Chapter 1: Website hoạt động như thế nào ?

#1. Feeling
Về sự cảm nhận (nhìn bằng mắt)/ tương tự như fresher tester :v

I. Góc nhìn của người dùng (user/interface)
Sử dụng website:
1. Truy cập vào website thông qua 1 đường dẫn link, ví dụ: facebook.com ; sis.hust.edu.vn
2. Chọn tính năng và sử dụng
3. Trong 1 lần sử dụng, có thể sử dụng nhiều tính năng một lúc, như xem video, chat...
- Đôi khi, thao tác bị lỗi, thì 'thấy' có thông báo hiện lên. hoặc đang dùng, cũng thấy có thông báo.
ví dụ: notification khi có tin nhắn mới

- Một vài website trông đẹp khi xem trên máy tính bàn (desktop), tuy nhiên trông khá xấu trên điện thoại
ví dụ: website của mình chẳng hạn :v
//todo: bổ sung link

II. Góc nhìn của lập trình viên
(sử dụng từ ngữ chuyên ngành -> sau này dùng để search google)
1.Truy cập vào website thông qua 1 đường dẫn link
ví dụ: facebook.com => đây gọi là 1 URL

2. Chọn tính năng và sử dụng
=> đây gọi là 'quá trình điều hướng trang" (navigate)

ví dụ:
từ trang facebook, xem tính năng profile
=> facebook.com điều hướng sang trang facebook.com/profile (navigate)
/profile => gọi là route

3. Đôi khi bị lỗi
=> popup hiện lên => gọi là modal
- Giao diện trong đẹp trên máy tính, tuy nhiên, dùng điện thoại trông xấu
=> gọi là 'không' responsive (màn hình không co giãn)

Tất cả các keyword ở trên, là tích lũy theo thời gian...

#2. Xây dựng Chức năng của một website

Một website dù phức tạp đến đâu, đều xoay quanh các tính năng về CRUD.
CRUD là viết tắt của Create, Read, Update và Delete.

1. Create (C) : thêm mới

Create: tạo ra 1 cái gì đấy mới, tạo ra thêm dữ liệu
Ví dụ: bạn đăng 1 bài post facebook để cập nhật trạng thái
=> góc nhìn dev: user CREATE a new post

2. Read (R) : hiển thị (đọc thông tin hiển thị)

Read: là chức năng giúp hiển thị thông tin
ví dụ: bạn lướt newFeed facebook để cập nhật thông tin của bạn bè, 'đọc' các thông tin mới.
=> góc nhìn dev: hiển thị (READ) các bài post

3. Update (U): cập nhật thông tin

ví dụ: bạn đăng 1 bài post, phát hiện lỗi chính tả => nhấn nút edit để sửa, sau đấy nhấn nút 'save' để xác nhận cập nhật
=> góc nhìn dev: thực hiện update thông tin với bài post cần cập nhật

4. Delete (D): xóa thông tin

ví dụ: bài post bạn đăng lên tạo sóng gió trên mạng xã hội, gây cho bạn nhiều phiền phức. vì vậy bạn lựa chọn option delete để khiến cho bài post biến mất
=> góc nhìn dev: thực hiện delete bài post được yêu cầu

5. Xây dựng tính năng

Một tính năng dù phức tạp đến đâu, đều phân theo CRUD. Nếu khác, đấy chính là logic xử lý

=> cần tính toán xử lý trước khi thực hiện 1 hành động.

Bất kỳ dự án nào, tính năng đều gói gọn với CRUD
=> học 1 framework/language => thực hành CRUD

#3. Mô hình hoạt động
Mô hình client-server:
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

1. Định nghĩa
- Client (người dùng/user): người sử dụng website bằng công cụ,
ví dụ: trình duyệt web (browser: Google Chrome, Firefox...)

user dùng máy tính (hoặc mobile) -> mở browser -> truy website và sử dụng.

Bạn A ngồi nhà, mở máy tính lên, mở tiếp Google Chrome => gõ facebook.com => enter và sử dụng (tính là 1 client)

Bạn B đi học trung tâm, mở điện thoại lên, dùng brower và vào facebook.com => sử dụng (tính là 1 client)
....

=> 1 website có thể có nhiều người dùng (clients) cùng 1 lúc (tại 1 thời điểm)

- Server (máy chủ): gồm 2 thành phần
+ tên miền (domain) : là 1 tên dễ nhớ, giúp client có thể nhớ, gõ vào trình duyệt và sử dụng
+ nơi hosting files: là 1 máy tính trên cloud (kết nối internet), lưu trữ tất cả mã nguồn của website

Hiểu đơn giản, server là 1 máy tính hoạt động 24/24, giúp 1 website hoạt động.

2. Mô hình
Bạn A, dùng máy tính mở Google Chrome, gõ Facebook.com => enter và sử dụng Facebook.

Ở đây, bạn A dùng google Chrome lướt Facebook => tính là 1 client

bạn A dùng FireFox lướt FB => cũng tính là 1 client

Bạn A gõ vào Facebook.com, trình duyệt web trả ra giao diện website vì:
đường link facebook.com đã được 1 server 'kiểm soát'.

Ở đây, mô hình sẽ là yêu cầu <-> phản hồi (request <-> response)

client -> gửi request lên server
(Bạn A gõ facebook.com, enter và ngồi chờ website load giao diện Facebook.com - mong muốn sử dụng website)

server -> gửi response cho client (Server nhận đc yêu cầu của client, sẽ tạo giao diện website và gửi lại kết quả - giao diện website facebook)

#4. Vận Hành Một Website

1. Chạy localhost

(local tức là chạy tại máy tính cá nhân, bạn bè (của bạn) không thể dùng đường link localhost để chạy và sử dụng website được)

bạn mở tính tính, coding, gõ câu lệnh để chạy website, và mở browser.
ví dụ: chạy http:localhost:3000 => đây là đường link local, bạn sử dụng được vì:

+ client: chính là bạn luôn (mở website, gõ đường link và sử dụng)
+ server: còn ai ở đây nữa (ngoài bạn thì còn ai khác :v).
việc bạn code, sau đấy gõ lệnh sẽ giúp chạy server tại máy tính bạn.

nhờ có server được chạy, thì đường link 'localhost' mới sống => sử dụng được website

+ ưu điểm: bạn kiểm soát mã nguồn, thích làm gì thì làm => đây là quá trình xây dựng website từ con số 0 (làm từ đầu)

+ nhược điểm: chạy local (máy tính của bạn), người khác không thể sử dụng được localhost

=> gõ vào máy tính khác nó không ra gì đâu :D

2. Chạy với hosting

(LƯU Ý QUAN TRỌNG: bước này mình không hướng dẫn trong các khóa học của mình)

gồm 2 bước: mua tên miền và mua hosting

B1: Tên miền (domain) là 1 tên dễ nhớ, bằng chữ, ví dụ: facebook.com, twitter.com

B2: Mua hosting và deploy code của bạn lên đấy

Việc mua hosting sẽ cho bạn 1 địa chỉ IP (IP của server, ví dụ: 192.168.1.69), đồng thời cung cấp nguồn tài nguyên để bạn triển khai (deploy) code local lên lưu trữ tại đây.

Sau khi deploy xong, bạn cần mapping domain với cái IP hosting này.

Hiểu đơn giản là như này:
B1: bạn bỏ tiền ra mua 1 tên miền (domain). Tên miền là duy nhất (unique),
thành ra, bạn phải nghĩ ra và mua 1 cái tên chưa được sử dụng trước đấy.

công dụng của nó là mua 1 cái tên dễ viết, dễ nhớ và không đụng hàng (unique)

ví dụ: bạn nhớ trang google.com chứ không nhớ 192.168.1.69
thì ở đây: google.com chính là tên miền

B2: mua 1 máy tính server (chạy trên cloud).
Máy tính trên cloud sẽ có 1 địa chỉ IP, ví dụ 192.168.1.96
Chúng ta cần địa chỉ này để có thể truy cập vào nó.

Sau khi truy cập vào server này, các bạn cần tải code (local) lên trên này lưu trữ,và cài đặt các phần mềm cần thiết,

ví dụ như database chẳng hạn.

=> đây gọi là bước deploy, giúp chuyển code từ máy tính local lên 1 máy tính ở cloud.

B3: Cấu hình tên miền trỏ vào địa chỉ IP server.

Tức là: khi người dùng gõ ten-mien-cua-ban.com => máy tính sẽ tự động trỏ tới 192.168.1.96 (địa chỉ server)

Do máy tính server hoạt động 24/7 => bạn có thể truy cập và sử dụng website ở bất cứ đâu, bất cứ khi nào bạn muốn

- ưu điểm: website chạy thực tế, 'sống' 24/7, ai cũng có thể truy cập và sử dụng
- nhược điểm: tốn tiền :v

#5. Dữ Liệu Để Hiển Thị Website

Dữ liệu hiển thị là cái 'mắt' chúng ta thấy. gồm 2 loại, dữ liệu động (dynamic) và dữ liệu tĩnh (static).

1. Dữ liệu tĩnh (static)
Dữ liệu static là các dữ liệu cố định, và không thay đổi theo thời gian. Ví dụ: tên/ảnh thương hiệu website

Do tính chất 'cố định', chúng ta thường 'hard-code' fix cứng trong mã nguồn (source code), viết 1 lần và dùng mãi mãi.

Ưu điểm: dữ liệu được lưu trong source code, all-in-one
Nhược điểm: khó thay đổi khi có yêu cầu sửa đổi

2. Dữ liệu động (dynamic)
Dữ liệu dynamic là dữ liệu thay đổi theo thời gian (ngược hoàn toàn với static).
ví dụ: hôm nay bạn lướt facebook, post hiện lên nó khác, ngày mai nó khác và thay đổi liên tục.
Ưu điểm: thay đổi liên tục theo thời gian
Nhược điểm: cần code logic để xử lý :D

3.Lưu trữ dữ liệu
Có dạng lưu trữ dữ liệu phổ biến: lưu trữ data trong file và lưu trữ dữ liệu trong 'phần mềm'

- Lưu trữ trong file: ví dụ, bạn lưu trữ thông tin user trong file .txt, file .csv
ưu điểm: thuận tiện cho người không coding và quen thuộc với các phần mềm cơ bản (word, excel...)
nhược điểm: lưu trữ data không hiệu quả, nếu như data gồm nhiều loại dữ liệu, và khối lượng kích thước lớn

- Lưu trữ trong 'phần mềm': các phần mềm hỗ trợ việc lưu trữ data, gọi là các hệ quản trị cơ sở dữ liệu hiểu đơn giản là database.

ưu điểm: lưu trữ data hiệu quả, đảm bảo lưu trữ được nhiều loại dữ liệu và khối lượng dữ liệu lớn
nhược điểm: cần 'học' cú pháp để có thể sử dụng các phần mềm này

#6. Website Khi Có Lưu Trữ Data

Data được lưu trữ ở đâu ?

1. Ví dụ: người dùng thực hiện hành động đăng nhập

B1: nhập username/password
B2: nhấn nút đăng nhập
B3: nhận kết quả

Thì bí mật đứng sau là gì: Tại bước 2 chuyển qua bước 3, website sẽ cần phải làm 1 cái gì đấy, cụ thể là:
- website sẽ cần phải kiểm tra thông tin người dùng cung cấp, và thông tin 'nó đã lưu trữ từ trước đấy - tập users'
- nếu thông tin cung cấp là chính xác => thành công

Vậy website lưu trữ data ở đâu ? => database

Chúng ta sẽ dùng database để lưu trữ dữ liệu.

#7. Database

Database là cách chúng ta 'tổ chức và lưu trữ' dữ liệu website một cách hiệu quả.

Ví dụ: website có 100k users, bạn cần tìm thông tin của bạn (1 rows/record) trong 100k ấy. Nếu không 'tổ chức' dữ liệu, thì làm sao có thể giảm thiểu thời gian tìm kiếm ?

Với database, chúng ta lưu trữ dữ liệu với 2 cách:
- Sử dụng dữ liệu quan hệ (Relational Database - SQL) : MySQL, Postgres, Oracle, SQL Server...
- Sử dụng dữ liệu phi quan hệ (Non-Relational Database / NoSQL): MongoDB, Redis...

1. SQL (Relational Database)
- Là cơ sở dữ liệu quan hệ
- Dữ liệu được lưu trữ trong các tables. Mỗi tables có nhiều rows, fields.
- Các table có 'quan hệ với nhau'

2. NoSQL
- Là cơ sở dữ liệu phi quan hệ
- Dữ liệu được tổ chức dưới dạng 'documents' => object

3. Lưu ý khi học Database (Với beginners)
- Nên biết cách câu lệnh query trước (raw query) trước khi sử dụng các công cụ ORM/ODM

- Lý do: bản chất của ORM/ODM là giúp code ngắn đi cho dev, tuy nhiên, nó vẫn 'convert' ra raw query (vì database chỉ có thể hiểu raw query, không hiểu các cú pháp ORM/ODM)

Thành ra, trong trường hợp ORM/ODM bị lỗi, chúng ta cần check câu lệnh 'raw query' nó tạo ra để biết được lỗi đang ở đâu :D

Ví dụ: muốn tìm kiếm user có id =1 trong tables users

với ORM/ODM: findUserById(1)

với raw query: select * from users where id = 1 ; //sql
db.collection.find({ id: 1 }) //nosql

- Nguồn học:
SQL: https://www.w3schools.com/sql/default.asp

NoSQL: https://www.mongodb.com/docs/manual/reference/method/js-collection/

Câu hỏi đặt ra là nên học cái nào trước ???

Chapter 2 : Web Server với Node.JS

#8. What is Node.JS

1.Web server

Để viết (tạo ra) một web server, chúng ta cần 'sử dụng' các ngôn ngữ backend, có thể kể tới như: Java, PHP, Python, Ruby... và Javascript.

