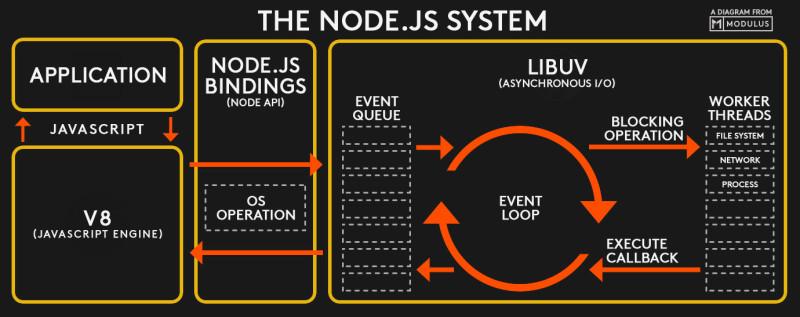
1. Nodejs Basic
2. Nodejs overview
3. Nodejs là gì?

* NodeJS là một nền tảng được xây dựng trên “V8 Javascript engine” – trình thông dịch thực thi mã JavaScript được viết bằng c++ và Javascript.( ѕử dụng mô hình event driven, môi trường đa nền tảng)
* Node JS hoàn toàn không phải một ngôn ngữ lập trình mà chỉ đơn giản là môi trường runtime, cho phép chạy Javascript bên ngoài trình duyệt.
* Các request tạo ra một luồng xử lý yêu cầu mới và chiếm RAM của hệ thống. 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.
* 

1. Js engine

* Một động cơ là một chương trình
* Nó dịch cú pháp javascript sang mã máy và thực thi nó để nhận được kết quả.
* Các JavaScript Engine chủ yếu được phát triển và cài đặt cho các trình duyệt web (web browser)

1. Synchronous và asynchronous

* Synchronous có nghĩa là xử lý đồng bộ, chương trình sẽ chạy theo từng bước và chỉ khi nào bước 1 thực hiện xong thì mới nhảy sang bước 2, khi nào chương trình này chạy xong mới nhảy qua chương trình khác.
  + Ưu: Chương trình sẽ chạy theo đúng thứ tự và có nguyên tắc nên sẽ không mắc phải các lỗi về tiến trình không cần thiết.
  + Nhược: Chương trình chạy theo thứ tự đồng bộ nên sẽ sinh ra trạng thái chờ và là không cần thiết trong một số trường hợp
* Asynchronous là xử lý bất động bộ, nghĩa là chương trình có thể nhảy đi bỏ qua một bước nào đó. Các hành động có thể хảу ra đồng thời hoặc chí ít, mặc dù các hành động bắt đầu theo thứ tự nhưng kết thúc thì không. Một hành động có thể bắt đầu (ᴠà thậm chí kết thúc) trước khi hành động trước đó hoàn thành.
  + Ưu: Có thể xử lý nhiều công việc một lúc mà không cần phải chờ đợi
  + Nhược: Nếu một chương trình đòi hỏi phải có quy trình thì bạn không thể sử dụng

1. Non blocking I/O

* Blocking : Khi có 1 request thì sẽ đóng kết nối server và clients đến khi request đó hoàn thành
* Non Blocking I/: Các request sẽ không phụ thuộc vào nhau. Các thao tác này không nhất thiết phải đồng bộ và lần lượt mà có thể được xử lý không đồng bộ và có thể chạy phía dưới nền để Nodejs có thể tiếp tục xử lý các lệnh tiếp theo
* Điểm khác nhau khi đọc ghi dữ liệu: Thay vì ghi dữ liệu vào out stream và đọc dữ liệu từ input stream chúng ta sẽ đọc và ghi dữ liệu vào bộ nhớ đệm. Có thể hiểu bộ nhớ đệm ở đây là một bộ nhớ tạm thời thay vì tương tác trực tiếp.

1. Event loop

* Call stack:
  + Stack là một vùng nhớ đặc biệt phục vụ cho quá trình thực thi các dòng lệnh mà cụ thể là các hàm.
  + Mỗi khi một hàm được triệu gọi thì nó sẽ được đẩy vào một hàng đợi đặc biệt có tên là stack. Stack là một hàng đợi kiểu LIFO (Last In First Out).
* Heap: là vùng nhớ được dùng để chưa kết quả tạm phục vụ cho việc thực thi các hàm trong stack. Heap càng lớn thì khả năng tính toán càng cao.
* Event loop: là thứ cho phép node.js thực hiện các tác vụ I/O không đồng bộ
  + Nhiệm vụ của Event Loop rất đơn giản đó là đọc Stack và Event Queue. Nếu nhận thấy Stack rỗng nó sẽ nhặt Event đầu tiên trong Event Queue và handler (callback hoặc listener) gắn với Event đó và đẩy vào Stack.
  + Web APIs: bản chất Runtime của Javascript chỉ có 1 luồng và không thể chạy multi-thread, vì thế browser đã viết thêm một Web APIs để bọc runtime này lại. Web APIs này sẽ giúp cho JS có thể hoạt động một cách bất đồng bộ như multi-thread. (Ajax-call api, setTimeOut, setInterval, dom event)
  + Callback Queue: Như tên của nó là hàng đợi các callback do thằng Web APIs ở trên trả về.( Microtask > Macrotask > Render Queue.)

1. Event driven

* Lập trình hướng sự kiện là một mô hình lập trình.
* Luồng của chương trình được xác định bởi các sự kiện.
* Node.js sử dụng lập trình hướng sự kiện, ngay sau khi khởi động máy chủ, nó sẽ khởi tạo các biến, khai báo các hàm và chờ sự kiện xảy ra. Đó là lý do giải thích tại sao Node.js nhanh hơn so với các công nghệ tương tự khác.

1. Nodejs dùng cho

* Các ứng dụng máy chủ web: Hệ thống Notification, Websocket server, Any Real-time Data Application, Ứng dụng truy vấn tới NoSQL database

1. Tại sao nên dùng Nodejs

* Miễn phí
* Trọng lượng nhẹ
* Nhanh
* Dễ dàng xếp chồng đầy đủ (cả FE và BE đều là javascript)
* Số lượng lớn các mô-đun do cộng đồng đóng góp
* Được hỗ trợ bởi tất cả các công ty công nghệ lớn

1. Event Emitter

* Node.js dựa trên kiến trúc hướng sự kiện không đồng bộ trong đó một số đối tượng nhất định được gọi là emitters định kỳ phát ra (emit) các sự kiện (Events) khiến các Listener Object được gọi.
* Khi đối tượng EventEmitter phát ra một sự kiện, tất cả các hàm được gắn vào sự kiện cụ thể đó được gọi một cách đồng bộ.
* MỘt số phương thức:
  + AddListener(event, listener)
    - Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể
  + On(event, listener)
    - Thêm một Listener vào phần cuối của mảng các Listener cho một sự kiện cụ thể
  + 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
  + RemoveListener(event, listener)
    - Xóa một Listener ra khỏi mảng các Listener cho một sự kiện nào đó.
  + RemoveAllListeners([event])
    - Xóa tất cả Listener của một sự kiện

1. Micro task và macro task

* Có 2 loại task queue đó là macroTask Queue và microTask Queue, macroTask còn có tên gọi khác là Task Queue và microTask cũng có tên gọi là Job Queue. Hai task này nhìn chung là như nhau , những task trong Job Queue ưu tiên hơn những task có trong Task Queue. Thứ tự chạy của các task trong hai queue này như sau:
  + Ở mỗi chu kỳ của event-loop, nó sẽ check xem call stack có rỗng không, nếu rỗng:
  + Lấy một task cũ nhất trong Task Queue ra chạy.
  + Sau khi task trong Task Queue chạy xong, event loop sẽ nhìn vào Job Queue (chứa microTask) và chạy hết tất cả các tasks trong này.
  + Kết thúc chu kỳ event-loop quay lại bước 1 cho đến khi chay hết task queue.
* + Macro Task: SetTimeout, SetInterval....là những Asynchronous methods trong JS. Callbacks của những Asynchronous methods này sẽ được xếp loại vào Macro Task.
* + Micro Task: Promises...là Asynchronous class trong JS. Callbacks của Resolved Promises/Rejected Promises được xếp loại là Micro Task.

