhất định. Hàm này là không đồng bộ. gọi lại tham số cuối cùng sẽ được thêm vào như là một người biết lắng nghe cho sự kiện ‘*listen’*. Xem thêm *net.Server.listen (path)*

**Server.listen (handle, [callback])**

Các đối tượng có thể được thiết lập đễ xủa lý một máy chủ hoặc socket, hoặc đối tượng *{fd: <n>}.*

Lắng nghe trên một mô tả tập tin không được hỗ trợ trên Window.

Chức năng này là không đồng bộ. gọi lại tham số cuối cùng sẽ được thêm vào như là một người biết lắng nghe cho sự kiện ‘*listen’*. Xem them *net.Server.listen().*

**Server.close ([callback])**

Dừng server và chấp nhận các kết nối mới. Xem *net.Server.close().*

* **Class: http.ServerRequest**

Đối tượng này được tạo ra trong nội bộ của một máy chủ HTTP - không phải bởi người sử dụng - và thông qua như là đối số đầu tiên để một listener ‘request’. Request thực hiện các giao diện Readable Streams.

**Request.method**

Yêu cầu một chuỗi. chỉ đọc. ví dụ: *‘GET’, ‘DELETE’.*

**Request.url**

Yêu cầu chuỗi URL, câu lệnh nay chỉ chứa các URL theo thực tế trong HTTP request.

Nếu request là:

*GET /status?name=ryan HTTP/1.1\r\n*

*Accept: text/plain\r\n*

*\r\n*

*Thì request.url sẽ là: ‘/status?name=ryan’*

Nếu bạn muốn để phân tích các URL thành các phần của nó, bạn có thể sử dụng

*equire('url').parse(request.url).* Ví dụ:

*node> require('url').parse('/status?name=ryan')*

*{ href: '/status?name=ryan',*

*search: '?name=ryan',*

*query: 'name=ryan',*

*pathname: '/status' }*

Nếu bạn muốn trích xuất các params từ chuỗi truy vấn, bạn có thể sử dụng *require('querystring').parse*, hoặc thông qua các đối số thứ hai *require('url').parse.* Phân tích cú pháp. Ví dụ:

*node> require('url').parse('/status?name=ryan', true)*

*{ href: '/status?name=ryan',*

*search: '?name=ryan',*

*query: { name: 'ryan' },*

*pathname: '/status' }*

**request.headers**

Đọc bản đồ duy nhất của tên header và giá trị. Tên header là *lower-cased*.

Ví dụ:

*// In một cái gì đó như:*

*// {User-agent ':' curl/7.22.0 '*

*// Host: '127 .0.0.1:8000 ',*

*// accept: '\*/\*' }*

*console.log(request.headers);*

**request.trailers**

Chỉ đọc HTTP trailer (nếu có).   
**request.httpVersion**

Phiên bản giao thức HTTP như một chuỗi. Chỉ đọc. Ví dụ: *'1.1 ', '1.0'*. Ngoài ra *request.httpVersionMajor* là số nguyên đầu tiên và *request.httpVersionMinor* là thứ hai.

**request.setEncoding ([encoding])**

Thiết lập mã hóa cho cơ chế request. Xem *stream.setEncoding()* để biết thêm thôngtin.  
**request.pause ()**

Tạm dừng request từ các sự kiện phát ra. Rất hữu ích để tăng tốc tải lên.  
**request.resume ()**

Tiếp tục lại request tạm dừng.  
**request.connection**

Các đối tượng *net.Socket* kết hợp với kết nối. Với hỗ trợ HTTPS, sử dụng *request.connection.verifyPeer()* và *request.connection.getPeerCertificate()* để có được thông tin xác thực của client.

* **Class: http.ServerResponse.**

Đối tượng này được tạo ra trong nội bộ của một HTTP server - không phải bởi người sử dụng. Nó được thông qua như là tham số thứ hai của sự kiện 'request'. Response thực hiện các giao diện Writable Stream.

**response.writeHead (statusCode [reasonPhrase], [headers])**

Gửi một response headers để đáp ứng yêu cầu. statusCode là một trạng thái HTTP có 3 chữ số mã, giống như 404. Tùy chọn có thể đưa ra một reasonPhrase như là đối số thứ hai. Ví dụ:

*var body = 'hello world';*

*response.writeHead(200, {*

*'Content-Length': body.length,*

*'Content-Type': 'text/plain' });*

Câu lệnh này chỉ được gọi là một lần trên một tin nhắn và nó phải được gọi trước khi *response.end()* được gọi. Nếu bạn gọi *response.write()* hoặc *response.end()* trước khi gọi, header ngầm định sẽ được tính toán và gọi hàm này cho bạn.

Lưu ý: rằng *Content-Length* được đưa ra trong các byte không phải ký tự. Ví dụ trên làm việc vì chuỗi 'hello world' chứa các byte ký tự duy nhất. Nếu nó chứa các ký tự được mã hóa cao hơn thì sau đó *Buffer.byteLength()* nên được sử dụng để xác định số byte trong một đoạn mã hóa nhất định. Và Node không kiểm tra

*Content-Length* và độ dài của thân đã được truyền đi bằng nhau hay không.

response.statusCode

Khi sử dụng các header ẩn (không gọi response.writeHead() một cách rõ ràng), hàm điều khiển các mã trạng thái đó sẽ được gửi đến cho client khi các header nhận được. Ví dụ:

response.statusCode = 404;

Sau khi response header đã được gửi cho client, hàm này chỉ ra các mã trạng thái được gửi.

**Response.setHeader (name, value)**

Thiết lập một giá trị header duy nhất cho header ẩn. Nếu header này đã tồn tại trong phần header được gửi, giá trị của nó sẽ được thay thế. Sử dụng một mảng các chuỗi ở đây nếu bạn cần phải gửi nhiều header với cùng một tên. Ví dụ:  *response.setHeader ("Content-Type", "text / html");*

hoặc

*response.setHeader ("Set-Cookie", "loại = ninja", "ngôn ngữ = javascript "]);*

**Response.sendDate**

Khi giá trị là true, date header sẽ được tự động tạo ra và gửi response nếu nó không phải là đã có trong các header. Mặc định là true.

* **http.ClientResponse.**

Đối tượng này được tạo ra khi thực hiện một yêu cầu với http.request(). Nó được thông qua với sự kiện 'response' của đối tượng yêu cầu. response thực hiện các giao diện Readable Stream.

**response.statusCode**

3 chữ số mã trạng thái của HTTP response. VÍ DỤ: 404.

**response.httpVersion**

Các phiên bản HTTP của máy chủ để kết nối. Có thể hoặc là '1.1 'hoặc '1.0'. Ngoài ra *response.httpVersionMajor* là số nguyên đầu tiên và

*response.httpVersionMinor* là thứ hai.

**response.headers**

Đối tượng response headers .  
**response.trailers**

Trailer đối tượng response.  
**response.setEncoding ([encoding])**

Thiết lập mã hóa cho cơ thể phản ứng. Xem *stream.setEncoding ()* để biết thêm thông tin.

**response.pause ()**

Tạm dừng response từ các sự kiện phát ra. Hữu ích để tăng tốc tải về.  
*response.resume ()* Tiếp tục lại một response bị tạm dừng.

* 1. **Socket.io**

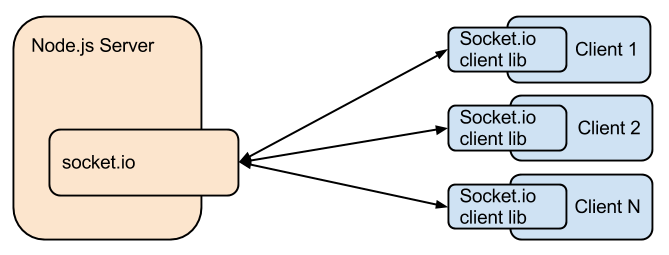
Socket.io là một thư viện Javascript, với đặc trưng mạnh mẽ và dễ sử dụng, thì socket.io cho phép ta tạo ra các ứng dụng realtime trên trình duyệt cũng như các ứng dụng thiết bị di động. Socket.io đang dần trở nên quen thuộc và được hỗ trợ của nhiều nhà phát triển lớn (từ Microsoft Office, Yammer, Zendesk, Trello… tới những đội hackathon, những start up trẻ).

Socket.io gồm 2 phần:

* Về phía client: bộ thư viện viết cho web (JavaScript), IOS, Android.
* Về phía server: bộ thư viện viết bằng Javsscript và dùng cho các máy chủ Node.js
  + 1. **Các đặc điểm của Socket.io**

Nhìn lại về websocket thì Trong trình duyệt mà được hỗ trợ WebSockets protocol, một kết nối giữa server và browser được tạo ra quanh HTTP và được gọi là một “HTTP hanshake” (cái bắt tay). Một kết nối được tạo ra trình duyệt và server mở ra một cổng giao tiếp liên tục thông qua một TCP socket. Nó sẽ hỗ trợ cho cả việc gửi và truy vấn message trên một cổng kết nối. Điều này giúp server load ít hơn, giảm số message bị trễ, “and unify PUSH communication using standalone connection”. Tuy nhiên WebSocket mắc phải vấn đề đó là HTTP proxies, firewall và hosting provider. Khi Websocket sử dụng một phương thức giao tiếp ngoài HTTP, một phần nhiều trong số đó chưa được hỗ trợ và block bất cứ kết nối socket nào. Vấn đề này chỉ được giải quyết khi sử dụng thư viện trừu tượng mà có thể dễ dàng thay đổi giữa các giao thức dựa trên resoures có sẵn. Vì vậy Socket.io được xây dựng để giải quyết các vấn đề đó và luôn sẵng sàng được sử dụng cho NodeJS developer.

* **Các đặc điểm nổi bật của Socket.io.**
* Là một module của Node.js.
* Được xây dựng nhằm mục đích tạo ra real time NodeJS application. Socket.io cung cấp cho lập trình viên các đặc trưng như event, room và tự động phục hồi lại kết nối.
* Khi chúng ta include Socket.io module vào trong ứng dụng của mình nó sẽ cung cấp cho chúng ta hai object đó là: socket server quản lý functionality phía server và socket client điều khiển funtionality phía client.
* Khi client muốn kết nối tới Socket.io server, nó sẽ gửi cho server một “handshake HTTP request”. Server sẽ phân tích request đó với những thông tin cần thiết trong suốt quá trình kết nối. Nó sẽ tìm cấu hình của middleware mà đã được đăng ký với server và thực thi chúng trước khi đưa ra sự kiện kết nối. Khi kết nối thành công thì connection event listener được thực thi, tạo ra một instance mới của socket có thể coi như định danh của client mà mỗi một client kết nối tới sẽ có 1 định danh. Các bạn có thể thấy rõ khi xem hình dưới đây.



****Hình 2.3****Quá trình kết nối của client tới socket.io

* Một module khác của Node.js là LightStreamer-adapter cũng có tạo các kết nối từ client tới server nhưng không trực tiếp mà thông qua LightStreamer Server, đó là các máy chủ theo thời gian thực và nằm ngoài tiến trình của Node.js Server.
  + 1. **Phương thức hoạt động của socket.io trong Node.js**

Với Node.js, ta chỉ cần biết vài hàm cở bản như requires () để import thư viện. Công việc còn lại, chỉ cần dùng socket.io.

