**BỘ THÔNG TIN VÀ TRUYỀN THÔNG**

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

------------------------------



**BÁO CÁO THỰC TẬP TỐT NGHIỆP ĐẠI HỌC**

***Đề tài* :**

**NGĂN CHẶN TẤN CÔNG INJECTION VÀO NOSQL**

**Người hướng dẫn TS. HUỲNH TRỌNG THƯA**

**Sinh viên thực hiện NGUYỄN BÁ LÂM**

**Mã số sinh viên N14DCA141**

**Lớp D14CQAT01-N**

**Khóa 2014**

**Hệ ĐẠI HỌC CHÍNH QUY**

**TP.HCM, tháng 8/2018**

LỜI CẢM ƠN

Kính thưa quý thầy cô !

Trong quá trình thực hiện đề tài “ **Ngăn chặn tấn công Injection vào NoSQL**” em đã nhận được nhiều sự giúp đỡ, tạo điều kiện của tập thể Ban giám Hiệu, Khoa Công Nghệ Thông Tin, các thầy cô, cán bộ các phòng ban chức năng Học Viện Công Nghệ Bưa Chính Viễn Thông tại TP.HCM .Em xin bày tỏ làm cảm ơn chân thành về sự giúp đỡ đó.

Đặc biệt ,em xin bày tỏ lòng biết ơn sâu sắc tới ThS.Huỳnh Trọng Thưa. Thầy đã hướng dẫn tận tình, truyền đạt kiến thức, chỉ bảo em hoàn thành đề tài này.

Sau cùng , em xin kính chúc các quý thầy cô Khoa Công Nghệ Thông Tin và ThS. Huỳnh Trọng Thưa thật dồi dào sức khỏe , niềm tin để tiếp tục ương mần thế hệ mai sau.

Trân trọng !

TP.Hồ Chí Minh,Ngày 8 tháng 8 năm 2018

Sinh viên thực hiện

**Nguyễn Bá Lâm**

MỤC LỤC

MỞ ĐẦU

[CHƯƠNG I : TÌM HIỂU VỀ NOSQL DATABASE 9](#_Toc521446569)

[1.1 NoSQL là gì ? 9](#_Toc521446570)

[1.1.1 Định nghĩa 9](#_Toc521446571)

[1.1.2 Một số thuật ngữ liên quan. 9](#_Toc521446572)

[1.2 Phân loại cơ sở dữ liệu NoSQL 10](#_Toc521446573)

[1.2.1 Key – Value Store 10](#_Toc521446574)

[1.2.2 Document Databases 11](#_Toc521446575)

[1.2.3 Column Family Stores 12](#_Toc521446576)

[1.2.4 Graph Database 13](#_Toc521446577)

[1.3 MongoDB và một số thuật ngữ 14](#_Toc521446578)

[1.3.1 Khái niệm 15](#_Toc521446579)

[1.3.2 Một số thuật ngữ hay sử dụng trong MongoDB 15](#_Toc521446580)

[1.4 Các kiểu dữ liệu trong MongoDB 16](#_Toc521446581)

[1.5 Một số câu lệnh dùng trong MongoDB 17](#_Toc521446582)

[1.5.1 MongoDB – Tạo Database 17](#_Toc521446583)

[1.5.2 MongoDB – Xóa Database 17](#_Toc521446584)

[1.5.3 MongoDB – Tạo Collection 17](#_Toc521446585)

[1.5.4 MogoDB - Xóa Collection 17](#_Toc521446586)

[1.5.5 MongoDB – Chèn Document 17](#_Toc521446587)

[1.5.6 MongoDB – Truy vấn Document 18](#_Toc521446588)

[1.5.7 MongoDB - Cập nhập Document 18](#_Toc521446589)

[1.5.8 MongoDB - Xóa Document 19](#_Toc521446590)

[CHƯƠNG II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS 20](#_Toc521446591)

[2.1 Tổng quan về Node.js 20](#_Toc521446592)

[**2.1.1** Node.js là gì ? 20](#_Toc521446593)

[**2.1.2** Các đặc tính của Node.js 20](#_Toc521446594)

[**2.1.3** Node.js làm việc như thế nào 20](#_Toc521446595)

[**2.1.4** NPM : The Node Package Manager 21](#_Toc521446596)

[2.2 Khái niệm về Callback trong Node.js 22](#_Toc521446597)

[2.3 Khái niệm Event Loop 23](#_Toc521446598)

[2.4 Lớp Event Emitter 24](#_Toc521446599)

[2.5 Buffer trong Node.js 26](#_Toc521446600)

[2.6 Stream trong Node.js 27](#_Toc521446601)

[2.7 Đọc ghi file trong Node.js 29](#_Toc521446602)

[2.8 Đối tượng toàn cục 30](#_Toc521446603)

[2.9 Module trong Node.js 32](#_Toc521446604)

[CHƯƠNG III : TÌM HIỂU CÁCH TẤN CÔNG VÀ NGĂN CHẶN INJECTION VÀO NOSQL QUA ỨNG DỤNG WEB NODE.JS 35](#_Toc521446605)

[3.1 Tổng quan về Injection NoSQL 35](#_Toc521446606)

[3.1.1 Cơ chế hoạt động 35](#_Toc521446607)

[3.1.2 Các Loại Tấn Công 36](#_Toc521446608)

[CHƯƠNG IV : XÂY DỰNG ỨNG DỤNG WEB ĐƠN GIẢN BẰNG NODE.JS NGĂN CHẶN TẤN CÔNG INJECTION VÀO NOSQL 38](#_Toc521446609)

[4.1 Tổng quan về ứng dụng Facebook Massenger 38](#_Toc521446610)

[4.1.1 Các công nghệ sử dụng trong việc xây dựng ứng dụng 38](#_Toc521446611)

[4.1.2 Một số tính năng của ứng dụng Facebook Massenger 38](#_Toc521446612)

[4.2 Tiến hành cài đặt và chạy chương trình trên Ubuntu 38](#_Toc521446613)

[4.3 Cài đặt một số tính năng tiêu biểu 39](#_Toc521446614)

[4.3.1 Đăng nhập và Đăng kí 39](#_Toc521446615)

[4.3.2 Lưa trữ và kết nối MongoDB 41](#_Toc521446616)

[4.3.3 Quản lý user và Group chat bằng session 41](#_Toc521446617)

[4.3.4 Ngăn chặn tấn công Injection vào NoSQL 41](#_Toc521446618)

[CHƯƠNG V : KẾT LUẬN 42](#_Toc521446619)

[5.1 Kết quả đạt được 42](#_Toc521446620)

[5.2 Hạn chế 42](#_Toc521446621)

[DANH MỤC TÀI LIỆU THAM KHẢO 43](#_Toc521446622)

DANH MỤC KÍ HIỆU VÀ TỪ VIẾT TẮT

|  |  |
| --- | --- |
| **Từ viết tắt** | **Ý nghĩa** |
| RBDMS | Relational Database Management System |
| NoSQL | Not Relational SQL |
| JSON | JavaScript Object Notation |
| SQL | Structured Query Language |
| NPM | Node Package Manager |
| API | Application Programming Interface |
| HTTP | HyperText Transfer Protocol |

**DANH SÁCH CÁC HÌNH , BẢNG**

Hình 1.1. Key –Value Database 14

Hình 1.2. Key –Value Store.. 16

Hình 1.3. Key-Value và Document 17

Hình 1.4. Document Store 17

Hình 1.5. Super Column Family 18

Hình 1.6. Graph Database Store 19

Hình 1.7 . NoSQL Graph Database 20

Hình 1.8 MongoDB 23

Hình 2.1 . Cơ chế hoạt động của Node.js 31

Hình 2.2. Event Loop 37

Hình 2.3 Cấu trức ứng dụng web 49

Hình 3.1 Cơ chế hoạt động của NoSQL 52

Hình 4.1 Cấu trúc ứng dụng Facebook Messenger 55

Bảng 1. So sánh SQL và NoSQL 17

Bảng 2. Một số kiểu dữ liệu phổ biến **20**

Bảng 3. Phương thức của đối tượng toàn cục của console trong node.js 43

Bảng 4. Phương thức của đối tượng toàn cục của process trong node.js 44

**MỞ ĐẦU**

Ngày nay, khi Internet được phổ biến rộng rãi ,các tổ chức đều có nhu cầu giới thiệu thông tin của mình trên xa lộ thông tin cũng như thực hiện các phiên giao dịch trực tuyến. Vấn đề nảy sinh là vi phạm ứng dụng của các ứng dụng web ngày càng mở rộng thì khả năng xuất hiện lỗi và bị tấn công càng cao, trở thành đối tượng cho nhiều người tấn công với các mục đích khác nhau. Đôi khi cũng chỉ đơn giản là để thử tài hoặc đùa bởn với người khác.

Cùng với sự phát triển không ngừng của Internet và các dịch vụ trên Internet, số lượng dịch vụ tấn công trên Internet cũng tăng theo cấp số nhân. Trong khi các phương tiện thông tin đại chúng ngày càng nhắc nhiều đến những khả năng truy cập thông tin của Internet, thì các tài liệu chuyên môn bắt đầu đề cập đến nhiều vấn đề bảo đảm và an toàn dữ liệu cho các máy tính kết nối vào mạng Internet.

Vì thế vấn đề bảo đảm và an toàn dữ liệu cho các máy tính được kết nối vào mạng Internet là cần thiết nhằm mục đích bảo vệ dữ liệu, bảo vệ thông tin doanh nghiệp, người dùng và bảo vệ hệ thống.

Tuy đã có nhiều cố gắng từ các nhà lập trình ứng dụng Web nhưng vẫn không thể đảm bảo ngăn chặn toàn bộ vì công nghệ Web đang phát triển nhanh chóng nên dẫn đến nhiều khuyết điểm mới phát sinh. Sự tấn công không nằm trong một vài khuôn khổ mà linh động và tăng lên tùy vào những sai sót của nhà quản trị hệ thống cũng như của những người lập trình ứng dụng.