1. Git và node module
   1. Git

* Git là một hệ thống quản lý phiên bản phân tán (distributed version control system). Nhờ Git, việc quản lý code và làm việc nhóm của developer trở nên đơn giản, thuận tiện hơn.( Version control system, Free, Open source, Fast and efficient)
* Version control : Một phần mềm, Theo dõi các thay đổi của mã nguồn, Quản lý mã nguồn theo thời gian, Tăng tốc làm việc nhóm(Git, SVN, CVS)
* Working directory (git add)=> staging area (git commit)=> repository
* Staging area nghĩa là khu vực sẽ lưu trữ những thay đổi của bạn trên tập tin để nó có thể được commit, vì muốn commit tập tin nào thì tập tin đó phải nằm trong Staging Area.
* Git command:
  + Git init: Lệnh này biến một thư mục thành một kho lưu trữ Git trống. Sau khi init ta có thể add và comment
  + Git add: thêm file vào khu vực staging area
  + Git commit: Ghi lại những thay đổi được thực hiện đối với tệp vào local repository. Mỗi cam kết có một ID duy nhất.
  + Git status: Trả về trạng thái hiện tại của repository
* Git branch: Một nhánh đại diện cho một dòng phát triển độc lập.
  + Git checkout: Lệnh git checkout cho phép bạn điều hướng giữa các nhánh được tạo bởi git branch
  + Git merge: Git merge sẽ kết hợp nhiều chuỗi cam kết thành một lịch sử thống nhất. Trong các trường hợp sử dụng thường xuyên nhất, hợp nhất git được sử dụng để kết hợp hai nhánh
* Git remote:
  + Git clone: Để tạo một bản sao làm việc cục bộ của một kho lưu trữ từ xa hiện có. Git sao chép để sao chép và tải kho lưu trữ về máy tính.
  + Git remote: Để kết nối kho lưu trữ cục bộ với kho lưu trữ từ xa. Một kho lưu trữ từ xa có thể có một tên được đặt để tránh phải nhớ URL của kho lưu trữ
  + Git pull: sẽ kéo các thay đổi từ remote repo sang máy tính cục bộ.
  + Git push: Gửi cam kết cục bộ đến kho lưu trữ từ xa. Git push yêu cầu hai tham số: kho lưu trữ từ xa và nhánh mà push dành cho
* Giao thức của remote repository: Local repo, http reo, ssh repo
  1. Node module
* Node module: Module là giống như các thư viện trong C, C#, Java, … Mỗi module chứa một tập các hàm chức năng có liên quan đến một “đối tượng” của Module.
* Các loại module:
  + Built-in modules: http, fs, path,..
  + External module là những mô-đun bạn tự viết hoặc tải về từ internet như: express, jwt, ajv, ...
* Node\_modules nơi bạn có thể đặt mô-đun bên ngoài và nhập mô-đun đó vào bất kỳ đâu bên trong ứng dụng của mình mà không cần trỏ đường dẫn chính xác đến nó, giống như các mô-đun tích hợp sẵn.

1. Npm và web server
   1. Npm

* Npm:
  + Viết tắt của "Node Package Manager"
  + Một Cơ quan đăng ký phần mềm, chứa hơn 800.000 gói mã
  + Các nhà phát triển mã nguồn mở sử dụng npm để chia sẻ phần mềm.
  + Đăng ký tư nhân có phí
  + Đầu tiên được tạo dưới dạng trình quản lý gói cho Node.js
* Why use npm:
  + Dùng miễn phí
  + Mạng CDN rộng khắp thế giới
  + CLI hữu ích
  + Dễ học
  + Cộng đồng lớn
* Package.json:
  + Để quản lý các gói cài đặt cục bộ bằng npm thì cách tốt nhất là thông qua file package.json, chính là file nằm trong thư mục gốc của project. File JSON này chứa các nội dung:
    - Các gói thư viện lập trình mà project sử dụng.
    - Cho phép xác định phiên bản chính xác của các gói thư viện lập trình được sử dụng.
    - Các gói bạn xây dựng có thể chia sẻ dễ dàng với các lập trình viên khác trên toàn cầu thông qua npm.
* Package-lock.json:
  + Mục đích của package-lock.json chính là việc ngăn chặn những tính huống được mô tả phía trên : Cài đặt các dependencies từ cùng một file package.json nhưng lại dẫn đến 2 phiên bản cài đặt khác nhau.
* Npm cli:
  + Tương tự như cài đặt npm nhưng nhanh hơn
  + Được sử dụng trong môi trường tự động (nền tảng thử nghiệm, CI / CD)
  + Thực hiện cài đặt sạch sẽ các phụ thuộc
  + Chỉ có thể sử dụng khi package-lock.json tồn tại
  + node\_modules sẽ tự động bị xóa trước khi chạy ci.
  1. Web server
* Khái niệm: Web server hay còn gọi là máy chủ web, trong đó được kết nối và liên kết mạng máy tính mở rộng. Máy chủ web được cài đặt các chương trình để phục vụ ứng dụng web, chứa toàn bộ dữ liệu và nắm quyền quản lý. Web server có thể lấy thông tin requess từ phía trình duyệt web và gửi phần hồi tới máy khách thông qua HTTP hoặc giao thức khác.
  + Thuật ngữ máy chủ web có thể đề cập đến phần cứng hoặc phần mềm, hoặc cả hai đều hoạt động cùng nhau.
  + Phần cứng:
    - Nó lưu trữ phần mềm máy chủ web, các tệp thành phần của trang web.
    - Kết nối với Internet
    - Hỗ trợ trao đổi dữ liệu vật lý với các thiết bị khác được kết nối với web.
  + Phần mềm:
    - Một ứng dụng
    - Hiểu các URL
    - Hiểu giao thức HTTP
    - Cung cấp nội dung web cho khách hàng web
    - Có thể truy cập thông qua tên miền hoặc địa chỉ IP
* Route: là một cơ chế để chuyển hướng các yêu cầu đến một số enpoint, một chức năng nhất định.
* URL parameter:
  + (còn được gọi là "chuỗi truy vấn") là một cách để cấu trúc thông tin bổ sung cho một URL nhất định
  + Các thông số được thêm vào cuối URL sau ký hiệu ‘?’
  + Nhiều tham số có thể được bao gồm, phân tách bằng ký hiệu ‘&’
  + Params và query:
    - Params: chứa các tham số đường tuyến (trong phần đường dẫn của URL): Phân biệt route, tính route
    - Query: Chứa các tham số truy vấn URL(sau dấu ?)
* Request header:
  + Cho phép máy khách chuyển thông tin bổ sung đến máy chủ
  + Cung cấp ngữ cảnh yêu cầu như:
    - Chấp nhận: loại phản hồi
    - Tác nhân người dùng: thông tin về ứng dụng của yêu cầu.
    - Chấp nhận mã hóa: loại mã hóa của phản hồi
* Request body:
  + Nội dung yêu cầu là dữ liệu do khách hàng gửi đến máy chủ web
  + Thường được sử dụng với các phương thức POST và PUT
  + Nội dung thường nhập JSON hoặc XML, đôi khi là nhị phân.
* Response:
  + HTTP response được thực hiện bởi máy chủ đến máy khách
  + Cung cấp cho khách hàng tài nguyên mà khách hàng yêu cầu
  + Thông báo cho khách hàng rằng hành động mà họ yêu cầu đã được thực hiện.
  + HTTP response chứa:
    - A status line.
    - A series of HTTP headers, or header fields.
    - A message body, which is usually needed