* Server: Tạo một đối tượng socket bằng phương thức listen (). Phương thức này chờ đợi một yêu cầu kết nối từ client.
* Client: Kết nối đến server bằng phương thức connect (url, {port: server\_port}).
* Socket.io cung cấp 3 event chính là connect, message và disconect. Chúng được kích hoạt khi client.server:
* Connect: tạo kết nối
* Message: nhận được thông điệp
* Disconnect: ngắt kết nối

Ví dụ: khai báo cho socket nhận một sự kiện “message”

*socket.on("message", function(msg){*

*// console.log("Received: "+ msg);*

*});*

* Để gửi dữ liệu, ta dùng phương thức send(). Dữ liệu có thể là đối tượng (được chuyển thành JSON) và sẽ nhận được qua sự kiện message.

Ví dụ:

*Socket.send (“hello word”);*

* Socket.io có thể gửi và nhận các event tự tạo với phương thức emit(). Hai phía gửi và nhận phải biết được tên của event đó để thực hiện giao tiếp:

Ví dụ:

*// client gửi một dòng message “welcome” lên event “hello”*

*Socket.emit (“hello”, {msg: “welcome”});*

*// server nhận sự kiện event đưa lên*

*Socket.on (“hello”, function (data){*

*Console.log (data);*

*});*

* 1. **Hệ Cơ Sở Dữ Liệu MONGODB**

Trong những gương mặt góp phần làm suy tàn đế chế SQL thì MongoDB nổi lên là một CSDL đáng tin cậy và dễ dùng nhất. Mongo viết bằng C++. Nó thích hợp cho các ứng dụng tầm trung trở lên. Nếu tỉ lệ lượng dữ liệu ghi vào CSDL của ứng dụng lớn hơn lượng đọc thì đây càng là lựa chọn hợp lý.

MongoDB là một CSDL có khả năng mở rộng, hiệu suất cao, mã nguồn mở và hướng

văn bản.

Trước khi đi vào tìm hiểu kỹ hơn về MongoDB, chúng ta làm quen với một số khái niệm cơ bản của MongoDB:

* Văn bản (Document) là đơn vị cơ bản của dữ liệu trong MongoDB, nó tương đương với một dòng trong CSDL quan hệ
* Bộ sưu tập (Collection) có thể được coi như tương đương với một bảng.
* MongoDB có thể lưu trữ nhiều CSDL độc lập, mỗi CSDL này có các bộ sưu tập và điều khoản riêng của mình
* MongoDB đi kèm với một trình tiện ích JavaScript đơn giản nhưng mạnh mẽ, nó hữu ích trong quản trị và thao tác dữ liệu.
* Mỗi văn bản có một khóa đặc biệt, đó là “\_id”, nó là duy nhất trong bộ sưu tập của văn bản.
* **Văn bản**

Văn bản là một khái niệm quan trọng trong MongoDB. Văn bản bao gồm tập hợp các khóa với các giá trị tương ứng.

Ví dụ: *{"greeting" : "Hello, world!"}*

Văn bản trên gồm một khóa là “greeting”, với giá trị là “Hello, world!”. Các văn bản có thể chứa nhiều cặp khóa/giá trị.

Ví dụ: *{"greeting" : "Hello, world!", "foo" : 3}*

Một số lưu ý:

* Các cặp khóa / giá trị trong văn bản được sắp xếp. Văn bản trên sẽ khác với văn bản sau

*{"foo" : 3, "greeting" : "Hello, world!"}*

* Khóa trong văn bản là một chuỗi
* MongoDB phân biệt chữ hoa chữ thường
* Văn bản trong MongoDB không được chứa những khóa giống nhau. Ví dụ văn bản sau là không hợp lệ

*{"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!"}*

* **Bộ sưu tập**

Bộ sưu tập là một nhóm các văn bản. Nếu văn bản tương đương với dòng trong CSDL quan hệ thì bộ sưu tập tương đương với bảng.

Bộ sưu tập là một Schema-Free, nghĩa là các văn bản có hình dạng khác nhau có thể cùng được lưu trữ trong 1 bộ sưu tập.

Ví dụ các văn bản sau có thể cùng được lưu trong một bộ sưu tập:

*{"greeting" : "Hello, world!"}*

*{"foo" : 5}*

Bộ sưu tập được xác định bởi tên của nó là một chuỗi UTF-8

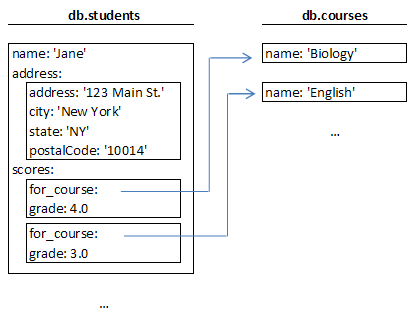
Các đặc trưng của MongoDB:

* Lưu trữ hướng văn bản: Văn bản theo phong cách JSON với những lược đồ động đơn giản
* Hỗ trợ chỉ mục đầy đủ: chỉ mục trên bất kỳ các thuộc tính
* Tính sao lặp và tính sẵn sàng cao: mở rộng
* Auto-sharding: mở rộng theo chiều ngang mà không ảnh hưởng đến chức năng
* Truy vấn: đa dạng, truy vấn dựa trên văn bản
* Cập nhật nhanh:
* Map/Reduce
* GridFS: lưu trữ file với bất kỳ kích cỡ nào mà không làm phức tạp ngăn xếp
* Hỗ trợ thương mại: hỗ trợ doanh nghiệp, đào tào, tư vấn
  + 1. **Thiết kế lược đồ**

Với MongoDB, chúng ta ít phải “chuẩn hóa” hơn so với khi làm việc với lược đồ quan hệ vì trong MongoDB không có khái niệm liên kết (join). Nói chung, với mỗi đối tượng (object) mức cao nhất, ta sẽ có một bộ sưu tập (collection) dữ liệu.

Một bộ sưu tập không phải cho tất cả các lớp (class), thay vào đó, các đối tượng sẽ được nhúng vào đó.

**Hình 1.3** minh họa có 2 bộ sưu tập: students và courses. Các văn bản student được nhúng văn bản address và văn bản score. Trong đó, văn bản Score được tham chiếu đến Courses.



****Hình 2.4** Minh họa bộ sưu tập**

So sánh với lược đồ quan hệ: ta cần lưu Score vào bảng riêng và dùng khóa ngoài liên kết với Student.

* **Nhúng hay Tham chiếu**

Một câu hỏi quan trọng trong thiết kế lược đồ Mongo là: “Đối tượng này có cần một bộ sưu tập của riêng nó không hay nên nhúng vào trong các đối tượng trong các bộ sưu tập khác?” Trong cơ sở dữ liệu quan hệ, mỗi tiểu mục có thể trở thành một bảng riêng biệt. Trong Mongo, nó không được khuyến cáo, việc nhúng các đối tượng hiệu quả hơn nhiều. Chúng ta cũng có thể đặt ra câu hỏi “Tại sao tôi không muốn nhúng đối tượng này?”

Tại sao tham chiếu lại chậm. Ta xem ví dụ sau. Chúng ta có một đối tượng Student và cần thực hiện:

*print( students.address.city );*

Phép toán này sẽ luôn được thực hiện nhanh nếu Address là một đối tượng nhúng, và được lưu ở RAM nếu Student được lưu ở RAM.

Tuy nhiên, với truy vấn:

*print( students.scores[0].for\_course.name );*

Nếu đó là lần đầu truy cập đến khóa này thì trình tiện ích phải thực hiện truy vấn:

*students.scores[0].for\_course = db.courses.findOne({\_id:\_course\_id\_to\_find\_});*

* **Các luật cơ bản**
* Các đối tượng “lớp thứ nhất” là các đối tượng ở mức cao nhất, có bộ sưu tập của riêng mình.
* Các đối tượng miêu tả chi tiết các mục thường được nhúng
* Các đối tượng mà theo mô hình đối tượng có chứa quan hệ nói chung nên được nhúng
* Quan hệ nhiều – nhiều thường được tham chiếu.
* Các bộ sưu tập chỉ với một vài đối tượng có thể tồn tại một cách an toàn giống như bộ sưu tập riêng lẻ, được lưu trữ nhanh chóng trong bộ nhớ máy chủ ứng dụng.
* Các đối tượng nhúng khó khăn để tham chiếu hơn là các đối tượng mức cao.
* Sẽ khó khăn hơn để có một cái nhìn mức hệ thống đối với các đối tượng nhúng. Ví dụ: Sẽ dễ thực hiện truy vấn tìm 100 sinh viên có điểm cao nhất hơn nếu Score không bị nhúng.
* Nếu dữ liệu được nhúng lớn, có thể đạt đến giới hạn kích thước của một đối tượng.
* Nếu hiệu suất là quan trọng, hãy nhúng.
* Một số ví dụ
* Customer/Order/ Order Line-Item: Customers, Orders nên có một bộ sưu tập riêng.
* Line-Items nên là một mảng các mục cần mua và được nhúng trong đối tượng Order
* Hệ thống Blog: Posts cần có bộ sưu tập riêng. Post Author có thể có bộ sưu tập riêng hoặc nếu đơn giản chỉ là địa chỉ mail của tác giả thì cho thành một trường trong Posts. Comments được nhúng trong Posts
* **Lựa chọn chỉ mục**

Một khía cạnh thứ hai khi thiết kế lược đồ là việc lựa chọn chỉ mục. Việc đánh chỉ mục làm cho việc thực hiện truy vấn nhanh hơn. Một truy vấn bình thường cần vài phút, có thể được thực hiện ngay lập tức với việc sử dụng chỉ mục.

Trong MongoDB:

* Trường \_id được đánh chỉ mục tự động
* Những trường mà theo đó các khóa được tìm kiếm nên được đánh chỉ mục
* Những trường sắp xếp nói chung nên được đánh chỉ mục

Lưu ý rằng việc thêm vào chỉ mục chỉ làm chậm quá trình ghi vào bộ sưu tập mà không làm chậm quá trình đọc. Vì vậy, sử dụng nhiều chỉ mục với những bộ sưu tập mà tỉ lệ read:write cao. Với những bộ sưu tập mà ghi nhiều hơn đọc, sử dụng chỉ mục là rất tốn kém.

* + 1. **Chỉ mục**

Chỉ mục làm tăng hiệu suất truy vấn lên rất nhiều. Điều quan trọng là nghĩ xem xét tất cả các loại truy vấn cần trong ứng dụng để xác định những chỉ mục liên quan. Khi đã xác định xong, việc tạo ra các chỉ mục trong MongoDB là khá dễ dàng.

* **Các khái niệm cơ bản**

Chỉ mục là một cấu trúc dữ liệu, thu thập thông tin về giá trị của các trường trong các văn bản của một bộ sưu tập. Cấu trúc dữ liệu này được sử dụng trong tối ưu truy vấn Mongo để sắp xếp nhanh các văn bản trong một bộ sưu tập.