Vì vậy, đề tài “ ngăn chặn tấn công Injection vào Nosql “này được thực hiện với mục đích tìm hiểu, phân tích các lỗ hổng bảo mật trong các ứng dụng Web sử dụng cơ sở dữ liệu Nosql .Từ đó tìm ra những giải pháp khắc phục tốt nhất để mang đến cho người sử dụng những ứng dụng web an toàn.

# 

# 

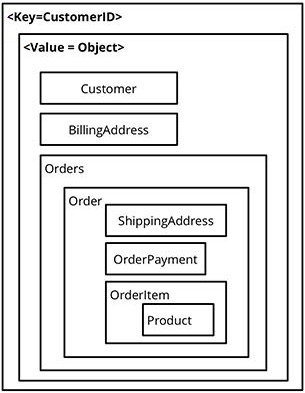
# CHƯƠNG I : TÌM HIỂU VỀ NOSQL DATABASE

* 1. NoSQL là gì ?
     1. Định nghĩa

NoSQL là một xu hướng cơ sở dữ liệu mà không dùng mô hình dữ liệu quan hệ để quản lý dữ liệu trong lĩnh vực phần mền. NoSQL có nghĩa là No-Relational (NoRel) – Không rằng buộc. Tuy nhiên thuật ngữ đó ít phổ dụng hơn và ngày ngày người ta thường dịch NoSQL thành Not Only SQL – Không chỉ SQL .

Vào năm 2009, Eric Evans, nhân viên của Rackspace giới thiệu lại thuật ngữ NoSQL trong một hội thảo về cơ sở dữ liệu nguồn mở phân tán. Thuật ngữ NoSQL đánh dấu bước phát triển của thế hệ database mới: distributed (phân tán) + non-relational (không ràng buộc).

Dữ liệu được lưa dưới dạng object . mặc dù bị trùng lặp nhưng truy vấn rất nhanh và đơn giản.



Hình 1.1. Key/ Value Database

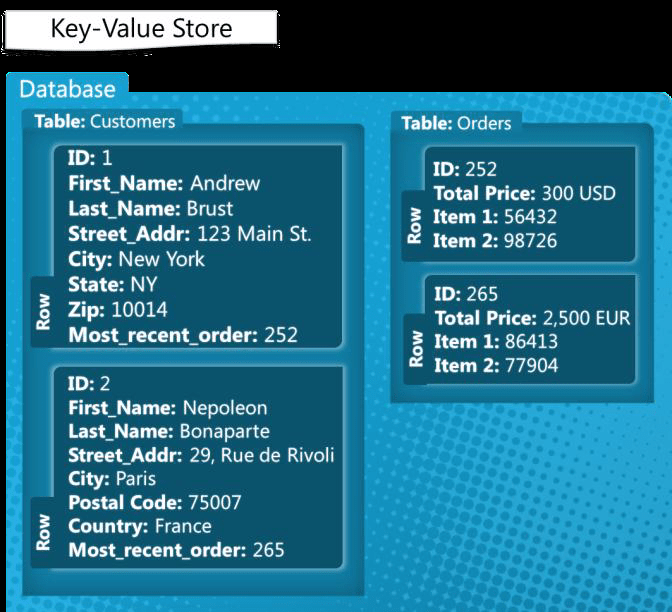
* + 1. Một số thuật ngữ liên quan.
* Relational (Rằng buộc) thuật ngữ sử dụng đến các mối quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ (RDBMs) sử dụng mô hình khóa gồm 2 loại khóa: khóa chính và khóa phụ( primary key + foreign key) để rằng buộc dữ liệu nhàm thể hiện tính nhất quán dữ liệu từ các bẳng khác nhau.
* Non- Relational (Không rằng buộc ) là khái niệm không sử dụng các rằng buộc dữ liệu cho nhất quán dữ liệu ở NoSQL database.
* High Availability (Tính sẵn sàng) : Do chấp nhân sự trùng lặp trong lưa trữ nên nếu một node( commodity machine) nào đó bị chết cũng không ảnh hưởng tới toàn bộ hệ thống.
* High Scalability ( khả năng mở rộng ) : Gần như không có một giới hạn cho dữ liệu và người dùng trên hệ thống.
* Eventual consistency (Nhất quán cuối) : Tính nhất quán của dữ liệu không cần phải đảm bảo ngay tức khác sau mỗi phép write. Một hệ thống phân tán chấp nhận những ảnh hưởng theo phương thức lan truyền và sau một khoảng thơi gian ( không phải ngay tức khắc), thay đổi sẽ di đến mọi điểm trong hệ thống, tức là cuối cùng ( eventually) dữ liệu trên hệ thống sẽ trở lại trạng thái nhất quán
* Vertical scalable ( khả năng mở rộng theo chiều dọc ): Khi dữ liệu lớn về lượng, phương pháp tăng cường khả năng lưa trữ và xử lý bằng việc cải tiến phần mền và cải thiện phần cứng trên một máy tính đơn lẻ được gọi là khả năng mở rộng chiều dọc.
* Horizontal scalable (khả năng mở rộng chiều ngang) : Khi dữ liệu lớn về lượng , phương pháp tăng cường khả năng lưa trữ và xử lý dùng nhiều máy tính phân tán .Phân tán dữ liệu được hộ trợ bởi các phần mền tức cơ sở dữ liệu.
* Distributed Data ( phân tán dữ liệu ): Mô hình lưa trữ phân tán các file hoặc dữ liệu ra nhiều máy tính khác nhau trong mạng LAN hoặc Internet dưới sự kiểm soát của phần mền .
* Deployment Flexibility ( Triển khai linh hoạt ): Việc bổ sung thêm/loại bỏ các node , hệ thống sẽ tự động nhận biết để lưa trữ mà không cần phải can thiệp bằng tay. Hệ thống cũng không đòi hỏi cấu hình mạnh , đồng nhất .
* Durability ( Lưa trữ tốt) : Dữ liệu có thể tồn tại trong bộ nhớ máy tính nhưng đồng thời cũng được lưa trữ lại đĩa cứng.
  1. **Phân loại cơ sở dữ liệu NoSQL**

Tính đến thời điểm hiện tại thì có hơn 255 cơ sở dữ liệu NoSQL (<http://nosql-database.org/>) . Các loại cơ sở dữ liệu này không giống nhau về các lưa trữ. Và chúng được chia thành những nhóm sau.

* + 1. Key – Value Store

Key –value stores là kiểu lưa trữ đơn giản nhất trong các loại cơ sở dữ liệu NoSQL đồng thời nó cũng là kiểu lưa trữ cho tất các hệ quản trị cơ sở dữ liệu NoSQL.

Ý tưởng chính của kiểu Key – Value là sử dụng bảng băm ( hash-table) , nới có một khóa duy nhất và một con trỏ chỉ đến một mục dữ liệu cụ thể. Các mô hình key/value thì đơn giản và dễ thực hiện nhất. Nhưng nó sẽ không hiệu quả khi bạn chỉ truy vấn hay update đến một phần dữ liệu trong phần giá trị.



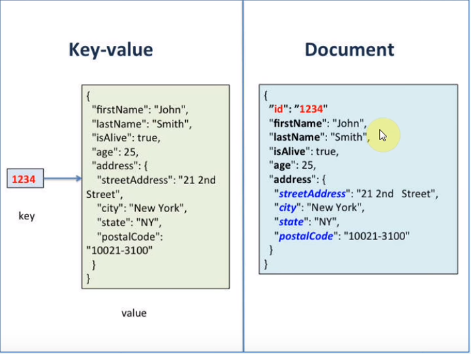
Hình 1.2. Key – Value Store

Với sự đơn giản của cách lưa trữ Key-Value làm cho các cơ sở dữ liệu loại này rất phù hợp với các ứng dụng cần truy xuất nhanh chóng và khả năng mở rộng cao.

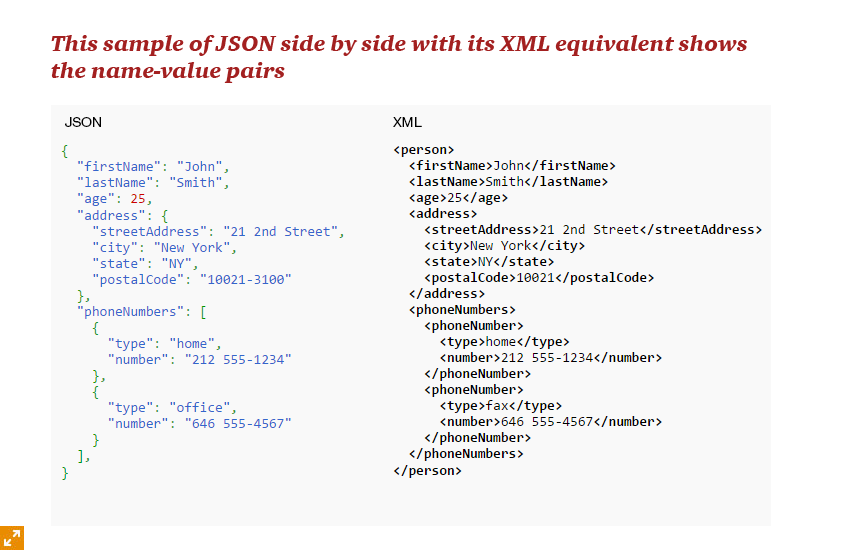
Một số cơ sở dữ liệu NoSQL tiêu biểu cho dạng này: Aerospike, CouchDB, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, Berkeley DB

* + 1. Document Databases

Cơ sở dữ liệu Document được thiết kế để quản lý và lưa trữ dữ liệu ở dạng document. Nó được lấy cảm hứng tử Lotus Noted và nó cũng có kiểu lưa trữ tương tự key –value stores. Mô hình Document database nó giống như tập hợp các tài liệu , trong các tài liệu nó có các key - value. Nhưng document này được lưa ở định dạng JSON. Trong document có nhiểu cặp key-value và các cặp key-value này được phép lồng nhau. Mỗi document sẽ có một key, kiểu cơ sở dữ liệu document này hộ trợ truy vấn hiệu quả hơn.