Chúng ta không thể sử dụng các công cụ ở Frontend như HTML, CSS... để tạo nên server, tuy nhiên Javascript là 1 ngoại lệ.

Javascript có thể chạy được ở Frontend, hoặc thậm chí được dùng để viết server Backend :D

Node.JS chính là công cụ (môi trường) giúp chúng ta có thể thực thi code Javascript trên Server.

Node.JS là platform (môi trường), không phải library (thư viện) hay framework.

Khi nói về học backend với Node.JS, hàm ý là chúng ta sử dụng Javascript cho backend, và học cách sử dụng các thư viện, framework viết backend với javascript

Việc này tương tự với:
Windows là platform. Chúng ta sử dụng phần mềm Microsoft Word
—
Windows là node.js. Chúng ta sử dụng website Microsoft Word . Dev là người code ra Microsoft Word bằng cách sử dụng ngôn ngữ javascript

2. Cài đặt Node.JS

Download version mới nhất : https://nodejs.org/en/download/

Download version v14.17.0 (recommend) : https://nodejs.org/download/release/v14.17.0/

Sau khi cài đặt thành công Node.JS, công việc này sẽ:

- Cài đặt môi trường Node.JS
- Cài đặt NPM ứng với version Node.js đã cài đặt

Kiểm tra kết quả bằng cách sử dụng câu lệnh:
node -v
npm -v

3. Setup VSCode

Download VSCode: https://code.visualstudio.com/download

Setup auto format code / press ctrl + S

#9. NPM - Node Package Manager
NPM chính là công cụ quản lý các thư viện của Node.js

Trang chủ: https://www.npmjs.com/

Why ?

NPM là công cụ giúp chúng ta quản lý các thư viện được xây dựng sẵn cho node.js, bao gồm miễn phí và trả phí.

Nó chính là nơi 'lưu trữ' các thư viện mà lập trình viên chúng ta sẽ 'kéo về' và dùng thôi :v

Ngoài ra, NPM cung cấp giao diện dòng lệnh (CLI) để chúng ta có thể cài đặt các thư viện cần thiết một cách dễ dàng

Khi cài đặt thành công Node.JS, chúng ta đã cài đặt NPM CLI

Kiểm tra bằng câu lệnh: npm -v

Một vài câu lệnh thường dùng:

Tài liệu: https://docs.npmjs.com/cli/v8/commands/npm-install

Câu lệnh sử dụng nhiều nhất:
npm i library_name

Hoặc viết đầy đủ là **npm install library_name**
- file package.json
- 1 vài câu lệnh

#10. Hello world với Node.js (basic)

1. Hello world với browser

- Tạo file HTML
- Đính kèm file Javascript vào HTML

2. Hello world với node.js (chạy với node)

Để chạy file javascript, sử dụng câu lệnh:
node path_to_file.js

3. Lý do gọi Node.JS là backend

Với cách làm thứ 2, chúng ta đang chạy file ở server, và trả ra kết quả, và không phụ thuộc vào browser (như cách làm 1)

=> chúng ta gọi Node.JS là backend

4. Setup Git

Quản lý mã nguồn dự án thông qua Git

#11. Các thành phần của URL (bổ trợ)
Tài liệu: https://www.ibm.com/docs/en/cics-ts/5.2?topic=concepts-components-url

URL (Uniform Resource Locator), hiểu đơn giản là đường link để truy cập tới 1 website.
ví dụ:
https://www.typescriptlang.org/docs/handbook/2/basic-types.html
https://www.ibm.com/docs/en/cics-ts/5.2?topic=concepts-components-url

http://www.example.com/software/index.html

Cấu trúc của URL: **scheme://host:port/path?query**
Lấy ví dụ: http://www.example.com/software/index.html

trong đó:
- A scheme: định nghĩa protocol để truy cập nguồn tài nguyên trên internet, bao gồm HTTP (không có SSL) và HTTPs (có SSL)
SSL : Secure Sockets Layer => giúp website an toàn hơn

=> scheme = http

- A host: nơi chứa (lưu trữ server), có thể gọi là domain name
host = www.example.com

host name có thể chứa PORT number
Tại server, 1 server có thể sử dụng nhiều **'process'** tại 1 thời điểm. Để 'định danh' từng process, PORT được sử dụng.

PORT number (16 bit), có thể lên tới 65535

Khi client truy cập vào 1 hostname (URL của server), server sẽ 'tự động' sử dụng PORT hay dùng (well-known) để khởi tạo.
ví dụ: port hay dùng (mặc định)
FTP (File Transfer Protocol) : 21
Telnet: 23
HTTP : 80
HTTPS: 443

=> khi truy cập vào http://www.example.com, thực chất là đang truy cập vào:
http://www.example.com:80
https://facebook.com === https://facebook.com:443

=> với dev, dùng PORT để có thể chạy nhiều project 1 lúc tại local,
vì : **http://localhost === http://localhost:80**

- A path: nguồn tài nguyên ở host mà client muốn truy cập
ví dụ:
https://www.typescriptlang.org/docs/handbook/2/basic-types.html
https: facebook.com/watch

path = /docs/handbook/2/basic-types.html

- A query string: nếu được sử dụng, thì nó sẽ cung cấp các thông tin mà client truyền lên server
ví dụ: https://www.google.com/search?q=ip

query string : q=ip

#12. Hello world với Node.js (advance)

Trong #10, chúng ta đã biết cách sử dụng node.js để chạy file javascript, bằng cách sử dụng **node path_file_js**

#11 giúp chúng ta hiểu được ý nghĩa của 1 URL

1. Tạo server website

Tài liệu: https://nodejs.dev/en/learn/introduction-to-nodejs/

```
//Load HTTP module (thư viện đã có sẵn khi cài đặt node.js)
const http = require("http");

const hostname = "127.0.0.1"; //cái này === http://localhost
const port = 3000;

//Create HTTP server and listen on port 3000 for requests
const server = http.createServer((req, res) => {
  //Set the response HTTP header with HTTP status and Content type
  res.statusCode = 200;
  res.setHeader("Content-Type", "text/plain");
  res.end("Hello World\n");
});

//listen for request on port 3000, and as a callback function have the port listened on logged
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});


//tự động chạy trên localhost
// server.listen(port, () => {
//     console.log(`Server running at http://${hostname}:${port}/`);
// });
```

2. Fix lỗi trùng port

```
Error: listen EADDRINUSE: address already in use
Emitted 'error' event on Server instance at:
  code: 'EADDRINUSE',
```

- Cần đọc thông tin lỗi, và 'THINK' tại sao lỗi

EADDRINUSE: address already in use
// ADDRINUSE === address in use

https://stackoverflow.com/questions/39632667/how-do-i-kill-the-process-currently-using-a-port-on-localhost-in-windows

Cách dễ nhất:

npx kill-port PORT

hoặc cách dễ hơn nữa:

- Khởi động lại máy tính :v
hoặc

Chạy project tại port khác :v

#13. Cài Đặt Thư Viện Node.JS với NPM
với các thư viện có sẵn của Node.js, chúng ta có thể chạy được web server (video #12), tuy nhiên, we want more

với việc sử dụng NPM, chúng ta có thể cài đặt các thư viện cần thiết để xây dựng server (ngoài các package có sẵn)

1. Câu lệnh npm init
npm init
giúp tạo ra file **package.json** và các thông tin cần thiết: app name, version...

2. Cài đặt thư viện
Chỉ có thể thực hiện, khi đã có file package.json

- Cài 1 package:
npm install <package-name>

- cài 1 package với version cụ thể:
npm install <package-name>@<version>

- Cài đặt express:
https://www.npmjs.com/package/express

npm i --save-exact express@4.18.2

3. Running tasks
npm run <task-name>

```
{
  "scripts": {
    "start-dev": "node server-development.js",
    "start": "node server-production.js"
  }
}
```

---
ví dụ với webpack:

```
{
  "scripts": {
    "watch": "webpack --watch --progress --colors --config webpack.conf.js",
    "dev": "webpack --progress --colors --config webpack.conf.js",
    "prod": "NODE_ENV=production webpack -p --config webpack.conf.js"
  }
}
```

thay vì phải gõ câu lệnh dài dòng như vậy, thay bằng:
$ npm run watch
$ npm run dev
$ npm run prod

#14. Hello world với Express

1. Các framework xây dựng website với Node.js

https://expressjs.com/en/resources/frameworks.html

KOAJS
HAPI

- Sử dụng express vì: dễ dàng customize

2. Hello world
https://expressjs.com/en/starter/hello-world.html

- update file package.json để chạy project

#15. Do we need Babel ?

Minh họa lỗi:
Sử dụng keyword 'import' thay vì 'require'

1. Lịch sử version của javascript
https://www.w3schools.com/js/js_versions.asp

Các version sau thêm nhiều chức năng mới cho JS, tuy nhiên, Node.JS có hiểu ???

2. Babel (Compiler)
Tài liệu:https://babeljs.io/

- Input: tất cả cú pháp hiện có của Javascript
- Output: chuyển về cú pháp JS version cũ, mà 'ứng dụng' chạy JS có thể hiểu

Ưu điểm: Giúp code chạy được ở nhiều 'ứng dụng' mà không cần quan tâm về 'version'. Developer có thể 'code theo phong cách' tùy thích. Dùng version nào của JS cũng không lo bị lỗi

Nhược điểm: Cần setup cho ứng dụng chạy với babel

- Mô hình hoạt động:

+ Chạy ở dev:
B1: Coding (source1)
B2: Dịch code sang source2 và lưu trong memory
B3: chạy code từ source2 (ở đây, không tạo thêm source code mới)

+ Chạy ở production:
B1: Coding (source1)
B2: Dịch code. từ source1 dịch code sang source2 (tạo ra source code mới)
B3: chạy code source2 trên production

3. Phát triển ứng dụng với Babel

- Ứng dụng frontend (ví dụ React): cần Babel dịch code để project có thể chạy trên nhiều loại Browser và các version cũ của Browser (phần này phụ thuộc vào client)

- Ứng dụng backend (xây dựng server với backend): **optional**
Do backend chỉ cần Node.JS, và thông thường, phụ thuộc vào developer, hay dùng các version mới của Node.js, **nên không cần thiết phải dùng Babel**

Với các framework, và sử dụng Typescript, thì 'nên dùng Babel'

Đối với dự án tự viết => không cần thiết để giảm thiểu độ phức tạp (complicated)

4. Hướng giải quyết khi không sử dụng Babel

- Khuyến khích sử dụng các version mới của Node.JS

và :
https://stackoverflow.com/questions/31354559/using-node-js-require-vs-es6-import-export

https://stackoverflow.com/questions/58384179/syntaxerror-cannot-use-import-statement-outside-a-module

#16. Mô hình hoạt động của Express
- Mô hình client server hoạt động như thế nào?

Tài liệu:
https://expressjs.com/en/4x/api.html#express
https://expressjs.com/en/4x/api.html#app.get.method

1.Cách express hoạt động

```
const express = require('express') //import express
const app = express() // tạo express application
const port = 3000  // init port

//khai báo routes
//req (request), res(response) là 2 object trong môi trường Node.js
app.get('/', (req, res) => {
  res.send('Hello World!')
})

//run server trên port đã khởi tạo trước đấy
//nạp các thông tin khai báo ở trên rồi chạy (ví dụ như nạp routes)
app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

2. Trải nghiệm của user (client)

- Vào đường link **http://localhost:3000/** hoặc **http://localhost:3000** đều trả ra kết quả 'hello world'
- Vào các link khác, ví dụ http://localhost:3000/abc => hiện thông báo Cannot GET /abc

Lưu ý: http://localhost:3000/ và http://localhost:3000 có tác dụng như nhau,
gọi là homepage (trang chủ),
trình duyệt tự động bỏ dấu '/', tương tự như URL, http://localhost và http://localhost:80 ...

- Quá trình website vận hành:
Trước khi client vào website, server đã được chạy lên, sẵn sàng để hoạt động.

Khi server được chạy lên thì:
- sẽ chạy trên 1 port. ví dụ ở trên là http://localhost:3000

- server biết 'nó có thể xử lý' các routes nào (URL) bằng cách nạp thông tin đầu vào

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})
```

=> đây chính là đoạn code nói cho server biết, khi người dùng vào route '/', thì nó cần gửi 'Hello World'

Khi client vào website, gõ http://localhost:3000 hoặc http://localhost:3000/

và nhấn Enter => tạo một request lên server, muốn server hiển thị nội dung ứng với route '/'

Server nhận được yêu cầu (req) của client, Lục tìm trong đống 'route' nó được khai báo

trả về kết quả với res (Hello World)

Trường hợp client gõ http://localhost:3000/abc , tức là yêu cầu nội dung ứng với route '/abc'
Server nhận yêu cầu và lục tìm trong đống 'route' nó được khai báo, không thấy có
=> trả ra lỗi 'Cannot GET /abc'

=> Sau này, ứng với 1 tính năng (screen - màn hình) của 1 website, sẽ xuất điểm với route để biết server sẽ làm gì

Vậy làm sao để khai báo thêm route với Express ?

#17. More routes

1. Khai báo thêm route

Để khai báo thêm route với Express, cách đơn giản nhất là code thêm vào :v

Tương tự như:

```
app.get('/abc', (req, res) => {
  res.send('ABC')
})
```

app.METHOD(PATH, HANDLER)

app ở đây là ứng dụng express

method: là HTTP request method, viết thường (sẽ đề cập tới sau)
method GET sẽ nói với Express cần trả ra nội dung cho client
Path: đường link (route) trên server
Handler: function để xử lý khi route được match

res.send là cách gửi nội dung về client
'ABC' là định dạng text (String)

Vậy có cách nào để gửi nội dung HTML, thay vì String ?