Chúng ta có thể khởi tạo chỉ mục bằng cách gọi hàm ensureIndex() và cung cấp một văn bản với một hoặc nhiều khóa để đánh chỉ mục. Ví dụ đánh chỉ mục cho trường name trong *students db.students.ensureIndex({name:1});*

Hàm ensureIndex() chỉ khởi tạo chỉ mục nếu nó chưa tồn tại. Để kiểm tra việc tồn tại chỉ mục trên bộ sưu tập students, ta có thể chạy hàm

*db.students.getIndexes().*

Khi một bộ sưu tập được đánh chỉ mục trên một khóa nào đó, truy cập ngẫu nhiên trên biểu thức truy vấn có chứa khóa đó sẽ được thực hiện rất nhanh. Nếu không được đánh chỉ mục, MongoDB phải soát tất cả các văn bản để kiểm tra giá trị của khóa đó trong truy vấn.

* ***Chỉ mục mặc định***

Một chỉ mục luôn luôn được tạo ra là \_id. Chỉ mục này là đặc biệt và không thể bị xóa. Chỉ mục \_id là duy nhất cho các khóa của nó.

* ***Các khóa nhúng***

Với MongoDB chúng ta thậm chí có thể đánh chỉ mục trên các khóa bên trong văn bản nhúng. Ví dụ

*db.students.ensureIndex({"address.city": 1})*

* ***Văn bản như là khóa***

Các trường được đánh chỉ mục có thể là bất kỳ loại nào, bao gồm cả văn bản.

* ***Mảng***

Khi giá trị của trường được đánh chỉ mục của văn bản là một mảng. MongoDB đánh chỉ mục mỗi phần tử của mảng đó.

* **Chỉ mục hỗn hợp các khóa.**

Ngoài chỉ mục khóa đơn, MongoDB còn hỗ trợ đánh chỉ mục hỗn hợp nhiều

khóa. Giống như đánh chỉ mục cơ bản, chúng ta sử dụng hàm ensureIndex()

để khởi tạo chỉ mục.

*db.things.ensureIndex({j:1, name:-1});*

Khi khởi tạo một chỉ mục, số đi cùng với khóa là hướng của chỉ mục, 1: tăng dần, -1: giảm dần. Hướng không ảnh hưởng đến việc truy cập ngẫu nhiên nhưng quan trọng nếu bạn đang làm các truy vấn sắp xếp hoặc phân loại trên chỉ mục hỗn hợp.

Nếu chúng ta có một chỉ mục hỗn hợp trên nhiều trường, chúng ta có thể sử dụng nó để truy vấn trên các tập hợp con đầu của các trường đó. Ví dụ ta có chỉ mục trên (a, b, c), ta có thể sử dụng nó để truy vấn trên (a), (a, b),

(a, b, c).

* **Chỉ mục thưa thớt**

Chỉ mục thưa thớt là chỉ mục mà chỉ bao gồm các văn bản có trường được đánh chỉ mục. Bất kỳ văn bản nào bị thiếu trường đánh chỉ mục thưa thớt đều không được lưu vào trong chỉ mục. Các chỉ mục là thưa thớt vì bị thiếu những văn bản không có giá trị của trường được đánh chỉ mục.

Chỉ mục thưa thớt, theo định nghĩa, là không đầy đủ và hoạt động khác với chỉ mục đầy đủ. Khi sử dụng chỉ mục thưa thớt để sắp xếp, một vài văn bản trong bộ sưu tập sẽ không được trả về. Đó là do chỉ những văn bản được đánh chỉ mục mới được trả về.

*db.people.ensureIndex({title : 1}, {sparse : true})*

*db.people.save({name:"Jim"})*

*db.people.save({name:"Sarah", title:"Princess"})*

*db.people.find({title:{$ne:null}}).sort({title:1})*

*// returns only Sarah*

* **Chỉ mục duy nhất**

MongoDB hỗ trợ đánh chỉ mục duy nhất, đảm bảo rằng không có văn bảo nào được chèn mà giá trị của khóa được đánh chỉ mục lại trùng với văn bản đã tồn tại. Để tạo ra một chỉ mục đảm bảo ràng không có 2 văn bản có cùng giá trị cho 2 trường firstname và lastname ta làm như sau:

*db.things.ensureIndex({firstname: 1, lastname: 1}, {unique: true});*

* **Khóa bị thiếu**

Khi một văn bản được lưu vào bộ sưu tập với việc đánh chỉ mục duy nhất, bất kỳ khóa được đánh chỉ mục nào bị thiếu sẽ được chèn vào với giá trị null. Vì vậy, không được phép chèn nhiều văn bản bị thiếu cùng một khóa được đánh chỉ mục.

*db.things.ensureIndex({firstname: 1}, {unique: true});*

*db.things.save({lastname: "Smith"});*

*// Next operation will fail because of the unique index on firstname.*

*db.things.save({lastname: "Jones"});*

* **Giá trị lặp lại**

Chỉ mục duy nhất không cho phép một khóa có giá trị nhân bản. Nếu bạn muốn đánh chỉ mục bằng mọi giá, hãy giữ văn bản đầu tiên trong CSDL và xóa tất cả các văn bản có giá trị bị nhân bản, thêm tùy chọn dropDups

*db.things.ensureIndex({firstname : 1}, {unique : true, dropDups : true})*

* **Xóa chỉ mục**

Xóa tất cả các chỉ mục trên bộ sưu tập:

*db.collection.dropIndexes();*

Xóa chỉ mục đơn:

*db.collection.dropIndex({x: 1, y: -1})*

Chạy trực tiếp như một lệnh mà không cần hỗ trợ:

*// note: command was "deleteIndexes", not "dropIndexes", before MongoDB v1.3.2*

*// remove index with key pattern {y:1} from collection foo*

*db.runCommand({dropIndexes:'foo', index : {y:1}})*

*// remove all indexes:*

*db.runCommand({dropIndexes:'foo', index : '\*'})*

* **ReIndex**

Lệnh reIndex sẽ xây dựng lại tất cả các chỉ mục cho bộ sưu tập.

*db.myCollection.reIndex ()*

*/ / giống như:*

*db.runCommand ({reIndex: 'myCollection'})*

Thông thường, điều này là không cần thiết. Chúng ta có thể làm điều này nếu kích thước bộ sưu tập đã thay đổi đáng kể hoặc không gian đĩa được sử dụng bởi các chỉ mục có vẻ lớn bất thường.

reIndex sẽ bị chậm với các bộ sưu tập lớn.

Các lệnh sửa CSDL tái tạo lại tất cả các chỉ mục trong CSDL.

**Lưu ý**

* Chỉ mục trong MongoDB phân biệt chữ hoa chữ thường
* Chỉ số thông tin được lưu giữ trong bộ sưu tập system.indexes, chạy db.system.indexes.find () để xem ví dụ.
* **Hiệu suất chỉ mục**

Việc đánh chỉ mục thực hiện rất nhanh. Cập nhật được thực hiện nhanh hơn vì MongoDB có thể tìm thấy các văn bản cần cập nhật rất nhanh chóng. Tuy nhiên, với việc sử dụng chỉ mục, khi ghi dữ liệu vào bộ sưu tập, các khóa sau đó phải được thêm vào trường chỉ mục. Như vậy, chỉ mục chỉ tốt cho bộ sưu tập có số lượng đọc nhiều hơn rất nhiều số lượng ghi. Đối với các bộ sưu tập chú trọng ghi, việc sử dụng chỉ mục, trong một số trường hợp, có thể phản tác dụng. Hầu hết các bộ sưu tập đều chú trọng đọc, vì vậy mà chỉ mục là tốt trong hầu hết các tình huống.

* ***Sử dụng Sort() mà không cần chỉ mục***

Chúng ta có thể sử dụng sort() để trả về dữ liệu được sắp thứ tự mà không cần phải dùng chỉ mục nếu dữ liệu cần trả về là nhỏ (< 4 MB). Đối với những trường hợp này tốt nhất là sử dụng hàm limit() và sort() cùng nhau.

* + 1. **Sao chép**

Có lẽ công việc quan trọng nhất của bất kỳ quản trị viên MongoDB là đảm bảo sao cho sao chép được thiết lập và hoạt động đúng. Sao chép có thể được sử dụng hoàn toàn để dự phòng và toàn vẹn dữ liệu hoặc có thể được sử dụng cho mục đích cao hơn như mở rộng đọc, sao lưu nóng,…

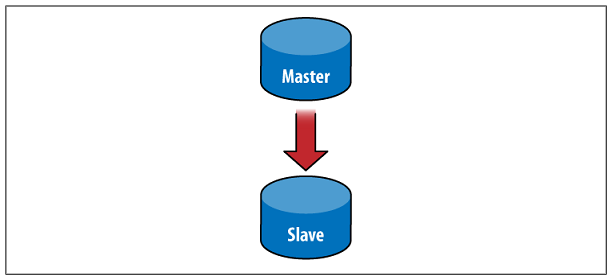
MongoDB hỗ trợ sao chép dữ liệu không đồng bộ giữa các máy chủ. Tại một thời điểm, chỉ có 1 máy chủ hoạt động để ghi (primary hay master).

Có hai hình thức sao chép.

* Master-Slave Replication
* Replica Sets.
* **Master-Slave Replication**

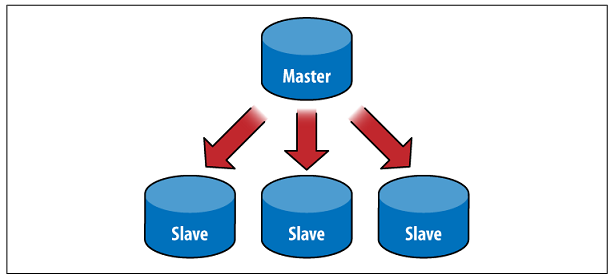
Sao chép Master-slave là mô hình sao chép phổ biến nhất được hỗ trợ bởi MongoDB. Mô hình này rất linh hoạt và có thể được sử dụng để sao lưu, dự phòng, mở rộng đọc, …

Hình 2.5 minh họa mô hình Master – Slave bao gồm 2 nút, một nút làm Master, nút còn lại làm Slave

****

****Hình 2.5**** Mô hình Master – Slave hai nút

Hình 2.3 minh họa mô hình Master – Slave bao gồm 4 nút, một nút làm Master, 3 nút còn lại làm Slave



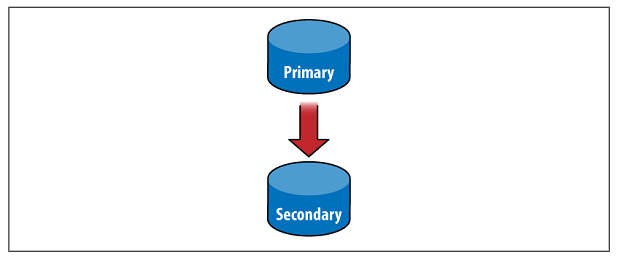
****Hình 2.6**** Mô hình Master – Slave bốn nút

Để thiết lập cần khởi động nút master và một hoặc nhiều nút slave, các nút này đều biết địa chỉ của nút master. Để khởi động master, chạy mongod --master. Để khởi động slave, chạy mongod --slave --source master\_address, trong đó master\_address là địa chỉ của nút master vừa được khởi động

* **Replica Sets**

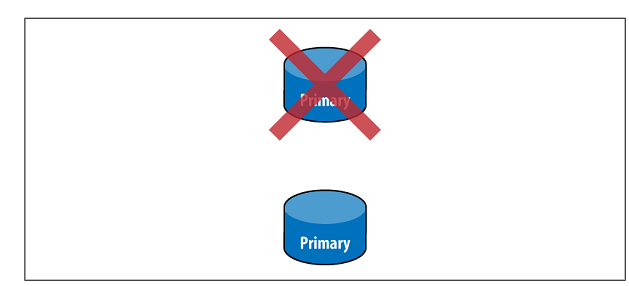
Replica Sets là một cụm master-slave tự động chịu lỗi. Replica Sets không có một master cố định: một master được bầu chọn và có thể thay đổi đến nút khác nếu master bị sập [1].