Hình 1.3 Key-Value và Document



Hình 1.4 . Document Store

− Cơ sở dữ liệu Document phù hợp cho việc lưa trữ và quản lý dữ liệu có kích thước lớn như tài liệu văn bản, tin nhắn, cũng như biểu diễn một thực thể cơ sở dữ liệu là Product hay Customer

− Một số cơ sở dữ liệu NoSQL tiêu biểu cho dạng này : Apache Jackrabbit, CouchDB, IBM Lotus Notes Storage Format (NSF), MongoDB, Terrastore, ThruDB, OrientDB, RavenDB,...

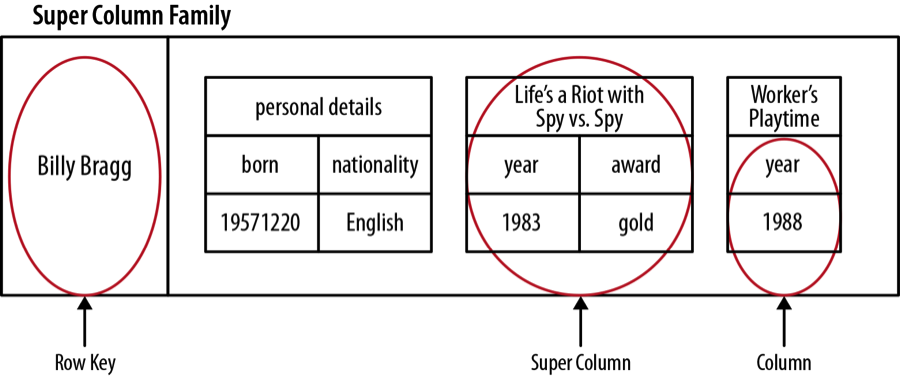
* + 1. Column Family Stores

Cơ sở dữ liệu Column Family Stores được tạo ra để lưa và xử lý một lượng dữ liệu lớn trên nhiều server. Cơ sở dữ liệu Column-family lưa trữ dữ liệu trong nhiều cột trong mỗi dòng với key cho từng dòng. Column families là một nhóm dữ liệu các dữ liệu liên quan được truy cập cùng nhau.

Với các cơ sở dữ liệu Column Family , chúng ta cần quan tâm đến các khái niệm sau :

− Column Family ( Họ cột) : Một column family là cách thức dữ liệu được lưa trữ trên đĩa. Tất cả dữ liệu trong một cột sẽ được lưa trên cùng một file. Một column family có thể chứa super column hoặc column.

− Super column ( Siêu cột) : Một super column có thể được dùng như một dictionnary ( kiểu từ điển ) . Nó là một column có thể chứa những column khác ( mà không phải là super column ).



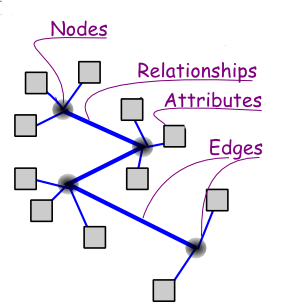
Hình 1.5. Super Column Family

.

Một số cơ sở dữ liệu NoSQL tiêu biểu cho dạng này :Hadoop/HBase – Apache, BigTable – Google, Cassandra - Facebook/Apache, Hypertable - Zvents Inc/Baidu, Cloudera, SciDB, Mnesia, Tablets,…

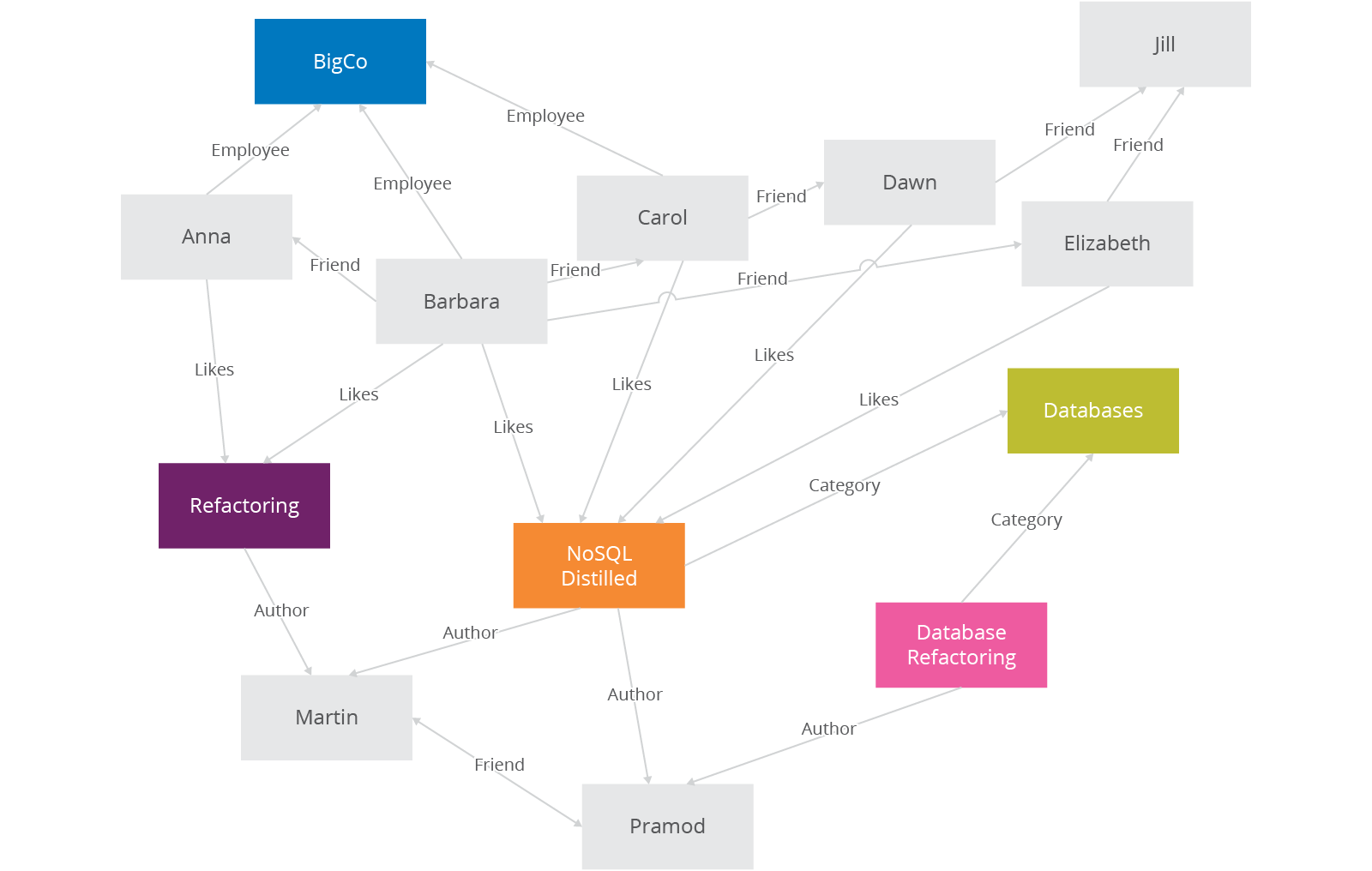
* + 1. Graph Database

Cơ sở dữ liệu đồ thị là một cơ sở dữ liệu dạng đồ thị sử dụng các cấu trúc đồ thị với nút (nodes) các mối quan hệ ( relationships) , các thuộc tính (properties) để mô tả và lưa trữ dữ liệu



Hình 1.6. Graph Database Store

Trong thực tế,hầu hết các trang web mạng xã hội đều sử dụng một số hình thức của garph database để làm những việc mà chúng ta đã biết như :kết bạn, bạn của bạn.



Hình 1.7. NoSQL Graph Database

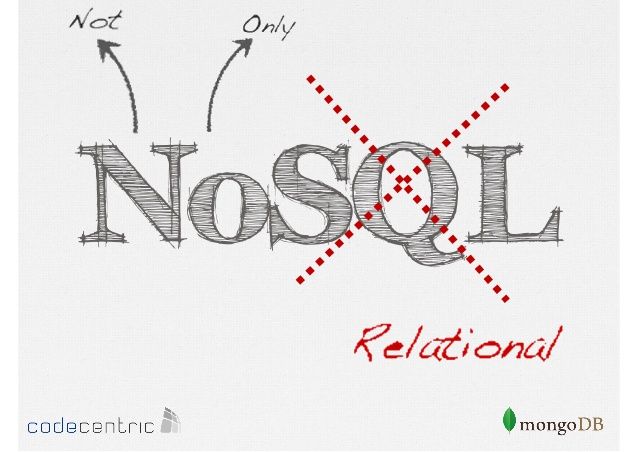
Một vấn đề đối với việc mở rộng Graph Database là rất khó khăn để tìm thấy một đồ thị con độc lập, có nghĩ là rất khó khăn để ta phân tán Graph Database thành nhiều mảnh .

Một số sản phẩm tiêu biểu của Graph database là : Neo4j , Sones , AllegroGraph , Core Data, Dex, FlockDB, InfoGrid, OpenLink Virtuoso…

* 1. **MongoDB và một số thuật ngữ**
     1. Khái niệm

MongoDB là một cơ sở dữ liệu mã nguồn mở và là cơ sở dữ liệu NoSQL hàng đầu, được hàng triệu người sử dụng. MongoDB được viết bằng C++.