1. Express và ejs
   1. Express

* Expressjs là một framework được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. Expressjs hỗ trợ các method HTTP và midleware tạo ra API vô cùng mạnh mẽ và dễ sử dụng.
* app.use () được sử dụng để gắn kết chức năng phần mềm trung gian hoặc gắn kết với một đường dẫn cụ thể, chức năng phần mềm trung gian được thực thi khi đường dẫn cơ sở khớp.
  1. Middleware:
* Khái niệm: những đoạn mã trung gian nằm giữa các request và response. Nó nhận các request, thi hành các mệnh lệnh tương ứng trên request đó. Sau khi hoàn thành nó response (trả về) hoặc chuyển kết quả ủy thác cho một Middleware khác trong hàng đợi.
  + Một loạt các functions được gọi bởi router layer trước khi request cuối cùng được thực thi
  + Function nhận các request, response của 1 chu kỳ HTTP request/response
  + Function có thể chình sửa request, response trước khi gửi chúng đến next middleware function
  + Có thể update response hoặc chấm dứt nếu cần thiết
  + Function nhận vào ba tham số: đối tượng request, đối tượng response và next
* Các kiểu middleware:
  + Application-level middleware: khi viết trong đối tượng app
  + Router-level middleware: express.Router()
  + Error-handling middleware: phục vụ việc sử lý lỗi: app.use(fn)(có 4 tham số)
  + Built-in middleware: express.static. Dựa trên thư viện serve-static, được dùng để cung cấp các nội dung tĩnh trong trang Web, ví dụ như các trang HTML tĩnh, các file hình ảnh, css, js, ...
  + Third-part middleware: Sử dụng Third-party sẽ giúp chúng ta thêm các chức năng cho Web App của mình mà không cần mất nhiều công implement.(cookie parser, body-parser)
  1. Ejs
* EJS (Embedded JavaScript) là một ngôn ngữ tạo khuôn mẫu đơn giản cho phép bạn tạo đánh dấu HTML bằng JavaScript đơn giản.
* app.set**(**'views', path.join**(**\_\_dirname, 'views'**))**;
* app.set**(**'view engine', 'ejs'**)**;
* Chúng ta nhúng JS vào trong file ejs bằng cú pháp: <% <js\_code> %>
* **<div>**
* **<p>**
* <% var myName = "John Does"%>
* Hello <%-myName%>!
* **</p>**
* **</div>**
* res.render**(**'path/file\_name', **{**variable: value**})**;

1. Restful-API
   1. HTTP protocols

* Http (HyperText Transfer Protocol) là giao thức truyền tải siêu văn bản được sử dụng trong www dùng để truyền tải dữ liệu giữa Web server đến các trình duyệt Web và ngược lại. Giao thức này sử dụng cổng 80 (port 80) là chủ yếu.
* là một giao thức lớp ứng dụng, cho phép tìm nạp tài nguyên, bắt đầu bởi người nhận
* Các phương thức:
  + POST – create a new resource
  + GET – read-only access
  + PUT – update an existing resource (or create a new one)
  + PATCH – update an existing resource (\*)
  + DELETE – remove a resource
  1. Apis
* API là viết tắt của Application Programming Interface (Giao diện chương trình ứng dụng) – phương thức trung gian kết nối các ứng dụng và thư viện khác nhau.
* Nó cung cấp khả năng truy xuất đến một tập các hàm hay dùng, từ đó có thể trao đổi dữ liệu giữa các ứng dụng.
* API sử dụng mã nguồn mở, dùng được với mọi client hỗ trợ XML, JSON.
* API có khả năng đáp ứng đầy đủ các thành phần HTTP: URI, request/response headers, caching, versioning, content forma…. Bạn có thể sử dụng các host nằm trong phần ứng dụng hoặc trên IIS.
* Mô hình web API dùng để hỗ trợ MVC như: unit test, injection, ioc container, model binder, action result, filter, routing, controller. Ngoài ra, nó cũng hỗ trợ RESTful đầy đủ các phương thức như: GET, POST, PUT, DELETE các dữ liệu.
* Được đánh giá là một trong những kiểu kiến trúc hỗ trợ tốt nhất với các thiết bị có lượng băng thông bị giới hạn như smartphone, tablet…
  1. Rest architecture
* REST (REpresentational State Transfer) là từ viết tắt của Chuyển trạng thái đại diện
* là một kiểu kiến trúc (không phải là một giao thức hay một tiêu chuẩn) sử dụng Giao thức HTTP.
* Cung cấp quyền truy cập vào các tài nguyên - mỗi tài nguyên được xác định bởi các URI.
* Truyền biểu diễn trạng thái của tài nguyên tới người yêu cầu hoặc enpoint
* Rest rằng buộc: Client-Server, stateless(server k lưu thông tin của request), cacheable( lưu thông tin nhận từ server vào bộ nhớ đệm), uniform interface( thiết kế url thông nhất, chỉ có 1 url) , layer system(truy cập nhiều server để lấy dữ liệu), code on command(kiểu dữ liệu trả về )
  1. Restful APIs
* là là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource dựa trên các rest rằng buộc
* Một kiến trúc máy chủ-máy khách được tạo thành từ máy khách, máy chủ và tài nguyên, với các yêu cầu được quản lý thông qua HTTP giao tiếp không trạng thái.
  1. Một số status code phổ biến
* 200 OK – Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
* 201 Created – Trả về khi một Resouce vừa được tạo thành công.
* 204 No Content – Trả về khi Resource xoá thành công.
* 304 Not Modified – Client có thể sử dụng dữ liệu cache, resource server không đổi gì.
* 400 Bad Request – Request không hợp lệ
* 401 Unauthorized – Request cần có xác thực.
* 403 Forbidden – bị từ chối không cho phép.
* 404 Not Found – Không tìm thấy resource từ URI
* 405 Method Not Allowed – Phương thức không cho phép với user hiện tại.
* 410 Gone – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
* 415 Unsupported Media Type – Không hỗ trợ kiểu Resource này.
* 422 Unprocessable Entity – Dữ liệu không được xác thực
* 429 Too Many Requests – Request bị từ chối do bị giới hạn