2. Trả ra dynamic content

```
res.send('<h1>some html</h1>');
```

Ưu điểm: Nội dung động
Nhược điểm:
- Không maintain được code
- Nếu template được định nghĩa content động.
Ví dụ, cùng là template gửi email, tuy nhiên, gửi cho A sẽ có tên A, gửi cho B sẽ có tên B

Để giải quyết vấn đề trên chúng ta sẽ cần view engine (template engine):v

#18. Template (View) Engine
template: tức là bộ khung định hình sẵn
ví dụ template wedding chẳng hạn, người bán hàng đưa sẵn template, bạn chỉ cần chọn và fill tên vào thôi :v

https://expressjs.com/en/guide/using-template-engines.html

1. Template Engine
Template Engine (View engine) giúp chúng ta có thể tạo 'static template files'.

At runtime. template engine sẽ biến đổi file view thành HTML để gửi cho client.

Express hỗ trợ các template engines nổi tiếng như : Pug, Mustache, EJS, Jade, Handlebars...

2. So sánh các loại template engine
List danh sách hỗ trợ: https://expressjs.com/en/resources/template-engines.html

So sánh nhanh:
https://strongloop.com/strongblog/compare-javascript-templates-jade-mustache-dust/

Lựa chọn EJS vì:
- Nếu bạn đã dùng .JSP (Java) hoặc .Blade (Laravel/PHP) thì syntax nó giống hệt :v

- Nhiều bạn sẽ bảo rằng, ngoài lý do trên, thì tại sao lại không chọn cái khác. Lý do vì:
Nếu để thiết kế 1 giao diện đẹp, ít code, dễ maintain thì không dùng node.js để render giao diện.
=> học React :v (1 công cụ chuyên về Frontend)

3. Cấu hình EJS cho project
Tài liệu:
https://github.com/mde/ejs
https://ejs.co/

- Cài đặt EJS
https://www.npmjs.com/package/ejs

npm install --save-exact ejs@3.1.8

https://expressjs.com/en/4x/api.html#app.set

views, the directory where the template files are located.
Eg: **app.set('views', './views')**. This defaults to the views directory in the application root directory.

view engine, the template engine to use. For example,
to use the EJS template engine:

app.set('view engine', 'ejs').

=> https://expressjs.com/en/4x/api.html#app.render

Chapter 3: Project Structure

#19. ENV (Environment Variables)

1. Tại sao cần env
- Tham số môi trường, là các tham số thiết lập chung. Thay vì 'hardcode', chúng ta sẽ cấu hình tập trung lại
Ví dụ:
Thông tin kết nối tới database gồm:
USERNAME và PASSWORD.

Nếu chọn giải pháp 'harcode', mỗi lần user update password, thì dự án có bao nhiêu chỗ dùng 'PASSWORD' sẽ cần cập nhật
bấy nhiêu nơi.

=> ENV tương tự như ENUM/CONSTANT, giúp định nghĩa tại 1 nơi, và sử dụng nhiều nơi.

2. File .env
- Là file dùng định nghĩa các tham số môi trường
- Cách tham số được khai báo theo định dạng : KEY=VALUE
tên KEY thông thường viết hoa (constant)
ví dụ: DB_USERNAME = hoidanit

- Lưu ý:
+ giá trị lưu trong file sẽ bị convert sang string
+ để thêm comment, sử dụng dấu #
+ không nên đẩy file .env lên public repos như Github... nếu như chứa thông tin nhạy cảm: username, password...

+ tạo thêm file .env.example cho trường hợp trên (nếu cần share .env)

3. Sử dụng với Node.JS
- Cài đặt package dotenv:
https://www.npmjs.com/package/dotenv
npm install --save-exact dotenv@16.0.3
- Create .env, .env.example
- Update .env, update .gitignore
- Update file app với .env

LƯU Ý: Mỗi lần update .env, cần chạy lại project để cập nhật.

#20. DevTool - Nodemon
Tài liệu: https://www.npmjs.com/package/nodemon

Nodemon là công cụ giúp ứng dụng node.js tự động 'restart' mỗi khi có file thay đổi (bị nodemon phát hiện :v)
Mặc định, nodemon sẽ quan sát các file sau: .js, .mjs, .coffee, .litcoffee

Muốn nhiều hơn, thì cần cấu hình nodemon :v
https://github.com/remy/nodemon#usage

1. Sử dụng với node.js
- Cài đặt:
npm install --save-dev nodemon@2.0.20

- update file package.json để chạy với nodemon, **thay node => nodemon**

- Lưu ý: khi thay đổi file .env, cần chạy lại nodemon

2. devDependencies

- thêm tham số: NODE_ENV=development

Tài liệu:
https://docs.npmjs.com/specifying-dependencies-and-devdependencies-in-a-package-json-file

"dependencies": Packages required by your application in production. (các package cần thiết để chạy production)

"devDependencies": Packages that are only needed for local development and testing. (các package chạy local dùng cho
developer, nếu chạy production thì vẫn 'ok', tuy nhiên có rủi ro về hiệu năng :D)

#21. Static files

Tài liệu:
https://expressjs.com/en/starter/static-files.html
https://expressjs.com/en/4x/api.html#express.static

Static files: Images, CSS files, JS files

```
const path = require('path')
app.use('/static', express.static(path.join(__dirname, 'public')))
```

Now, you can load the files that are in the public directory:

http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html

Download file images:
https://drive.google.com/file/d/1twpyRjFwSFYujiHxkM0X6_I9_RAGrUGe/view?usp=share_link

#22. Mô hình MVC

Mô hình MVC - MVC Design Pattern

Design Pattern là cách 'giải pháp thiết kế' để xử lý bài toán một cách hiệu quả nhất.

Với mô hình xây dựng website, mô hình MVC là cơ bản nhất, cũng như được sử dụng rộng rãi.

MVC là viết tắt của Model - View - Controller

MVC giúp viết code theo tổ chức, dễ dàng maintain và scale (mở rộng) ứng dụng.

1. View

View là kết quả nhìn thấy của client.
Với mô hình client - server, client gửi request lên server, server gửi response lại cho client,

view chính là kết quả người dùng nhận được.

2. Controller

Controller chính là nơi điều hướng dữ liệu (data).

Hình dung 1 cách đơn giản:

Bạn muốn mua 1 cốc cafe thì cần tìm 'đúng người bán cafe', không thể tìm người bán cơm, bán nước được...

Cafe (view) và người bán cafe (controller) giúp bạn đạt được kết quả mong muốn.

Tuy nhiên, nếu website chỉ có view và controller, thì khối lượng công việc dành cho controller rất lớn.
code sẽ nhiều và không thể maintain được

Người bán cafe sẽ kiêm các vị trí: phục vụ, pha chế, giao hàng, sao kê lợi nhuận...
=> cần thuê thêm người, và tập trung vào nhiệm vụ chính : quản lý nhà hàng và thuê nhân viên làm từng nhiệm vụ nhỏ

đấy là lý do Model ra đời, giúp giảm thiểu gánh nặng cho controller.

Trong công ty IT, thì controller chính là leader, break việc nhỏ ra cho model. Kết quả đạt được sẽ đưa cho Tester kiểm tra (view)

3. Model

Model (theo lập trình hướng đối tượng, giúp mô hình hóa 'code' thành các thành phần có thể 'tái sử dụng' và 'định nghĩa tiêu chuẩn để sử dụng data) trong ứng dụng.

Ở đây, với ứng dụng website, Model sẽ tạo hình thù của data lưu trữ (các đối tượng sử dụng website), đồng thời, sẽ chịu trách nhiệm truy vấn thông tin (CRUD)

Như vậy, bằng cách sử dụng Model, Controller sẽ 'san bớt việc' cho Model. giúp code 'tường minh hơn'

#23. Tổ chức các thư mục project

- Tạo controllers
- Tạo services
- Tạo routes

...

#24. Áp dụng mô hình MVC với Node.js (Part 1)

- Chia code theo mô hình:

View Engine

Routes

#25. Áp dụng mô hình MVC với Node.js (Part 2)

- Chia code theo mô hình:

Controller

Chapter 4: Setup Docker & Database SQL

#26. Why Docker ?

1.Mình biết đến Docker như thế nào ?

- Học ở trường

- **Chuyện đi làm**

2.Docker có tác dụng gì ?

Build , **Ship then Run**

#27. Sử dụng Docker

1.Sử dụng Docker

- Download Docker: https://docs.docker.com/desktop/install/windows-install/

2. Tài liệu học Docker

Khóa học docker siêu cơ bản:
https://www.youtube.com/playlist?list=PLncHg6Kn2JT4kLKJ_7uy0x4AdNrCHbe0n

#28. Docker Hub

Tương tự như github, docker hub là nơi lưu trữ & chia sẻ image docker (free)

Chúng ta cần docker hub vì:

- Có nơi lưu trữ images docker (nếu muốn chia sẻ cho người khác dùng)
- Lưu trữ free

#29. Relational Database

Relational Database : Cơ sở dữ liệu quan hệ (SQL), có thể kể tới: MySQL, SQL Server, Postgres, Oracle...

1. DBeaver

Download: https://dbeaver.io/download/

DBeaver (bản Community Free) giúp kết nối tới các hệ cơ sở dữ liệu, all-in-one

2. Kết nối database

- Tạo database MySQL với docker

B1: Đảm bảo rằng máy tính đã chạy docker (với Windows, MacOs thì chạy sẵn Docker Desktop)

B2: Tạo database với Docker Compose

- Đảm bảo rằng, ví dụ ổ C của máy tính, còn đủ tài nguyên (dung lượng) để lưu trữ.

Download file cài đặt:

https://drive.google.com/file/d/1DfSjZTM8RYXbKCde5gRg_HYI3YJWNtGa/view?usp=share_link

Chạy câu lệnh:

docker compose -f mysql.yml -p nodejs-sql up -d

-f: file name

-p: project name

-d : detach, run as background

Bạn nào dùng MacOS, chạy câu lệnh trên bị lỗi, thì xem mục 5 để biết cách khắc phục

B3: Tạo connection trong DBeaver

Lưu ý:

PORT: 3307

Username/password: root/123456

DBeaver không phải là database, nó là phần mềm giúp truy cập và sử dụng database. Database được tạo bên trong Docker :v

4. Lưu ý khi sử dụng

- Sau này, mỗi lần cần sử dụng database MySQL, đảm bảo rằng 'đã chạy docker trước'. Vì không có docker, thì cũng không có database

- Không cần 'update docker' để hạn chế mất dữ liệu => Guide hướng dẫn :v

5. Chạy MySQL với MacOS

Link fix lỗi Docker (Apple Silicon/M1 Preview) MySQL "no matching manifest for linux/arm64/v8 in the manifest list entries"
https://stackoverflow.com/questions/65456814/docker-apple-silicon-m1-preview-mysql-no-matching-manifest-for-linux-arm64-v8

Link download (dành cho MacOS):
https://drive.google.com/file/d/1aVHfvBCKnIw2N-65Yps405iwNwxe0DO-/view?usp=sharing

#30. Sử Dụng MySQL với Node.JS

NPM có 2 package là: mysql và mysql2
https://stackoverflow.com/questions/25344661/what-is-the-difference-between-mysql-mysql2-considering-nodejs

1.Cài đặt
https://www.npmjs.com/package/mysql2

npm install --save-exact mysql2@2.3.3

2. Tạo fake data cho database
Create table: https://www.w3schools.com/sql/sql_create_table.asp
https://www.w3schools.com/sql/sql_autoincrement.asp

Insert row: https://www.w3schools.com/sql/sql_insert.asp

```
CREATE TABLE Users (
    id int NOT NULL AUTO_INCREMENT,
    email varchar(255),
    name varchar(255),
    city varchar(255)
);

INSERT INTO Users
VALUES (1, 'eric@gmail.com', 'hoidanit', 'hanoi');
```

3. Kết nối tới database
- Lưu ý: Cần chạy Docker để có database
https://github.com/sidorares/node-mysql2#using-prepared-statements

```
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  database: 'test',
  port: 3307,
  password: '123456
});
```
=> đưa các tham số database vào file .env

#31. Tái Sử Dụng Connection

1.Chia tách file connections

- Việc chia tách file connections, **giúp chúng ta không phải code lại 'đống shit' nhiều lần**, đúng theo nguyên tắc DRY (don't repeat yourself)

- Tạo 1 file để kết nối tới database, sau này, mỗi lần cần kết nối database, thì chỉ cần import và sử dụng

2. File kết nối

Lưu ý: các tham số kết nối tới database, không HARDCODE, thay vì vậy, lấy từ file .env.

Sau này, thay đổi thông tin kết nối tới database, thì chỉ cần update file .env, mà không cần phải sửa code

Với cách làm này, chúng ta đã tái sử dụng được code, tuy nhiên, nó có đảm bảo được hiệu năng sẽ cao, hay hiểu đơn giản là, ứng dụng đã 'đủ tốt' ?

#32. Connection Cost - Tạo Mới Connection ?

Các vấn đề gặp phải với cách làm hiện tại:

1. A new connection every time

Với code hiện tại, mỗi 1 lần chúng ta cần lấy dữ liệu dưới database, thì cần tạo mới 1 connection.