MongoDB là một cơ sở dữ liệu đa nền tảng, hoạt động trên các khái niệm Collection và Document, nó cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng



Hình 1.8 MongoDB

* + 1. Một số thuật ngữ hay sử dụng trong MongoDB
* \_id - Là trường bắt buộc có trong mỗi document. Trường \_id đại diện cho một giá trị duy nhất trong document MongoDB. Trường \_id cũng có thể được hiểu là khóa chính trong document.
* Collection - Là nhóm của nhiều document trong MongoDB. Collection có thể được hiểu là một bảng tương ứng trong cơ sở dữ liệu RDBMS (Relational Database Management System). Collection nằm trong một cơ sở dữ liệu duy nhất. Các collection không phải định nghĩa các cột, các hàng hay kiểu dữ liệu trước
* Cursor - Đây là một con trỏ đến tập kết quả của một truy vấn. Máy khách có thể lặp qua một contrỏ để lấy kết quả
* Database - Nơi chứa các Collection, giống với cơ sở dữ liệu RDMS chúng chứa các bảng. Mỗi Database có một tập tin riêng lưu trữ trên bộ nhớ vật lý. Một mấy chủ MongoDB có thể chứa nhiều Database.
* Document - Nơi chứa các Collection, giống với cơ sở dữ liệu RDMS chúng chứa các bảng. Mỗi Database có một tập tin riêng lưu trữ trên bộ nhớ vật lý. Một mấy chủ MongoDB có thể chứa nhiều Database.
* Field - Là một cặp name – value trong một document. Một document có thể có không hoặc nhiều trường. Các trường giống các cột ở cơ sở dữ liệu quan hệ
* JSON -. Con người có thể đọc được ở định dạng văn bản đơn giản thể hiện cho các dữ liệu có cấu trúc. Hiện tại JSON đang hỗ trợ rất nhiều ngôn ngữ lập trình.
* Index - Là những cấu trúc dữ liệu đặc biệt, dùng để chứa một phần nhỏ của các tập dữ liệu một cách dễ dàng để quét. Chỉ số lưu trữ giá trị của một fields cụ thể hoặc thiết lập các fields, sắp xếp theo giá trị của các fields này
  1. **Các kiểu dữ liệu trong MongoDB**

|  |  |  |
| --- | --- | --- |
| Type | Number | Alias |
| Double | 1 | “ double” |
| String | 2 | “ string” |
| Object | 3 | “object” |
| Array | 4 | “array” |
| Binary data | 5 | “binData” |
| Undefined | 6 | “undefined” |
| Boolen | 8 | “bool” |
| Data | 9 | “date” |
| Null | 10 | “null |

Bảng 2 : Một số kiểu dữ liệu phổ biến

* Chuỗi : Đây là kiểu dữ liệu được sử dụng phổ biến nhất để lưu giữ dữ liệu
* Số nguyên : Kiểu dữ liệu này được sử dụng để lưu một giá trị số
* Boolean : Kiểu dữ liệu này được sử dụng để lưu giữ một giá trị Boolean (true/false).
* Double : Kiểu dữ liệu này được sử dụng để lưu các giá trị số thực dấu chấm động.
* Mảng : Kiểu dữ liệu này được sử dụng để lưu giữ các mảng hoặc danh sách hoặc nhiều giá trị vào trong một key
* Object : Kiểu dữ liệu này được sử dụng cho các Document được nhúng vào
* Null : Kiểu dữ liệu này được sử dụng để lưu một giá trị Null
* Date: Kiểu dữ liệu này được sử dụng để lưu giữ date và time hiện tại trong định dạng UNIX time
* Code : Kiểu dữ liệu này được sử dụng để lưu giữ JavaScrip code vào trong Document
  1. **Một số câu lệnh dùng trong MongoDB**
     1. MongoDB – Tạo Database

Lệnh **use DATABASE\_NAME** trong MongoDB được sử dụng để tạo cơ sở dữ liệu. Lệnh này sẽ tạo một cơ sở dữ liệu mới, nếu nó chưa tồn tài, nếu không thì, lệnh này sẽ trả về cơ sở dữ liệu đang tồn tại.

Cú pháp : use **DATABASE\_NAME**

* + 1. MongoDB – Xóa Database

Lệnh **db.dropDatabase()** trong MongoDB được dùng để xóa một cơ sở dữ liệu đang tồn tại.

Vd: Nếu bạn muốn xóa 1 cơ sở dữ liệu **mydb** có sẵn

>use mydb

switched to db mydb

>db.dropDatabase()

>{ "dropped" : "mydb", "ok" : 1 }

>

* + 1. MongoDB – Tạo Collection

Phương thức **db. createCollection(name, options)** trong MongoDB được sử dụng để tạo Collection. Với name kiểu chuỗi là tên Collection, options là một Document

>db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )

{ "ok" : 1 }

>

* + 1. MogoDB - Xóa Collection

Phương **thức db. Collection.drop()** được sử dụng để xóa một Collection từ cơ sở dữ liệu.

db.COLLECTION\_NAME.drop()

Phương thức drop() sẽ trả về true , nếu Collection đã chọn xóa thành công, nếu không sẽ trả về false.

>db.mycollection.drop()

true

>

* + 1. MongoDB – Chèn Document

Để chèn dữ liệu vào trong Collection trong MongoDB sử dụng phương thức **insert()** hoặc **save()**

>db.COLLECTION\_NAME.insert(document)

**Ví dụ :**

>db.mycol.insert({

\_id: ObjectId(7df78ad8902c),

title: 'MongoDB Overview',

description: 'MongoDB is no sql database',

by: 'tutorials point',

})

* + 1. MongoDB – Truy vấn Document

Để truy vấn dữ liệu từ Collection trong MongoDB ta sử dụng phương thức find()

Cú pháp:

>db.COLLECTION\_NAME.find()

Để hiển thị các kết quả theo một cách đã được định dạng, bạn có thể sử dụng phương thức **pretty()**.

Cú pháp :

>db.mycol.find().pretty()

Ngoài phương thức find(), trong MongoDB còn có phương thức findOne() sẽ chỉ trả về một Document

* + 1. MongoDB - Cập nhập Document

Phương thức **update()** hoặc **save()** trong MongoDB được sử dụng để cập nhật Document vào trong một Collection. Phương thức update() cập nhật các giá trị trong Document đang tồn tại trong khi phương thức save() thay thế Document đang tồn tại với Document đã truyền trong phương thức save() đó.

* Phương thức update() cập nhật các giá trị trong Document đang tồn tại

Cú pháp :

>db.COLLECTION\_NAME.update(SELECTIOIN\_CRITERIA, UPDATED\_DATA)

* Phương thức save() thay thế Document đang tồn tại với Document mới đã được truyền trong phương thức save() này.

Cú pháp :

>db.COLLECTION\_NAME.save({\_id:ObjectId(),NEW\_DATA})

* + 1. MongoDB - Xóa Document

Phương thức remove() trong MongoDB được sử dụng để xóa Document từ Collection. Phương thức remove() nhận hai tham số. Tham số đầu tiên deletion criteria xác định Document để xóa, và tham số thứ hai là justOne.

* **deletion criteria :** (Tùy ý) Xác định Document để xóa.
* **justOne :** (Tùy ý) Nếu được thiết lập là true hoặc 1, thì chỉ xóa một Document.

Cú Pháp :

>db.COLLECTION\_NAME.remove(DELLETION\_CRITTERIA)

* Nếu có nhiều bản ghi và bạn chỉ muốn xóa bản ghi đầu tiên, thì thiết lập tham số **justOne** trong phương thức remove()

>db.COLLECTION\_NAME.remove(DELETION\_CRITERIA,1)

* Xóa tất cả Document

>db.mycol.remove()

>db.mycol.find()

# 

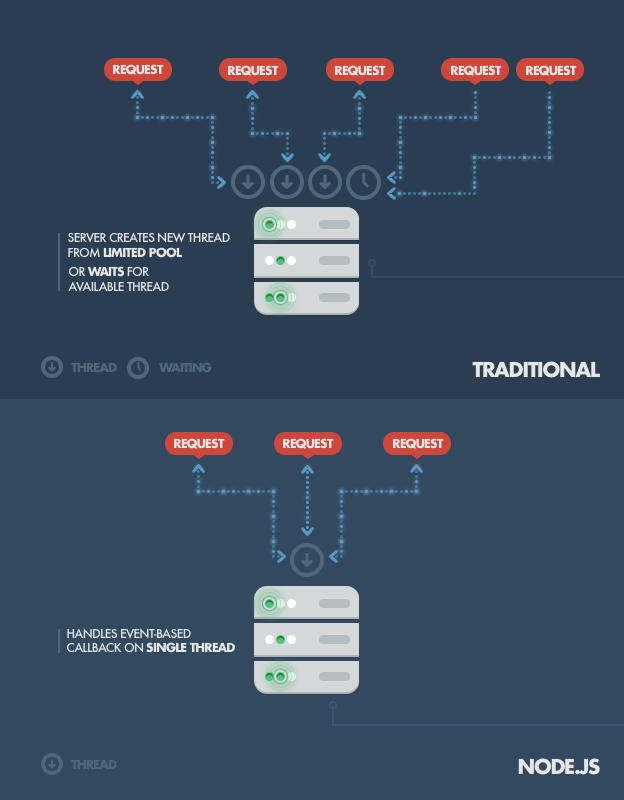
# CHƯƠNG II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

* 1. **Tổng quan về Node.js**
     1. Node.js là gì ?