Hình 2.7 mô phỏng mô hình Replica Sets gồm 2 nút.



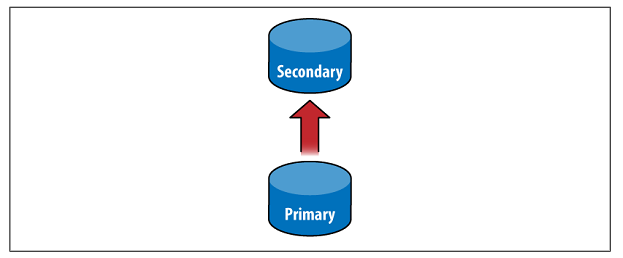
****Hình 2.7****Mô hình Replica Sets hai nút

Khi server chính chết, server cấp 2 chở thành server chính (hình 2.5).

****

**Hình 2.8**Replica Sets – Bầu chọn master mới

Nếu server chính ban đầu hoạt động trở lại, nó trở thành server cấp 2 (hình 2.6).

****

Hình 2.9 Server chính trở thành server cấp 2

* + 1. **Truy vấn**

Một trong những tính năng tốt nhất của MongoDB là hỗ trợ truy vấn động (ad hoc). Hệ thống hỗ trợ truy vấn động không yêu cầu bất cứ chỉ mục nào để tìm dữ liệu. Người dùng có thể tìm dữ liệu với việc sử dụng bất kỳ tiêu chuẩn nào. Với CSDL quan hệ, truy vấn động là chuẩn hóa.

* **Đối tượng biểu thức truy vấn**

MongoDB hỗ trợ một số các đối tượng truy vấn để lấy dữ liệu. Ví dụ, giả sử chúng ta muốn sử dụng trình MongoDB để trả về mọi văn bản trong bộ sưu tập *users.* Truy vấn sẽ được viết như sau:

*db.users.find({})*

Trong trường hợp này, lựa chọn (điều kiện) của chúng ta là trống, nó phù hợp với mọi văn bản trong bộ sưu tập. Chúng ta xem thêm một số ví dụ:

*db.users.find({'last\_name': 'Smith'})*

Ở đây, lựa chọn của chúng ta là tất cả các văn bản mà thuộc tính *last\_name* là *Smith.*

* **Các tùy chọn truy vấn**
* ***Lựa chọn các trường***

Ngoài các biểu thức truy vấn, truy vấn MongoDB còn có thể thêm vào các tham số. Ví dụ, chúng ta muốn các số CMT của tất cả người có họ là *Smith*, ta có thực hiện truy vấn:

// lấy trường ssn của các văn bản có last\_name == 'Smith':

*db.users.find({last\_name: 'Smith'}, {'ssn': 1});*

// lấy tất cả các trường ngoại trừ trường thumbnail đối với tất cả các văn bản.

*db.users.find({}, {thumbnail:0});*

Chú ý rằng, trường \_id luôn luôn được trả về ngay cả khi không yêu cầu

* ***Sắp xếp***

Truy vấn MongoDB có thể trả về kết quả được sắp xếp. Để trả về tất cả các văn bản mà trường last\_name được sắp xếp theo thứ tự tăng dần, ta viết truy vấn sau:

*db.users.find({}).sort({last\_name: 1});*

* ***Bỏ qua và giới hạn***

MongoDB luôn luôn hỗ trợ bỏ qua và giới hạn để phân trang một cách dễ dàng. Ví dụ ta muốn bỏ qua 20 họ đầu tiên và giới hạn kết quả đến 10, ta viết truy vấn sau:

*db.users.find().skip(20).limit(10);*

*db.users.find({}, {}, 10, 20); // giống như lệnh trên nhưng không rõ ràng.*

* ***SlaveOk***

Khi thực hiện truy vấn ở một hoặc nhiều bản sao, trình tiện ích gửi yêu cầu đến master, để thực hiện truy vấn đối với slave, truy vấn có thể chạy với tùy chọn slaveOk.

*db.getMongo().setSlaveOk(); // cho phép truy vân slave*

*db.users.find(...)*

* ***Con trỏ***

Các truy vấn CSDL được thực hiện với phương thức find(), với kỹ thuật này một con trỏ được trả về. Con trỏ sau đó được sử dụng lặp đi lặp lại để lấy tất cả các văn bản mà truy vấn trả về. Chúng ta có thể xem ví dụ sau:

*> var cur = db.example.find();*

*> cur.forEach( function(x) { print(tojson(x))});*

*{"n" : 1 , "\_id" : "497ce96f395f2f052a494fd4"}*

*{"n" : 2 , "\_id" : "497ce971395f2f052a494fd5"}*

*{"n" : 3 , "\_id" : "497ce973395f2f052a494fd6"}*

*>*

Như vậy, MongoDB là một CSDL hướng văn bản, lưu trữ dữ liệu dưới cặp khóa/giá trị. Các đối tượng trong MongoDB thường được nhúng trong các đối tượng mức cao hơn để tăng tốc độ xử lý truy vấn. Để tăng tốc độ truy vấn, người ta cũng thường đánh chỉ mục cho những bộ sưu tập có tỉ lệ đọc:ghi cao. MongoDB thực hiện truy vấn để lấy dữ liệu thông qua các biểu thức truy vấn cùng các tham số cần thiết. Với những dự án mà tỉ lệ lượng dữ liệu ghi vào CSDL lớn hơn lượng đọc thì lựa chọn MongoDB sẽ mang lại hiệu quả cao.

Để làm quen với truy vấn trong MongoDB, người đọc có thể tham khảo một số lệnh tương đương với truy vấn SQL chuẩn mà chúng tôi trình bày trong Phụ lục của báo cáo này.

* 1. **GIT – hệ thống quản lý phiên bản phân tán**



Hình 2.10 giới thiệu GIT

* + 1. **Giới thiệu về GIT**

Trong khoảng thời gian maintenance 1991-2002 của Linux kernel project, những thay đổi được phân tán dưới dạng patches hoặc archived file. Năm 2002, được sử dụng miển phí BitKeeper, là một DVCS có bản quyền. Tuy nhiên, đến năm 2005,  mối quan hệ giữa cộng đồng phát triển Linux kernel project và nhà sở hữu BitKeeper có vấn đề, cho nên họ không được sử dụng license miễn phí của BitKeeper nữa. Do đó, cộng đồng phát triển Linux (đặc biệt là Linus Torvalds, người tạo ra Linux) đã quyết định phát triển Git dựa trên những gì học được trong thời gian sử dụng BitKeeper. Và cho đến nay Git đã trở thành một trong các phần mềm quản lý mã nguồn phổ biến nhất.

Git là một hệ thống quản lý phiên bản ( version control system – VCS ) dùng để theo dõi các thay đổi trong các tập tin và cách phối hợp sao cho nhiều người có thể cùng làm việc trên những tập tin đó. Git được sử dụng chính trong lĩnh vực phát triển phần mềm, nhưng Git cũng có thể được sử dụng để theo dõi các thay đổi trong bất kỳ tập tin nào. Git là hệ thống điều khiển phiên bản phân tán, có tốc độ xử lý khá nhanh, đảm bảo toàn vẹn dữ liệu và hỗ trợ cho các workflow phân tán, phi tuyến tính.

Cũng như hầu hết các hệ thống quản lý phiên bản phân tán khác, mọi thư mục Git trên mỗi máy tính đều là một kho chứa (repository) có lịch sử làm việc và khả năng theo dõi phiên bản đầy đủ, độc lập với mạng và máy chủ trung tâm (central server).

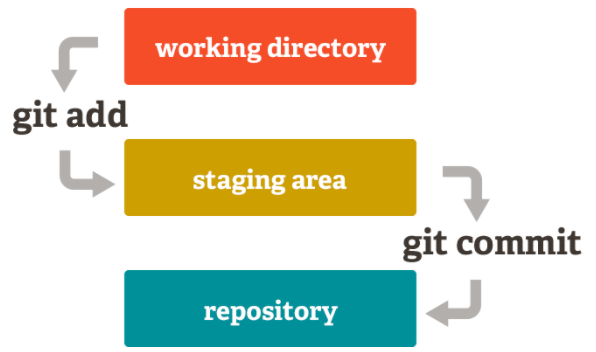
Git có khả năng chạy trên nhiều hệ điều hành khác nhau như Linux, Windows, Mac OSX…

* + 1. **Đặc điểm của GIT**

Khi vào trang web chính thức của Git, chúng ta sẽ thấy các đặc điểm của Git như: fast-version-control (quản lý phiên bản nhanh), distributed-even-if-your-workflow-isnt (phân tán cho dù quy trình làm việc của ta không cần đến), distributed-is-the-new-centralized (phân tán cũng là một cách tập trung mới), local-branching-on-the-cheap (phân nhánh local với chi phí thấp), everything-islocal (mọi công việc đều thực hiện local).