1. Authentication và authorization
   1. Authentication

* là quá trình xác minh danh tính của người dùng thông qua việc thu thập các thông tin xác thực và sử dụng các thông tin xác thực đó để xác nhận danh tính của người dùng.
* Vì sao dùng authentication:
  + Cho phép các tổ chức giữ an toàn cho mạng của họ bằng cách chỉ cho phép những người dùng (hoặc quy trình) đã xác thực truy cập vào các tài nguyên được bảo vệ của nó
  + Authorization bắt đầu nếu thông tin xác thực là hợp pháp
  1. Authorization
* là quá trình cho phép người dùng đã xác thực truy cập vào tài nguyên bằng cách xác định xem họ có quyền truy cập hệ thống hay không
  1. Phân biệt
* Xác thực là quá trình xác thực danh tính của người dùng đã đăng ký trước khi cho phép truy cập vào tài nguyên được bảo vệ.
* Ủy quyền là quá trình xác nhận rằng người dùng được xác thực đã được cấp quyền truy cập vào các tài nguyên được yêu cầu.
* Quá trình xác thực luôn xảy ra trước quá trình ủy quyền.
  1. JWT
* Là một chuỗi json được mã hóa
* Bao gồm 3 phần
  + Header: Loại token (mặc định là JWT - Thông tin này cho biết đây là một Token JWT) và thuật toán đã dùng để mã hóa.
  + Payload: Phần thông tin mà muốn đặt trong chuỗi
  + Signature: Phần chữ ký bí mật, được tạo ra bởi mã hóa phần Header cùng với phần Payload kèm theo đó là một chuỗi secret (khóa bí mật)
* Nên dùng JWT để Authentication hoặc trao đổi thông tin
* Chúng ta có thể lưu JWT ở phía clients (memory, local/ session cookie,, local storage,...). Nhưng JWT không được khuyến khích lưu ở local storage của trình duyệt bởi:
  + JWT token vẫn còn được lưu lại sau khi người dùng tắt browser nên session vẫn có thể restore cho đến khi JWT token hết hạn.
  + Local storage của trang web có thể bị truy cập từ Javascript, như vậy dữ liệu đã không còn được bảo vệ.
  1. Access control
* Là quá trình mà quyền truy cập vào các tài nguyên đó bị hạn chế đối với một số lượng người dùng nhất định
  1. Local Storage, Session Storage và Cookie
* Local Storage: Bạn có thể lưu trữ giá trị các biến vào localStorage của trình duyệt, biến đã lưu vào đây thì không bị quá hạn. Biến sẽ tồn tại cho đến khi bạn xóa nó bằng JavaScript hoặc xóa cache của trình duyệt.
* Session storage: SessionStorage là nơi lưu trữ biến trong trình duyệt. Dữ liệu bạn lưu ở đây sẽ tự động bị xóa khi đóng tab. Các biến trong sessionStorage không được gửi lên Server khi request. Thông tin lưu trữ trong sessionStorage lưu nhiều hơn cookie
* Cookie:
  + Thông tin được gửi lên server.
  + Có thể đọc ở phía máy chủ khác với Local/Session Storage chỉ đọc được ở phía máy khách.
  + Có hạn sử dụng: mỗi cookie thường có một khoảng thời gian sống nhất định và tùy vào mục đích sử dụng sẽ có khoảng thời gian sử dụng khác nhau, ngoài hạn sử dụng cookie còn có thời gian sống (max-age).

1. Data-modeling và validation
   1. Data modeling

* Là quá trình tạo ra một mô hình dữ liệu để dữ liệu được lưu trữ trong cơ sở dữ liệu. Mô hình dữ liệu này là một biểu diễn khái niệm của các đối tượng dữ liệu, sự liên kết giữa các đối tượng dữ liệu khác nhau và các quy tắc.
  + Mô hình hóa dữ liệu giúp thể hiện trực quan dữ liệu và thực thi các quy tắc kinh doanh, tuân thủ quy định và chính sách của chính phủ về dữ liệu. Mô hình dữ liệu đảm bảo tính nhất quán trong quy ước đặt tên, giá trị mặc định, ngữ nghĩa, bảo mật đồng thời đảm bảo chất lượng của dữ liệu.
  + Mục đích của data modeling là tạo ra phương pháp hiệu quả nhất để lưu trữ thông tin, đông thời vẫn vẫn cung cấp các quy trình truy cập và báo cáo hoàn chỉnh.
  + Có thể được so sánh với lộ trình, bản thiết kế của kiến trúc sư hoặc bất kỳ sơ đồ chính thức nào giúp hiểu sâu hơn về những gì đang được thiết kế.
* Quá trình data modeling:
  + Xác định các thực thể. Xác định các sự vật, sự kiện hoặc khái niệm được biểu diễn trong tập dữ liệu sẽ được mô hình hóa.
  + Xác định các thuộc tính chính của mỗi thực thể. Mỗi loại thực thể có thể được phân biệt với tất cả các loại khác bởi vì nó có một hoặc nhiều thuộc tính duy nhất, được gọi là thuộc tính.
  + Xác định mối quan hệ giữa các thực thể. Bản thảo sớm nhất của mô hình dữ liệu sẽ chỉ rõ bản chất của các mối quan hệ mà mỗi thực thể có với các thực thể khác.
  + Ánh xạ các thuộc tính cho các thực thể một cách hoàn chỉnh.
  + Gán các khóa khi cần thiết và quyết định mức độ chuẩn hóa cân bằng giữa nhu cầu giảm dư thừa với các yêu cầu về hiệu suất. Chuẩn hóa là một kỹ thuật để tổ chức các mô hình dữ liệu (và cơ sở dữ liệu mà chúng đại diện) trong đó các định danh số, được gọi là khóa, được gán cho các nhóm dữ liệu để biểu thị mối quan hệ giữa chúng mà không lặp lại dữ liệu.
  + Hoàn thiện và xác nhận mô hình dữ liệu. Mô hình hóa dữ liệu là một quá trình lặp đi lặp lại cần được lặp lại và tinh chỉnh khi nhu cầu kinh doanh thay đổi.
* Trạng thái :
  + Mô hình dữ liệu khái niệm: Thông thường, chúng bao gồm các lớp thực thể, các đặc điểm và ràng buộc của chúng, mối quan hệ giữa chúng và các yêu cầu liên quan về bảo mật và tính toàn vẹn của dữ liệu.(tên bảng)
  + Mô hình dữ liệu logic: Chúng ít trừu tượng hơn và cung cấp chi tiết hơn về các khái niệm và mối quan hệ trong miền đang được xem xét.(tên thuộc tính)
  + Mô hình dữ liệu vật lý: (hoàn hiện tên thuộc tính và dữ liệu)
* Lợi ích của mô hình hóa dữ liệu?
  + Giảm lỗi trong phát triển phần mềm và cơ sở dữ liệu.
  + Tăng tính nhất quán trong tài liệu và thiết kế hệ thống trong toàn doanh nghiệp.
  + Cải thiện hiệu suất ứng dụng và cơ sở dữ liệu.
  + Dễ dàng lập bản đồ dữ liệu trong toàn bộ tổ chức.
  + Cải thiện giao tiếp giữa các nhà phát triển và nhóm thông minh kinh doanh.
  + Dễ dàng và tăng tốc quá trình thiết kế cơ sở dữ liệu ở cấp độ khái niệm, logic và vật lý.
  1. Validation
* Tại sao chúng ta cần Xác thực?
  + nhằm mục đích xác minh xem giá trị của một mục dữ liệu có đến từ tập hợp các giá trị được chấp nhận (hữu hạn hoặc vô hạn) đã cho hay không.
  + Xác thực phía máy khách của bạn không đủ tin cậy
  + Dễ bị Man trong các cuộc tấn công giữa và máy chủ không bao giờ được tin tưởng phía máy khách.
* Data validation và Business Rules validation:
  + Data: Xác thực dữ liệu tập trung vào việc xác minh một mục dữ liệu.
  + Business: một khái niệm rộng hơn và gần với cách thức hoạt động và hành vi của doanh nghiệp. Vì vậy nó chủ yếu tập trung vào hành vi xác thực cũng phụ thuộc vào dữ liệu, nhưng trong phạm vi rộng hơn.( Customer can place only two orders in one day).