Node.js là một mã nguồn được xây dựng dựa trên nền tảng Javascript V8 Engine, nó được sử dụng để xây dựng các ứng dụng web như các trang video clip, các forum và đặc biệt là trang mạng xã hội phạm vi hẹp. NodeJS là một mã nguồn mở được sử dụng rộng bởi hàng ngàn lập trình viên trên toàn thế giới. NodeJS có thể chạy trên nhiều nền tảng hệ điều hành khác nhau từ WIndow cho tới Linux, OS X nên đó cũng là một lợi thế. Node.js cung cấp các thư viện phong phú ở dạng Javascript Module khác nhau giúp đơn giản hóa việc lập trình và giảm thời gian ở mức thấp nhất

* + 1. Các đặc tính của Node.js
* **Không đồng bộ**: Tất cả các API của Node.js đều không đồng bộ (none-blocking), nó chủ yếu dựa trên nền của NodeJS Server và chờ đợi Server trả dữ liệu về. Việc di chuyển máy chủ đến các API tiếp theo sau khi gọi và cơ chế thông báo các sự kiện của Node.js giúp máy chủ để có được một phản ứng từ các cuộc gọi API trước (Realtime).
* **Chạy rất nhanh**: Node.js được xây dựng dựa vào nền tảng V8 Javascript Engine nên việc thực thi chương trình rất nhanh.
* **Đơn luồng nhưng khả năng mở rộng cao**: Node.js sử dụng một mô hình luồng duy nhất với sự kiện lặp. cơ chế tổ chức sự kiện giúp các máy chủ để đáp ứng một cách không ngăn chặn và làm cho máy chủ cao khả năng mở rộng như trái ngược với các máy chủ truyền thống mà tạo đề hạn chế để xử lý yêu cầu. Node.js sử dụng một chương trình đơn luồng và các chương trình tương tự có thể cung cấp dịch vụ cho một số lượng lớn hơn nhiều so với yêu cầu máy chủ truyền thống như Apache HTTP Server.
* **Không đệm**: Node.js không đệm bất kì một dữ liệu nào và các ứng dụng này chủ yếu là đầu ra dữ liệu.
* **Có giấy phép**: Node.js đã được cấp giấy phép bởi MIT License..
  + 1. Node.js làm việc như thế nào

Ý tưởng chính của Node. js là sử dụng non-blocking, hướng sự vào ra dữ liệu thông qua các tác vụ thời gian thực một cách nhanh chóng. Bởi vì, Node js có khả năng mở rộng nhanh chóng, khả năng xử lý một số lượng lớn các kết nối đồng thời bằng thông lượng cao. Nếu như các ứng dụng web truyền thống, các request tạo ra một luồng xử lý yêu cầu mới và chiếm RAM của hệ thống thì việc tài nguyên của hệ thống sẽ được sử dụng không hiệu quả. Chính vì lẽ đó giải pháp mà Node js đưa ra là sử dụng luồng đơn (Single-Threaded), kết hợp với non-blocking I/O để thực thi các request, cho phép hỗ trợ hàng chục ngàn kết nối đồng thời.



Hình 2.1 Cơ chế hoạt động của Node.js

* + 1. NPM : The Node Package Manager

Một số các module NPM phổ biến nhất hiện nay là:

* [**expressjs.com/**](http://expressjs.com/) - Express.js, một Sinatra-inspired web framework khá phát triển của Node.js, chứa rất nhiều các ứng dụng chuẩn của Node.js ngày nay.
* [**connect**](https://github.com/senchalabs/connect#readme)- Connect là một mở rộng của HTTP server framework cho Node.js, cung cấp một bộ sưu tập của hiệu suất cao "plugins" được biết đến như là trung gian; phục vụ như một nền tảng cơ sở cho Express
* [**socket.io**](http://socket.io/) and [**sockjs**](https://github.com/sockjs) - Hai thành phần Server-side websockets components nổi tiếng nhất hiện nay.
* [**Jade**](http://jade-lang.com/) - Một trong những engines mẫu, lấy cảm hứng từ HAML, một phần mặc định trong Express.js.
* [**mongo**](https://www.npmjs.com/package/mongodb)and [**mongojs**](https://github.com/mafintosh/mongojs) - MongoDB hàm bao để cung cấp các API cho cơ sở dữ liệu đối tượng trong MongoDB Node.js
* [**redis**](https://github.com/NodeRedis/node_redis) - thư viện Redis client.
* [**coffee-script**](https://www.npmjs.com/package/coffee-script) - CoffeeScript trình biên dịch cho phép developers viết các chương trình Node.js của họ dùng Coffee.
* [**underscore**](https://www.npmjs.com/package/underscore)**(**[**lodash**](https://lodash.com/)**,**[**lazy**](https://www.npmjs.com/package/lazy)**)** - Thư viện tiện ích phổ biến nhất trong JavaScript, package được sử dụng với Node.js, cũng như hai đối tác của mình, hứa hẹn hiệu suất tốt hơn bằng cách lấy một cách tiếp cận thực hiện hơi khác nhau.
* [**forever**](https://www.npmjs.com/package/forever)- Có lẽ là tiện ích phổ biến nhất để đảm bảo rằng một kịch bản nút cho chạy liên tục. Giữ quá trình Node.js của bạn lên trong sản xuất đối mặt với bất kỳ thất bại không ngờ tới.
  1. **Khái niệm về Callback trong Node.js**

Callback có tính chất không đồng bộ tương đương cho một hàm. Một hàm callback được gọi khi hoàn thành một nhiệm vụ cụ thể. Tất cả các API của Node đều được viết theo các cách của hàm callback.

Ví dụ, một hàm để đọc file bắt đầu với việc đọc file và trả về phần điều khiển để môi trường thực hiện quyết định lệnh tiếp theo có thể thực thi. Khi phần I/O (đọc/ghi) file được hoàn thành, nó sẽ gọi về một hàm callback, với nội dung của file là tham số. Do đó sẽ không có blocking hoặc chờ khi đọc/ghi File. Nó làm cho Node.js có hiệu năng cao hơn, như có số lượng request cao hơn mà không cần phải chờ kết quả trả về.

* Ví dụ Blocking Code

Tạo một dòng text với input.txt với nội dung

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

Tạo một file main.js với nội dụng

var fs = require("fs");

var data = fs.readFileSync('input.txt');

console.log(data.toString());

console.log("Ket thuc chuong trinh");

Kết quả chạy chương trình :

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

Ket thuc chuong trinh

* Ví dụ Non-Blocking Code

Tạo 1 file với tên input.txt với nội dung

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

* Hàm main.js

ar fs = require("fs");

fs.readFile('input.txt', function (err, data) {

if (err) return console.error(err);

console.log(data.toString());

});

console.log("Ket thuc chuong trinh");

Kết quả khi chạy chương trình :

Ket thuc chuong trinh

Chương II : TÌM HIỂU VỀ LẬP TRÌNH NODE.JS

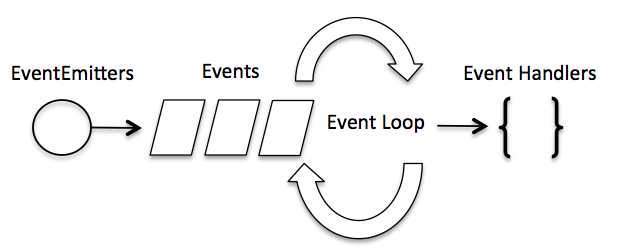
Hai ví dụ trên giải thích định nghĩa cách gọi blocking và non-blocking. Ví dụ đầu tiên chỉ ra rằng chương trình khóa cho đến khi nó đọc file và chỉ tiếp tục chạy vài giây sau đó, chương trình thứ 2 không đợi cho việc đọc file và tiếp tục in "Ket thuc chuong trinh" cùng thời điểm thực hiện chương trình.

* 1. **Khái niệm Event Loop**

Node.js là ứng dụng đơn luồng nhưng có hỗ trợ việc xử lí đồng thời thông qua các định nghĩa về sự kiện và callback. Như tất cả các API của Node.js có tính chất không đồng bộ và được xử lí đơn luồng, nó sử dụng hàm **async** để duy trì sự đồng thời. Node.js sử dụng Observer Pattern. Các Thread trong Node.js giữ một Event Loop và bất cứ khi nào có tác vụ nào hoàn thành, nó sẽ kích hoạt sự kiện tương ứng để báo cho Event Listener sẵn sàng thực hiện..

Node.js sử dụng rất nhiều sự kiện, đó là lý do tại sao Node.js khá nhanh so với các sản phẩm công nghệ khác. Ngay khi Node khởi động Server của nó, nó sẽ nhanh chóng khởi tạo các biến, khai báo các hàm và sau đó đơn giản là chờ đợi các sự kiện xảy ra.

Trong ứng dụng xử lý sự kiện, nhìn chung vòng lặp chính lắng nghe các sự kiện, và sau đó trigger đến hàm callback khi một trong những sự kiện được phát hiện.



Hình 2.2 Event Loop

Trong khi các Event là khá tương tự như các hàm callback . Điểm khác nhau nằm ở chỗ, hàm callback gọi khi một hàm không đồng bộ và trả về kết quả của nó trong khi phần xử lí sự kiện làm việc trên Observer Pattern. Hàm này sẽ lắng nghe các sự kiện, đóng vai trò như một Observers (Người quan sát). Bất cứ khi nào một sự kiện phát sinh, các hàm Listener của nó sẽ bắt đầu thực thi. Node.js có nhiều sự kiện có sẵn thông qua **events** Module và lớp **EventEmitter**có thể dựa vào để bind sự kiện và lắng nghe sự kiện.

Trước khi sử dụng event Module, bạn sử dụng phương thức require() để khai báo như sau:

var events = require('events');

var eventEmitter = new events.EventEmitter();

Sau đó, để gắn kết Event Handler với một sự kiện, bạn sử dụng cú pháp sau:

eventEmitter.on('eventName', eventHandler);

Bạn có thể kích hoạt một sự kiện bởi sử dụng phương thức emit() của EventEmitter:

eventEmitter.emit('eventName');

Ví dụ minh họa Event Loop trong Node.js

var events = require('events');

var eventEmitter = new events.EventEmitter();

var connectHandler = function connected() {

console.log(' ket noi thanh cong!');

eventEmitter.emit('data\_received');

}

eventEmitter.on('connection', connectHandler);

eventEmitter.on('data\_received', function(){

console.log('Du lieu duoc tiep nhan thanh cong.');

});

eventEmitter.emit('connection');

console.log("Ket thuc chuong trinh.");

Kết quả của chạy chương trình :

ket noi thanh cong!

Du lieu duoc tiep nhan thanh cong.

Ket thuc chuong trinh.

* 1. **Lớp Event Emitter**

Nhiều đối tượng trong Node.js sinh ra các sự kiện, ví dụ **net.Server** sinh ra một sự kiện mỗi khi có một kết nối ngang hàng đến nó, hay **fs.readStream** sinh ra sự kiện khi một file được mở. Tất cả các đối tượng này đều là sự thể hiện của lớp **events.EventEmitter** trong Node.js.