Bạn cần lấy thông tin user => kết nối tới db, table users và thực hiện query => trả ra kết quả
Lấy thông tin products => kết nối db, table product….. => trả ra kết quả

Viễn cảnh:
Website (source code backend) đặt tại việt nam, database đặt tại USA. Máy bay cũng có delay, ví dụ thời tiết, chính trị…

Thì kết nối tới database (chẳng may cá mập cắn cáp) cũng có delay. Việc phải 'chạy đi chạy lại' giữa Việt Nam và USA 'nó tốn thời gian' và 'không có tính tái sử dụng'

Thực tế là: do chúng ta chạy database với docker và môi trường đang test là localhost (máy tính chúng ta), nó là perfect, nên chưa thấy nhược điểm:D

2. Connection never close
Khi chúng ta truy vấn dữ liệu, các công việc cần làm là:
Khởi tạo kết nối (fill vào user/password…), sau đấy thực thi query (truy vấn dữ liệu)

Mô hình khởi tạo => thực thi

Nếu chúng ta thực thi 10 câu query data, thì sẽ cần tạo 10 connections.

Chúng ta có 1000 users (1000 máy tính khác nhau), tại 1 thời điểm, thực hiện 1 câu query => 1000 connections

Như vậy, chưa cần biết nó có 'ảnh hưởng' như thế nào, tuy nhiên, **connection 'không đóng'** nó sẽ tiềm ẩn rủi ro.

Cách giải quyết đơn giản nhất, là **'đóng' connection sau mỗi lần sử dụng**, bằng cách sử dụng các hàm có sẵn của thư viện, như là close, destroy…

3. Database Rate Limit

Database, suy cho cùng vẫn là '1 phần mềm' chạy trên máy tính.

Đã là phần mềm, thì nó 'tiêu tốn' tài nguyên. Thành ra, cho dù 'database' không có cơ chế 'giới hạn' tài nguyên sử dụng, thì hệ điều hành máy tính có cơ chế để 'ép buộc/giới hạn' mức độ sử dụng tài nguyên của 1 phần mềm.

Thực tế là database nào cũng có cơ chế 'rate' limit, tức là 'giới hạn chịu đựng' của nó.

Ở đây, chúng ta cần quan tâm tới 'mức độ chịu đựng' của database vì:

1 connection tới database, thực hiện query data, là đã khiến 'database phải hoạt động'
=> tốn tài nguyên (RAM/CPU...)

10 connections => tài nguyên x10
1000 => x1000

Sẽ tới 1 lúc đạt tới mức 'giới hạn' của database => database tự động sử dụng cơ chế 'auto restart'.

Nếu không restart, thì nó 'đứng im bất động', không biết nên làm gì :v

Vậy làm sao để tránh 3 nhược điểm nêu trên, đấy là lúc chúng ta cần quan tâm tới 'cách thiết kế kết nối tới database' như thế nào cho hiệu quả ?

#33. Connection Pool Pattern

Update code theo Pool

https://github.com/sidorares/node-mysql2#using-connection-pools

Todo...

#34. Test Performance Query Database

Download file test:

https://drive.google.com/file/d/1gFVxpGxHuK-gs1WIgZ1lil_XHab6c3xt/view?usp=sharing

Chapter 5: CRUD với Node.JS
#35. Design NavBar

Link navbar: https://www.w3schools.com/howto/howto_js_topnav.asp

#36. Design Form Add New User

HTML Legend tag: https://www.w3schools.com/tags/tag_legend.ASP

#37. Express và Req.body

1.Cấu hình

Để lấy được data từ client gửi lên server (request), chúng ta cần sử dụng thư viện để làm điều đấy.

Thêm vào file server.js :

app.use(express.json( )); // Used to parse JSON bodies
app.use(express.urlencoded( )); //Parse URL-encoded bodies

Tài liệu:
https://stackoverflow.com/questions/9304888/how-to-get-data-passed-from-a-form-in-express-node-js

https://expressjs.com/en/api.html#express.urlencoded

https://expressjs.com/en/api.html#express.json

2. Lưu ý khi sử dụng form HTML

Để server có thể lấy được data từ form gửi lên server, thông qua **req.body,** chúng ta phải dùng thuộc tính (attribute) **là name, không phải là id** cho phần tử HTML muốn lấy giá trị.

Các phần tử của HTML trong form dùng để lấy giá trị bao gồm: input (text, checkbox), select...

Sự khác biệt giữa sử dụng **id và name** khi sử dụng với form
https://stackoverflow.com/questions/1397592/difference-between-id-and-name-attributes-in-html

#38. Chức Năng Create User
Câu lệnh SQL Insert:
https://www.w3schools.com/sql/sql_insert.asp

Query với giá trị được truyền động, ví dụ:

```
connection.query(
 'SELECT * FROM `Users` WHERE `name` = ? AND `age` > ?',
 ['Hỏi Dân IT' , 45],
 function(err, results) {
   console.log(results);
 }
)
```

Trong ví dụ trên, **2 biến name và age** được truyền động, giá trị của nó được truyền ngay phía sau câu lệnh query, sử dụng 1 array. Để sử dụng giá trị truyền động, chúng ta **dùng dấu ?**

Array giá trị của các phần tử truyền động:
['Hỏi Dân IT' , 45] => thứ tự của array chính là thứ tự của các biến truyền động

Ở đây: biến **name = 'Hỏi Dân IT'**
Biến **age = 45;**

#39. Design List Users

- Tạo routes get list all user (navigate). Sửa lại giao diện. Khi vào giao diện sẽ hiển thị list user , còn create user là 1 chức năng
- Hardcode table list user, có 2 nút edit và delete

File table: https://www.w3schools.com/css/tryit.asp?filename=trycss_table_hover

Backup (nếu link trên không dùng được):

Phần CSS:

```
<style>
table {
  border-collapse: collapse;
  width: 100%;
}
th, td {
  padding: 8px;
  text-align: left;
  border-bottom: 1px solid #ddd;
}
tr:hover {background-color: coral;}
</style>
```

Phần Code HTML Table:

```
<table>
  <tr>
    <th>First Name</th>
    <th>Last Name</th>
    <th>Points</th>
  </tr>
  <tr>
    <td>Peter</td>
    <td>Griffin</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>Lois</td>
    <td>Griffin</td>
    <td>$150</td>
  </tr>
</table>
```

- Update connection với promise:

```
// update get the client
const mysql = require('mysql2/promise');

const [rows, fields] = await connection.execute('SELECT * FROM `table`);
```

Ở đây, vế bên phải (await connection.execute('SELECT * FROM `table`)) trả ra 1 array gồm 2 phần tử, vì vậy, việc viết:
const [rows, fields] có nghĩa là: rows là phần tử đầu tiên, fields là phần tử thứ 2 của array kết quả.

Cú pháp viết như trên gọi là destructing assignment (mình đã đề cập trong khóa react ultimate), chi tiết tham khảo link sau:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Ví dụ:
const [a, b] = [10, 20];

Áp dụng với bài toán của chúng ta;
const [rows, fields] = await connection.execute('SELECT * FROM `table`);

Giả dụ: await connection.execute('SELECT * FROM `table`) = [10, 20]

Thì const [rows, fields] = [10, 20]

=> rows = 10 ; (phần tử đầu tiên của mảng)
Fields = 20 ; (phần tử thứ 2 của mảng)

#40. Query List Users

- Viết câu lệnh query list user
- Tách code sang services, đúng theo mvc
  https://ejs.co/#promo

- Truyền data từ controller sang view:
  https://stackoverflow.com/questions/65977136/how-to-pass-data-from-express-to-ejs

  Thông thường, để trả ra 1 view, tại controller chúng ta sử dụng:
  res.render("my-view.ejs")

  Để truyền data từ controller sang view, sử dụng thêm tham số trong hàm ở trên:

  res.render("my-view.ejs", { data-pass-here }

  Data được truyền sang view sẽ được viết dưới dạng object, x <- y

- Để kiểm tra dữ liệu (object/array…) bên file view, chúng ta sẽ convert sang dạng string.
  Tài liệu tham khảo:
  https://www.w3schools.com/js/js_json_stringify.asp

  Bằng cách sử dụng J**SON.stringify( ), chúng ta có thể convert biến sang dạng String.**

#41. Display List Users

Sử dụng ejs loop (vòng lặp): lưu ý từng dấu đóng mở ngoặc

Cách dùng vòng lặp truyền thống, ví dụ vòng lặp for:
https://stackoverflow.com/questions/38179077/what-is-the-proper-way-to-loop-through-an-array-in-an-ejs-template-after-an-ajax

```
<% for (let i = 0; i < reports.length; i++) { %>
  <li >
    <div >
      <span>
       <%= report[ i ].type %>
      </span>

      <span>
       <%= report[ i ].dateTime %>
      </span>
    </div>
  </li>
<% } %>.
```

Với cách làm này, **chúng ta cần biến i để biết đang truy cập tới object nào**

Cách dùng vòng lặp theo object, ví dụ forEach:
https://stackoverflow.com/questions/49264980/how-do-you-do-a-for-loop-for-each-in-ejs

```
<% students.forEach(function(student) { %>
  <li>
    Name:    <%= student.name %>
  </li>
<% }); %>
```

Sử dụng vòng lặp forEach thì không cần quan tâm tới chỉ số của mảng như cách làm truyền thống (với vòng lặp for)

#42. Design View Edit User

- Design view html for edit
- Sử dụng <a/> tag => method get

#43. Route params

Tài liệu:
https://expressjs.com/en/guide/routing.html

Ở đây, chúng ta cần truyền động tham số trên URL, mục đích để lấy được thông tin tương ứng (với user) để hiển thị.

Ví dụ, khi khai báo
Route path: /users/**:userId**
thì userId chính là một tham số "truyền động". Nhờ có dấu 2 chấm trước tên biến, Express sẽ biết được chúng ta đang muốn "truyền động" tham số có tên userId.

Khi người dùng vào
Request URL: http://localhost:3000/users/34
thì Express sẽ hiểu là số 34 là 1 tham số truyền động, và nếu chúng ta sử dụng cú pháp:

req.params, console.log sẽ trả ra kết quả: { "userId": "34" }

Như vậy, để khai báo động 1 tham số trên đường link URL thì:

- Khai báo route như thông thường, tuy nhiên, khai kèm theo tham số, bắt đầu bằng dấu hai chấm, ví dụ: /hoidanit/**:name**
  (name chính là tham số truyền động)
- Để lấy ra giá trị của tham số truyền động trên đường link URL, sử dụng:
  req.params

#44. Get User By Id

- Query user by id: sử dụng với điều kiện Where

- Fill data to form update: để có thể điền thông tin người dùng (lấy theo ID) vào form, sử dụng **thuộc tính value của tag input**

#45. Update A User

Để cập nhật người dùng, chúng ta cần biết ID của người dùng.

Tuy nhiên, tham số này chúng ta cần, nhưng người dùng không cần.
Vì vậy, với backend, có 1 mẹo (trick) hay dùng, đấy là tạo 1 thẻ input, và ẩn nó đi với CSS.

Câu lệnh SQL Update: https://www.w3schools.com/sql/sql_update.asp

Để chuyển trang, sử dụng:

res.render('route-to-render')

Tài liệu: https://expressjs.com/en/api.html#res.redirect

#46. Delete Form Delete User
Design form delete User

#47. Delete User By Id
Câu lệnh SQL Delete: https://www.w3schools.com/sql/sql_delete.asp

#48. Cải Thiện Trải Nghiệm Giao Diện với Javascript

Tài liệu:
https://drive.google.com/drive/folders/1zsj2kRKWoeVEKnMsjtqJwD_z0jb5aK3-?usp=share_link

1.Trước khi xóa user, hiển thị popup confirm ?

Thông thường, việc phải điều hướng qua lại 'nhiều trang' gây cảm giác khó chịu cho người dùng, vì vậy, đối với các chức năng Create/Edit/Delete, chúng ta thưởng hiển thị pop-up ngay tại trang người dùng thao tác.

Ở đây, khi người dùng cần xóa 1 user, chúng ta chỉ cần hiển thị popup xác nhận. Nếu người dùng confirm là có (yes), thì chúng ta xóa luôn.

Cách làm này có 2 ưu điểm:
- Tăng sự trải nghiệm của người dùng (không cần phải điều hướng qua lại nhiều trang)
- Ở phía backend, không cần phải tạo route mới (ứng với giao diện confirm xóa)

Tuy nhiên, cách làm này có 1 nhược điểm, đấy chính là: Nếu không Refresh (loát lại trang), thì làm sao có thể gọi lên server , tức là gọi vào hàm xử lý (handler) của 1 route.

Bấy lâu nay, chúng ta đang đi theo nguyên tắc. User vào 1 route => gọi lên server, server sẽ tìm xem xem là ứng với router đấy, controller nào sẽ chịu trách nhiệm xử lý.

Bài toán chúng ta gặp phải là, user đã vào trang rồi, sau đấy thao tác thêm/sửa/xóa, thì làm sao có thể gọi lên controller (khi không refresh trang).

AJAX sinh ra để giải quyết vấn đề trên.

2.Sử dụng AJAX để gọi apis

AJAX = Asynchronous JavaScript And XML

Nó là công cụ, giúp các bạn không cần load lại trang, nhưng vẫn gọi được lên server.

Ở đây, chúng ta sử dụng JQuery để làm điều đấy.

Nhờ có công cụ này, trải nghiệm của người dùng sẽ được tăng lên.

#49. Ưu Nhược Điểm Của Cách Làm Hiện Tại

1. SSR - Server Side Rendering

SSR là cách tạo ra giao diện website bằng server.

Tức là, theo mô hình client - server, ứng với request của client gửi lên, server sẽ nhìn theo route (link website), sau đấy render ra giao diện tương ứng.

Ví dụ, vào /users => server sẽ render ra giao diện ứng với route /users

Client chỉ làm 1 nhiệm là gửi yêu cầu, và nhận kết quả.
Phần xử lý để tạo ra kết quả => server làm.

Như vậy, hiểu đơn giản, phần 'hard work' là server chịu trách nhiệm.

Ưu, nhược điểm của SSR:

- Ưu điểm: client dùng sẽ sướng, vì tốn ít tài nguyên (RAM/CPU) của client.

  Ví dụ, 1 người dùng máy tính yếu sử dụng website vẫn rất mượt mà, vì máy tính client chỉ hiển thị kết quả, không cần phải tính toán xử lý gì cả.

- Nhược điểm: nếu website có nhiều người dùng => server bị quá tải, vì với từng client, nó cần thời gian để tính toán để trả ra kết quả => website bị chậm, giật lag

Trong thực tế, chúng ta có 1 cách làm nữa, là CSR (client side rendering), giúp giảm thiểu gánh nặng cho server.
Hiểu đơn giản là nó khắc phục nhược điểm của SSR ở trên, đồng thời, vẫn đảm bảo được phần nào ưu điểm đã có sẵn của SSR.

Phần CSR, mình sẽ đề cập chi tiết khi các bạn học lộ trình Frontend của mình (với React), và mình cũng sẽ ôn tập lại SSR tại đấy.

2.Kiểm soát version của database

- Cách tạo database, table ?
- Cách sử đổi database, ví dụ cần update/delete/create thêm fields ?

- Làm sao có thể quản lý version, ví dụ, hôm nay code lỗi, muốn quay lại quá khứ ?
- Tạo dữ liệu fake cho database ?

...

#50. Học Gì Tiếp Theo với SQL ?

1.Sử dụng SQL với Node.JS

Bên cạnh việc sử dụng SQL raw (tức là viết câu lệnh sql thuần túy), các bạn có thể tìm hiểu và sử dụng ORM (object relational mapping)

Use SQL với Node.js
- suggest sử dụng với sequelize

2. Luyện tập project

Việc luyện tập project, giúp các bạn có thêm kiến thức, và củng cố những điều đã học.

Khóa học Fullstack SERN:
https://www.youtube.com/playlist?list=PLncHg6Kn2JT6E38Z3kit9Hnif1xC_9VqI

Chapter 6: NoSQL - MongoDB
#51. Lịch Sử Ra Đời của Database
1.Gắn liền với Internet
Sự phát triển của database, gắn liền với sự phát triển của Internet.

Tài liệu: https://en.wikipedia.org/wiki/Relational_database

Relational database management systems (RDBMS), được hiểu là cơ sở dữ liệu quan hệ, được sử dụng với SQL.

RDBMS được phát triển từ năm 1970, giúp lưu trữ dữ liệu website và các ứng dụng của business (ví dụ như app desktop chẳng hạn).

Tên gọi cũng đã nói lên 1 phần, đấy là 'relational'. Các model cần có mối quan hệ với nhau. Data được lưu trữ trong database, thông qua table, fields, rows…

Mục tiêu của RDBMS : "one size fits all"

Thuật ngữ 'NoSQL' lần đầu tiên được sử dụng vào năm 1998, với sự bắt đầu bùng nổ của web 2.0

Nếu như giải quyết được tất cả các bài toán trong thực tế, thì đã không có khái niệm NoSQL.

Ví dụ Facebook:
2004 : 0 users
2008: 100 M
4/2009: 200 M
9/2009: 300 M
5/2010: 400 M
….
12/2016: 1.8 B

2. Các loại DB Engines của RDB
Oracle Database
MySQL
Microsoft SQL Server
PostgreSQL
IBM Db2
SQLite
MariaDB

#52. What is NoSQL ?

1. Định nghĩa

Tài liệu: https://hostingdata.co.uk/nosql-database/

Thông thường: NoSQL = not only sql

Ý tưởng của NoSQl, không phải để thay thế SQL, mà là cung cấp giải pháp để giải quyết vấn đề tốt hơn, so với hệ cơ sở dữ liệu quan hệ.

The idea with NoSQL isn't so much to replace SQL as it is to provide a solution for problems that aren't solved well with traditional RDBMS (Relational Database Management System)

Thực tế:

NOSQL DEFINITION: Next Generation Database Management Systems mostly addressing some of the points:

being non-relational, distributed, open-source and horizontally scalable.

(không bị ràng buộc bởi quan hệ, phân tán, mã nguồn mở và có thể mở rộng theo chiều ngang)

====

horizontal-vs-vertical-scaling

Khi bạn tăng cân, quần áo mặc nó không đủ vừa vặn để dùng, thì:

horizontal scaling (chiều ngang), tức là mua thêm quần áo về mặc

vertical scaling (chiều dọc), tức là sửa đổi quần áo đang có.

---

với database:

horizontal scaling: tức là mua thêm server (có thể cùng cấu hình) với server hiện tại để chia sẻ gánh nặng => phân tán data

vertical scaling : mua thêm tài nguyên cho server hiện tại => tập trung data

https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling

==========

2. Danh sách NoSQL Database

Tài liệu: https://hostingdata.co.uk/nosql-database/

Hiện có > 225 loại NoSQL Database

Một số loại hay dùng:

Loại Document Store:

- Elastic (https://www.elastic.co/)

- MongoDB (https://www.mongodb.com/)

Loại Key Value / Tuple Store:
- Redis (https://redis.io/)

Loại Graph Database Management Systems
- Neo4J (https://neo4j.com/)
....

3. Trending
https://www.christof-strauch.de/nosqldbs.pdf

https://www.ccs.neu.edu/home/kathleen/classes/cs3200/20-NoSQLMongoDB.pdf

https://blog.nahurst.com/visual-guide-to-nosql-systems

Link tài liệu backup:
https://drive.google.com/file/d/1Ftuz_dhWA2TIolTpvThMCNAnaMZqgg1f/view?usp=share_link

1. The continuous growth of data volumes (to be stored)
2. The growing need to process larger amounts of data in shorter time

Không có khái niệm "one size fits all".
RDB: read than write

The Flexibility: có những loại data ko phù hợp với RDBMS => more flexible

The performance: (FEE) khi for upgrade (large data)

Scaling không đồng nghĩa với fast (performance) => đồng nghĩa với size (data)

#53. Why MongoDB ?
Tài liệu: https://www.mongodb.com/

1. Định nghĩa

NoSQL: being non-relational, distributed, open-source and horizontally scalable.

MongoDB là một loại NoSQL database (type document), lưu trữ dữ liệu dưới dạng BSON format. (based on JSON - B stands for binary)

MongoDB được phát triển từ 2008, với sự bùng nổ của web 2.0 (điển hình là các mạng xã hội)

2. Tại sao dùng MongoDB

MongoDB là 1 loại NoSQL, và **NoSQL thiết kế không phải để giải quyết tất cả bài toán có trong thực tế.**

Vì vậy, khi dùng MongoDB cần phân tích để lấy được điểm mạnh của mongodb.

- Dữ liệu mềm dẻo (flexible):
Nếu dữ liệu của bạn nó 'không cố định định dạng', MongoDB là lựa chọn hoàn hảo.

Bạn có thể thay đổi dữ liệu bất cứ khi nào, mà không ảnh hưởng tới dữ liệu cũ (Tính toàn vẹn dữ liệu của RDB)

Ví dụ:
Sử dụng RDB, với fields phoneNumber => bạn cần gán type, có thể là number hoặc string. Tại 1 thời điểm, chỉ 1 type duy nhất
phoneNumber = 19001000;
phoneNumber = "+84123456";

với MongoDB:
{phoneNumber: 19001000 }

{phoneNumber: "+84123456" }

```
{phoneNumber: {
        home: 19001000,
        work: "+84123456"
}}
```

-Tính mở rộng

3.Lưu ý khi dùng MongoDB

Use the right tool for the job

MongoDB thiết kế không phải để giải quyết tất cả bài toán, mà chỉ giải quyết một vài bài toán trong thực tế.

Tài liệu tham khảo:
http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/

Không nên dùng MongoDB khi:

- Dữ liệu có mối quan hệ với nhau chặt chẽ (relationship)
- Thực hiện nhiều join, query phức tạp để lấy data.
- Dữ liệu (data) nhỏ

Nêu dùng MongoDB khi:

- Chưa hình dung ra được Model xây dựng, dữ liệu lưu như thế nào cũng được
- Dữ liệu data lớn (cần mở rộng theo chiều ngang - vertical)
- Cần xây dựng nhanh ứng dụng, mà không cần quan tâm tới hiệu năng

#54. What is Mongoose ?
Tài liệu: https://mongoosejs.com/
1.Mongoose là gì
Mongoose là 1 thư viện của Node.JS, giúp xử lý và thao tác với MongoDB thông qua ODM.

ODM: Object Data (Document) Mapping

Hiểu đơn giản, thay vì đọc tài liệu của MongoDB, chúng ta đọc tài liệu của Mongoose, sau đấy làm theo.

So sánh với Mongodb:
https://www.mongodb.com/developer/languages/javascript/mongoose-versus-nodejs-driver/

Lý do sử dụng Mongoose:

Chỉ với 1 lý do duy nhất, nó là ODM.

Tương tự khi thao tác với SQL, chúng ta dùng ORM (Object Relational Mapping)
(với Node.JS, mình có giới thiệu sử dụng Sequelize)

Cái lợi ích của ODM:

Validate (Process) dữ liệu trước khi lưu xuống database Mongo. Tức là, tạo khái niệm Schema (Model)

2. Cài đặt Mongoose

Lưu ý: *không phải cài đặt thư viện này*: https://www.npmjs.com/package/mongodb

Mà cài đặt thư viện sau:
https://www.npmjs.com/package/mongoose

Câu lệnh cài đặt: npm install --save-exact mongoose@6.7.2

#55. Cài Đặt MongoDB Compass

Link download google drive(version giống video):
https://drive.google.com/file/d/1aVZVj1pY83wdVbn327w1bjOWsCB-XF5a/view?usp=share_link

Version cũ hơn:
https://github.com/mongodb-js/compass/releases

Link download từ trang chủ: https://www.mongodb.com/try/download/compass

#56. Cài Đặt MongoDB với Docker

Download file cài đặt:
https://drive.google.com/file/d/1QAug_XP8AOv_R7GkYd9hWmRGf4GSKIX2/view?usp=share_link

Sử dụng câu lệnh sau: **docker compose -p nodejs-mongdb up -d**

Check kết nối tới database: **mongodb://root:123456@localhost:27018**

#57. Create Connection

Lưu ý: dùng git checkout sang nhánh khác

Tài liệu: https://mongoosejs.com/docs/connections.html

Handle lỗi khi kết nối: https://mongoosejs.com/docs/connections.html#error-handling

```
mongoose.connect('mongodb://localhost:27017/test').
  catch(error => handleError(error));

// Or:
try {
  await mongoose.connect('mongodb://localhost:27017/test');
} catch (error) {
  handleError(error);
}
```

URL kết nối cho Mongoose trong khóa học:
mongodb://root:123456@localhost:27018

Check trạng thái đã kết nối hay chưa:
https://stackoverflow.com/a/70723327

Code backup:

```
 const dbState = [
{ value: 0,  label: "disconnected" },
{ value: 1, label: "connected" },
{value: 2,label: "connecting"},
{value: 3, label: "disconnecting"}
];

const state = Number(mongoose.connection.readyState);
console.log(dbState.find(f => f.value === state).label, "to db"); // connected to db
```

#58. Connection Options

Đẩy các tham số kết nối vào file .env

Tài liệu: https://mongoosejs.com/docs/connections.html#options

Sửa code để cho kết nối tới database trước khi chạy server
https://stackoverflow.com/questions/592396/what-is-the-purpose-of-a-self-executing-function-in-javascript

Ở đây, chúng ta sử dụng 1 công cụ là 'self running' function.

Cú pháp sử dụng:

```
;(( ) => {
//code here
}) ( )
```

Lưu ý: cần dấu chấm phẩy (;) semicolon trước self-running function để nó chạy chính xác.
Viết SAI:

```
app.use( '/', appRoute) // không có dấu (; ) kết thúc
(( ) => {
//code here => gây ra bugs
}) ( )
```

Viết ĐÚNG:

```
app.use( '/', appRoute); // có dấu (; ) kết thúc
(( ) => {
//code here => gây ra bugs
}) ( )
```

Về loại functions này, mình đã giải thích trong khóa React Advance, khi đề cập về 'arrow function' cũng như 'anonymous functions' với javascript

Ở đây, chúng ta dùng như trên để cho ngắn gọn, cũng như có thể sử dụng được async await.
Với self function, chúng ta không cần gọi tới nó, nó sẽ tự động được thực thi.

#59. Create Database

Tạo Schema và Model

Tài liệu:
https://mongoosejs.com/docs/index.html
https://mongoosejs.com/docs/guide.html

Code Backup:

```
const mongoose = require("mongoose");
const kittySchema = new mongoose.Schema({
        name: String
})
const Kitten = mongoose.model("Kitten", kittySchema);
const cat = new Kitten({ name: 'Silence' });
cat.save();
```

Khi kết nối tới MongoDB, mà không quy định tên của database => mongoose tự động tạo database tên là 'test'.

Các khái niệm ánh xạ giữa MongoDB và SQL:

database - database
collection - table
document - row

#60. Create Schema & Model

Phân biệt Schema và Model

Tài liệu:
https://stackoverflow.com/a/22950402

Schema: quy định 'hình thù' data sẽ lưu vào database.
Khi định nghĩa Schema, chúng ta không cần định nghĩa trường id, thư viện sẽ tự sinh cho chúng ta, **dưới dạng _id**

_id này được tự sinh, là 1 chuỗi uuid (unique - độc nhất) không lo bị trùng :D

Trong thực tế, ngoài việc quy định 'hình thù' cho data được lưu, chúng ta còn định nghĩa các method sẽ có cho Model (ngoài các method đã có sẵn), cũng như validate dữ liệu tại đây.

Model: là công cụ giúp thao tác với database, ví dụ như Query (tìm kiếm), Create (thêm), Update (sửa), Delete (xóa)...

Model đại diện cho document (bản ghi được lưu trữ trong database). Chúng ta dùng code (thông qua Model) để có thể thay đổi database, chứ không sửa đổi trực tiếp database

#61. Create A User

Phân biệt Save và Create với Mongoose:
Tài liệu:
https://stackoverflow.com/questions/38290684/mongoose-save-vs-insert-vs-create

Chúng ta có thể dụng hàm save( ) hoặc hàm create ( ) để tạo mới 1 document với MongoDB

```
const Tour = require('../models/tourModel');

 // method 1
 const newTour = await Tour.create(req.body);
```

Với hàm create( ), chúng ta thao tác trực tiếp thông qua Model.

```
 // method 2
 const newTour = new Tour(req.body);
 await newTour.save();
```

Với hàm save( ), chúng ta cần tạo mới 1 đối tượng (bản sao - instance) của Model, sau đấy sử dụng (thao tác gián tiếp)

#62. Display List Users
Tài liệu:
https://mongoosejs.com/docs/queries.html

https://mongoosejs.com/docs/models.html#querying

Để có thể hiển thị dữ liệu, chúng ta cần sử dụng công cụ 'query' của mongoose.

Chúng ta sử dụng hàm find ( ) để query tất cả users:
https://mongoosejs.com/docs/api.html#model_Model-find

Cú pháp sử dụng : **Model.find({ condition })**

Khi sử dụng **Model.find({ }) //truyền vào object rỗng,** tức là tìm tất cả và không bị ràng buộc bởi điều kiện

Ví dụ khi tìm kiếm có điều kiện:

Model.find({ name: 'hoidanit' }) // tìm tất cả documents có name là 'hoidanit'

#63. Update A User
Tài liệu:
https://mongoosejs.com/docs/api.html#model_Model-updateOne

Cú pháp sử dụng:
await Model.updateOne({ condition }, { data-update });

Tác dụng của hàm exec:
https://stackoverflow.com/questions/31549857/mongoose-what-does-the-exec-function-do

Với hàm Exec( ), kết quả trả về là promise => có tác dụng nếu có lỗi xảy ra, thông tin lỗi chi tiết hơn.

Khi code, mà không có lỗi, thì dùng exec( ) hay không, thì kết quả vẫn đúng. Amazing :v

#64. Delete A User
Tài liệu:
https://stackoverflow.com/questions/42715591/mongodb-difference-remove-vs-findoneanddelete-vs-deleteone

Hàm remove không được dùng nữa ~DEPRECATED~
https://mongoosejs.com/docs/api/query.html#query_Query-remove

Thay vào đó, sử dụng hàm delete
https://mongoosejs.com/docs/api/query.html#query_Query-deleteOne

Chapter 7: RESTful APIs

#65. Setup Postman

Postman là công cụ giúp chúng ta có thể test nhanh sự hoạt động của 1 APIs.

Với Frontend, sử dụng Postman để test APIs backend gửi cho.
Với Backend, sử dụng Postman để test APIs trước khi gửi cho Frontend (và sử dụng trong quá trình làm việc khi viết APIs)

Postman có thể sử dụng online với browser, hoặc cài đặt trên máy tính. Tuy nhiên, nên cài đặt và sử dụng trên máy tính để trải nghiệm tất cả tính năng 1 cách tốt nhất.

Link download: https://www.postman.com/downloads/

Link video hướng dẫn: (xem từ phút 13). Các khái niệm về JSON, APIs... mình sẽ giải thích ở trong các video kế tiếp của khóa học này.

https://www.youtube.com/watch?v=L3Z4KfyCZOI&list=PLncHg6Kn2JT6E38Z3kit9Hnif1xC_9VqI&index=38

Lưu ý: Postman là một công cụ rất basic, và nhà tuyển dụng mặc định quan niệm bạn biết cách sử dụng nó (tương tự như GIT)

#66. Setup dự án Frontend (To do with docker)

Video này cancel, nên sẽ bỏ qua video này các bạn nhé. Không ảnh gì tới khóa học, cứ xem tiếp video #67

#67. Vai Trò của Web Server

Khi xây dựng website, chúng ta có mô hình client -> server.

Ở đây, và cụ thể là khóa học này, chúng ta đang học cách xây dựng server (với node.js).

Server chính là thứ kiểm soát dữ liệu website, thông qua code logic (mô hình MVC) và thao tác trực tiếp với database.

Chúng ta đang sử dụng công cụ SSR (server side rendering), viết cả code giao diện + code logic + thao tác với database trên cùng 1 project (1 server)

Mô hình: client (viết = node.js) => server (viết = node.js). ALL IN ONE

Tuy nhiên, trong thực tế, chúng ta sử dụng React, Vue, Angular… để viết giao diện website, làm giảm đi vai trò của server (server chỉ code logic + thao tác database, không phụ trách phần giao diện nữa).

Đây là cách chúng ta sử dụng CSR (client side rendering - khóa học về React mình đề cập và giải thích cụ thể hơn).

Tại sao chúng ta lại có CSR, thay vì chỉ dùng mình SSR:
Mô hình: client (viết = React/vue…) => server (viết = node.js). Chia tách thành 2 projects
Client: team FE (frontend)
Server: team BE (backend)

Có 2 lý do chính về việc chia tách theo mô hình:

- Giảm thiểu gánh nặng cho server (khi không cần render giao diện)
- Tính mở rộng (scale) và security: frontend chính là giao diện 'phơi ra internet', ai cũng có thể tiếp xúc.
  Trong khi đấy, backend là nơi kiểm soát dữ liệu. Nếu chúng ta mất quyền kiểm soát dữ liệu thì chuyện gì sẽ xảy ra ?

  Thứ quan trọng nhất của 1 website thực tế, không phải là code, mà là 'dữ liệu' người dùng, thứ lưu trong database của bạn.

Vậy câu hỏi đặt ra, là làm cách nào để client và server có thể giao tiếp được với nhau ?

#68. JSON & APIs

Với mô hình client - server, sử dụng CSR (client side rendering), chúng ta cần công cụ 'giao tiếp' để giúp điều này thành hiện thực.

1.JSON
Tài liệu: https://www.w3schools.com/js/js_json_intro.asp

Khi xây dựng ứng dụng theo tư duy lập trình hướng đối tượng (OOP), chúng ta quan niệm các thành phần cấu tạo nên website là 'đối tượng'.

Vì vậy, 'đối tượng' (class) là thứ quan trọng nhất (cốt lõi). Javascript cũng không ngoại lệ.

Chỉ có điều, JS - nó dùng Object { }
Ví dụ:
```
const mineObj = {
        name: "Hỏi Dân IT",
        age: 25
}
```

Tuy nhiên, khi giao tiếp giữa client và server, **server không thể 'tự động' hiểu được kiểu dữ liệu Object. Nó chỉ hiểu được string (chuỗi).**

Vì vậy, JSON ra đời, giúp việc chuyển đổi qua lại giữa Object và String trở nên dễ dàng hơn bao giờ hết.

JSON stands for JavaScript Object Notation

Ví dụ:
```
const mineJSON = "{
        name: "Hỏi Dân IT",
        age: 25
}"
```

2. Sử dụng JSON

Có 2 hàm phổ biến nhất hay dùng với JSON

- Convert từ string sang object: JSON.parse( )
  https://www.w3schools.com/js/js_json_parse.asp

- Convert từ object sang string: JSON.stringify( )

https://www.w3schools.com/js/js_json_stringify.asp

3. APIs

APIs: https://en.wikipedia.org/wiki/API
Viết tắt của: application programming interface

Client (ví dụ React), giao tiếp với Server (ví dụ Node.js) thông qua APIs.

Ví dụ:
Ứng dụng React chạy tại: localhost:3000

Ứng dụng Node.js chạy tại: localhost:8000

APIs lấy danh sách users: localhost:8000/get-all-user

Như vậy, hiểu một cách đơn giản nhất, APIs là 1 URL

Dưới góc nhìn của client: Tôi không quan tâm nó viết như thế nào, điều tôi quan tâm, là tôi dùng cái 'đường link ấy', tôi có dữ liệu để hiển thị lên giao diện.

Góc nhìn của server: đối với tôi, APIs là 1 route tôi viết, chỉ có điều, tôi phải trả ra dữ liệu theo 1 định dạng đã thống nhất với Frontend, cụ thể, đấy là JSON.

Mình họa APIs trong thực tế:
https://documenter.getpostman.com/view/10808728/SzS8rjbc

https://api.covid19api.com/summary

#69. Restful là gì
Tài liệu: https://aws.amazon.com/what-is/restful-api/

1.API
API chính là cầu nối giữa 'clients' và 'resources' trên website

ví dụ về **client**: React/web browser
resource: chính là server backend, resources chính là tài nguyên (ví dụ data lưu ở database)

2. REST là gì
Representational State Transfer (REST):
- là 1 kiến trúc phần mềm (software architecture)
- quy định cách thức API 'nên hoạt động' như thế nào cho hiệu quả với mạng internet.

Developer khi thiết kế APIs có nhiều dạng 'kiến trúc' để lựa chọn.
ngoài REST, chúng ta còn có Graph (ví dụ GraphQL của facebook)

- Khi APIs được thiết kế theo tiêu chuẩn REST, chúng ta gọi là REST APIs.

- Khi web services (website) sử dụng kiến trúc REST, gọi là RESTful web services.
Thuật ngữ RESTful API ám chỉ RESTful web APIs, và chúng ta có thể dùng REST API hay RESTful API giống hệt nhau.

3. Thành phần chính của RESTful APIs.

- Client Request:

Mỗi nguồn tài nguyên cần 1 định danh, với REST, **đấy là 1 URL (còn gọi là endpoint)**

RESTful APIs thường được sử dụng với HTTP, vì vậy, chúng ta cần quan tâm tới HTTP methods, thứ nói cho server biết cần làm gì:

+ GET
+ POST
+ PUT
+ DELETE

- Server Response:
+ Status
+ Message body

#70. Status Code

Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Sử dụng Status Code cho HTTP request để 'minh họa' kết quả phản hồi cho 1 request.

Chia thành 5 loại chính:

Informational responses (100 – 199)

Successful responses (200 – 299)

Redirection messages (300 – 399)

Client error responses (400 – 499)

Server error responses (500 – 599)

#71. GET Method

Tài liệu: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET

Sử dụng method GET để 'yêu cầu lấy dữ liệu' (request data)

Các hành động trigger 'GET method':
+ F5 lại website (cần GET -> lấy dữ liệu để hiển thị)
+ thẻ <a><a/>

Lưu ý: không gửi kèm body (payload) với GET request.

#72. GET All Users API
Tài liệu:
- Về cách đặt tên: (là danh từ, phân biệt nhờ method sử dụng)
https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design#organize-the-api-design-around-resources

- Hàm JSON và status khi sử dụng với express:
https://expressjs.com/en/api.html#res.json
https://expressjs.com/en/api.html#res.status

1. Cấu trúc của 1 API
- Gồm:
+ mã lỗi (status)
+ body (nội dung data trả về)

2. APIs get users
- status: 200 (lấy data thành công)
- data trả về cho user (json)

#73. POST Method

Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

1. Mục đích

- Method post được sử dụng để gửi data lên server => mục đích muốn server có sự thay đổi (về data)
ví dụ: tạo mới users

2. Cấu trúc

- Method POST, ngoài URL, còn gửi kèm data ở phần body.

- Data sẽ được định dạng thông qua 'content-type', ví dụ:
+ application/x-www-form-urlencoded:
+ multipart/form-data
+ text/plain

#74. Create User API

- Thành phần của API:
https://stackoverflow.com/questions/12806386/is-there-any-standard-for-json-api-response-format

- Phân biệt application/json and application/x-www-form-urlencoded

https://stackoverflow.com/questions/9870523/what-are-the-differences-between-application-json-and-application-x-www-form-url

#75. PUT Method
Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT

- Tương tự như method POST, cần gửi kèm data lên server (request body)

- Sử dụng để update data => ảnh hưởng 1 records (trong khi gọi post nhiều lần => tạo nhiều records)

#76. Update User API
Update user
- by id
- method update

#77. PUT vs PATCH
https://stackoverflow.com/questions/24241893/should-i-use-patch-or-put-in-my-rest-api

https://stackoverflow.com/questions/28459418/use-of-put-vs-patch-methods-in-rest-api-real-life-scenarios

https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design#define-api-operations-in-terms-of-http-methods

- update 1 fields, hay update toàn bộ ?

#78. DELETE Method
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE

- Sử dụng để xóa tài nguyên trên server

#79. Delete User API

- get user by id

or
delete directly

Chapter 8: Project Practices

#80. Giới thiệu Project thực hành

Todo (coming soon). Các bạn bỏ qua video này và xem tiếp #81

#81. Model Customers

Gồm các thông tin:

customer

- name
- address
- phone
- email
- image (data type = string)
- description

#82. Giải pháp lưu trữ file với MongoDB

Tài liệu:
https://www.mongodb.com/docs/manual/reference/limits/

https://www.mongodb.com/community/forums/t/process-of-storing-images-in-mongodb/15093/3

Limit của MongoDB:

- Thông thường, mongoDB lưu trữ dưới dạng BSON (Binary JSON)

- Giới hạn của loại dữ liệu này là 16MB (với MongoDB v6). Khi mới ra đời là 2MB

- Vì vậy, để lưu trữ file lớn, không thể lưu theo cách thông thường.

Có 3 giải hay dùng để lưu trữ file với MongoDB

1. Lưu trữ dưới dạng BSON
Ưu điểm:
- Lưu nguyên file cần lưu vào database (dưới dạng BSON)
- Không cần xử lý nhiều (coding)

Nhược điểm:
- Tốc độ truy cập chậm, không tối ưu (vì cần thao tác đọc/ghi database với file)
- Bị giới hạn dung lượng bởi BSON (16MB với MongoDB v6)

Cách dùng:
- Định dạng type là Buffer để lưu trữ file

2. Lưu trữ dưới dạng GridFS (file system)
Ưu điểm:
- Lưu nguyên file cần lưu vào database (dưới dạng đặc biệt - GridFS)
- Đáp ứng được nhu cầu lưu trữ file lớn (tương tự BLOB của SQL)
- Max file size ???

https://stackoverflow.com/questions/21331141/whats-the-maximum-size-for-gridfs-on-mongodb

Nhược điểm:
- Tốc độ truy cập chậm, không tối ưu (vì cần thao tác đọc/ghi database với file)
- Nếu lưu nhiều file lớn sẽ dẫn tới kích thước database bị phình :v
- Để thao tác với định dạng này, cần sử dụng API trực tiếp của MongoDB (thay vì thông qua ORM Mongoose)

Cách dùng:

Tài liệu:
https://www.freecodecamp.org/news/gridfs-making-file-uploading-to-mongodb/

Ý tưởng:
- Sử dụng API (hàm) GridFS của MongoDB
- File gốc ban đầu sẽ được phân tán nhỏ ra, và lưu vào 2 collections (do MongoDB tự tạo):

+ **chunk** collection (stores the document parts) : lưu trữ file (break nhỏ ra dưới dạng binary)

+ **file** collection (stores the consequent additional metadata) : lưu trữ thông tin file, ví dụ filename, id..

- Để query 1 file, chúng ta sẽ lại dựa vào API GridFS, sau đấy truyền vào (filename, id...) để lấy lên.

3. Lưu trữ file không dùng database

- Với 2 cách làm ở trên, đều yêu cầu sử dụng database để lưu trữ.
- Có 2 cách để lưu trữ file (không dùng database)

Cách 1: Sử dụng FTP/FTPs (bạn mua server, rồi lưu file trên đấy) :

Cách 2: Sử dụng các dịch vụ, ví dụ như AWS S3

Ưu điểm:
- Không lưu trữ file dưới database
- Tốc độ truy cập nhanh

Nhược điểm:
- Với cách làm 1, cần quan tâm tới việc backup (nếu mất data), tính mở rộng (nếu lưu nhiều file), tính an toàn (security)
nếu lưu thông tin nhạy cảm (thông tin cá nhân khách hàng...)

- Với cách làm 2 là perfect, chỉ mỗi cái tốn tiền. Cách 1 tốn tiền đầu tư, phần còn lại tự lo.
Còn cách này, dùng bao nhiêu trả bấy nhiêu. pay as you go

Cách dùng:
- lưu file vào server (có thể là server đang code hoặc server remote)
- Chỉ lưu đường dẫn tới ảnh vào database

#83. Setup lưu trữ file với Node.js

- Với Node.js, các thư viện nổi tiếng thường dùng để upload file:
+ **multer** : https://www.npmjs.com/package/multer
+ **formidable**: https://www.npmjs.com/package/formidable
+ **busboy**: https://www.npmjs.com/package/busboy
+ **express-fileupload**: https://www.npmjs.com/package/express-fileupload

bonus multer:
https://stackoverflow.com/questions/46622473/expressjs-how-to-require-multer-in-another-file

1. Lựa chọn thư viện
- Chọn express-fileupload vì:
https://github.com/richardgirges/express-fileupload/issues/69
+ Hỗ trợ Express
+ Đơn giản

- Với các thư viện còn lại:
+ Việc dùng thư viện nào phụ thuộc vào đánh giá của các bạn (từ hiệu năng, cách viết code...)
+ Hiểu cách dùng 1 cái, các cái còn lại dùng tương tự (mindset), chỉ khác nhau cách khai báo (cú pháp viết code)
+ Mình sẽ cố gắng sẽ có video youtube nói về các thư viện trên.

2.Cài đặt thư viện
https://www.npmjs.com/package/express-fileupload

npm install --save-exact express-fileupload@1.4.0

3. Cách dùng
Example: https://github.com/richardgirges/express-fileupload/tree/master/example

Setup for express....

#84. API Upload files

- use Form data => to send File

- upload single file

- upload multiple files

- Viết services để tận dụng

#85. Tối Ưu Upload Files

Gồm các bước:

- Tạo đường link (path) lưu ảnh
- Tạo tên ảnh (để lưu không bị trùng)
- Hỗ trợ upload multiple images

Source code video này (chỉ nên tham khảo, cố gắng tự code theo ý mình hiểu trước)

Link download:

https://drive.google.com/file/d/1otKABzkcJh4YUz06kaKm0YJC_vLdOzhh/view?usp=share_link

#86. Create a customer API

Tạo APIs để thêm mới 1 customer

Yêu cầu:

- API (endpoint): **POST /customers**
- **Sử dụng Form Data để gửi kèm file**

- Kết quả trả ra dưới dạng:

```
{
   "EC": 0,
   "data": [   //customers object  ]
}
```

#87. Create array of customers API (sử dụng khi import files)

Tài liệu: https://stackoverflow.com/a/47643316

Lưu ý: không dùng vòng lặp for, sau đấy dùng hàm create/save để xử lý trường hợp này, vì hiệu năng không cao. Thay vào đấy, dùng insertMany

Tạo APIs để thêm mới nhiều customers một lúc

Yêu cầu:

- API (endpoint): **POST /customers-many**
- **Không cần upload files**

- Kết quả trả ra dưới dạng:

```
{
   "EC": 0,
   "data": [   //customers object  ]
}
```

#88. Bài tập GET all customer APIs

Tạo APIs để lấy tất cả danh sách của customers:

Yêu cầu:

- API (endpoint): **GET /customers**
- Kết quả trả ra dưới dạng:

```
{
   "EC": 0,
   "data": [   //customers object  ]
}
```

#89. Bài tập Update a customer

Tạo APIs để cập nhật thông tin của customer (không cần update image - tức là upload file, chỉ làm dạng đơn giản)

Yêu cầu:

- API (endpoint) : **PUT /customers**
- Body truyền lên server (thông tin dùng để update) **: id, name, email, address**

#90. Soft Delete với Mongodb
Tài liệu:
https://www.npmjs.com/package/mongoose-delete

npm install --save-exact mongoose-delete@0.5.4

Soft Delete là việc chúng ta không xóa dữ liệu khỏi database.
Lý do: xóa là mất, không restore được.

Ngày nay, dữ liệu là cái quan trọng nhất (big data). Bạn xóa bài post Facebook => bài post đấy bạn không thấy nữa,
nhưng, có chắc rằng Facebook đã xóa bài post của bạn ?

Soft Delete giúp tạo thêm 1 (hoặc 1 vài) fields để đánh dấu dữ liệu là xóa hay chưa.

Ý tưởng:

- Thêm trường cho model (ví dụ model Customer)

#91. Delete a Customer API
Tài liệu: https://github.com/dsanel/mongoose-delete
API Delete a customer:
Endpoint: DELETE /customers
Body: id

#92. Bài tập Delete Array Customers
1.Static method
Tài liệu: https://mongoosejs.com/docs/guide.html#statics

2. API Delete Array Customer (todo)

#93. Query String
Tài liệu:
https://en.wikipedia.org/wiki/Query_string
https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design#filter-and-paginate-data

1. Cấu trúc
Ví dụ:
https://example.com/path/to/page?name=eric&color=purple
http://localhost/over/here?name=eric

Query string là một phần của URL, được sử dụng để truyền thêm data từ client lên server (lấy động data)

Với url: https://example.com/path/to/page?name=eric&color=purple
thì đoạn **?name=eric&color=purple** được gọi là query string

Query string dùng dấu hỏi để phân cách (dấu hiệu để server biết), và dùng dấu & để truyền nhiều biến.

field1=value1&field2=value2&field3=value3...

Khi gọi URL: https://example.com/path/to/page?name=eric&color=purple
Server sẽ xử lý như sau:
https://example.com/path/to/page => đây là 1 route trên server
?name=eric&color=purple => đây là query string.

Từ query string này, cần tách biệt ra thông tin cần lấy, cụ thể ở đây là:
name=eric
color=purple

Server sẽ dựa vào giá trị của 'name' và 'color' để trả ra kết quả tương ứng.

2. URL encoding
encoding/decoding
Một vài ký tự không thể có trong URL (ví dụ space), hiểu đơn giản là ký tự đặc biệt
Ví dụ, dấu # dùng để minh họa ID cho document

Vì vậy, sử dụng URL encoding để giải quyết vấn đề trên.
https://en.wikipedia.org/wiki/Percent-encoding#Percent-encoding_reserved_characters

Ví dụ: https://www.google.com/search?q=my+nam+is+eric

SPACE is encoded as '+' or "%20"

#94. Req.query
Tài liệu: https://www.npmjs.com/package/qs
http://expressjs.com/en/api.html#req.query

Vấn đề đặt ra, với URL: **http://localhost/customers?name=eric&address=ha+noi**
Làm cách nào để server lấy được giá trị name=eric, và cách khai báo route như thế nào cho trường hợp trên.

Với express, các bước cần làm:

1. Định nghĩa route
GET /customers
=> không cần định nghĩa phần query string, vì đơn giản, đây là phần truyền động

2. Lấy dữ liệu từ query string bằng cú pháp : req.query (tương tự dùng req.body, req.files)
- Hướng dẫn sử dụng postman tab params (query params)

Bonus: behind the scenes => Express sử dụng thư viện 'qs' để parse URL

3. Tổng kết
- Việc sử dụng query params không phát sinh việc khai báo thêm route
- Tận dụng lại GET /path
- Nếu truyền lên query string, thì req.query trả ra data, ngược lại, không truyền lên => req.query = { } //object rỗng
- Được sử dụng để filter dữ liệu

#95. Req.params
Tài liệu:
http://expressjs.com/en/api.html#req.params

Tương tự khi sử dụng req.query, req.params giúp chúng ta 'truyền động' tham số từ client lên server

Ví dụ:
http://localhost/customers/eric
http://localhost/customers/hary

=> ở đây, name được truyền động

1.Cấu trúc
Khác với query string, khi dùng 'params', chúng ta không sử dụng dấu ? hay &
Muốn truyền nhiều tham số, sử dụng '/'
ví dụ: http://localhost/customers/eric/ha+noi => name=eric; address=ha noi

2. Lấy dữ liệu từ phía server
- Khai báo route:
Khác với query string, chúng ta cần phải định nghĩa route (để quy định tham số, vì chỉ truyền lên value, không truyền key,
thì server làm sao biết được value ứng với key nào)

ví dụ: /customers/:name/:address
=> để khai báo params, chúng ta sử dụng dấu ":" và tên của params đấy.

- **Lấy giá trị params thông qua : req.params**

3. Tổng kết
- Cần khai báo routes mới khi sử dụng với req.params
- Cách làm này chỉ phù hợp khi cần query động data với thông tin truyền lên ít.
Nếu truyền nhiều thông tin => sử dụng req.query

#96. Limit với URL (Giới Hạn của URL)

Tài liệu:
https://stackoverflow.com/questions/812925/what-is-the-maximum-possible-length-of-a-query-string
https://stackoverflow.com/questions/417142/what-is-the-maximum-length-of-a-url-in-different-browsers

- Khi sử dụng query string, URL sẽ dài thêm ra. và câu hỏi đặt ra là có bị giới hạn hay không ?

Best practice:
- Giới hạn độ dài của URL : tốt cho SEO và chạy ổn định trên tất cả browser
- Khi cần truyền nhiều characters trên URL => chuyển qua body của POST (tương tự upload file)

#97. Pagination (offset/limit)

1. Khái niệm Pagination (phân trang)

Vấn đề tồn đọng:
- Với APIs GET all, ví dụ GET /customers => trả ra tất cả kết quả. điều này không khả thi (không áp dụng trong thực tế)

- Ví dụ bạn search Google keyword "eric" => Google trả ra 1.3B kết quả. nhưng:
+ Google chỉ hiển thị khoảng 20 kết quả
+ phần còn lại, muốn xem tiếp thì lăn chuột xuống cuối trang, chọn trang kế tiếp

=> đây chính là khái niệm phân trang (pagination)

2. Tại sao cần phân trang ?

- Đối với dữ liệu lớn (từ hàng trăm rows), tại 1 thời điểm, user chỉ quan tâm tới 10, 20... kết quả đầu tiên, phần còn lại không quan tâm
=> hiển thị tất cả cũng = thừa

- Khi dữ liệu lớn, thì việc fetching data (gọi APIs lấy dữ liệu) sẽ tốn thời gian. việc này tương tự như việc bạn download video 1GB và video 100MB

dung lượng càng lớn (khối lượng data), thì gian chờ đợi càng lâu (phụ thuộc vào tốc độ internet và tốc độ xử lý của server)

client (request) -> server (response)

=> việc phân trang giúp giải quyết:
+ Chỉ hiển thị dữ liệu cần thiết (dữ liệu khách hàng quan tâm)
+ Giảm thiểu thời gian chờ đợi của người dùng

3. Cách sử dụng pagination

Ví dụ: kết quả tìm kiếm thông tin khách hàng có 100 kết quả, chỉ hiển thị 10 kết quả 1 lần
=> có 10 trang
page 1: hiển thị C1->C10
page 2: hiển thị C11-> C20
page 3: hiển thị C21 -> C30
....
page 10: hiển thị C91 -> C100

Vậy, với 1 apis phân trang, cần có:
- giới hạn số lượng bản ghi tối đa lấy 1 lần là bn ? (max documents)
- muốn lấy kết quả ở trang thứ bao nhiêu ?

4. SQL pagination

Tài liệu: https://www.sqltutorial.org/sql-limit/

Sử dụng 2 keyword: offset và limit

limit: số lượng bản ghi tối đa lấy ra
offset: số lượng bản ghi bỏ qua

Ví dụ:
page 1: hiển thị C1->C10 => limit = 10, offset = 0
page 2: hiển thị C11-> C20 => limit = 10, offset = 10 (bỏ qua 10 kết quả, chính là phần page 1 -> lấy từ kết quả 11)
page 3: hiển thị C21 -> C30 => limit = 10, offset = 20 (bỏ qua 20 kết quả, chính là phần page 1, 2 -> lấy từ kết quả 21)

5. MongoDB pagination

limit: số lượng bản ghi tối đa lấy ra
skip: số lượng bản ghi bỏ qua

#98.Tính toán $limit và $skip
Tài liệu:
https://stackoverflow.com/questions/27992413/how-do-i-calculate-the-offsets-for-pagination

Bài toán: client truyền lên $page và $limit, cần tính toán ra $skip

Ví dụ, có 100 customers
page = 3, limit = 10 => tính toán skip
page = 3, limit = 20 => tính toán skip

$offset = ($page - 1) * $items_per_page;

#99. API Get Customers with pagination
- api truyền lên page = ? và limit = ?
- sử dụng req.query để lấy được $page & $skip
- Tính toán ra $limit và $skip.

#100. Filter (Dạng Basic)
Filter chính là cách người dùng sử dụng các tiêu chí đề ra để 'lọc' các dữ liệu hiển thị.
Ví dụ khi mua hàng online:

- Sắp xếp theo giá cả (cao, thấp..)
- Sắp xếp theo vị trí
- Sắp xếp theo rating...

Việc sử dụng Filter, giúp người dùng lấy được kết quả mà người ta mong muốn nhìn thấy (bỏ bớt đi các kết quả thừa thãi)

Tài liệu: toán tử '%like%' với Mongoose:
https://stackoverflow.com/questions/43729199/how-i-can-use-like-operator-on-mongoose

#101. Bài tập filter (Basic)
Tài liệu:
https://mongoosejs.com/docs/api/query.html
https://www.mongodb.com/docs/manual/reference/operator/query/
https://www.npmjs.com/package/api-query-params

Yêu cầu:
name : có chứa cụm từ 'eric'
address: là hn hoặc hcm

address: {$in: ['hn', 'hcm']}

Cài đặt thư viện:
npm install --save-exact api-query-params@5.4.0

#102. Query Builder (Advance)

Áp dụng thư viện để build dynamic query

Chapter 9: MongoDB Design Patterns

#103. Mongoose và MongoDB (Driver) khác nhau như thế nào ?

1. Tài liệu

- Mongoose:

+ https://www.npmjs.com/package/mongoose
+ https://mongoosejs.com/

- Mongodb (driver), dùng cho Node.js

Ở đây, thư viện này không phải là database, mà là công cụ để kết nối tới database MongoDB
=> lý do gọi là Mongodb driver

+ https://www.npmjs.com/package/mongodb
+ https://www.mongodb.com/home

2. Mongoose vs MongoDB (Driver)

- sử dụng hình ảnh minh họa cho dễ hiểu

3. Kiến thức đã học

- Setup Mongoose
- Code ODM với Mongoose (find, create, update...)

Việc kết nối tới Database MongoDB, hay việc read/write data là do Mongoose phụ trách

#104. MongoDB Driver

1. Setup

https://www.npmjs.com/package/mongodb

npm install --save-exact mongodb@4.13.0

https://stackoverflow.com/a/47840960

2. Connection

```
const { MongoClient } = require('mongodb');

// Connection URL
const url = 'mongodb://localhost:27017';
const client = new MongoClient(url);

// Database Name
const dbName = 'myProject';

async function main() {
  // Use connect method to connect to the server
  await client.connect();
  console.log('Connected successfully to server');
  const db = client.db(dbName);
  const collection = db.collection('documents');

  // the following code examples can be pasted here...
}
```

#105. Read/Write với Mongodb Driver
Tài liệu: https://www.mongodb.com/docs/drivers/node/current/quick-reference/

1. Read data (fetch)
https://github.com/mongodb/node-mongodb-native#find-all-documents

2. Write data (create/update/delete)
https://github.com/mongodb/node-mongodb-native#insert-a-document

#106. Data Modeling

Tài liệu:
https://www.mongodb.com/docs/manual/core/data-modeling-introduction/

Khi thiết kế model, chúng ta cần cân bằng giữa các yếu tố:
- Các tính năng mà ứng dụng phát triển
- Performance của database
- Cách thức đọc/ghi dữ liệu

Cần cân nhắc và cân bằng giữa việc ứng dụng sử dụng data (read/write/processing data) và thiết kế data lưu trữ.

1. Flexible Schema (tính mềm dẻo)

- Với mongodb, chúng ta không cần định nghĩa kiểu dữ liệu khi lưu xuống database.
    + documents ko nhất thiết phải có cùng fields và data type

- với mongoose => say no :v (vì thông qua Model)

2. Document Structure
Cần quan tâm tới cấu trúc của dữ liệu để chọn dạng lưu trữ phù hợp

Todo: add images
- Embedded Data (nhúng data)

- References (tham chiếu data)

#107. Embedded Data Models

Tài liệu:
https://www.mongodb.com/docs/manual/core/data-model-design/#embedded-data-models

1. Khái niệm
<todo> add images
Embed là cách chúng ta 'xử lý' data ở trong cùng 1 document , 'all in one'.

ví dụ:
```
{
  _id: <ObjectId1>,
  username: "123",
  contact: { phone : "848586", email: "abc@gmail.com"}, //embedded sub-document
  address: {city: 'ha noi', country: "vietnam" }, //embedded sub-document
}
```

2.Ưu, nhược điểm
- Ưu điểm:
+ Read data => sử dụng ít câu lệnh query hơn

- Nhược điểm:
+ Lưu thừa data
+ Update ?

#108. Database References

Tài liệu: https://www.mongodb.com/docs/manual/reference/database-references/

#109. MongoDB Design Pattern

https://www.mongodb.com/blog/post/building-with-patterns-a-summary

#110. Mongoose Subdocuments (Embedded data)

Tài liệu:

https://mongoosejs.com/docs/subdocs.html

1. Khái niệm

Subdocuments là các documents được embedded (nhúng) vào trong documents khác. Điều này có nghĩa là chúng ta có thể sử dụng nested schema

2. Single nested subdocuments.

```
const childSchema = new Schema({ name: 'string' , address: 'string'});

const parentSchema = new Schema({
  name: 'string',
  email: 'string',

  // Single nested subdocuments
  child: childSchema
});
```

equal:

```
const parentSchema = new Schema({
  name: 'string',
  email: 'string',

  // Single nested subdocuments
  child: {
          name: 'string',
          address: 'string'
  }
});
```

3.Array of subdocuments

```
const childSchema = new Schema({ name: 'string' , address: 'string'});

const parentSchema = new Schema({
  name: 'string',
  email: 'string',

  // Array of subdocuments
  child: [ childSchema ]
});
```

equal:

```
const parentSchema = new Schema({
  name: 'string',
  email: 'string',

  // Single nested subdocuments (không biết trước số lượng phần tử)
  child: [
          {
                  name: 'string',
                  address: 'string'
           }
]
});
```

4. Subdocuments vs Nested Documents

https://mongoosejs.com/docs/subdocs.html#what-is-a-subdocument

#111. Mongoose Reference Documents

Tài liệu:
https://mongoosejs.com/docs/populate.html

Populate là cách 'join' schema, tương tự như $lookup của MongoDB Driver

join === Primary key + foreign key => cần reference (key tham chiếu)

```
const personSchema = Schema({
  _id: Schema.Types.ObjectId,
  name: String,
  age: Number,
  stories: [{ type: Schema.Types.ObjectId, ref: 'Story' }]
});

const storySchema = Schema({
  title: String,
  description: String
});
```

Note:
ObjectId, Number, String, and Buffer are valid for use as refs.

#112. Design Models with Relationship

Tài liệu: http://learnmongodbthehardway.com/schema/schemabasics/

1. Các bước cần làm

- Xác định 'mối quan hệ' giữa các model:
 + quan hệ 1:1
 + quan hệ 1: n
 + quan hệ n:n

- Xác định yêu cầu của ứng dụng: Read or Write or Read & Write

- Xác định khối lượng data sẽ lưu và tính scale của ứng dụng (performance)

2. Models

Customers, Projects, Tasks, Users

1 Customer n Projects
1 Projects 1 Customer

=> 1 Customer n Projects

1 Projects n Task
1 Task 1 Project

1 Projects n Users
1 Users n Project

=> 1 Project n Task
=> n Project n Users

customers
- name
- address
- email
- image
- description

=> 1 Customer n Projects
=> 1 Project n Task
=> n Project n Users

projects
- name
- start date
- end date
- description
- customerInfor : embed
- usersInfor: [ref]
- leader: embed
- tasks: [ref]

tasks
- name
- description
- usersInfo: embed
- projectInfo: embed
- status
- startDate
- endDate

#113. Tạo Models

Source code video:
https://drive.google.com/file/d/1nPc-4-sUAz6QCnpUMV5NHVydUsf2xyTu/view?usp=share_link

Chapter 10: MongoDB Advance

#114. Bài Tập Tạo Mới Projects

- chưa add tasks
- chưa add users

```
{
    "type": "EMPTY-PROJECT",
    "name": "test project 1",
    "startDate": "24/12/2022",
    "endDate": "25/12/2023",
    "description": "just a test project",
    "customerInfor": {
        "name": "eric",
        "phone": "0123456789",
        "email": "hoidanit@gmail.com"
    },
    "leader": {
        "name": "aka",
        "email": "eric@gmail.com"
    }
}
```

#115. Thêm User vào Projects

```
{
  "type": "ADD-USERS",
  "projectId": "63a6fa8c1ccff6eca9304a5d",
  "usersArr": ["63a70afb8ebbb5bc38496958", "63a70afb8ebbb5bc38496123"]
}
```

#116. Fetch a Project (with Ref)

- Sử dụng populate
- Get project with users

#117. Bài Tập Về Projects

- Update/Delete a project
- Delete users from project

```
{
    "type": "REMOVE-USERS",
    "projectId": "63a6fa8c1ccff6eca9304a5d",
    "usersArr": ["63a70afb8ebbb5bc38496958", "63a70afb8ebbb5bc38496123"]
}
```

#118. Bài tập CRUD a Task

1. add new task

```
{
   "type": "EMPTY-TASK",
   "name": "task 1 learn mongodb",
   "description": "bla bla nosql",
   "status": "PENDING",
   "startDate": "25/12/2022",
   "endDate": "26/12/2022"
}
```

2. fetch all task

3. update a task

4. delete a task

#119. Bài Tập Thêm Users/Projects cho Task

```
{
   "id": "63a7f3f12964daaaffd23665",
   "usersInfo": {
       "name": "eric",
       "email": "hoidanit@gmail.com",
       "city": "hcm"
   },
   "projectInfo": {
      "name": "project test",
      "startDate": "25/12/2022",
      "endDate": "25/12/2023",
      "description": "bla bla"
   }
}
```

#120. Bài tập Add a Task to a Project

```
{
   "type": "ADD-TASKS",
   "projectId": "63a6fa8c1ccff6eca9304a5d",
   "taskArr": ["63a7f3f12964daaaffd23665", "63a7f42a2964daaaffd23668" ]
}
```

#121. Bài tập Get Tasks của Projects

Todo…

#122. Validate Data

Tài liệu:

https://stackoverflow.com/questions/60801039/do-i-need-to-use-hapi-joi-with-mongoose/60802635#60802635

https://mongoosejs.com/docs/validation.html

1. Mongoose Validation.

- Viết trực tiếp với Mongoose Schema
- Được 'fire' mỗi khi save/update

2. Javascript Validation

npm install --save-exact joi@17.7.0

#123. Bài Tập Validate Data

Todo..

Chapter 11: Tổng kết các kiến thức đã học

#124. Deploy Database With Mongodb Atlas

Link đăng ký tài khoản: https://www.mongodb.com/cloud/atlas/register

Các bước cần thực hiện:

1. Create Account

2. create database => get url connection

3. test connection with mongodb compass

4. run with localhost

#125. Deploy Backend NodeJS With Render
Chuẩn bị:
- có tài khoản github
- push source code lên github
Tham khảo: https://www.youtube.com/playlist?list=PLncHg6Kn2JT6nWS9MRjSnt6Z-9Rj0pAlo

Các bước thực hiện:

1. Tạo tài khoản render

2. Kết nối github

3. setup .env

4. deploy

6. check connection from render => to mongodb atlas

5. test with postman

#126. Giới thiệu boilerplate Node.JS/Mongoose
https://github.com/hagopj13/node-express-boilerplate

#127. Các kiến thức chưa đề cập
Todo...
#128. Login ???
Todo...
#129. Nhận xét về model của mongodb
Todo...
#130. What's next ?
Todo...

Lời Kết

Như vậy là chúng ta đã cùng nhau trải qua hơn 130+ videos về cách luyện mindset (cách tư duy) với Node.JS và SQL/MongoDB.

Dĩ nhiên rằng, trong quá trình quá trình thực hiện khóa học này, mình sẽ không thể tránh khỏi những sai sót,

Vì vậy, nếu thấy sai sót, các bạn cứ thoải mái đóng góp qua Fanpage Hỏi Dân IT nhé.
https://www.facebook.com/askITwithERIC

Với khóa học này, mình hi vọng nó sẽ là hành trang backend giúp các bạn hiểu rõ các thức vận hành của 1 backend server website, đồng thời **'use right tools for the job**', đặc biệt là với MongoDB

Sau khóa học này, sẽ là update và thêm mới các khóa học...

Hẹn gặp lại các bạn ở các khóa học tiếp theo ....
Hỏi Dân IT