1. Apis document

* Khái niệm:
  + Là một nội dung kỹ thuật có thể phân phối, chứa các hướng dẫn về cách sử dụng và tích hợp hiệu quả với một API.
  + Là một hướng dẫn tham khảo ngắn gọn chứa tất cả thông tin cần thiết để làm việc với API
* Tại sao nên sử dụng API tài liệu?
  + Cải thiện việc chấp nhận người dùng.
  + Nâng cao nhận thức.
  + Tiết kiệm thời gian và chi phí hỗ trợ.
  + Bảo trì dễ dàng hơn

1. Data overview
   1. What is database

* Data:
  + Dữ liệu là tập hợp của một đơn vị thông tin nhỏ riêng biệt
  + Trong máy tính, dữ liệu có thể hoán đổi cho nhau.
* Database:
  + là một tập hợp dữ liệu có tổ chức để có thể dễ dàng truy cập và quản lý.
  + Mục đích chính của cơ sở dữ liệu là vận hành một lượng lớn thông tin bằng cách lưu trữ, truy xuất và quản lý dữ liệu.
  1. Cơ sở dữ liệu tập trung (Centralized Database)
* Toàn bộ CSDL được lưu ở một dàn máy tính (có thể bao gồm một hoặc nhiều máy được điều hành chung như một máy).
* Ưu:
  + Nó đã làm giảm rủi ro quản lý dữ liệu.
  + Tính nhất quán của dữ liệu được duy trì vì nó quản lý dữ liệu trong kho lưu trữ trung tâm.
  + Cho phép các tổ chức thiết lập các tiêu chuẩn dữ liệu.
  + Nó ít tốn kém hơn.
* Nhược:
  + Kích thước của cơ sở dữ liệu tập trung lớn ,.
  + Không dễ để cập nhật một hệ thống cơ sở dữ liệu phong phú như vậy.
  + Nếu xảy ra bất kỳ lỗi máy chủ nào, toàn bộ dữ liệu sẽ bị mất.
  1. Cơ sở dữ liệu phân tán (Distributed Database)
* Dữ liệu không lưu trữ tập trung ở một máy mà được lưu trữ ở nhiều máy trên mạng và được tổ chức thành những CSDL con.
* Loại DDB:
  + DDB đồng nhất: Những hệ thống cơ sở dữ liệu thực thi trên cùng một hệ điều hành và sử dụng cùng một quy trình ứng dụng và mang cùng một thiết bị phần cứng.
  + DDB không đồng nhất: Các hệ thống cơ sở dữ liệu đó thực thi trên các hệ điều hành khác nhau theo các quy trình ứng dụng khác nhau và mang các thiết bị phần cứng khác nhau.
* Ưu:
  + Có thể phát triển mô-đun trong cơ sở dữ liệu phân tán, tức là hệ thống có thể được mở rộng bằng cách bao gồm các máy tính mới và kết nối chúng với hệ thống phân tán.
  + Một lỗi máy chủ sẽ không ảnh hưởng đến toàn bộ tập dữ liệu.
  1. Cơ sở dữ liệu quan hệ
* Khái niệm:
  + Lưu trữ dữ liệu dưới dạng hàng (tuple) và cột (thuộc tính), và cùng nhau tạo thành một bảng (quan hệ).
  1. NoSQL
* Khái niệm : NoSQL Database là cơ sở dữ liệu được xây dựng dành riêng cho mô hình dữ liệu và có sơ đồ linh hoạt để xây dựng các ứng dụng hiện đại, dữ liệu lớn và ứng dụng nền web thời gian thực.
* Các loại nosql:
  + Key-value storage: Đây là loại lưu trữ cơ sở dữ liệu đơn giản nhất, nơi nó lưu trữ mọi mục đơn lẻ dưới dạng khóa.
  + Document-oriented Database: Một loại cơ sở dữ liệu được sử dụng để lưu trữ dữ liệu dưới dạng tài liệu giống JSON(hướng tài liệu)
  + Graph Databases: Nó được sử dụng để lưu trữ một lượng lớn dữ liệu trong một cấu trúc giống như đồ thị(dữ liệu đồ thị)
  + Wide-column stores: dữ liệu được lưu trữ trong các cột lớn với nhau, thay vì lưu trữ trong các hàng.
  + Multi-Model: Hỗ trợ nhiều loại giao diện NoSQL.
* Ưu:
  + Nó cho phép tạo ra năng suất tốt trong việc phát triển ứng dụng vì không yêu cầu phải lưu trữ dữ liệu ở định dạng có cấu trúc.
  + Đây là một lựa chọn tốt hơn để quản lý và xử lý các tập dữ liệu lớn.
  + Nó cung cấp khả năng mở rộng cao.
  + Người dùng có thể nhanh chóng truy cập dữ liệu từ cơ sở dữ liệu thông qua key-value.
  1. ACID
* ACID là bốn thuộc tính quan trọng cần đảm bảo khi thực hiện bất kỳ thao tác giao dịch nào với cơ sở dữ liệu.
  + Atomicity(nguyên tử): mọi thay đổi về dữ liệu phải đảm bảo trọn vẹn. Tính chất này đảm bảo rằng tất cả các xử lý được thực hiện hoặc không có xử lý được thực hiện.
    - Nếu transaction xử lý xảy ra lỗi ở data hay các hệ thống thì tất cả các transaction sẽ bị hủy bỏ
  + Consistency: (Nhất quán) yêu cầu tính nhất quán dữ liệu cho database, trước hoặc sau một transaction được thực hiện dù lỗi hay không, thì database vẫn phải được giữ ở trạng thái hợp lệ
    - ví dụ dữ liệu phải phù hợp với các quy định được định nghĩa cho database
  + Isolation (Cô lập): quy định từng transaction khác nhau cần phải được thực hiện trong một môi trường độc lập, nếu có 2 transaction diễn ra tại cùng một thời điểm thì cần một cơ chế đảm bảo transaction này không ảnh hưởng tới transaction khác..
  + Durability(Bền): quy định rằng khi transaction được diễn ra (thành công hoặc rollback lại khi có lỗi) thì sau đó dù có bất cứ sự cố nào diễn ra với database (mất điện, server tràn bộ nhớ...) thì khi được khôi phục lại thì dữ liệu được khôi phục sẽ giữ nguyên trạng thái trước khi có sự cố (đảm bảo không có bất cứ transaction trước đó không bị mất)..
  1. So sánh sql và nosql