* Lớp EventEmitter trong Node.js

Lớp EventEmitter nằm trong events Module. Lớp này được truy cập qua cú pháp sau:

var events = require('events');

var eventEmitter = new events.EventEmitter();

Khi một EventEmitter gặp bất kì lỗi nào, nó sẽ sinh ra một Error Event. Khi một Listener mới được thêm, sự kiện 'newListener' sẽ được kích hoạt và một Listener sẽ bị loại bỏ, sự kiện 'removeListener' sẽ được kích hoạt.

Event Emitter cung cấp nhiều thuộc tính như **on** hay **emit**. Thuộc tính *on* được sử dụng để gắn kết một hàm với sự kiện, và *emit* dược sử dụng để kích hoạt một sự kiện.

Các phương thức của lớp EventEmitter trong Node.js

|  |  |
| --- | --- |
| Stt | Phương thức & Miêu tả |
| 1 | **addListener(event, listener)** Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể |
| 2 | **on(event, listener)** Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể |
| 3 | **once(event, listener)** Thêm một One-Time Listener cho sự kiện. Listener dạng này sẽ chỉ được gọi khi sự kiện được kích hoạt, sau đó nó sẽ bị xóa |
| 4 | **removeListener(event, listener)** Xóa một Listener ra khỏi mảng các Listener cho một sự kiện nào đó. |
| 5 | **removeAllListeners([event])** Xóa tất cả Listener của một sự kiện |
| 6 | **setMaxListeners(n)** Theo mặc định, lớp EventEmitters sẽ in một lời cảnh báo nếu bạn thêm nhiều hơn 10 Listener cho một sự kiện cụ thể. Việc này khá hữu ích, bởi vì nó sẽ giúp tìm ra các lỗi gây rò rỉ bộ nhớ. Tất nhiên, không phải tất cả các Emitters đều cần được giới hạn với con số là 10. Hàm này cho phép bạn tăng con số đó. Thiết lập nó về 0 để không giới hạn lượng Listener cần thêm |
| 7 | **listeners(event)** Trả về một mảng bao gồm các Listener cho một sự kiện cụ thể nào đó |
| 8 | **emit(event, [arg1], [arg2], [...])** Thực thi từng Listener với các tham số đã cho. Trả về true nếu sự kiện có các Listener, và false nếu không có |

**Các sự kiện của lớp EventEmitter trong Node.js**

|  |  |
| --- | --- |
| Stt | Sự kiện & Miêu tả |
| 1 | **newListener**   * **event** - Dạng chuỗi, biểu diễn tên sự kiện * **listener** - Tên hàm xử lý sự kiện   Sự kiện này được sinh bất cứ khi nào bạn thêm một Listener. Khi sự kiện này được kích hoạt, Listener có thể sẽ chưa được thêm vào mảng Listener của sự kiện |
| 2 | **removeListener** **event** - Dạng chuỗi, biểu diễn tên sự kiện   * **listener** - Tên hàm xử lý sự kiện   Sự kiện này xảy ra bất cứ khi nào có ai đó xóa một Listener. Khi một sự kiện được kích hoạt, Listener này chưa được xóa khỏi mảng Listener của sự kiện |

* 1. **Buffer trong Node.js**

Node.js cung cấp các lớp Buffer cho phép lưu trữ các dữ liệu thô như một mảng các số nguyên tương ứng với phần cấp phát bộ nhớ thô bên ngoài V8 heap.

Các lớp Buffer trong Node.js là các lớp toàn cục và có thể được truy cập trong ứng dụng mà không cần khai báo các Buffer Module bởi phương thức require() như các Module khác.

Buffer trong Node.js có thể được xây dựng theo nhiều cách khác nhau.

Cách 1 : Cú pháp cho việc tạo một Buffer cỡ 10:

var buf = new Buffer(10);

cách 2 : Cú pháp tạo một Buffer từ một mảng cho trước

var buf = new Buffer([10, 20, 30, 40, 50]);

Cách 3 : Cú pháp tạo một Buffer từ một chuỗi cho trước và với kiểu mã hóa tùy ý:

var buf = new Buffer("nguyen ba lam "utf-8");

* Ghi dữ liệu vào Buffer trong node.js

Cú pháp để ghi một Buffer trong node.js

buf.write(string[, offset][, length][, encoding])

Phương thức sẽ trả về số lượng các byte được ghi. Nếu bộ nhớ trong buffer là không đủ để đáp ứng cho toàn bộ chuỗi, nó sẽ ghi một phần của chuỗi đó.

* Đọc dữ liệu từ Buffer trong Node.js

Cú pháp : đọc dữ liệu từ Buffer

buf.toString([encoding][, start][, end])

Phương thức này giải mã và trả về một chuỗi từ dữ liệu đã được mã hóa trong Buffer bởi sử dụng bộ mã hóa cụ thể.

* Chuyển đổi Buffer thành JSON trong Node.js

Cú pháp :

buf.toJSON()

Phương thức này trả về một biểu diễn JSON cho đối tượng Buffer đã cho.

* Ghép nối các Buffer trong node.js

Cú pháp : Để nối ghép hai hoặc nhiều Buffer thành một Buffer trong Node.js, bạn sử dụng phương thức concat() như sau:

Buffer.concat(list[, totalLength])

Phương thức này trả về một Buffer mới.

* So sánh các Buffer trong Node.js

Cú pháp : Để so sánh hai Buffer trong Node.js, bạn sử dụng phương thức compare() như sau:

buf.compare(otherBuffer);

Trả về một giá trị dạng số thể hiện Buffer này là đứng trước, sau hay cùng thứ tự với Buffer kia

* Sao chép Buffer trong Node.js

Cú pháp : Để sao chép Buffer trong Node.js, bạn sử dụng phương thức copy() như sau:

buf.copy(targetBuffer[, targetStart][, sourceStart][, sourceEnd])

* chia nhỏ Buffer trong node.js

Cú pháp :

buf.slice([start][, end])

* 1. **Stream trong Node.js**

Stream là các đối tượng cho phép bạn đọc dữ liệu từ một nguồn và ghi dự liệu đến một đích. Trong Node.js, có 4 loại Stream.

**Readable** - Là Stream được sử dụng để cho hoạt động đọc

**Writable** - Là Stream được sử dụng cho hoạt động ghi

**Duplex** - Là Stream được sử dụng cho cả mục đích ghi và đọc

**Transform** - Đây là một kiểu Duplex Stream, khác ở chỗ là kết quả đầu ra được tính toán dựa trên dữ liệu bạn đã nhập vào.

Mỗi loại Stream là một sự thể hiện của đối tượng **EventEmitter** và ném một vài sự kiện tại các thời điểm khác nhau. Danh sách sau liệt kê một số sự kiện thường được sử dụng:

**data** - Sự kiện này được kích hoạt khi dữ liệu là có sẵn cho hoạt động đọc.

**end** - Sự kiện này được kích hoạt khi không còn dữ liệu nào để đọc nữa.

**error** - Sự kiện này được kích hoạt khi xảy ra bất kỳ lỗi nào trong việc đọc và ghi dữ liệu.

**finish** - Sự kiện này được kích hoạt khi tất cả dữ liệu đã được chuyển hết tới vùng hệ thống cơ sở.

* Đọc dữ liệu từ Stream trong Node.js

Tạo 1 file input.txt có nội dung

Báo cáo thực tập tốt nghiệp

Tạo 1 file main.js  Trong file này, đầu tiên bạn khai báo **fs** Module (đây là Module cho các hoạt động File I/O) bởi sử dụng phương thức require(). Sau đó sử dụng phương thức createReadStream() nhận tham số là tên của text file bạn đã tạo trước đó để đọc dữ liệu từ đó

var fs = require("fs");

var data = '';

var readerStream = fs.createReadStream('input.txt');

readerStream.setEncoding('UTF8');

readerStream.on('data', function(chunk) {

data += chunk;

});

readerStream.on('end',function(){

console.log(data);

});

readerStream.on('error', function(err){

console.log(err.stack);

});

console.log("Ket thuc chuong trinh");

Chạy chương trình kết quả thu được :

Ket thuc chuong trinh

Báo cáo thực tập tốt nghiệp

Ghi dữ liệu tới Stream trong Node.js

var fs = require("fs");

var data = 'timhieuvelaptrinhnodejs';

var writerStream = fs.createWriteStream('output.txt');

writerStream.write(data,'UTF8');

writerStream.end();

writerStream.on('finish', function() {

console.log("Ket thuc hoat dong ghi.");

});

writerStream.on('error', function(err){

console.log(err.stack);

});

console.log("Ket thuc chuong trinh");

Chạy chương trình kết quả thu được **:**

Ket thuc chuong trinh

Ket thuc hoat dong ghi.

Bây giờ, bạn mở output.txt đã được tạo trong thư mục hiện tại và kiểm tra nội dung kết quả thu được:

timhieuvelaptrinhnodejs

* 1. **Đọc ghi file trong Node.js**

Để khai báo **fs** Module để triển khai các hoạt động về File I/O trong Node.js. Cú pháp như sau:

var fs = require("fs")

* Mở một file trong node.js

Cú pháp :

fs.open(path, flags[, mode], callback)

* Lấy thông tin file trong Node.js

Cú pháp :

fs.stat(path, callback)

* Ghi dữ liệu file trong Node.js

fs.writeFile(filename, data[, options], callback)

Phương thức này sẽ ghi đè nếu file đã tồn tại.

* Đọc dữ liệu từ file trong Node.js

Cú Pháp:

fs.read(fd, buffer, offset, length, position, callback)

Phương thức này sẽ sử dụng tham số fd (viết tắt của File Descriptor) để đọc file. Nếu bạn muốn đọc file bởi sử dụng trực tiếp tên file thì bạn nên sử dụng phương thức khác.