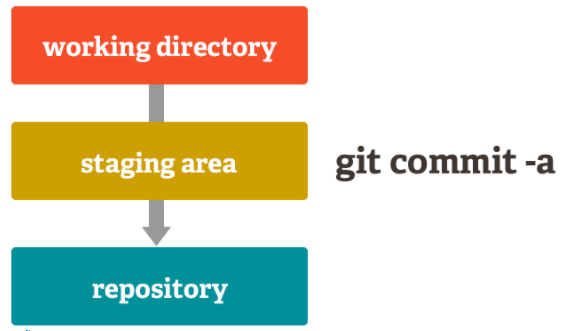
Các đặc điểm chính của Git:

* Branching & Merging
* Nói về Git là phải nói về mô hình phân nhánh ủa Git (branching model). Git cho phép và khuyến khích chúng ta nên có nhiều nhánh cục bộ có thể làm việc hoàn toàn độc lập với nhau. Và việc tạo mới, gộp lại hay xóa các kêt quả trong quá trình phát triển chỉ mất vài giây.
* Đáng chú ý là khi ta đưa vào một kho điều khiển từ xa (remote repositiry), ta không phải đưa vào tất cả các nhánh. Ta có thể chọn chỉ chia sẽ một, một vài hoặc tất cả trong số nhánh đó. Điều này giúp cho mọi người cảm thấy thoải mái để tiếp tục thử nghiệm những ý tưởng mới mà không phải lo lắng về việc phải lập kế hoạch làm thế nào và khi nào sẽ gộp vào hoặc chia sẽ các nhánh cho người khác. Và Git làm cho quá trình này trở nên dễ dàng.
* Nhỏ gọn và nhanh chong (Small and Fast)
* Hầu hết các hoạt động trên Git được thực hiện cục bộ, điều đó giúp Git có một lợi thế rất lớn về tốc độ xử lý, nhất là trên các hệ thống tập trung thường xuyên phải giao tiếp với một máy chủ.
* Git được xây dựng để làm việc trên Linux Kernel, có nghĩa là Git có khả năng xử lý hiệu quả các kho lưu trữ (repository) lớn ngay từ giai đoạn đầu. Tốc độ và hiệu suất là mục tiêu thiết kế chính của Git.
* Phân tán (Distributed)
* Distributed: một trong những tính năng hấp dẫn nhất của Distributed SCM mà Git có đó là phân tán. Điều này có nghĩa là thay vì thực hiện “checkout” source code hiện tại, ta có thể “clone” toàn bộ kho lưu trữ (respsitory).
* Nhiều bản sao lưu (Multipe Backups): Điều này có nghĩa là ngay cả khi ta đang sử dụng quy trình làm việc tập trung, mỗi người dùng đều có một bản sao lưu đầy đủ từ máy chủ. Bản sao này có thể được đẩy lên thay thế cho bản chính trên máy chủ trong trường hợp có sự cố xảy ra. Rõ ràng, nếu ta có nhiều bản sao thì việc phục hồi sẽ an toàn và sex thành công hơn rất nhiều.
* Workflow: vì bản chất của Git là phân tán và có hệ thống nhánh tuyệt vời nên hầu như các quy trình công việc (workflow) nào cũng có thể thực hiện tương đối dễ dàng trên Git.
* Bảo đảm dữ liệu (Data Assurance)
* Mô hình dữ liệu mà Git sử dụng đảm bảo tính toàn ven cho từng của dự án. Nó có cở chế để mỗi tập tin và việc hoàn tất được kiểm tra kỹ lưỡng. Không thể lấy bất cứ thứ gì ra khỏi Git ngoài các bit chính xác mà chúng ta đưa vào.
* Chúng ta cũng không thể thay đổi tập tin, ngày tháng, thông điệp hoàn tất (commit message) hay bất cứ dữ liệu nào trong kho của Git ma không thay đổi ID của chung sau đó. Điều này có nghĩa là nếu ta có một commit ID, ta có thể yên tâm là dự án của ta chính xác như khi nó đã được commit. Trong khi đó hầu hết các hệ thống kiểm soát tập trung đều không mặc đinh cung cấp tính toàn vện như vậy.
* Staging area (khu vực tổ chức)
* không giống như các hệ thống khác, Git có một vùng gọi là “staging area” hay “index”. Đây là một khu vực trung gian nơi các giao dịch được định dạng và kiểm tra lại trước khi hoàn tất.
* Git có thể nhanh chóng chạy các tập tin và hoàn tất chúng mà không cần phải hoàn tất tất cả các tập tin được sửa đổi trong thư mục làm việc hoặc phải liệt kê chúng trên command line trong quá trình thực hiện.



**Hình 2.11** hoạt động của GIT tại khu vực tôt chức

* Điều này cho phép ta chỉ thực hiện các phần của một tập tin được sửa đổi, cập nhật các thay đổi trong tập tin khi cần.
* Git cũng có thể không sử dụng tính năng này khi ta không muốn – chỉ cần thêm ‘-a’ vào lện commit để thêm tất cả thay đổi của tất cả các tập tin vào “staging area”.



**Hình 2.12** thao tác vận hành trong GIT

* Mã nguồn mở - miễn phí (Free & Open Source)
* Git là một phần mềm mã nguồn mở miễn phí được phát hành theo giấy GNU General Public License Versiin 2.0
  + 1. **Các thành phần quan trọng của Git**

Mỗi tập tin trong Git được quản lý dựa trên ba trạng thái: committed, modified, và staged. Committed có nghĩa là dữ liệu đã được lưu trữ một cách an toàn trong cơ sở dữ liệu. Modified có nghĩa là bạn đã thay đổi tập tin nhưng chưa commit vào cơ sở dữ liệu. Và staged là bạn đã đánh dấu sẽ commit phiên bản hiện tại của một tập tin đã chỉnh sửa trong lần commit sắp tới.

Điều này tạo ra ba phần riêng biệt của một dự án sử dụng Git: thư mục Git (Git Directory), thư mục làm việc (Working directory), và khu vực tổ chức (staging area).



Hình 2.13 các thành phần trong GIT

Thư mục Git (Git Directory) là nơi Git lưu trữ các "siêu dữ kiện" (metadata) và cơ sở dữ liệu cho dự án của bạn. Đây là phần quan trọng nhất của Git, nó là phần được sao lưu về khi bạn tạo một bản sao (clone) của một kho chứa từ một máy tính khác.

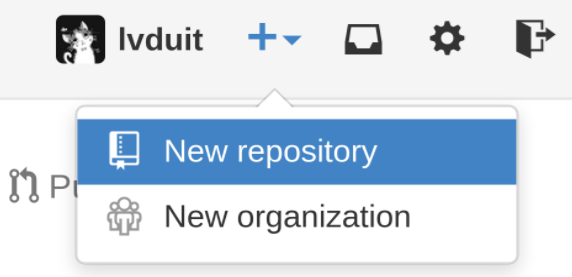
Thư mục làm việc (Working directory) là bản sao một phiên bản của dự án. Những tập tin này được kéo về (pulled) từ cơ sở dữ liệu được nén lại trong thư mục Git và lưu trên ổ cứng cho bạn sử dụng hoặc chỉnh sửa.

khu vực tổ chức (staging area) là một tập tin đơn giản được chứa trong thư mục Git, nó chứa thông tin về những gì sẽ được commit trong lần commit sắp tới. Nó còn được biết đến với cái tên "chỉ mục" (index), nhưng khu vực tổ chức (staging area) đang dần được coi là tên tiêu chuẩn.

Tiến trình công việc (workflow) cơ bản của Git:

* Ta thay đổi các tập tin trong thư mục làm việc.
* Ta tổ chức các tập tin, tạo mới ảnh của các tập tin đó vào khu vực tổ chức.
* Ta commit, ảnh của các tập tin trong khu vực tổ chức sẽ được lưu trữ vĩnh viễn vào thư mục Git.
* **Kho Repo (Repository)**

Repository hay được gọi tắt là Repo, đơn giản là nơi chứa tất cả những thông tin cần thiết để duy trì và quản lý các sửa đổi và lịch sử của toàn bộ project. Trong Repo có 2 cấu trúc dữ liệu chính là Object Store và Index. Tất cả dữ liệu của Repo đèu được chứa trong thư mục bạn đang làm việc dưới dạng folder ẩn có tên là .git



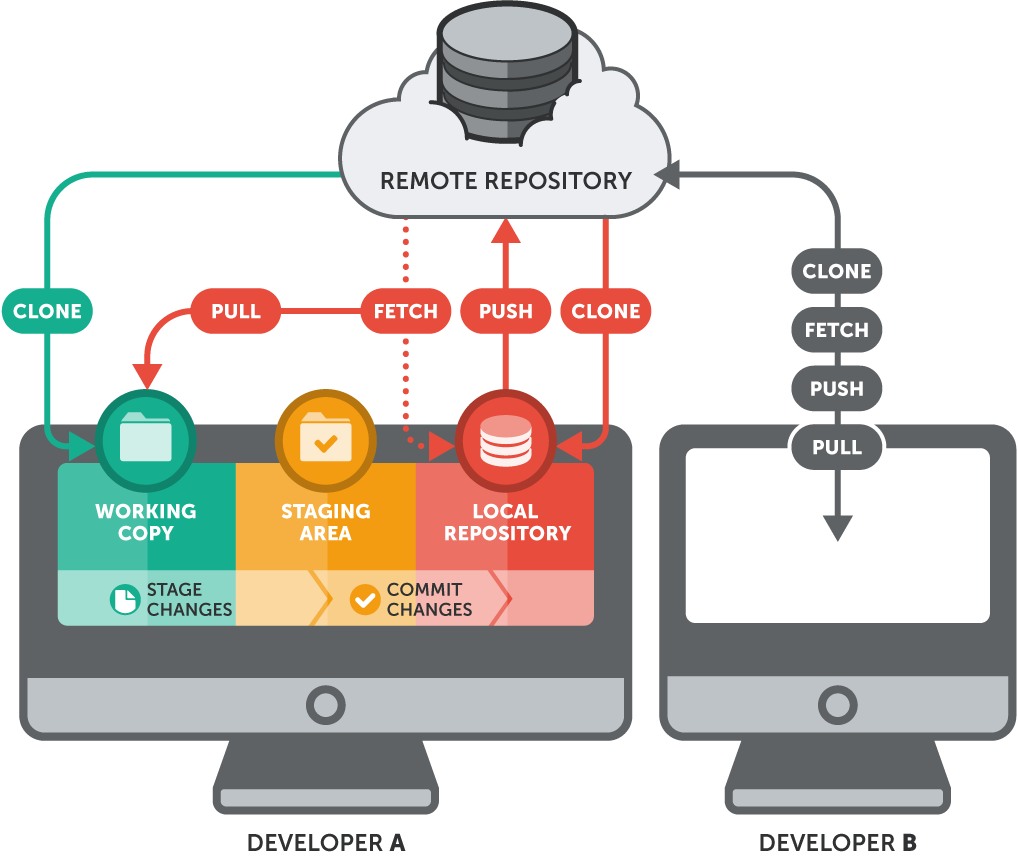
**Hình 2.14** kho repo

* **Remote repository và Local repository**

Đầu tiên, repository của Git được phân thành 2 loại là remote repository và local repository.

* Remote repository: Là repository để chia sẻ giữa nhiều người và bố trí trên server chuyên dụng.
* Local repository: Là repository bố trí trên máy của bản thân mình, dành cho một người dùng sử dụng.

Do repository phân thành 2 loại là local và remote nên với những công việc bình thường thì có thể sử dụng local repository. Khi muốn public nội dung công việc mà mình đã làm trên local repository, thì ta sẽ upload lên remote repository rồi public. Thêm nữa, thông qua remote repository bạn cũng có thể lấy về nội dung thay đổi của người khác.



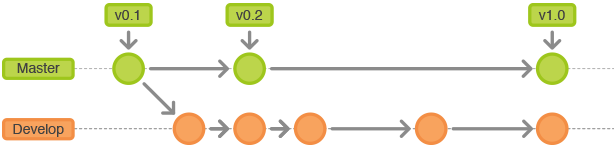
**Hình 2.15** sơ đồ Remote repository và Local repository

* **Nhánh (branch)**

Đây là một trong những thế mạnh của Git là nhánh. Với Git, việc quản lý nhánh rất dễ dàng. Mỗi nhánh trong Git gần giống như một workspace. Việc nhảy vào một nhánh để làm việc trong đó tương tự việc chuyển qua ngữ cảnh làm việc mới, sau đó có thể nhanh chóng quay lại ngữ cảnh cũ.

Nhánh (branch) được dùng để phát triển tính năng mới mà không làm ảnh hưởng đến code hiện tại.

Nhánh master là nhánh “mặc định” khi bạn tạo một repository.  
 Nhánh master thông thường là nhánh chính của ứng dụng. Ví dụ bạn thử nghiệm một tính năng mới và muốn không ảnh hưởng đến code chính bạn có thể tạo một nhánh mới và sau khi xong sẽ hợp nhất lại với nhánh master. Việc hợp nhất 2 nhánh lại được gọi là merge.



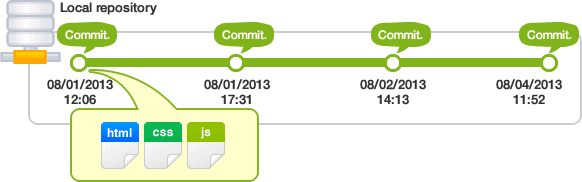
**Hình 2.16** sơ đồ nhánh trong GIT

**Commit (Ủy thác)**

Để ghi lại việc thêm/thay đổi file hay thư mục vào repository thì sẽ thực hiện thao tác gọi là Commit.

Khi thực hiện commit, trong repository sẽ tạo ra commit (hoặc revision) đã ghi lại sự khác biệt từ trạng thái đã commit lần trước với trạng thái hiện tại.

Commit này đang được chứa tại repository, các commit nối tiếp với nhau theo thứ tự thời gian. Bằng việc lần theo commit này từ trạng thái mới nhất thì có thể biết được lịch sử thay đổi trong quá khứ hoặc nội dung thay đổi đó.



Hình 2.17 sơ đồ vận hành của commit trong GIT

Các commit này, được đặt tên bởi 40 ký tự alphabet (mã md5 thì phải) không trùng nhau được băm từ thông tin commit. Bằng việc đặt tên cho commit, có thể chỉ định commit nào từ trong repository.

Mỗi commit đều có yêu cầu phải có commit message, để giải thích commit này là ta đã làm gì trong này.

* **Git Remote**

Để kết nối được với một repo khác người ta sử dụng một khái niệm gọi là remote.

Trên thực tế khi làm việc với nhau thì không như vậy, vì không phải máy ai cũng cài một “git server” để người khác kết nối được với mình. Thông thường thì chúng ta sẽ sử dụng một repo chung và các máy kết nối vào repo đó.

Có 2 “git repo server” được sử dụng nhiều là github.com và bitbucket.org.

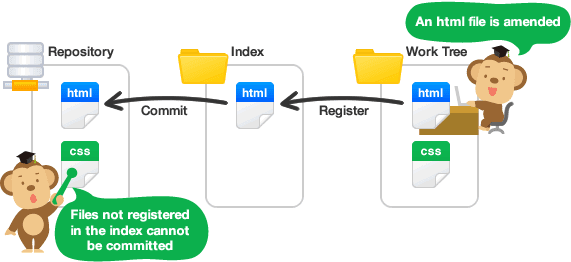
Trên thực tế khi có 2 người cùng làm việc với 1 project thì thông thường sẽ tạo một repo trên github hoặc bitbucket và repo trên máy người A sẽ kết nối với repo trên github và máy người B cũng kết nối với repo trên github/bitbucket. Từ đó source code của người A và người B sẽ được đồng bộ với nhau thông qua repo trên github/bitbucket.

Vì vậy, trước khi sử dụng git thì bạn nên đăng kí một tài khoản trên github.com hoặc bitbucket.org.

* **Working Tree và Index**

Trên Git, những thư mục được đặt trong sự quản lý của Git mà mọi người đang thực hiện công việc trong thực tế được gọi là working tree.

Và trên Git, giữa repository và working tree tồn tại một nơi gọi là index. Index là nơi để chuẩn bị cho việc commit lên repository.



**Hình 2.18** cách thức vận hành của working tree và Index

Trên Git, khi đã thực hiện commit thì trạng thái sẽ không được ghi trực tiếp trong repository từ working tree, mà sẽ ghi trạng thái đã được thiết lập của index được xây dựng ở giữa đó. Vì thế, để ghi lại trạng thái của file bằng commit thì trước hết cần thông báo file trong index.

* 1. **Heroku**
     1. **Giới thiệu heroku**



Hình 2.19 giới thiệu heroku

Heroku là dịch vụ cung cấp máy chủ miễn phí cho người dùng. Với cái giá phải trả 0$ cùng với vô vàn các addons hỗ trợ cực kỳ hữu ích thì đấy được coi là một trong những dịch vụ hấp dẫn khó cưỡng. Dù miễn phí nhưng nó có thể so sanh với các server trả phí.

Heroku là một server đa dạng cung cấp cho nhiều ứng dụng, ta có thể cài đặt web server như apace hoặc các phần mềm lên đó, herolu hỗ trợ nhiều ngôn ngữ lập trình như Node.js, Ruby, Python, Java, PHP, …

* + 1. **Cấu trúc của horuku.**

Heroku cung cấp một virtual machine (VM) cho người dùng triển khai code. Tất nhiên là nếu bạn muốn dùng free thì sẽ chỉ được triển khai tới 5 application thôi (5 VM). Còn đối với ứng dụng thực của bạn, Heroku sẽ cung cấp một URL subdomain  riêng. Vì thế mà project name của bạn phải không bị trùng.

Những workspace có nhiều components khác nhau như: code và resource files (không phải dynamic data files), database (Postgres), và log files.

Trên local desktop của user, Heroku sẽ dùng directory name để xác định cho project của use, đồng thời cũng là để Heroku hiểu được nội dung của chúng. Do đó bạn có thể có nhiều project khác nhau với directory riêng biệt, miễn là khi chạy Heroku commands  thì nhớ để đúng folder.

Điều duy nhất bạn cần phải lưu ý đó là tất cả mọi thứ đều chạy từ memory. Không hề có bất cứ persistent storage nào. Tôi xin nhắc lại – bạn không thể lưu trữ bất cứ file nào trên file server. Cho persistence, Heroku cung cấp một postgress SQL database cho bạn lưu trữ theo ý thích.

* + 1. **Một số tính năng của horuku.**
* Database miễn phí.

Heroku cho phép app bạn sử dụng hạn chế và số records được phép với các dạng free, tuy nhiên, đối với các website hoạt động lâu dài hoặc có lượng dữ liệu lớn thì phải chịu phí với các gói khác nhau. Chi phí này tính ra không hề rẻ và có thể còn đắc hơn cả giá thuê 1 VPS.

* SSL miễn phí.
* Hỗ trợ làm việc Team.
* Liên kết với Github đơn giản

# CHƯƠNG 3. PHÂN TÍCH THÀNH PHẦN DỮ LIỆU

* 1. **Mô tả chương trình.**

Ứng dụng này được xây dựng trên nền tảng nodejs, người dùng có thể sử dụng để chat trong nội bộ doanh nghiệp và hỗ trợ khách hàng thông qua kênh chat trực tuyến được gắn trên website thương mại của doanh nghiệp.

Khách hàng có thể chat trực tuyến với nhân viên hỗ trợ của doanh nghiệp mà không cần phải đăng ký hay đăng nhập, tin nhắn sẽ được gửi đến tài khoản nhân viên hỗ trợ trực tuyến của doanh nghiệp.

Nhân viên của doanh nghiệp phải đăng ký tài khoản để sử dụng, mỗi người dùng có thể tạo các kênh chat để mời mọi người cùng tham gia thảo luận, hoặc gửi tin nhắn trực tiếp đến user khác.

* 1. **Chức năng hệ thông**

Đăng ký, đăng nhập, đăng xuất: Cho nhiều người dùng truy cập phần mềm, có thể logout khi cảm thấy không cần thiết.

Ghi nhớ trạng thái ở trang login: Bất kỳ phần mềm nào cũng cần phải có chức năng này đề phòng khi người dùng tắt mở tab trình duyệt phải đăng nhập lại rất tốn thời gian và phiền cho người sử dụng.

Tìm kiếm: Tìm kiếm tên người dùng khác và tên kênh

Tạo kênh: Người dùng có thể tạo kênh mới để mời các thành viên khác tham gia chat

Thanh trạng thái: Người dùng có thể set trạng thái hiển thị đối với những người khác (trực tuyến, rời khỏi, bận hoặc ẩn).

Âm báo tin nhắn: khi nhận được tin nhắn mới sẽ có âm thanh thông báo cho người dùng biết.

* 1. **Các thành phần dữ liệu**

Với hầu hết các thời kỳ web, Hệ quản trị cơ sở dữ liệu quan hệ dựa trên SQL đã thống trị hầu hết các hệ Quản trị Cơ sở dữ liệu. Tuy nhiên, thời gian gần đây, một cách tiếp cận mới đã bắt đầu biết đến là NoSQL, tạo ra sự thay thế cho các hệ quản trị cơ sở dữ liệu quan hệ truyền thống.

NoSQL còn có nghĩa là Non-Relational - không ràng buộc. Tuy nhiên, thuật ngữ đó ít phổ dụng hơn và ngày nay người ta thường dịch NoSQL thành Not Only SQL - Không chỉ SQL. NoSQL ám chỉ đến những cơ sở dữ liệu không dùng mô hình dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mềm

Thuật ngữ NoSQL được giới thiệu lần đầu vào năm 1998 sử dụng làm tên gọi chung cho các cơ sở dữ liệu quan hệ nguồn mở nhỏ nhưng không sử dụng SQL cho truy vấn.

Vào năm 2009, Eric Evans, nhân viên của Rackspace giới thiệu lại thuật ngữ NoSQL khi Johan Oskarsson của Last.fm muốn tổ chức một hội thảo về cơ sở dữ liệu nguồn mở phân tán. Thuật ngữ NoSQL đánh dấu bước phát triển của thế hệ CSDL mới: một thế hệ CSDL không ràng buộc, phân tán, nguồn mở, khả năng mở rộng theo chiều ngang, có thể lưu trữ, xử lý từ một lượng rất nhỏ cho tới hàng petabytes dữ liệu trong hệ thống có độ chịu tải, chịu lỗi cao với những đòi hỏi về tài nguyên phần cứng thấp.

Dữ liệu được lưu ở dạng NoSQL không sử dụng mô hình quan hệ. NoSQL lưu trữ dữ liệu của mình theo dạng cặp giá trị “key – value”.

Do không sử dụng mô hình quan hệ nên sẽ không có ràng buộc nào cho việc nhất quán dữ liệu. Linh động, không cần phải định nghĩa cấu trúc dữ liệu trước khi tiến hành lưu trữ.

Thay cho việc lưu trữ dữ liệu vào các bảng có quan hệ với nhau như truyền thống, MongoDB lưu các dữ liệu cấu trúc dưới dạng giống với JSON(JavaScript Object Notation) và gọi tên là BSON.

Khả năng mở rộng tốt (distributed horizontally), khả năng cân bằng tải cao, tích hợp các công nghệ quản lý dữ liệu vẫn tốt khi kích thước và thông lượng trao đổi dữ liệu tăng.

Dữ liệu được lưu gồm có users( thông tin người dùng) , roles (vai trò), permissions(các quyền tương ứng với vai trò), rooms(kênh chat), message (tin nhắn).

Thông tin user gồm có username (tên), địa chỉ email, password (mật khẩu), roles (vai trò), avatar (ảnh đại diện), state(trạng thái).

Roles (vai trò) và permissions(các quyền) tương ứng gồm có admin (người quản trị), user (người dùng bình thường), guest (khách), livechat-agent (người hỗ trợ chat trực tuyến), livechat-manager (người quản lí các livechat-agent), owner (người tạo ra kênh), moderator (người quản lý kênh).

Messages (tin nhắn) gồm có các thông tin như id tin nhắn, id kênh, thời gian gửi tin nhắn, nội dung tin nhắn, thời gian chỉnh sửa.

Rooms(kênh chat) gồm có các thông tin như id, thời gian tạo, tên room, mảng các user có trong room, số lượng tin nhắn, thời gian cập nhật.

Ví dụ một document trong collection rooms:

{

"\_id" : "GENERAL",

"ts" : ISODate("2018-01-04T16:20:04.702Z"),

"name" : "general",

"usernames" : [ "ADMIN" ],

"msgs" : 1,

"\_updatedAt" : ISODate("2018-01-07T16:24:42.746Z")

}

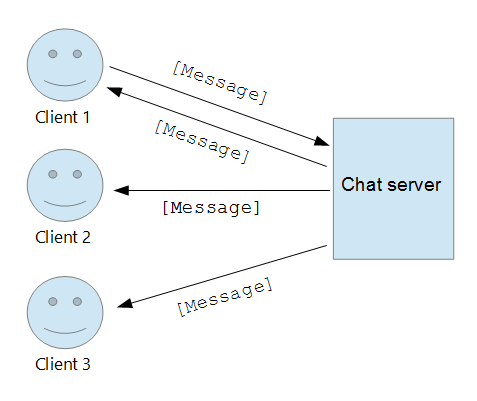
# CHƯƠNG 4: ỨNG DỤNG WEB CHAT TÍCH HỢP WEBSITE

* 1. **Giới thiệu.**

Ứng dụng chat trực tuyến cho phép 2 bên có thể trao đổi trực tiếp với nhau ngay tại thời gian thực.

* + 1. **Các chức năng chính.**
* Chat trực tuyến.
* Tạo, quản lý kênh chat**.**
  + 1. **Chi tiết chương trình.**

Mô hình tương tác của ứng dụng.



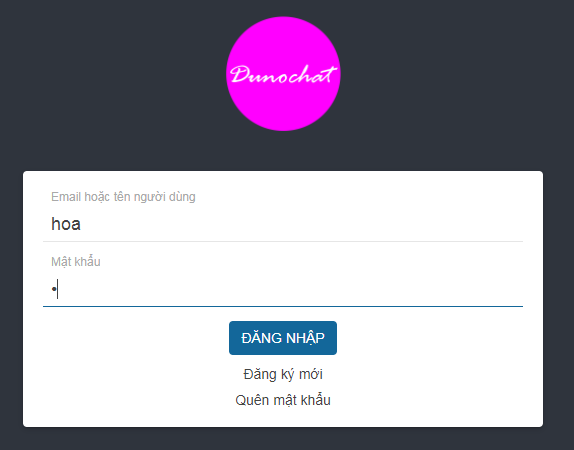
Hình 4.1 mô hình tương tác của ứng dụng.

Chat trực tuyến này từ phía client (có thể là khách, user hoặc admin) có thể tương tác với nhiều client 1, khi một client gửi tin nhắn cho một một client khác thì tin nhắn sẽ được đi qua server, tại server sẽ bắt các sự kiện rồi chuyển tiếp tin nhắn và thông báo cho các client nhận, đồng thời lưu tin nhắn lên database.

* **Đăng ký tài khoản.**

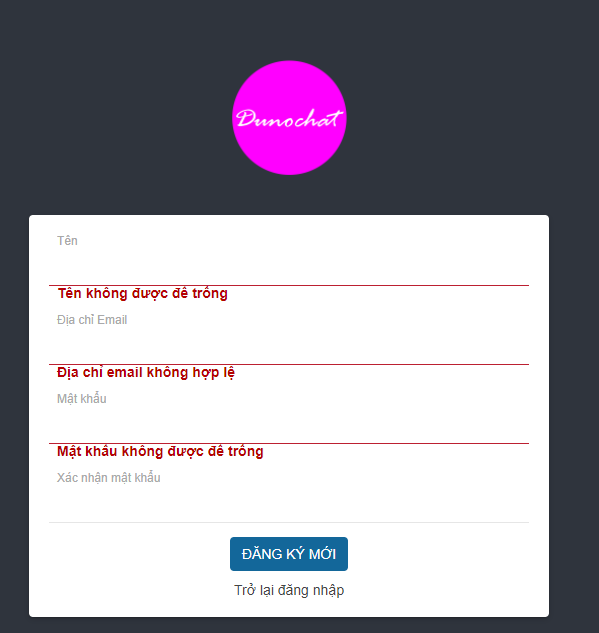
Trong phần này client (có thể là khách, user hoặc admin) sẽ trực để tiếp nhận yêu cầu từ client khác thông qua giao diện quản lý kênh.

Qua **Hình 4.3** ở dưới ta có thể thấy.



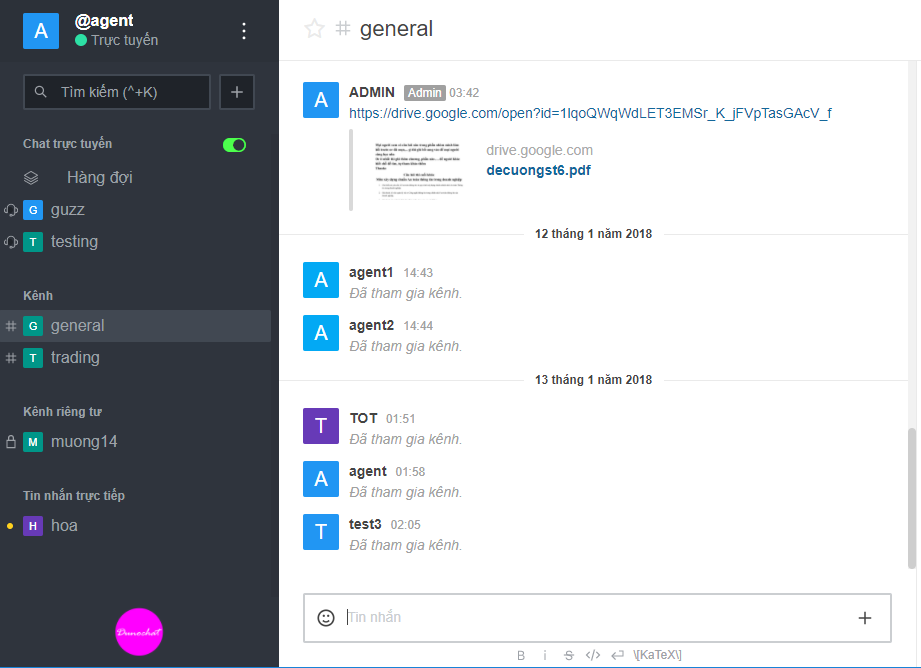
Hình 4. 2 giao diện đăng nhập

* Ở giao diện này có phải có 1 tài khoản để đăng nhập vào giao diện kênh, tại mục đăng nhập ta có thể tạo một tài khoản mới ở dưới nút đăng nhập, khi tạo tài khoản không thể thiếu, hoặc bỏ trống bất cứ mục nào. Hệ thống sẽ thông báo lỗi nếu ta bỏ sót bất cứ 1 trong các mục nào.



Hình 4. 3 giao diện đăng ký

* Khi đăng nhập hệ thống không cho phép bỏ trống tại các ô Email và mật khẩu.
* Khi đã điền đúng thông tin tài khoản đã đăng ký thì ta sẽ vào giao diện quản lý, trao đổi kênh như ở hình 5.5.

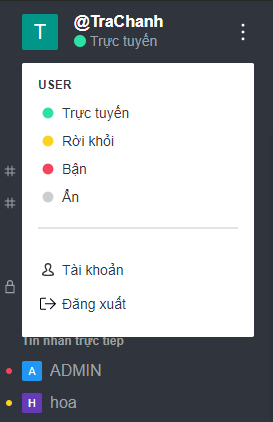


**Hình 4.4** giao diện quản lý, chat của quản lý

Ở giao diện này ta có thể thấy bên trái là khu vực kênh đã được tạo và ta có thể tương tác với các kênh trên đó. Bên phải là giao diện hiển thị nội dung tin nhắn.

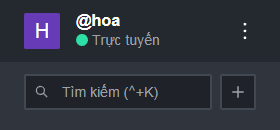
* Ở đây 1 kênh có thể có nhiều đối tượng tham gia trao đổi, tuy nhiên còn tùy thuộc vào tầm vực mà người tạo kênh muốn. Ở mục nay có 3 kiểu tầm vực kênh : kênh mở rộng, kênh riêng tư, và mục các kênh dành cho tin nhắn trực tiếp. Ta sẽ lướt từ trên xuống dưới để tìm hiểu kỹ hơn về ứng dụng.

Đầu tiên là mục hiển thị tên tài khoản, hình ảnh của tài khoản. Tại mục này ta thấy khi nhấp vào thì ta sẽ thấy được trạng thái hoạt động và các tác vụ có thể tương tác với giao diện hiện hành.



**Hình 4.5** giao diện hiển thị thông tin, trạng thái của tài khoản

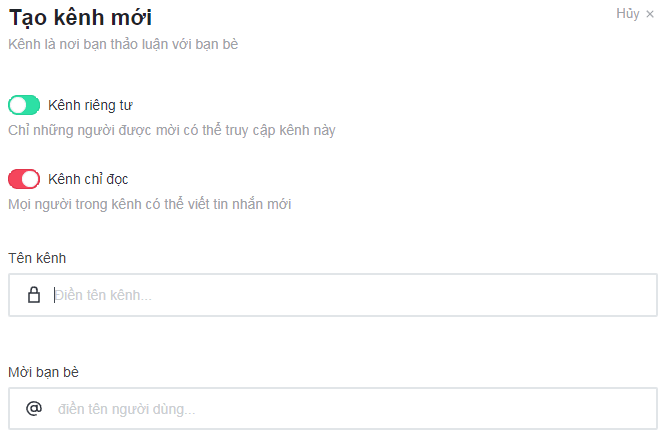
Chỉ mục tìm kiếm kênh, tạo kênh và live chat.



**Hình 4.6** giao diện tìm kiếm, tạo kênh

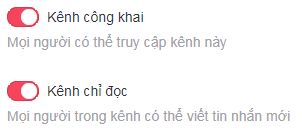
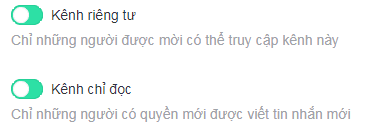
Tại ô tìm kiếm ta có thể tìm bất kỳ một kênh nào mà ta đã tạo trước đó và tùy tầm vực của kênh đó mà ta có thể bắt đầu kênh đó.

* Tại nút tạo kênh khi nhấn vào thì bên phải giao diện chính sẽ hiển thì giao diện tạo kênh.



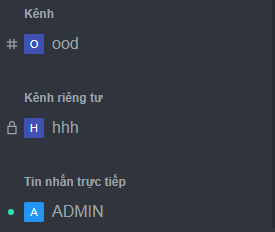
Hình 4.7 giao diện tạo kênh

Cũng đã nếu phía trên, thì tại giao diện tạo kênh này tùy vào mục đích trao đổi mà ta có thể lựa chọn các phương thức, tầm vực của kênh.



**Hình 4.8** các tầm vực của kênh

* Ở mục tầm vực kênh, như đã nói phía trên thì ta có 3 tầm vực cho kênh như: kênh mở rộng, kênh riêng tư, kênh có tin nhắn trực tiếp.



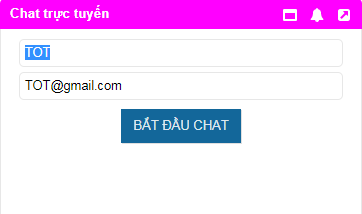
**Hình 4.9** giao diện tầm vực của kênh

* **Kênh mở rộng**: từ phần chọn kênh ta chọn phần kênh công khai (mọi người có thể truy cập kênh này) và kênh chỉ đọc (mọi người trong kênh có thể viết tin nhắn mới). Khi kênh này được tạo ra thì mọi tài khoản đều nhìn thấy và có thể tham gia vào kênh đó.
* **Kênh riêng tư**: ta chọn kênh riêng tư (chỉ có những người được mời có thể truy cập kênh) và kênh chỉ đọc (chỉ những người có quyền mới được viết tin nhắn mới). tại kênh này kênh được tạo ra được admin của kênh mời các tài khoản khác vào tham gia trao đổi kín. Khi một tài khoản truy cập vào giao diện quản lý kênh thì tạo mục kênh riêng tư sẽ không thấy được kênh riêng tư của tài khoản khác. Còn với lênh mở rộng thì ngược lại vậy.
* **Tại mục tin nhắn trực tiếp**: là các khu vực mà ta có thể chat với các tài khoản mà trước đó ta có liên hệ trước đó, mà lịch sử của tài khoản còn ghi lại, qua mục tìm kiếm ta có thể lựa chọn các tài khoản đã liên hệ rồi nó sẽ hiện tròn phần mục tin nhắn trực tiếp. Tại mục này ta có thể xem nó như 1 kênh mở rộng vậy, các tài khoản đều có thể chat với nhau thông qua kênh này mà không phải ràng buộc gì.

* **Chat trực tuyến:**

Khi một khách hàng nào đó có nhu cầu trao đổi thông tin thì sẽ truy cập vào mục chat trực tuyến đã được tích hợp trên website và nhập các thông tin cần thiết để bắt đầu chat với các người hỗ trợ.

Tại mục chat trực này từ phía khách không cần phải tạo tài khoản như lúc vào giao diện kênh.

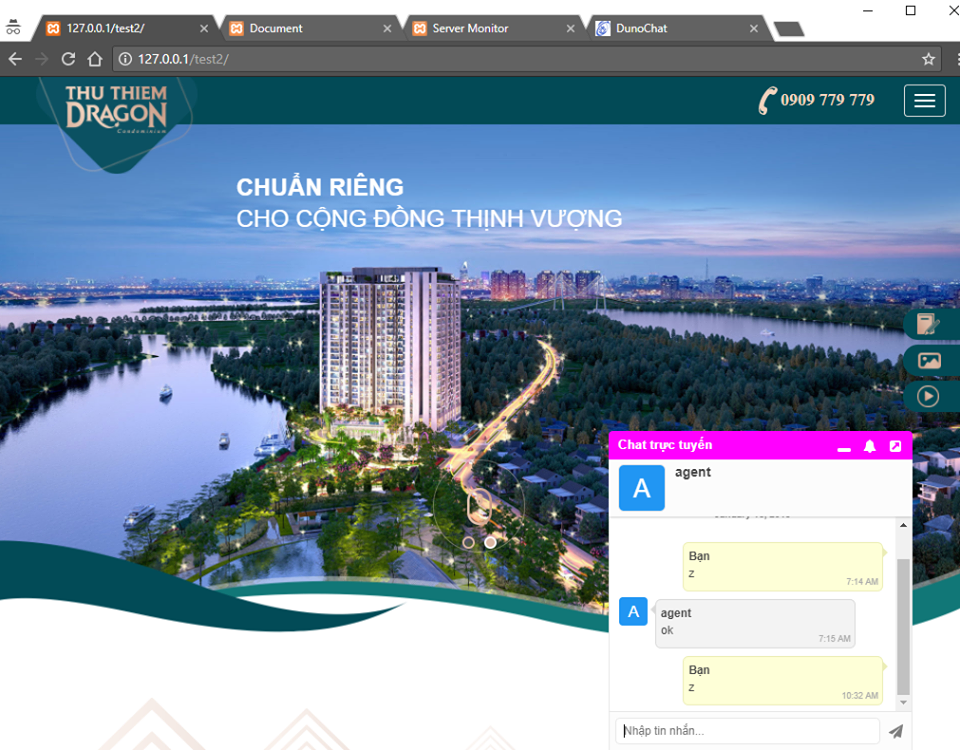


Hình 4.10 giao diện nhập thông tin để chat trực tuyến trên website khác

Phía client nhập các thông tin như Name (Tên) và Email (Tài khoản Email),

* Name (Tên): Ở đây có thể là bất cứ tên gì vì name cũng được lấy để bặt cho tên của room chat (phòng chat) với quản trị viên.
* Email: để có thể xác nhận và là thông tin liên lạc khi có sự cố hoặc một việc phần bản hồi nếu phía client đã offline (rời khỏi room chat).

Khi đã tạo điều đầy đủ thông tin thì ta có thể bắt đầu vào giao diện chat.



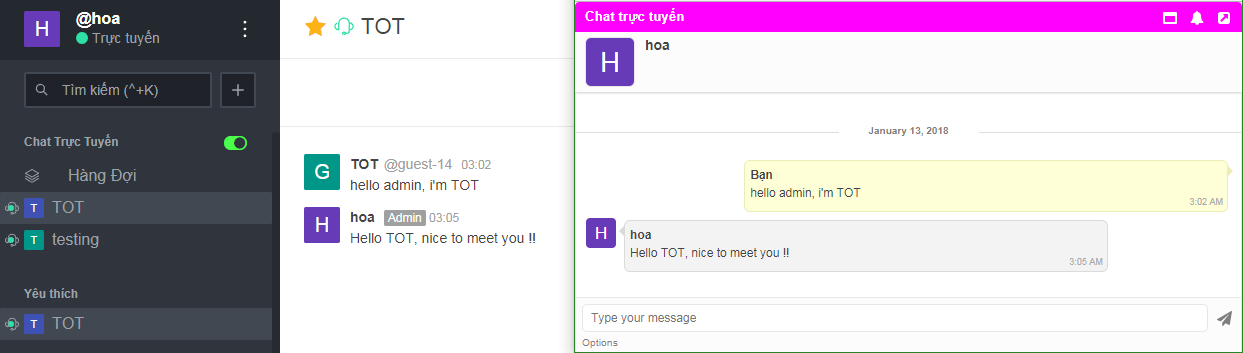
**Hình 4.11** giao diện chat trực tuyến gắn trên website khác

Khi vào giao diện chat ta có thể thấy được kênh cũng là tên của quản trị viên mà ta đang chat (nếu quản trị viên online).

* Mọi tin nhắn, thông tin chat đều được hiện bằng thời gian và ngày tháng.

Còn tại mục chat trực tuyến: ta cũng đã nói ở phía trên, khi một user truy cập vào giao diện chat trực tuyến nếu có quản trị viên online thì ta sẽ thấy được user cùng với thông điệp của user.

* Ngoài ra bên phải giao diện hiển thị tin nhắn trao đổi, ta còn có phần yêu thích để đánh dấu lại kênh mà ta yêu thích.



Hình 4.12 giao diện chat trực tuyến

# CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

* 1. **Kết luận.**

Qua giai đoạn nghiên cứu và thực hiện tiểu luận để trình bày, tổng kết một cách hệ thống và cụ thể các kiến thức nền tảng, các kiến thức về công nghệ để có thể hoàng thành ứng dụng. Thông qua đó ta thấy việc trao đổi thông tin cho nhau qua ứng dụng chat trở nên dễ dàng, thuận tiện và nhanh chóng hơn. Ứng dụng chạy tương đối ổn định.

Cửa sổ chat trực tuyến có thể tích hợp trên nhiều trang. Tuy nhiên ứng dụng bị lỗi hiển thị khi hoạt động trên trình duyệt của smartphone và chức năng quên mật khẩu còn bị lỗi.

Dưới sự hướng dẫn của thầy Đàm Quang Hồng Hải, nhóm đã hoàn thành được ứng dụng web chat của mình.

Tuy nhiên, do lượng thời gian làm tiểu luận cũng có hạn cũng như nhiều sự thiếu sót về kiến thức kiến thức, kinh nghiệm thực hành các công nghệ nhóm vẫn chưa nâng cao, hoàn thành tất cả các chỉ tiêu đề ra.

Và qua đó, trong thời gian tới nhóm sẽ cố gắng trau dồi, học hỏi thêm kiến thức, kinh nghiệm thực hành, nâng cao trình độ bản thân để nâng cấp, hoàn thành chỉnh chu ứng dụng nhất.

* 1. **Hướng phát triển.**

Để ứng dụng hoàn thiện hơn, nhóm chúng em cần cải thiện một số chức năng để áp dụng được rộng rãi trong các dự án lớn sau này.

Một số tính năng có thể phát triển thêm.

Sau này có thể phát triển thêm các chức năng như truyền file, voice chat và video chat.

Ngoài ra có thể phát triển thêm ứng dụng client trên mobile cho các hệ điều hành android và ios.

**TÀI LIỆU THAM KHẢO**

[1] <https://nodejs.org/en/>

[2] <https://socket.io/get-started/chat/>

[3] <http://getbootstrap.com/>

[4] <https://www.w3schools.com/>

[5] <https://docs.mongodb.com/>

[6] <http://luanvan.co/luan-van/tim-hieu-ve-nodejs-45178/>

**PHỤ LỤC**

**Thông tin giảng viên hướng dẫn  
 TS. Đàm Quang Hồng Hải** Khoa Mạng máy tính và Truyền thông  
 Trường Đại học Công nghệ Thông tin  
 Khu phố 6, phường Linh Trung, Quận Thủ Đức, TP. Hồ Chí Minh  
 Email: damhai@uit.edu.vn  
**Thông tin sinh viên thực hiện:  
*Phạm Kim Chấn Nguyên*** Sinh viên khóa 2012-2017 khoa Mạng máy tính và Truyền thông  
 Trường Đại học Công Nghệ Thông Tin  
 Điện thoại: 0935523302  
 Email: 12520295@gm.uit.edu.vn  
***Võ Đức Hòa*** Sinh viên khóa 2012-2017 khoa Mạng máy tính và Truyền thông  
 Trường Đại học Công Nghệ Thông Tin  
 Điện thoại: 01636304386  
 Email: 12520147@gm.uit.edu.vn