|  | SQL | NoSQL |
| --- | --- | --- |
| Loại hình | là cơ sở dữ liệu dựa trên bảng và các bảng này có quan hệ với nhau | có thể dựa trên cặp tài liệu, cặp khóa giá trị từng cặp một, cơ sở dữ liệu biểu đồ. |
| Khả năng mở rộng | Chiều dọc (Có thể thêm nhiều tầng vào tòa nhà này) | Chiều ngang (Có thể thêm nhiều tòa nhà vào khu vực lân cận) |
| Lưu trữ dữ liệu phân cấp | Không thích hợp cho việc lưu trữ dữ liệu phân cấp | Phù hợp cho kho lưu trữ dữ liệu phân cấp vì nó hỗ trợ phương thức cặp khóa – giá trị. |
| Mục đích sử dụng | Được thiết kế dành cho các ứng dụng xử lý giao dịch trực tuyến, trong giao dịch có độ ổn định cao và thích hợp để xử lý phân tích trực tuyến. | Được thiết kế phục vụ cho việc phân tích dữ liệu có cấu trúc chưa hoàn chỉnh. |
| Lựa chọn sử dụng | Những dự án đã có yêu cầu dữ liệu rõ ràng xác định quan hệ logic có thể được xác định trước. | Dự án yêu cầu dữ liệu không liên quan, khó xác định, đơn giản mềm dẻo khi đang phát triển. |
| Mô hình ACID và BASE | ACID | BASE |

1. Mysql
   1. SQL

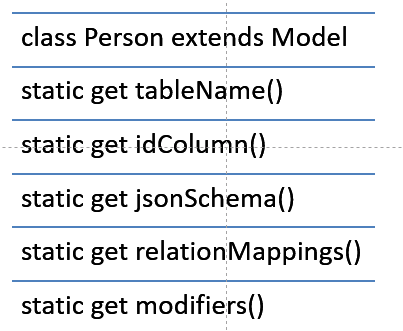
* SQL stands for the structured query language.
* SQL là ngôn ngữ chuẩn hóa được sử dụng để truy cập cơ sở dữ liệu.
* Gồm 3 phần:
  + Ngôn ngữ định nghĩa dữ liệu bao gồm các câu lệnh giúp bạn xác định cơ sở dữ liệu và các đối tượng của nó, ví dụ: bảng, view, trigger, procedure, v.v.
  + Ngôn ngữ thao tác dữ liệu chứa các câu lệnh cho phép bạn cập nhật và truy vấn dữ liệu.
  + Ngôn ngữ điều khiển dữ liệu cho phép bạn cấp quyền cho người dùng truy cập dữ liệu cụ thể trong cơ sở dữ liệu.
  1. MySQL
* Phần mềm mã nguồn mở được hỗ trợ bởi Oracle
* Có thể chạy trên nhiều nền tảng UNIX, Linux, Windows
* Đáng tin cậy, có thể mở rộng và nhanh chóng
* Hỗ trợ nhiều công cụ lưu trữ
  1. Primarykey
* Ràng buộc PRIMARY KEY xác định duy nhất từng bản ghi trong bảng.
* Khóa chính phải chứa giá trị DUY NHẤT và không được chứa giá trị NULL.
* Một bảng chỉ có thể có MỘT khóa chính;
* Khóa chính có thể chứa 1 hoặc nhiều cột
  1. Foreign key
* Ràng buộc FOREIGN KEY được sử dụng để ngăn chặn các hành động phá hủy liên kết giữa các bảng.
* FOREIGN KEY là một trường (hoặc tập hợp các trường) trong một bảng, tham chiếu đến KHÓA CHÍNH trong một bảng khác.
  1. Index
* Index trong SQL Server được tạo ra trên các cột trong bảng hoặc View. Chúng cung cấp một phương pháp giúp bạn nhanh chóng tìm kiếm dữ liệu dựa trên các giá trị trong các cột.
* Đầu tiên SQL Server sẽ tìm giá trị này trong Index, sau đó nó sử dụng Index để nhanh chóng xác định vị trí của dòng dữ liệu bạn cần tìm. Nếu không có Index, SQL Server sẽ thực hiện động tác quét qua toàn bộ bảng (table scan) để xác định vị trí dòng cần tìm.
* Câu lệnh CREATE INDEX được sử dụng để tạo chỉ mục trong bảng.
* Ví dụ: CREATE INDEX index\_name

ON table\_name ( column1, column2,...);

* Các chỉ mục được sử dụng để lấy dữ liệu từ cơ sở dữ liệu nhanh hơn so với cách khác.
* Các loại index: b-tree, hasd index,
  1. Basic sql command
* Query: OR | And, in | not in | between, Like, Limit, is null | is not null
* Query order: select, distinct, from, where, group by, having, order by
* Having: đặt các điều kiện vào các nhóm được tạo bởi mệnh đề GROUP BY.
* Các loại join: inner, left, right, full outer, self
* Insert ignore: giữ lại tập hợp bản sao bản ghi đầu tiên và loại bỏ phần còn lại. REPLACE giữ tập hợp bản sao cuối và xóa các phần trước.
  1. Transaction và câu lệnh dùng trong nó
* TRANSACTION trong SQL là tiến trình thực hiện một nhóm các câu lệnh SQL. Các câu lệnh này được thực thi một cách tuần tự và độc lập. Một Transaction được thực hiện thành công khi tất cả câu lệnh đều thành công, khi đó tất cả các thay đổi dữ liệu được thực hiện trong Transaction được lưu vào cơ sở dữ liệu. Tuy nhiên, nếu chỉ một trong số đó thất bại thì toàn bộ tiến trình sẽ thất bại, đồng nghĩa với việc dữ liệu phải rollback về trạng thái ban đầu (dữ liệu được khôi phục về trạng thái trước khi thực hiện Transaction).
* Xử lý transaction:
  + Lệnh Commit để lưu những thay đổi sau khi Transaction đã hoàn thành
  + Lệnh Rollback là để quay lại những điểm SAVEPOINT hoặc là quay lại trạng thái trước khi TranSaction được thực hiện nếu có bất kỳ một lệnh bị lỗi.
  + SAVEPOINT là tạo ra các điểm lưu trữ tại thời điểm nào đó.
  + RELEASE SAVEPOINT là xóa SAVEPOINT mà ta không muốn giữ
  + SET TRANSACTION là quy định các đặc tính cho Transaction đó được đọc (read only) hoặc đọc viết (read write)
  1. Phân biệt DELETE, TRUNCATE VÀ DROP trong sql
* DELETE xóa một hoặc tất cả các hàng từ một bảng dựa trên điều kiện và có thể được phục hồi lại.
* TRUNCATE xóa tất cả các hàng từ một bảng bằng cách phân bổ các trang bộ nhớ và không thể phục hồi lại
* DROP xóa hoàn toàn một bảng từ cơ sở dữ liệu.
  1. View
* View là đoạn lệnh truy vấn đã được viết sẵn và lưu bên trong cơ sở dữ liệu. Một View thì bao gồm 1 câu lệnh SELECT và khi bạn chạy View thì bạn sẽ thấy kết quả giống như khi bạn mở 1 Table. Các bạn có thể tưởng tượng nó giống như một Table ảo. Bởi vì nó có thể tổng hợp dữ liệu từ nhiều Table để tạo thành 1 Table ảo.
* View rất hữu dụng khi bạn muốn cho nhiều người người truy cập ở các permission khác nhau. Cụ thể là:
  + Hạn chế truy cập tới các Table cụ thể. Chỉ cho phép được xem qua View.
  + Hạn chế truy cập vào vào Column của Table. Khi truy cập thông qua View bạn không thể biết được tên Column mà View đó truy cập vào.
  + Liên kết các Column từ rất nhiều Table vào thành Table mới được thể hiện qua View.
  + Trình bày các thông tin tổng hợp(VD: sử dụng funtion như COUNT, SUM, ...)
* Sử dụng view để:
  + Sử dụng View để tập trung trên dữ liệu được xác định.
  + Sử dụng View để đơn giản hóa thao tác dữ liệu
  + Sử dụng View để tùy biến dữ liệu.
  + Sử dụng View để trích xuất dữ liệu tạo báo cáo.
  + Sử dụng View để bảo mật dữ liệu, che giấu thông tin.
  1. Wildcard
* là kí tự đại diện được sử dụng cùng với câu lệnh LIKE trong SQL để so sánh một giá trị với các giá trị tương tự.
* Các loại:
  + “%”: Đại diện cho 1 hoặc nhiều ký tự
  + “\_”: Đại diện cho 1 ký tự đơn
  1. Trigger, procedure, function
* **Trigger**: là một stored procedure được chạy tự động dựa trên những sự kiện mà nó được đăng ký. Khi các sự kiện này xảy ra trong database, thì nó cũng sẽ được thực thi.sẽ thực thi tự động mỗi khi có các sự kiện INSERT, DELETE, UPDATE xảy ra trong một table.
* **Procedure**: Procedure là một tập hợp các câu lệnh SQL dùng để thực thi một nhiệm vụ nhất định. Nó có thể xem như một hàm trong các ngôn ngữ lập trình.
* **Function**: Cũng giống với procedure.
  + Thủ tục lưu trữ có thể trả về giá trị zero, một hoặc nhiều giá trị. Trong khi hàm phải trả về một giá trị duy nhất (có thể là bảng).
  + Các hàm chỉ có thể có các tham số đầu vào cho nó trong khi thủ tục lưu trữ có thể có các tham số đầu vào hoặc đầu ra.
  + Hàm có thể được gọi từ thủ tục lưu trữ trong khi thủ tục lưu trữ không thể được gọi từ hàm.
  1. Chuẩn hóa cơ sở dữ liệu
* Gồm 3 quy tắc về dạng chuẩn:
  + Chuẩn 1 (1NF – First Normal Form): Giá trị được lưu trữ trong các ô phải là các giá trị đơn (scalar value) và trong bảng không có cột nào lặp lại.
  + Chuẩn 2 (2NF – Second Normal Form): Mọi trường không phải là khóa phải phụ thuộc vào khóa chính.
  + Chuẩn 3 (3NF – Third Normal Form): Mọi trường không phải là khóa chỉ phụ thuộc vào khóa chính mà thôi.