* Đóng File trong Node.js

Cú pháp :

fs.close(fd, callback)

* Xóa file trong Node.js

Cú pháp:

fs.unlink(path, callback)

* Tạo thư mục trong Node.js

Cú pháp :

fs.mkdir(path[, mode], callback)

* Đọc thư mục trong Node.js

Cú pháp :

fs.readdir(path, callback)

* Xóa thư mục trong Node.js

Cú pháp:

fs.rmdir(path, callback)

* 1. **Đối tượng toàn cục**

Khái niệm Toàn cục (Global) tức là mọi thứ đều có quyền truy cập đến. Trong Node.js cũng vậy, các đối tượng toàn cục là có sẵn cho tất cả Module. Chúng ta không cần khai báo hay import chúng bởi phương thức require() mà vẫn có thể sử dụng chúng một cách trực tiếp. Các đối tượng này có thể là các Module, các hàm, các chuỗi hoặc các đối tượng.

Trong Node.js, **\_\_filename** biểu diễn tên file của đoạn code đang được thực thi. Đây là đường dẫn tuyệt đối tới file chứa phần code này

Ví dụ : Để minh họa cho \_filenam trong Node.js, bạn tạo main.js có nội dung đơn giản sau:

console.log( \_\_filename );

Kết quả : in ra chính là đường dẫn tới main.js chứa đoạn code trên:

Trong Node.js, **\_\_dirname** biểu diễn thư mục chứa đoạn code mà đang được thực thi.

Ví dụ : Để minh họa cho việc lấy thông tin của \_dirname trong Node.js, bạn tạo main.js chứa đoạn code đơn giản sau:

console.log( \_\_dirname );

Kết quả : in ra là thư mục hiện tại chứa main.js:

* Hàm setTimeout(cb, ms)

Hàm toàn cục **setTimeout(cb, ms)** được sử dụng để chạy hàm callback có tên là **cb** sau một khoảng thời gian **ms** milisecond.

Hàm toàn cục này trả về một giá trị biểu diễn Timer để có thể được xóa bởi hàm clearTimeout().

* Hàm clearTimeout(t)

Hàm toàn cục clearTimeout( t ) được sử dụng để dừng một Timer đã được tạo bởi hàm setTimeout() trước đó. Tham số t chính là Timer được trả về từ hàm setTimeout().

* Hàm setInterval(cb, ms)

Hàm toàn cục **setInterval(cb, ms)** được sử dụng để chạy hàm callback có tên là **cb** một cách lặp đi lặp lại sau một khoảng thời gian **ms** millisecond.

Hàm toàn cục này trả về một giá trị biểu diễn Timer để có thể được xóa bởi hàm **clearInterval(t)**.

**console** trong Node.js là đối tượng toàn cục được sử dụng để in ra các mức độ thông báo khác nhau trên stdout và stderr. Đối tượng toàn cục này có sẵn nhiều phương thức được sử dụng để in thông tin mang tính cảnh báo, lỗi.

Bảng dưới trình bày các phương thức của đối tượng toàn cục console trong Node.js.

|  |  |
| --- | --- |
| Stt | Phương thức & Miêu tả |
| 1 | **console.log([data][,...])** In một dòng mới tới stdout. |
| 2 | **console.info([data][,...])** In một dòng mới tới stdout. |
| 3 | **console.error([data][,...])** In một dòng mới tới stderr |
| 4 | **console.warn([data][,...])** In một dòng mới tới stderr |
| 5 | **console.dir(obj[,options])** Sử dụng util.inspect trên obj và in chuỗi kết quả tới stdout. |
| 6 | **console.time(label)** Đánh dấu một mốc thời gian |
| 7 | **console.timeEnd(label)** Kết thúc Timer, ghi lại kết quả |
| 8 | **console.trace(message[,...])** In tới stderr dòng thông báo 'Trace :', theo sau bởi thông báo đã được định dạng và stack trace tới vị trí hiện tại |
| 9 | **console.assert(value[,message][,...])** Tương tự assert.ok(), nhưng thông báo lỗi ở đây được định dạng giống như phương thức util.format(message...). |

Bảng 3. Phương thức của đối tượng toàn cục của console trong node.js

## 

**process** trong Node.js là đối tượng toàn cục và có thể được truy cập từ bất cứ nơi đâu. Đối tượng toàn cục process này cung cấp rất nhiều phương thức tiện lợi được trình bày dưới đây.

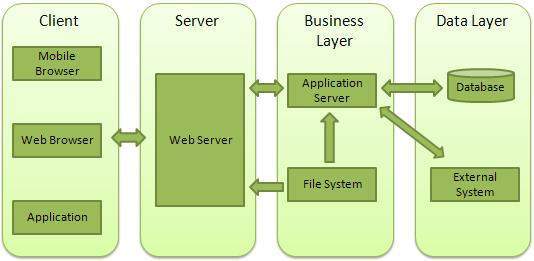
|  |  |
| --- | --- |
| Stt | Sự kiện & Miêu tả |
| 1 | **exit** Sự kiện xảy ra khi process chuẩn bị thoát. Không có cách nào để ngăn cản nó |
| 2 | **beforeExit** Sự kiện này xảy ra khi Node.js làm trống Event Loop và không còn cái gì khác để schedule nữa. |
| 3 | **uncaughtException** Sự kiện xảy ra khi xuất hiện một exception và không được bắt. Nếu một Listener được thêm vào exception này thì hành động mặc định sẽ không xảy ra |
| 4 | **Signal Events** Sự kiện xảy ra khi process nhận một tín hiệu giống như SIGINT, SIGHUP, ... |

Bảng 4. Đối tượng đối tượng toàn cục process trong node.js

* 1. **Module trong Node.js**

Web Server là một ứng dụng phần mềm có thể xử lý các HTTP request được gửi bởi HTTP Client (ví dụ: trình duyệt web) và trả về một trang web trong phản hồi tới Client. Web Server thường gửi các tài liệu html bên cạnh các ảnh cũng như style sheet và các đoạn Javascript.

Một ứng dụng web thường được chia thành 4 lớp như sau:



Hình 1.8 Cấu Trúc ứng dụng web

* **Client** - Lớp này bao gồm các trình duyệt web và các ứng dụng có thể tạo ra các HTTP request đến Web Server.
* **Server** - Lớp này bao gồm các Web Server có thể can thiệp các request được tạo bởi Client và trả về các phản hồi (response).
* **Business** - Lớp này bao gồm ứng dụng trên Server có thể được tận dụng bởi các Web Server để thực hiện các tiến trình xử lý cần thiết. Lớp này tương tác với lớp Data qua các chương trình bên ngoài.
* **Data** - Lớp này bao gồm các Database và bất kì các nguồn dữ liệu nào.

Node.js cung cấp **http** Module có thể được sử dụng để tạo các HTTP client và server. Dưới đây là phần kiến trúc thu nhỏ của HTTP Server được lắng nghe trên cổng 8081.

Đầu tiên, bạn tạo *server.js* có nội dung như sau:

var http = require('http');

var fs = require('fs');

var url = require('url');

http.createServer( function (request, response) {

var pathname = url.parse(request.url).pathname;

console.log("Request cho " + pathname + " da duoc nhan.");

fs.readFile(pathname.substr(1), function (err, data) {

if (err) {

console.log(err);

response.writeHead(404, {'Content-Type': 'text/html'});

}else{

response.writeHead(200, {'Content-Type': 'text/html'});

response.write(data.toString());

}

response.end();

});

}).listen(8081);

console.log('Server dang chay tai dia chi: http://127.0.0.1:8081/');

Kết quả khi chạy chương trình.:

Server dang chay tai dia chi: http://127.0.0.1:8081/

# 

# CHƯƠNG III : TÌM HIỂU CÁCH TẤN CÔNG VÀ NGĂN CHẶN INJECTION VÀO NOSQL QUA ỨNG DỤNG WEB NODE.JS

* 1. **Tổng quan về Injection NoSQL**

Chăc hẳn chúng ta khá quên thuộc với khái niệm SQL Injection đây là lỗ hổng phổ biến cũng như nguy hiểm với các trang web trên Internet. Mới đây với sự phát triển của những loại database mới như Graph Database và NoSql đã làm phong phú hơn lựa chọn database của những dự án phần mền. Với những ưa điểm của mình:

* + Mã nguồn mở
  + Khả năng mở rộng linh hoạt
  + Phù hợp với điện toán đám mây

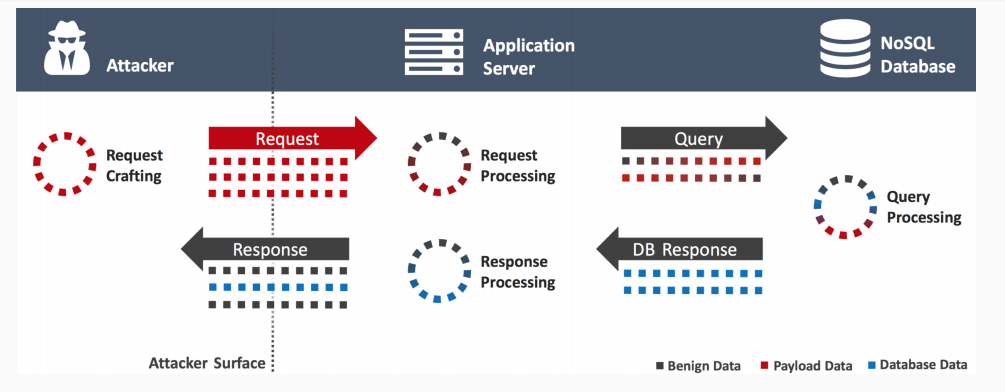
Các cơ sở dữ liệu như MongoDB hay redis là những lựa chọn đáng cân nhắc với những hệ thống lớn, một khía cạnh có lẽ không được cân nhắc đến nhất là ở việt nam đó chính là tính bảo mật của cơ sở dữ liệu này.

Trong bài viết này mình sẽ tìm hiểu về một số lỗ hổng thường gặp trong hệ quản trị cơ sở dữ liệu NoSQL, mà cụ thể là MongoDb hệ quản trị cơ sở dữ liệu Nosql phổ biến nhất hiện nay.