1. Objection

* ORM cho NodeJs
* Được xây dựng trên SQL query builder có tên là knex
* Lợi ích
  + Dễ sử dụng
  + Xác định mô hình và mối quan hệ
  + Cách đơn giản để CRUD Object
  + Xác thực lược đồ Json



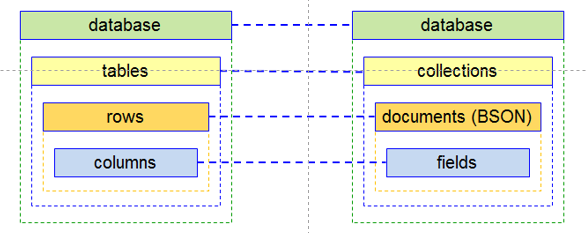
* Relation:
  + BelongsToOneRelation
  + HasManyRelation
  + ManyToManyRelation
  + HasOneThroughRelation

1. Knex
   1. Orm

* ORM (Object Relational Mapping), là một kỹ thuật/cơ chế lập trình thực hiện ánh xạ CSDL sang các đối tượng trong các ngôn ngữ lập trình hướng đối tượng như Java, C# …(các table tương ứng các class, mối ràng buộc giữa các table tương ứng quan hệ giữa các class ‘has a’ , ‘is a’).
* Ưu:
  + OOP: ORM giúp lập trình viên tập trung vào lập trình hướng đối tượng
  + Năng suất hơn: viết code ít hơn, dễ hiểu hơn. Phù hợp các case CRUD (Create, Read, Update, Delete)
  + Khả năng sử dụng lại code.
* Nhược:
  + Khả năng truy vấn bị hạn chế, nhiều trường hợp ta vẫn phải dùng native SQL để truy vấn database.
  + Khó tối ưu câu lệnh SQL (do câu lệnh SQL được ORM tự động sinh ra).
  1. Knex
* Là Mô-đun trình tạo truy vấn SQL, Kết nối gộp.
* knex query builder là giao diện được sử dụng để xây dựng và thực thi các truy vấn SQL tiêu chuẩn.
* Cần xác định số nhận dạng là tên bảng hoặc tên cột

1. MongoDB

* Khái niệm :
  + Một loại NoSQL DB
  + Tài liệu database
  + Cấu trúc dữ liệu linh hoạt
  + Cơ sở dữ liệu nguồn mở
  + 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.
  1. MongoDB basic
* Mg Db: can contain one or more collection
* Collection: can contain different types of document(object)
* Document: key-value



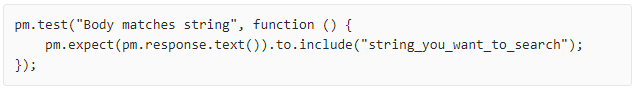
* 1. Query
* db.collection.find()

1. Mongoose

* Là mongo object data modeling cho nodejs (mô hình hóa đối tượng)
* Bắt dữ liệu phải có schema trước. Model của Mongoose sau đó được ánh xạ tới một MongoDB document thông qua định nghĩa schema của Model.
* Không hỗ trợ promise
* Ưu:
  + Cung cấp một sự trừu tượng.
  + Giá trị mặc định và dữ liệu xác thực.
  + Các chức năng có thể được gắn vào model trong MongooseJS.
  + Linh hoạt và dễ đọc hơn.
* Query..exec(callback)
* Khi bạn sử dụng.lean () Mongoose trả về các đối tượng JSON bình thường thay vì các tài liệu nặng về bộ nhớ và tài nguyên.
* Static schema function: ta có thể tự định nghĩa một hàm dùng như các hàm có sẵn
* Middleware: Các chức năng mà bạn có thể xác định trong các mô hình Mongoose của mình sẽ được thực thi để sửa đổi dữ liệu trước hoặc sau khi lưu.
  + userSchema.pre('save', function capitalizeNames(next) {  
     this.firstName = capitalize(this.firstName);  
     this.lastName = capitalize(this.lastName); next();   
    });

1. Unit test framework
   1. MochaJS

* là một khung kiểm tra JavaScript dành cho các chương trình Node.js, có hỗ trợ trình duyệt, kiểm tra không đồng bộ, báo cáo phạm vi kiểm tra và sử dụng bất kỳ thư viện assertion nào.
* Hook: Mocha cung cấp các hook before (), after (), beforeEach () và afterEach (). Chúng nên được sử dụng để thiết lập các điều kiện tiên quyết và dọn dẹp sau các bài kiểm tra của bạn.( “BDD”-style interface)
* Hỗ trợ xử lý bất đồng bộ với callback, async await và promise
* EXCLUSIVE TESTS – only(): chỉ chạy cái được đặt là only
* INCLUSIVE TESTS – skip(): bỏ qua case hiện tại
* Interface:
  + BDD: describe(), context(), it(), specify(), before(), after(), beforeEach(), afterEach()
  + TDD: suite(), test(), suiteSetup(), suiteTeardown(), setup(), teardown()
* Assertion library: should.js, express.js, chai, better-assert, Unexpected
  1. Jest
* Jest is an open JavaScript testing library from Facebook.
* Set up and teardown: afterEach, beforeEach, afterAll, beforeAll
* Testing async: callback, promise,reject, resolve, async-await
* Mock Functions & .mock property
  + Mock Functions giúp dễ dàng kiểm tra các liên kết giữa các mã bằng cách xóa việc triển khai thực tế của một chức năng.
  + Ghi lại các lần gọi đến hàm (và các tham số được truyền trong các lệnh gọi đó).
  + Chụp các intance của hàm khởi tạo khi được khởi tạo bằng Mocking modules mới.
  + Cho phép cấu hình thời gian thử nghiệm của các giá trị trả về.
  + Tất cả các Mock Functions đều có .mock property đặc biệt này
* Assertion: Chai , expect
  + The Expect / Should API of CHAI covers the BDD assertion styles.
  + The Assert API of CHAI covers the TDD assertion style.
  1. Postman
* Với Postman, bạn có thể viết và chạy các bài kiểm tra cho từng yêu cầu bằng ngôn ngữ JavaScript.
* Kiểm tra Postman về cơ bản là mã JavaScript được thực thi sau khi yêu cầu được gửi, cho phép truy cập vào đối tượng pm.



1. Mock
   1. Khái niệm

* Mocking: tạo các đối tượng mô phỏng hành vi của các đối tượng thực
  + Mocking chủ yếu được sử dụng trong kiểm thử đơn vị.
  + Một đối tượng đang được kiểm tra có thể có sự phụ thuộc vào các đối tượng (phức tạp) khác.
  + Cô lập hành vi của đối tượng mà bạn muốn thay thế các đối tượng khác bằng cách mô phỏng hành vi của đối tượng thực.
  + Hữu ích nếu các đối tượng thực không thực tế để đưa vào bài kiểm tra đơn vị.
* Mock thư viện: sinon(mochajs) và jest mock

1. Models on Units testing
   1. Test-driven development (TDD)

* là một quá trình phát triển phần mềm dựa trên sự lặp lại của một chu kỳ phát triển rất ngắn: các yêu cầu được chuyển thành các trường hợp thử nghiệm rất cụ thể, sau đó phần mềm được cải tiến để vượt qua các thử nghiệm mới.
* circle: Thêm một bài kiểm tra, Chạy tất cả các bài kiểm tra và xem liệu bài kiểm tra mới có thất bại hay không, Viết mã, Chạy thử nghiệm, Refactor mã, lặp lại
* Acceptance Test Driven Development (ATDD) xác định Tiêu chí chấp nhận và Kiểm tra chấp nhận trong quá trình tạo user stories, trong giai đoạn đầu phát triển.
  1. BDD
* Là một quy trình phát triển phần mềm ban đầu xuất hiện từ Phát triển theo hướng kiểm tra (TDD).
* BDD sử dụng các ví dụ để minh họa hoạt động của hệ thống được viết bằng ngôn ngữ dễ đọc và dễ hiểu cho mọi người tham gia vào quá trình phát triển.
* Đặc điểm:
  + Chuyển từ suy nghĩ trong “bài kiểm tra” sang suy nghĩ trong “hành vi”
  + Ngôn ngữ tự nhiên (ai cũng có thể hiểu được), rất dễ mô tả
  + Được thúc đẩy bởi giá trị kinh doanh
  + Mở rộng phát triển theo hướng kiểm tra (TDD) bằng cách sử dụng ngôn ngữ tự nhiên mà các bên liên quan không liên quan đến kỹ thuật có thể hiểu
  + Các khung công tác BDD như Cucumber hoặc JBehave là một công cụ hỗ trợ, đóng vai trò “cầu nối” giữa Ngôn ngữ Kinh doanh và Kỹ thuật

1. The concept of unit testing
   1. Unit testing

* là phần nhỏ nhất có thể kiểm tra được của một ứng dụng như hàm, lớp, thủ tục, giao diện. Unit testing là một phương pháp mà các đơn vị mã nguồn riêng lẻ được kiểm tra để xác định xem chúng có phù hợp để sử dụng hay không.
* Cơ bản được viết và thực hiện bởi các nhà phát triển phần mềm để đảm bảo rằng mã đáp ứng thiết kế và yêu cầu của nó và hoạt động như mong đợi.
* Ưu:
  + Phát hiện vấn đề sớm.
  + Dễ dàng bảo trì và tái cấu trúc.
  + Chi phí sửa lỗi ít hơn.
  + Gỡ lỗi đơn giản.
* Best practice:
  + Bài kiểm tra đơn vị chỉ phải làm một việc
  + Unit Test phải chạy độc lập
  + Unit Test không được phụ thuộc vào môi trường
  + Kiểm tra chức năng thay vì triển khai
  + Kiểm tra hành vi nơi công cộng; hành vi riêng tư liên quan đến chi tiết triển khai
  + Tránh thử nghiệm các thành phần giao diện người dùng
  + Unit Test phải dễ đọc và dễ hiểu
  + Tạo các quy tắc đảm bảo rằng bạn cần chạy Kiểm tra đơn vị (và chúng cần phải vượt qua) trước khi bạn được phép Đăng ký mã của mình trong Hệ thống kiểm soát mã nguồn
  + Thực sự là đơn vị, không phải tích hợp. Các bài kiểm tra đơn vị và tích hợp có các mục đích khác nhau. Cả bài kiểm tra đơn vị và hệ thống được kiểm tra không được truy cập vào tài nguyên mạng, cơ sở dữ liệu, hệ thống tệp, v.v., để loại bỏ ảnh hưởng của các yếu tố bên ngoài.
  1. Integration testing
* kiểm tra tích hợp hoặc giao diện giữa các thành phần, tương tác với các phần khác nhau của hệ thống như hệ điều hành, hệ thống tệp và phần cứng hoặc giao diện giữa các hệ thống.
* tuân theo hai cách tiếp cận được gọi là cách tiếp cận 'Từ trên xuống' và cách tiếp cận 'Từ dưới lên' như thể hiện trong hình ảnh bên dưới
* Type:
  + Big Bang integration testing: tất cả các thành phần hoặc mô-đun được tích hợp đồng thời, sau đó mọi thứ được kiểm tra tổng thể.
  + Top-down integration testing : Kiểm tra diễn ra từ trên xuống dưới, theo luồng điều khiển hoặc cấu trúc kiến trúc
  + Bottom-up integration testing: Kiểm tra diễn ra từ dưới cùng của luồng điều khiển trở lên. Các thành phần hoặc hệ thống được thay thế bởi trình điều khiển
* best practice:
  + Làm thử nghiệm tích hợp trước khi thử nghiệm đơn vị
  + Không kiểm tra logic nghiệp vụ với kiểm tra tích hợp
  + Biết tại sao kiểm thử tích hợp khác với kiểm thử đơn vị
  + Giữ các bộ thử nghiệm của bạn riêng biệt
  + Đăng nhập rộng rãi
  + Không dừng lại ở thử nghiệm tích hợp