* + 1. Cơ chế hoạt động

Theo như trang Mongo DB Developer FAQ có nói :"... with MongoDB we are not building queries from strings, so traditional SQL injection attacks are not a problem “

Tức là MongoDB có thể tránh được tất cả các loại SQL injection truyền thống, . Nhưng thay vào đó các hacker lại sự dụng một kĩ thuật đặc thù cho NoSQL đó là NoSQL



Hình 3.1 Cơ chế hoạt động

**C**ũng như SQL Injection để có thể thực hiện Nosql Injection hacker cũng phải truy vấn lên server ,dựa vào request từ client server sẽ truy vấn đến Database và nhận lại kết quả từ database để thực hiện trả cho client. Nhiệm vụ của hacker là phải làm sao để hiểu sai request dẫn đến một thực hiện một câu truy vấn không mong muốn đến database cuối cùng hacker sẽ lấy được thông tin mong muốn hoặc nghiêm trọng hơn là xóa dữ liệu trong cơ sở dữ liệu.

Điểm khác nhau giữa Nosql Injection và SQL Injection truyền thống

* câu truy vấn cho khối dữ liệu không cấu trúc.
* các hệ quản trị cơ sở dữ liệu rất đa dạng và khác biệ nhau khá nhiều trong cách tổ chức dữ liệu cũng như truy vấn.
* cho phép truy cập trực tiếp client-database thông qua RESTfull API.
  + 1. Các Loại Tấn Công

1. *Chuyển định dạng string thành biểu thức điều kiện*

Bắt đầu với truy vấn trên MongoDB dùng để truy vấn chi tiết đăng nhập người dùng :

db.usersdata.find({ username: "nguyenbalam", password: "123456"})

được xem như một truy vấn bình thường và tìm thấy kết quả.

Tuy nhiên, hacker có thể thay thế bằng một Biểu thức logic thay vì String, thường dùng nhất là "&ne" với ý nghĩa not equals(tìm ra những gì không bằng "") hoặc "&gt" ( lớn hơn).

Vd: Hacker gửi lên server một request có dạng

[https://example.org/login?user=[%24]&password[%24ne](https://example.org/login?user=%5b%24%5d&password%5b%24ne)]=

Khi đó câu query của chúng ta sẽ có dạng :

db.usersdata.find({ username: {"$ne": ""}, password: {"$ne": ""}})

Truy vấn này luôn trả về tất cả các row trong collection mà có username và password lớn hơn rỗng. Điều này thật sự rất nguy hiểm!

Để ngăn chặn điều này, chúng ta cần:

* Đọc dữ liệu vào theo định dạng String. Framework Meteor mặc định tất cả các input text nhập vào đều ở định dạng String.
* Một lần nữa kiểm tra tất cả dữ liệu nhập vào từ người dùng. Framework Meteor cung cấp một module rất mạnh đó là check. Nó có thể kiểm tra xem dữ liệu bạn đưa vào có phải dự liệu bạn mong muốn hay không.

VD: check(username, String)

Function này sẽ xuất hiện Exception nếu như username không phải là String và không cho chạy đoạn code phía sau. Nếu username là String thì tiếp tục chạy như bình thường.

1. *Nối chuỗi trong query*

Nếu như dev viết một câu truy vấn dạng này:

var string query = “{ username: ‘“ + user + “’, password: ‘” + password + “’ }”

Đây rõ ràng có thể thực hiện tương tự như SQL Injection mà chúng ta đã được học. Nếu:

username = admin’, $or: [ {}, { ‘a’:’a

password = ’ } ], $comment:’hacked’

thì câu query sẽ trở thành

{ username: ‘admin’ , $or: [ {}, { ‘a’: ‘a’, password: ‘’ } ], $comment: ‘hacked’ }

Và hacker đã bypass thành công bước Authentication.

Tương tự như phần 1, chúng ta cần quy định cứng dữ liệu nhập vào & một lần nữa kiểm tra chúng. Ngoài ra, hãy sử dụng thư viện Publish/Subcribe mà Meteor cung cấp. Publish/Subcribe giúp bạn quản lý dữ liệu được truyền từ server & nhận tại client. Chỉ truyền những thứ cần thiết đến client, không truyền tất cả. Điều này giúp giảm thiểu được nguy cơ bị khai thác dữ liệu sau khi cuộc tấn công NoSQL Injection.

# CHƯƠNG IV : XÂY DỰNG ỨNG DỤNG WEB ĐƠN GIẢN BẰNG NODE.JS NGĂN CHẶN TẤN CÔNG INJECTION VÀO NOSQL

* 1. **Tổng quan về ứng dụng Facebook Massenger**



Hình 4.1 Cấu trúc ứng dụng Facebook Messenger

* + 1. Các công nghệ sử dụng trong việc xây dựng ứng dụng
* Môi trường : Node.js
* Cơ sở dữ liệu : MongoDB
* Framework: Meteor
  + 1. Một số tính năng của ứng dụng Facebook Massenger
* Đăng nhập/ Đăng kí
* Chat với nhau trong một group chat
* Kết nối và lưa trữ database bằng MongoDB
* Quản lý user và group hiện tại bằng session
* Phân biệt các dòng chat
* Ngăn chặn Injection Vào NoSQL
  1. **Tiến hành cài đặt và chạy chương trình trên Ubuntu**

Mở Terminal

**B1** : Cài đặt Node.js

* Sudo apt-get update
* Sudo apt-get install nodejs nmp

**B2**. Cài đặt Meteor

* curl https://install.meteor.com/ | sh

**B3:** Cài đặt cái module

* nmp install meteor

**B4**: Reset lại Project

* meteor reset

**B5** : Chạy project ( mở 1 Terminal mới lên cd đến thư mục chứa project)

* meteor

**B6** : Sau khi chạy project ta truy cập vào mogodb

* meteor mongo
* Thực hiện những thao tác thêm , xóa , sửa trên MongoDB.
  1. **Cài đặt một số tính năng tiêu biểu**
     1. Đăng nhập và Đăng kí

Tiến hành cài đặt trong file App.js

class App extends Component {

username\_handleKeyPress(event) {

if (event.which === 13 && event.currentTarget.value.trim() != "") {

$('.password').focus();

}

}

password\_handleKeyPress(event) {

if (event.which === 13 && event.currentTarget.value.trim() != "") {

if ($('.username').val().trim() === ""){

$('.username').focus();

}

else{

var username =ReactDOM.findDOMNode(this.refs.userName).value.trim();

var password = ReactDOM.findDOMNode(this.refs.passWord).value.trim();

check(username, String);

check(password, String);

Meteor.call('my-login', {

username: username,

password: password

}, (err, res) => {

if (err) {

alert("Something went wrong!");

}

else {

if (res === true){

Session.set('currentUser', username); }

else if (res === false){

Meteor.call('my-check-username', {

username: username

}, (err, res) => {

if (err) {

alert("Something went wrong!");

}

else{

if (res === true){

alert("Password is wrong!");

}

else if (res === false){

var choose = confirm("Do you want to register?");

if (choose){

Meteor.call('register', {

username: username,

password: password

}, (err, res) => {

if (err) {

alert("Something went wrong!");

}

else{

alert("Done! Please login again!");

}

});

}

}

}

});

}

}

});

}

}

}

* + 1. Lưa trữ và kết nối MongoDB

Tiến hành cài đặt trong file connect-db.js

import { Mongo } from 'meteor/mongo';

export const groupsCollection = new Mongo.Collection('groupsdata');

export const messagesCollection = new Mongo.Collection('messagesdata');

if (Meteor.isServer) {

Meteor.publish('groupsdata', function tasksPublication() {

return groupsCollection.find();

});

Meteor.publish('messagesdata', function tasksPublication(currentRoom) {

return messagesCollection.find({ room: currentRoom}, { sort: { time: 1 }});

});

}

* + 1. Quản lý user và Group chat bằng session

Tiến hành cài đặt trong file App.js

Khi người dùng đăng nhập thành công thì chúng ta tạo Session currentUser bằng cú pháp:

Session.set('currentUser', username);

Khi người dùng đăng xuất chúng ta xóa Session currentUser bằng cú pháp:

Session.set('currentUser', undefined);

* + 1. Ngăn chặn tấn công Injection vào NoSQL

Tiến hành cài đặt trong file App.js

Khi người dùng nhập dữ liệu vào trường Username Và Password chúng ta tiến hành kiểm tra dữ liệu nhập vào có đúng chuẩn hay không bằng hàm check theo cú pháp.

check(username, String);

check(password, String);

# 

# CHƯƠNG V : KẾT LUẬN

* 1. **Kết quả đạt được**

Đã xây dựng được một ứng dụng chat real-time an toàn phòng chống được Injection vào cơ sở dữ liệu(MongoDB) cho phép các thành viên trong nhóm chat với nhau mà không phải lo sợ về tính bảo mật.

Trong quá trình thực tập , bản thân em đã học hỏi và tìm hiểu thêm được một ngôn ngữ lập trình mới và những cơ sở dữ liệu mới.

* 1. **Hạn chế**

Tuy nhiên, do thời gian ngắn và khả năng lập trình của em không tốt lắm cộng với số lượng công việc cũng như tìm hiểu các công nghệ mới cho nên đề tài của em chưa được hoàn thiện về mặt thực hành. Các chức năng chưa được đầy đủ như dự kiến. Mong nhận được nhiều ý kiến từ thầy cô và giúp em hoàn thiện sản phẩm hơn.

# DANH MỤC TÀI LIỆU THAM KHẢO

**Danh Mục**

1. <https://www.owasp.org/images/e/ed/GOD16-NOSQL.pdf>
2. <https://scotch.io/@401/mongodb-injection-in-nodejs>
3. <https://zanon.io/posts/nosql-injection-in-mongodb>
4. <https://technopy.com/mongodb-injection-how-to-hack-mongodb-html/>
5. <https://viblo.asia/p/tim-hieu-ve-nosql-Zzb7vDNYMjKd>
6. <https://kipalog.com/posts/NOSQL-co-dong-nghia-voi-NO-Injection>
7. <https://viblo.asia/p/tim-hieu-ve-nosql-Zzb7vDNYMjKd>
8. <https://vietjack.com/nodejs/index.jsp>
9. <https://blog.sqreen.io/mongodb-will-not-prevent-nosql-injections-in-your-node-js-app/?utm_content=buffer25729&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer>