TÀI LIỆU KHÓA HỌC
NodeJS Pro - Tự Học từ số 0
(MVC, REST APIs, SQL/MongoDB)

Tác giả: Hỏi Dân IT & Eric

Version: 1.0

Chapter 1: Bắt buộc xem - Không bỏ qua chương học này
Hướng dẫn sử dụng khóa học hiệu quả, đạt chất lượng cao nhất

#1. Hướng Dẫn Sử Dụng Khóa Học Hiệu Quả

Bạn vui lòng "xem video lần lượt" theo trình tự. Vì khóa học như 1 dòng chảy, video sau sẽ kế thừa lại kết quả của video trước đó.

1. Dành cho học viên "có ít thời gian"
Nếu bạn vội, cần học nhanh, hoặc "bạn đã biết rồi", thì "vẫn xem video, cơ mà không cần code theo".

Lưu ý: vẫn xem qua tài liệu khóa học để biết "video hướng dẫn gì".
Đã "Không xem video", thì cần "đọc giáo án".

Có như vậy mới biết khóa học nó làm cái gì.

2. Dành cho học viên "thông thường"
Nguyên tắc:
- Xem video lần lượt
- Xem video kết hợp với giáo án. Bạn không cần take note, vì những điều quan trọng đã có trong giáo án

- Bạn vui lòng code theo video.
Nếu bạn "code theo ý bạn", vui lòng "không hỏi khi có bugs".

Câu chuyện này giống như việc bạn đi khám bệnh, nhưng không tin lời bác sĩ
=> Nếu bạn giỏi, bạn làm luôn bác sĩ, còn đi khám bệnh làm gì.

- Bạn có thể "code theo ý bạn muốn", sau khi "đã kết thúc khóa học"

- Nếu bạn có thắc mắc (hoặc có ý tưởng/nhận thấy bugs), take note lại, bạn hỏi, rồi mình giải đáp.
Chứ không phải là "tự ý làm theo điều các bạn muốn".
Vì đa phần, các bugs trong khóa học mình đã fix hết rồi.

Nên là yên tâm để học theo bạn nhé.

3. Về cách code

Bạn vui lòng code theo video, từ cách đặt tên biến, hàm. Vì mình đã tuân theo "convention tối thiểu" khi bạn đi làm đấy

4. Về bài tập thực hành

Đối với bài tập thực hành, bạn cứ code theo cách bạn hiểu, và kết hợp với "search Google, stackoverflow..."

Ưu tiên cách học thông qua các trang tài liệu chính thức (documents), search Google và đọc stackoverflow.
(đây là cách làm mình hướng dẫn trong khóa học)

Trường hợp đã thử mọi cách không có kết quả, đây là lúc sử dụng các công cụ AI:

- Hạn chế việc, copy/paste solution (tức là hỏi, lấy đáp án, và không hiểu logic thực hiện) từ các ứng dụng AI, ví dụ như ChatGPT

- Với beginners, cần phải xây nền móng thật vững trước khi sử dụng các ứng dụng AI. Nên nhớ, copy/paste nhưng phải hiểu bạn đang làm gì.

- **Nếu bạn sử dụng AI, nên thông qua việc hỏi - phản biện**, giống như việc bạn nhờ AI để thực hiện search/deep-research.

- Nên tự đặt câu hỏi là làm sao có thể code được như vậy, cần học những kiến thức gì, và quan trọng nhất, đừng quên bước thực hành.

Vì không có thực hành, tất cả chỉ là lý thuyết.

#2 + #3 + #4 : Tài Liệu Khóa Học

//todo

#5. Demo kết quả đạt được

Dự án: xây dựng website bán laptop với Node.js

Lưu ý: đây là dự án thiên hướng "fullstack", tức rằng bạn sẽ code từ A tới Z, cả frontend lẫn backend

1. Công nghệ sử dụng

Backend: (Node.js với TypeScript)

- **Express** : bootstrapping project với Express version 5

- Authentication/Authorization với Passport.js

- **Prisma**: sử dụng ORM (object relational mapping) để mô hình hóa Model
  + có học cách tư duy phân tích database (ràng buộc relationship)

- Build tool: **ts-node, nodemon**

Frontend: HTML, CSS và Javascript

- View Engine: EJS
- AJAX để gọi APIs (không cần reload page)

Database: MySQL (phần mềm MySQL WorkBench)

2. Triển khai dự án

Dự án được chạy tại localhost và không triển khai lên hosting, vì:

- rất ít hosting FREE hỗ trợ nodejs + mysql
- hosting FREE không lưu trữ ảnh upload

3. Học viên nào có thể học ?

Để học được khóa học này, học viên cần:

+ Biết cú pháp cơ bản của HTML, CSS và Javascript

+ Biết cú pháp của TypeScript

Tham khảo (nếu bạn chưa biết về TypeScript) tại đây

Khóa học này dành cho những bạn:

- Muốn 1 khóa học (và chỉ 1), có thể làm ra 1 website với JavaScript/TypeScript
- Muốn tìm hiểu về Backend node.js sử dụng mô hình MVC

#6. Về Quyền Tác Giả

#7. Cách Dùng Udemy - Hỗ Trợ Hỏi Đáp Q&A

Lưu ý: không bỏ qua video này. Xem để biết cách sử dụng Udemy, cũng như cách đặt Q/A khi cần hỗ trợ (support)

1. Sử dụng trên máy tính

Xem hướng dẫn tài liệu chi tiết tại đây

- Cách bắt đầu sử dụng khóa học (bắt buộc xem)

- Cách đặt câu hỏi cho khóa học (bắt buộc xem)
    - Hướng dẫn cách sử dụng Q&A
    - Hướng dẫn cách liên hệ Instructor qua Message

- Cách sử dụng phím tắt

- Take note trực tiếp trên video đang xem

2. Sử dụng trên điện thoại

Udemy có hỗ trợ ứng dụng trên điện thoại Android/IOS

Xem hướng dẫn tài liệu chi tiết tại đây

#8. Thông Tin Tác Giả Hỏi Dân IT

Về tác giả:

Mọi thông tin về Tác giả Hỏi Dân IT, các bạn có thể tìm kiếm tại đây:

Website chính thức: https://hoidanit.vn/

Youtube "Hỏi Dân IT" : https://www.youtube.com/@hoidanit

Tiktok "Hỏi Dân IT" : https://www.tiktok.com/@hoidanit

Fanpage "Hỏi Dân IT" : https://www.facebook.com/askITwithERIC/

Udemy Hỏi Dân IT: https://www.udemy.com/user/eric-7039/

Nếu bạn muốn nói chuyện với mình (giao lưu trao đổi võ công :), có thể xem mình livestream trực tiếp tối thứ 2 & thứ 5 hàng tuần trên Youtube Hỏi Dân IT

Chapter 2: Setup Environment
Cài đặt & chuẩn bị môi trường thực hiện dự án

#9. Chuyện Cài Đặt Công Cụ (Bắt Buộc Xem)

1.Mục đích
Chương học này sẽ hướng dẫn chi tiết cách cài đặt các công cụ cần thiết để phục vụ cho khóa học. Vì vậy, bạn vui lòng không bỏ qua video nào, xem lần lượt theo thứ tự

Có 2 sai lầm mà các bạn hay gặp phải, đặc biệt là những bạn "đã biết 1 chút"

Sai lầm 1: Bỏ qua các video cài đặt công cụ vì bạn cho rằng bạn "đã biết rồi"

Hãy nhớ rằng, **cài công cụ là 1 phần, đang còn phải "cấu hình" nó nữa**.

Khóa học được sinh ra, và đã tối ưu. Bạn chỉ dành 5 tới 10 phút để xem video, đổi lại tiết kiệm cho bạn cả giờ đồng hồ ngồi mò mẫm.

Sai lầm 2: Không quan tâm tới version của phần mềm

Khi thực hiện khóa học, bạn vui lòng download và cài đặt version phần mềm giống như video. **Điều này sẽ đảm bảo môi trường thực thi code là giống nhau. (hạn chế tối đa bug có thể xảy ra)**

2. Version Phần Mềm theo thời gian

Bạn đừng sợ phần mềm (công cụ) nó thay đổi version, hay thậm chí là "chê bai" version cũ. Vì vốn dĩ, công nghệ nó là vậy, luôn thay đổi theo thời gian.

Bạn yêu cầu "version mới nhất" cho cái bạn học, mình đã làm điều đó tại thời điểm quay video khóa học, tuy nhiên, sẽ là cố định 1 version.

Lý do: công nghệ sẽ cập nhật theo thời gian. Cho dù bạn muốn hay không, hoặc thậm chí lắp tên lửa vào đít, đuổi cũng không kịp.
Những cái ngày hôm nay, bạn cho là mới nhất, qua ngày mai, nó đã là cũ.

Điều bạn cần làm là: học 1 version, và quan trọng hơn, là bạn học, bạn cần hiểu nó

Sau đấy, nếu cần thiết, bạn học version mới hơn. Điểm khác biệt ở đây, là khi học version mới, bạn không phải là người bắt đầu từ số 0 (do đã có base từ version cũ)

Chỉ không học version cũ khi và chỉ khi: sản phẩm của version cũ không dùng được

Đấy là lý do tại sao các khóa học của mình, khi học xong, mình mới hướng dẫn nâng cấp version.

#10. Cài đặt Node.js

Tài liệu: https://nodejs.org/en

1.Nodejs là gì ?
Nodejs không phải là thư viện (library), không phải framework của JavaScript.

Nodejs là môi trường để bạn thực thi code javascript, tại browser và server.

Bạn học Node.js, về bản chất, là học các thư viện/framework (viết bằng JavaScript), nên bạn cần cài đặt môi trường Nodejs để có thể thực thi code JavaScript

Điều này tương tự với:

Bạn học cách sử dụng Microsoft Excel (javascript)
Bạn cần cài hệ điều hành Windows để có thể học nó (nodejs)

2. Cài đặt Nodejs

Sai lầm của beginners, là không quan tới tới version của phần mềm. Nên nhớ, công nghệ nó thay đổi theo thời gian, vì vậy, để hạn chế tối đa lỗi tối đa, bạn nên dùng version phần mềm như khóa học hướng dẫn.

Điều này tương tự với:
Bạn đang chơi 1 con game rất ngon trên Windows 7, bạn vác lên Windows 10 để chạy, có điều gì để đảm bảo rằng "sẽ không có lỗi xảy ra" ?

Trong khóa học này, mình sử dụng **version Node.js là 22.13.0.**

Vì vậy, để hạn chế tối đa lỗi có thể xảy ra, bạn vui lòng cài đặt chính xác version nodejs ở trên

Khi code giống nhau, môi trường thực thi code giống nhau (version nodejs), thì rất hiếm khi lỗi xảy ra.

Nếu đây là lần đầu tiên bạn học (coding) một dự án với Node.js, mình khuyến khích sử dụng duy nhất 01 version Node.js (dễ quản lý)

Chỉ sử dụng nhiều version Node.js, khi và chỉ khi, trên bạn có nhiều dự án Node.js, và mỗi dự án yêu cầu một version Node.js khác nhau. (hướng dẫn tại mục 3 bên dưới)

Link tải nodejs v22.13.0:
https://nodejs.org/download/release/v22.13.0/

Sau khi cài đặt xong, kiểm tra bằng cách gõ câu lệnh:

node -v

3. Trường hợp dùng nhiều version Nodejs

Lưu ý: bạn cần gỡ nodejs trước khi cài nvm

//áp dụng cho windows
https://github.com/coreybutler/nvm-windows

Video hướng dẫn cài nvm cho window, xem tại video #11

//áp dụng cho macos
Video hướng dẫn cài nvm cho mac, xem tại đây

https://dev.to/ajeetraina/how-to-install-and-configure-nvm-on-mac-os-5fgi

#11. Sử Dụng Node.JS với NVM (Extra)

Lưu ý: bạn cần gỡ nodejs trước khi cài nvm

//áp dụng cho windows
https://github.com/coreybutler/nvm-windows

Video hướng dẫn cài nvm cho window, xem tại đây

//áp dụng cho macos
Video hướng dẫn cài nvm cho mac, xem tại đây

https://dev.to/ajeetraina/how-to-install-and-configure-nvm-on-mac-os-5fgi

#12. Cài đặt Visual Studio Code (VSCode)

Công cụ code trong dự án sử dụng VSCode, 1 Editor hoàn toàn miễn phí

Link download:
https://code.visualstudio.com/download

#13. Cấu hình Visual Studio Code

1. Format Code
Setup Format on Save
Mục đích: Mỗi lần nhấn Ctrl + S , code sẽ được auto format trông cho đẹp/dễ nhìn

2. Cài đặt Extensions

Lưu ý: off các extension như eslint, prettier … để tránh xung đột

Fact: đi làm, người ta cấu hình eslint, prettier..thông qua code, vì mỗi 1 dự án (1 khách hàng 1 yêu cầu), cài global qua extension thì cái nào cũng giống cái nào

Đồng thời, với rule trên sẽ đảm bảo mọi thành viên trong team sẽ có cấu hình giống nhau

Các extensions cài đặt thêm:
- Code Spell Checker : hỗ trợ check chính tả khi đặt tên tiếng anh
- Auto Complete Tag : hỗ trợ code nhanh HTML

#14. Tại sao mình dùng VScode ?

Có rất nhiều IDE hỗ trợ bạn code:

- Visual Studio Code (gọi tắt là VSCode)
- WebStorm, Intellij...
- Sublime Text
- Notepad, Notepad ++ 😂

IDE/Editor thực chất là công cụ code, giúp gợi ý code và phát hiện lỗi
=> dùng công cụ code nào mà bạn "thoải mái nhất"

Mình chọn VScode vì:
- miễn phí
- có gợi ý code và phát hiện lỗi
- theme dark
- hỗ trợ git out-of-the-box
- support mạnh mẽ cho ngôn ngữ JavaScript/TypeScript (Java/PHP...)

Trong khóa học này, chỉ cần bạn code giống mình, dùng IDE nào không quan trọng, điều quan trọng, chính là cách chúng ta code ra làm sao.

Recommend: dùng VScode, để đảm bảo bạn và mình giống nhau 100%, từ coding cho tới debug.

Yên tâm 1 điều là: điều quan trọng nhất chính là khả năng bạn "tư duy" (mindset), bạn có thể dùng những điều học được, để áp dụng sang IDE bạn thích.

Fact: mình đã từng rơi vào công ty (khá to), cơ mà không mua license bản quyền phần mềm (trong khi không cho dùng crack)
=> bắt buộc phải dùng các công cụ free @@

#15. Cài đặt và sử dụng Git

Nếu bạn chưa biết gì về Git, xem nhanh tại đây (miễn phí)
Khóa học Git trả phí, tham khảo tai đây

- Sử dụng Git theo nguyên tắc:
1. Học xong video nào, commit đẩy lên Github/Gitlab
=> tạo cơ hội để thực hành câu lệnh của Git, ví dụ:
git add
git commit
git push...

2. Git là công cụ "mặc định bạn phải biết" khi đi làm phần mềm
=> điều 1 ở trên giúp bạn thực hành

3. Thói quen học xong video nào, đẩy code lên Git, giúp bạn tạo ra bản "backup" cho project của bạn

Ví dụ máy tính bạn bị hỏng đột xuất/bị mất
=> vẫn còn code, chỉ cần pull về code tiếp mà không phải code từ đầu.

4. Trong trường hợp bạn bị bug
=> bạn có thể gửi link github/gitlab cho mình xem => support fix bug

=> Mục đích sử dụng git ở đây là : backup code + thực hành công cụ đi làm mà bạn "phải biết" nếu muốn đi thực tập/đi làm.

#16. Cài đặt Google Chrome

Ở đây, sử dụng google chrome vì nó là ứng dụng phổ biến nhất (trình duyệt web được dùng nhiều nhất)

Bạn nên dùng Google Chrome (thay vì Firefox/Edge...) để đảm bảo rằng thao tác sử dụng giữa bạn và mình là giống nhau (tránh gây khó khăn không cần thiết)

Lưu ý: sử dụng version tiếng anh
=> change language

Mục tiêu:

- Sử dụng Google Chrome để chạy ứng dụng web
- Ngôn ngữ hiển thị là Tiếng Anh
- Set default app là google chrome (nếu nó mở app, thì chạy với google chrome)

  Xem hướng dẫn setup default app cho windows tại đây

Chapter 3: Hello world với Node.js và Express

Làm quen và viết chương trình đầu tiên với Node.js và framework Express, sử dụng JavaScript/TypeScript

#17. Tổng quan về chapter

//todo

#18. Node.JS có thể làm gì ?

Nhắc lại khái niệm Node.js:

Node.js là môi trường giúp bạn thực thi code JavaScript ở mọi nơi, không giới hạn tại trình duyệt web (browser).

Bạn có thể xây dựng backend server, mobile app, desktop app… với node.js

Tuy nhiên, khi đề cập tới keyword “node.js”, người ta thường ám chỉ là công việc của backend developer (mặc dù nodejs có thể hỗ trợ làm frontend/mobile app…)

Kinh nghiệm rút ra khi search google:

- Nếu bạn phát triển ứng dụng tại phía server (backend), bạn nên sử dụng keyword **nodejs**

- Nếu bạn phát triển ứng dụng khác, chỉ nên đính kèm tên công nghệ bạn đang sử dụng, ví dụ react, vue…

Ví dụ: với tính năng download file, bạn có thể thực hiện tại backend (server) hoặc tại frontend (browser)

Nếu bạn muốn thực hiện tại backend, bạn search: download file **nodejs**

Nếu bạn muốn thực hiện tại frontend, ví dụ reactjs, bạn search: download file **reactjs** (mặc dù reactjs cũng sử dụng nodejs làm môi trường để thực thi code javascript)

#19. Việc làm về Node.js

Lưu ý: Chỉ áp dụng với thị trường Việt Nam

Để nắm rõ nhu cầu của thị trường việc làm về công nghệ Node.js như thế nào, **bạn cần dành thời gian nghiên cứu qua các trang tuyển dụng việc làm** (làm khảo sát - research)

Các tiêu chí khi làm khảo sát, bao gồm:

- **Vị trí (location) bạn sinh sống**. Việc làm tại Hà Nội, Hồ Chí Minh, sẽ khác với Đà Nẵng, Huế...

- **Yêu cầu của công việc**, như bằng cấp, số năm kinh nghiệm làm việc

- **Thời điểm bạn làm khảo sát**: thông thường thời điểm từ tháng 3 tới tháng 9 có số lượng công việc nhiều nhất (quý 2 và quý 3).

  Cuối năm (quý 4) và đầu năm (quý 1) số lượng việc làm sẽ ít hơn (do sự thay đổi nhân sự của công ty là không đáng kể/nhảy việc)

- Nguồn khảo sát, bao gồm mạng xã hội, trang web chính thức của công ty và các trang web tuyển dụng việc làm, có thể kể tới như (ví dụ minh họa):

  https://itviec.com/

  https://topdev.vn/

  https://www.topcv.vn/

  https://www.vietnamworks.com/

  https://www.linkedin.com/

#20. Hello World với Node.js (CLI)
CLI: command line interface, tức là bạn sẽ thao tác thông qua dòng lệnh

1.Chuẩn bị

Đảm bảo rằng bạn đã cài đặt Git và Node.js **(version 22.13.0)**

Nếu bạn chưa biết gì về Git, xem nhanh tại đây (miễn phí)
Khóa học Git trả phí, tham khảo tại đây

Bạn vui lòng sử dụng chính xác nodejs v22.13.0 để hạn chế tối đa lỗi xảy ra
(mình đã giải thích tại video hướng dẫn cài đặt nodejs, xem tại #10)

Kiểm tra version nodejs cài đặt, sử dụng câu lệnh: **node -v**

2. Thực hành

Bước 1: Tạo folder chứa source code, đặt tên: nodejs-pro-hoidanit
Lưu ý: tên đường dẫn trên máy tính của bạn, không nên đặt tên có dấu hoặc chứa ký tự đặc biệt, ví dụ
C:/User/Tự học nodejs/project thực hành

Thay bằng:
C:/User/Tu-hoc-nodejs/project-thuc-hanh

Bước 2: Tạo file app.js
//nội dung
console.log(“Hello World”

Bước 3: Thực thi file
Sử dụng câu lệnh: node app.js

#21. Cách đẩy dự án lên Github/Gitlab của chính bạn

Mục đích: giúp bạn backup code, tránh tình trạng máy tính bị hư, mất hết code.
Đồng thời, tạo thói quen tốt, học xong video nào, thực hành git ứng với video đấy (dễ kiểm soát code)

Nếu bạn chưa biết gì về Git, học ngay và luôn tại đây (miễn phí)
Khóa học Git trả phí, tham khảo tại đây

Bước 1: Tạo repository trên github/gitlab

Bước 2: Đẩy dự án lên repository vừa tạo
git init
git add .
git commit -m "init project"
git add origin .... (copy paste :)

git push origin main

Nếu muốn đẩy nhánh master:
git checkout -b master
git push origin master

Bước 3: Minh họa khi code xong 1 video
git add
git commit
git push …

#22. NPM là gì

NPM, viết tắt của Node Package Manager, là trang web cung cấp thông tin về các thư viện mã nguồn mở sử dụng với Node.JS

NPM được sở hữu bởi Github (Microsoft mua lại từ 2018)

1. Cách sử dụng website NPM (basic)

Bước 1: search theo keyword
Ví dụ: react

Bước 2: Đọc thông tin của thư viện
Ví dụ: https://www.npmjs.com/package/react

2. Sử dụng npm để quản lý dự án Nodejs

Bạn không cần cài đặt npm tại máy tính bạn, vì khi cài đặt nodejs, nó đã tự động cài đặt thêm npm

Kiểm tra máy tính đã cài đặt npm chưa, sử dụng câu lệnh: **npm -v**

3. Sử dụng npm init project
https://docs.npmjs.com/cli/v11/commands/npm-init

Sử dụng câu lệnh : **npm init**

#23. Cài đặt Express

Tài liệu: https://www.npmjs.com/package/express

1.Express là gì ?
https://expressjs.com/

ExpressJS (tên gọi ngắn gọn: Express) là một framework dùng để xây dựng web applications và APIs.

Chúng ta sử dụng Express vì nó phổ biến, dễ dùng và được support mạnh mẽ trong môi trường của Node.js

2. Cài đặt Express

Sử dụng câu lệnh sau:
npm i --save-exact express@5.0.1

#24. Hello World với Express (JavaScript)

Tài liệu: https://expressjs.com/en/starter/hello-world.html

```
const express = require('express')
const app = express()
const port = 3000

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

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Truy cập:
http://localhost:8080/

#25. Setup TypeScript cho dự án Node.js
Tài liệu:
https://www.digitalocean.com/community/tutorials/setting-up-a-node-project-with-typescript

Bản chất của việc setup TypeScript
1.Bạn coding bằng cú pháp của TypeScript: mục đích là code được gợi ý, và code ít lỗi hơn
2. Bạn cần sử dụng thư viện để dịch code từ typescript sang javascript
3. Node.js sẽ thực thi code đã dịch (javascript)

Setup Nodejs với TypeScript:
Bước 1: Cài đặt thư viện
Sử dụng câu lệnh sau:

npm i --save-exact --save-dev typescript@5.7.3 @types/express@5.0.0 @types/node@22.10.7

Thư viện **typescript**, giúp dịch code typescript, sang code javascript
Thư viện **@types/express**, giúp bạn coding Express được gợi ý code
Thư viện **@types/node** giúp bạn coding nodejs được gợi ý code

Bước 2: Cấu hình typescript
// **npx tsc --init**

Tạo file: **tsconfig.json tại root**

```
{
  "compilerOptions": {
    "module": "commonjs",
    "esModuleInterop": true,
    "target": "es6",
    "sourceMap": true,
    "skipLibCheck": true,
    "outDir": "dist"
  }
}
```

//giải thích ý nghĩa
Chi tiết, tham khảo tại đây

Bước 3: Dịch code TypeScript sang JavaScript

npx tsc

Bước 4: Chạy dự án

node dist/app.js

#26. Hello World với Express (TypeScript)

//giải thích dependencies vs devDependencies

//Tạo src folder

//Update **tsconfig.json**
"rootDir": "src",

//update **package.json**

```
"main": "dist/app.js",
"scripts": {
  "start": "tsc && node dist/app.js",
  "dev": "tsc && node dist/app.js",
}
```

Chạy dự án với câu lệnh:
npm run dev

Hoặc

npm start

#27. Mô hình hoạt động của Express (Extra)

Mục đích: Hiểu về mô hình client server hoạt động như thế nào?

1.Cách express hoạt động

```
const express = require('express') //import express
const app = express() // tạo express application
const port = 8080 // 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:8080/** hoặc **http://localhost:8080** đều trả ra kết quả 'hello world'
- Vào các link khác, ví dụ http://localhost:8080/abc => hiện thông báo Cannot GET /abc

Lưu ý: http://localhost:8080/ và http://localhost:8080 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:
1.Trước khi client vào website, server đã được chạy lên, sẵn sàng để hoạt động.

2. Khi server được chạy lên thì:
- sẽ chạy trên 1 port. ví dụ ở trên là http://localhost:8080
- 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'

3. Khi client vào website, gõ http://localhost:8080 hoặc http://localhost:8080/

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 danh sách 'route' nó được khai báo, và trả về kết quả với res (Hello World)

4, Trường hợp client gõ http://localhost:8080/abc , tức là yêu cầu nội dung ứng với route '/abc'
Server nhận yêu cầu và lục tìm trong danh sách '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 ?

Chapter 4: Mô Hình MVC với Express

Một dự án Backend chuyên nghiệp, sẽ không thể thiếu cách tổ chức hệ thống code (structure). Ngoài ra, khi làm website, chúng ta sẽ không thể không biết đến mô hình MVC (Model - View - Controller)

#28. Tổng quan về chapter

//todo

#29. Setup DevTool

Mục tiêu: chạy project tại chế độ **'watch'**. Mỗi lần có file thay đổi, project sẽ tự động chạy lại

Tài liệu:
https://www.npmjs.com/package/ts-node
https://www.npmjs.com/package/nodemon

1.Cài đặt thư viện

npm i --save-exact --save-dev nodemon@3.1.9 ts-node@10.9.2

Cài đặt nodemon và ts-node thay vì sử dụng typescript

2. Cấu hình nodemon

https://github.com/remy/nodemon?tab=readme-ov-file#packagejson

```
{
  "watch": ["src"],
  "ext": "ts",
  "ignore": ["node_modules"],
  "exec": "ts-node ./src/app.ts"
}
```

//script dev: "dev": "nodemon"

#30. 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@16.4.7

- 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.

#31. More routes

1. Khai báo thêm route

Để khai báo thêm route với Express, cách đơn giản nhất là code thêm logic.
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)

#32. 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.

Tài liệu: 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://github.com/tj/consolidate.js
https://expressjs.com/en/starter/faq.html#which-template-engines-does-express-support

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

- 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 frontend, ví dụ React (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@3.1.10
npm i --save-exact --save-dev @types/ejs@3.1.5

https://expressjs.com/en/5x/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/5x/api.html#app.render

#33. Mô Hình MVC (Model - View - Controller)
Với xây dựng website, mô hình MVC là kiến trúc cơ bản nhất, cũng như được sử dụng rộng rãi nhất
MVC là viết tắt của Model - View - Controller, giúp viết code theo tổ chức, dễ dàng bảo trì (maintain) và mở rộng (scale) 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 mía 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.
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'.

#34. Tổ chức thư mục cho dự án

1.Tạo cấu trúc dự án theo mô hình MVC
- Tạo controllers
- Tạo models
- Tạo services
- Tạo routes

//tạo route
https://expressjs.com/en/guide/routing.html

#35. Cấu Hình Static Files

Static files: Images, CSS files, JS files, là các file tĩnh (ít thay đổi trong dự án)

//Tạo thư mục public
https://expressjs.com/en/starter/static-files.html

Download file hình ảnh trong video tại đây

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

Mục tiêu: chia tách code và sử dụng controller

Bước 1: Tạo view create a user
create.user.ejs

Bước 2: khai báo route
/create-user
Render ra view

Bước 3: sử dụng bootstrap để design giao diện nhanh - đẹp
https://getbootstrap.com/docs/5.3/getting-started/introduction/

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH" crossorigin="anonymous">

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>
```

#37. Design Giao Diện

Mục tiêu: design giao diện với Bootstrap, tham khảo nhanh tại đây

Tạo user form, với 3 thuộc tính, và button submit:

- Name
- Email
- Address

#38. HTML Form

Tài liệu: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form

Mục đích: sử dụng form để gửi kết quả từ client lên server

Cách sử dụng:

Bước 1: khai báo form, bọc ngoài dữ liệu cần gửi đi
- Tạo form tag
- Tạo method, action

Bước 2: Khai báo các thành phần trong form
- Sử dụng input và khai báo **name**

//Tại sao dùng name, mà không dùng id ?

Bước 3: Khai báo button, dùng để trigger form

Button cần có **type = submit**

Bước 4: Nhận kết quả tại phía server, với req.body
https://expressjs.com/en/5x/api.html#req.body

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

Tài liệu: https://expressjs.com/en/5x/api.html#express

Cấu hình Express:

```
//config req.body
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

Khai báo các thành phần trong form

- Sử dụng input và khai báo **name**

//Tại sao dùng name, mà không dùng id ?

#40. Setup Debug Node.js (Extra)

Tham khảo: https://stackoverflow.com/a/52911183

Hướng dẫn chi tiết cách debug với VSCode:
https://code.visualstudio.com/docs/editor/debugging

//package.json

```
"start:debug": "nodemon --inspect=9229 -e ts,tsx --exec node -r ts-node/register ./src/app.ts"
```

//launch.json

```
{
   "version": "1.0.0",
   "configurations": [
      {
         "name": "debug-hoidanit",
         "type": "node",
         "request": "launch",
         "cwd": "${workspaceRoot}",
         "runtimeExecutable": "npm",
         "runtimeArgs": [
            "run",
            "start:debug"
         ]
      }
   ],
}
```

Chapter 5: Sử dụng Database (SQL)
Sử dụng Node.js với database MySQL

#41. Tổng quan về chapter
//todo

#42. Database là gì ?

Tài liệu tham khảo:
SQL: https://www.w3schools.com/sql/default.asp
NoSQL: https://www.mongodb.com/docs/manual/reference/method/js-collection/

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

#43. Cài Đặt MySQL Workbench

Link tài liệu hướng dẫn:

- Windows: https://dev.mysql.com/doc/refman/5.7/en/windows-installation.html
- MacOS: https://dev.mysql.com/doc/refman/5.7/en/macos-installation.html

Link hướng dẫn dành cho MacOS: https://www.youtube.com/watch?v=2cvH0HRjZF8

Link tải file cài đặt:
https://dev.mysql.com/downloads/installer/

Download file cài đặt sử dụng trong video (windows) - link google drive tại đây
Lưu ý: version MySQL mình cài đặt (link google drive) là 8.0.41

Trong quá trình cài đặt (nếu có lỗi xảy ra), ví dụ như không initial database..., các bạn fix bằng cách chủ động tải version mới nhất về xem có bị lỗi không nhé.

#44. Tạo Fake Data với MySQL

Tài liệu: https://www.npmjs.com/package/mysql2
https://sidorares.github.io/node-mysql2/docs

Cài đặt thư viện:
npm i --save-exact mysql2@3.12.0

Bước 1: Tạo database và table cho dự án

- Tạo database cho dự án
- Tạo table users: fullName, email, address
- Tạo fake data cho table

#45. Setup MySQL với Node.js

Bước 2: Test connections
https://sidorares.github.io/node-mysql2/docs

Bước 3: Test query data
//todo

#46. Hiển thị Users (Part 1)

Bước 1: Tạo table với data fake
//tạo nhanh table với bootstrap
https://getbootstrap.com/docs/5.3/content/tables/#hoverable-rows

Bước 2: Lấy dữ liệu động từ database
//todo

Bước 3: Render dữ liệu với ejs
https://ejs.co/

Cần tìm hiểu: cách sử dụng biến số và cách sử dụng vòng lặp

#47. Hiển thị Users (Part 2)

Sử dụng vòng lặp với ejs
https://stackoverflow.com/a/71682826

//bonus video hướng dẫn cách debug

#48. Tạo mới user

Tài liệu:
https://sidorares.github.io/node-mysql2/docs/examples/queries/prepared-statements/insert

Để thực hiện việc tạo mới dữ liệu vào database, sử dụng câu lệnh insert
https://www.w3schools.com/sql/sql_insert.asp

1.SQL Injection
https://www.w3schools.com/sql/sql_injection.asp

2. Build Query
https://sidorares.github.io/node-mysql2/docs#using-prepared-statements

```
const sql = 'INSERT INTO `users`(`name`, `age`) VALUES (?, ?), (?,?)';
const values = ['Josh', 19, 'Page', 45];

const [result, fields] = await connection.execute(sql, values);
```

#49. Setup Absolute Import (TypeScript)

1.Phân biệt Absolute Path và Relative Path

Path (đường dẫn), thể hiện nơi lưu trữ file.

Absolute Path: đường link tuyệt đối (tức là đường link dẫn tới file được lưu trữ trên máy tính)

Relative Path: đường link tương đối, phụ thuộc vào thư mục chọn làm root

Bonus: Phân biệt **.. và .** trong đường link tương đối
https://stackoverflow.com/a/23242061

Double dot (..) , ám chỉ parent directory (moving to a parent directory)

Single dot (.), ám chỉ current directory (thư mục hiện tại)

2.Cấu hình Absolute Path với TypeScript
khai báo Typescript **baseUrl và paths**
https://www.typescriptlang.org/tsconfig/#baseUrl

Về typescript path: https://www.typescriptlang.org/tsconfig/#paths

npm i --save-dev --save-exact tsconfig-paths@4.2.0

"exec": "ts-node -r tsconfig-paths/register ./src/app.ts"

Lưu ý: khi chạy tại chế độ debug, sẽ bị lỗi. Lỗi này, sẽ được fix tại video tiếp theo

#50. Route Parameters

//fix bug absolute import khi chạy tại chế độ debug
https://github.com/TypeStrong/ts-node/issues/777#issuecomment-462071725

```
"start:debug": "nodemon --inspect=9229 -e ts,tsx --exec node -r ts-node/register -r tsconfig-paths/register ./src/app.ts"
```

Tài liệu:
https://expressjs.com/en/5x/api.html#req.params
https://expressjs.com/en/guide/routing.html#route-parameters

Mục tiêu: cần lấy được id của user cần xóa

Bước 1: Lấy động id của user (render trên table)

Bước 2: Xử lý hành động user click button delete với form

Bước 3: Lấy id truyền lên, thông qua req.params

#51. Xóa User

Tài liệu:
https://sidorares.github.io/node-mysql2/docs/examples/queries/prepared-statements/delete

//todo

#52. Xem chi tiết User

Tài liệu:
https://sidorares.github.io/node-mysql2/docs/examples/queries/prepared-statements/select

Lưu ý: khi dùng thẻ a để điều hướng trang, hoặc F5 (refresh), nó sẽ tương ứng với method GET của form

Bước 1: Cần lấy được id của user xem chi tiết

Bước 2: get user by id

Bước 3: Render dữ liệu ra view

#53. Cập nhật User

Tài liệu:
https://sidorares.github.io/node-mysql2/docs/examples/queries/prepared-statements/update

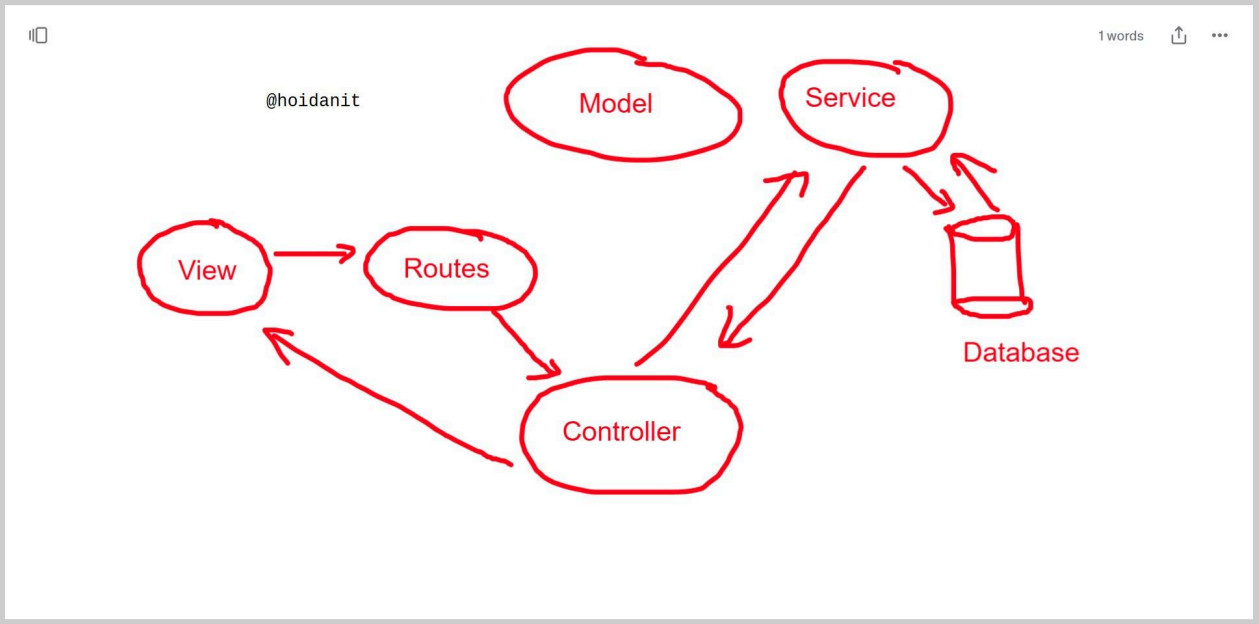
//todo

#54. Tổng kết về mô hình MVC

Mô hình MVC :
V (view): render ra giao diện
C (controller): xử lý yêu cầu (request) và trả ra kết quả cho view hiển thị
M (model): xử lý dữ liệu, phục vụ cho tr

Áp dụng vào framework Express:

Vẽ full luồng hoạt động với express (routes, services…)



#55. Nhận xét về cách làm hiện tại

Cách làm hiện tại, chính là mô hình SSR (server side rendering)

Mỗi lần client muốn truy cập tính năng của ứng dụng, sẽ gửi yêu cầu (request) lên server.

Server (dựa vào route), sẽ gửi lại phản hồi (response) tương ứng.

Hình thức kết nối xuống database, là viết câu lệnh SQL (raw query)

Ưu điểm:

Kiểm soát 100% code, vì bạn làm từ a tới z
Có thể tối ưu hóa theo ý muốn (vì bạn có thể tối ưu hóa câu lệnh query)

Nhược điểm:

Code viết dài
Đối với ứng dụng viết câu lệnh query đơn giản, thiên về CRUD (thêm/sửa/xóa), việc viết raw query không giúp tiết kiệm thời gian (tốn thêm thời gian để code, và yêu cầu cần hiểu rõ câu lệnh SQL)

=> Giải pháp đề ra để khắc phục nhược điểm của viết câu lệnh SQL (raw query) là sử dụng ORM (áp dụng với các dự án thiên về CRUD)

Chapter 6: Sử dụng ORM với Prisma
Áp dụng ORM với Node.js thông qua Prisma

#56. Tổng quan về chapter
//todo

#57. ORM là gì ?

Tham khảo: https://stackoverflow.com/a/1279678

Định nghĩa:
Object-Relational Mapping (ORM) là một kỹ thuật cho phép bạn truy vấn và thao tác dữ liệu từ cơ sở dữ liệu sử dụng mô hình hướng đối tượng.

Khi sử dụng ORM, bạn không viết câu lệnh SQL nữa (raw query). Thay vào đấy, bạn tương tác trực tiếp với đối tượng trong cùng một ngôn ngữ mà bạn đang sử dụng.

Ví dụ: bạn muốn lấy tất cả data của tables users
Raw query: select * from users
ORM: findAll( )

Lấy bản ghi (record) có id = 1
Select * from users where id = 1
findById(1)

Ưu/nhược điểm:

- Sử dụng ORM, bạn sẽ thao tác với code thông qua Model, giúp quản lý code tốt hơn (viết code ngắn đi, và theo tư duy hướng đối tượng)

Tuy nhiên:

- Bạn cần học cú pháp của ORM
- Đối với các dự án lớn (có độ phức tạp, yêu cầu tính tối ưu hóa cao), thì ORM sẽ có hiệu năng kém hơn so với câu lệnh query truyền thống (raw query)

#58. Sử dụng ORM với Node.js

1.Sử dụng ORM với Node.js
Có rất nhiều thư viện hỗ trợ ORM trong môi trường của node.js, có thể kể tới như:

Sequelize: https://sequelize.org/
https://github.com/sequelize/sequelize

TypeORM: https://typeorm.io/
https://github.com/typeorm/typeorm

Prisma: https://www.prisma.io/
https://github.com/prisma/prisma

Nổi tiếng nhất là Sequelize, tuy nhiên nó hỗ trợ không tốt cho TypeScript (sequelize phù hợp nhất nếu sử dụng JavaScript)

Việc lựa chọn thư viện nào không quan trọng. Điều quan trọng là chọn 1 thư viện, học và hiểu nó. Các thư viện khác, có thể cú pháp coding có sự khác biệt, nhưng mà tư duy giống nhau (đều là ORM)

Điều này tương tự bạn mua điện thoại android, phân vân nên dùng điện thoại của Samsung, Google, Xiaomi…
Mỗi hãng điện thoại sẽ có những tùy chỉnh khác nhau, nhưng "tư duy" khi sử dụng là giống nhau.

2. Sử dụng prisma
So sánh Prisma và typeORM:
https://www.prisma.io/docs/orm/more/comparisons/prisma-and-typeorm

Khóa học này, mình sử dụng Prisma, vì cách sử dụng đơn giản.

#59. Setup Prisma
Tài liệu:
https://www.prisma.io/docs/getting-started/setup-prisma/start-from-scratch/relational-databases-typescript-mysql

Bước 1: cài đặt thư viện
npm i --save-dev --save-exact prisma@6.3.0

VSCode extension:
https://marketplace.visualstudio.com/items?itemName=Prisma.prisma

Bước 2: setup
npx prisma init

//schema.prisma
```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}
```

//.env
```
DATABASE_URL="mysql://root:123456@localhost:3306/nodejspro"
```

Bước 3: Migrate tables
npm i --save-exact @prisma/client@6.3.0

//schema.prisma

```
model User {
  id       Int     @id @default(autoincrement())
  name    String?   @db.VarChar(255)
  email   String?   @db.VarChar(255)
  address  String?   @db.VarChar(255)
}
```

npx prisma migrate dev --name init

#60. Prisma Client (CREATE)

1.Create method
Tài liệu: https://www.prisma.io/docs/orm/prisma-client/queries/crud#create

2. Tái sử dụng connections
https://www.prisma.io/docs/orm/prisma-client/setup-and-configuration/databases-connections#prevent-hot-reloading-from-creating-new-instances-of-prismaclient

```
import { PrismaClient } from '@prisma/client'

const globalForPrisma = globalThis as unknown as { prisma: PrismaClient }

export const prisma =
  globalForPrisma.prisma || new PrismaClient()

if (process.env.NODE_ENV !== 'production') globalForPrisma.prisma = prisma
```

3. Enable logs
https://www.prisma.io/docs/orm/prisma-client/observability-and-logging/logging#log-to-stdout

https://stackoverflow.com/a/75174642

#61. Prisma Client (READ)

Tài liệu:
https://www.prisma.io/docs/orm/prisma-client/queries/crud#read

//todo

#62. Prisma Client (UPDATE)

Tài liệu:
https://www.prisma.io/docs/orm/prisma-client/queries/crud#update

//todo

#63. Prisma Client (DELETE)

Tài liệu:
https://www.prisma.io/docs/orm/prisma-client/queries/crud#delete

//todo

#64. Các thành phần của Prisma (Extra)
Mục đích của việc hiểu rõ các thành phần của Prisma, là chúng ta biết công cụ nó có thể làm gì, từ đó, tìm kiếm tài liệu chính xác hơn

Tham khảo: https://www.prisma.io/docs/orm

Các thành phần chính của prisma, bao gồm:
1.Prisma Schema
File: **schema.prisma**
Được dùng để định nghĩa mô hình dữ liệu cho dự án: sử dụng loại database nào, url kết nối vào database, **model (table) cho dự án**

Prisma Schema được dùng nhiều nhất để định nghĩa model (các đối tượng tham gia vào dự án), cũng chính là tables trong database

Tài liệu chi tiết: https://www.prisma.io/docs/orm/prisma-schema

2. Prisma Client
Đây là công cụ giúp bạn viết code truy vấn xuống database, thông qua ORM (đã thực hiện từ video #60 tới #63)

Tài liệu chi tiết: https://www.prisma.io/docs/orm/prisma-client

3. Prisma Migrate
Được sử dụng với 2 mục đích chính:
- Thực thi prisma schema để tạo ra tables bên trong database của bạn
- Version Control for Database Schema: kiểm soát lịch sử tạo database, ví dụ thêm/sửa/xóa table, columns...

4. Prisma CLI
Là giao diện dòng lệnh giúp bạn thực thi prisma schema, prisma migrate..., bắt đầu với keyword: **prisma**
Ví dụ: npx **prisma** init
npx **prisma** migrate dev --name init
https://www.prisma.io/docs/orm/prisma-migrate

5. Các công cụ khác
Có thể kể đến như Prisma Studio, cơ chế logging, caching data...

Chapter 7: Project thực hành
Phân tích, thiết kế database cho dự án thực hành để hiểu sâu hơn về mô hình MVC áp dụng với framework Express

#65. Nhìn lại các kiến thức đã học

1. Các công nghệ đã dùng

- Sử dụng Framework Express để chạy dự án

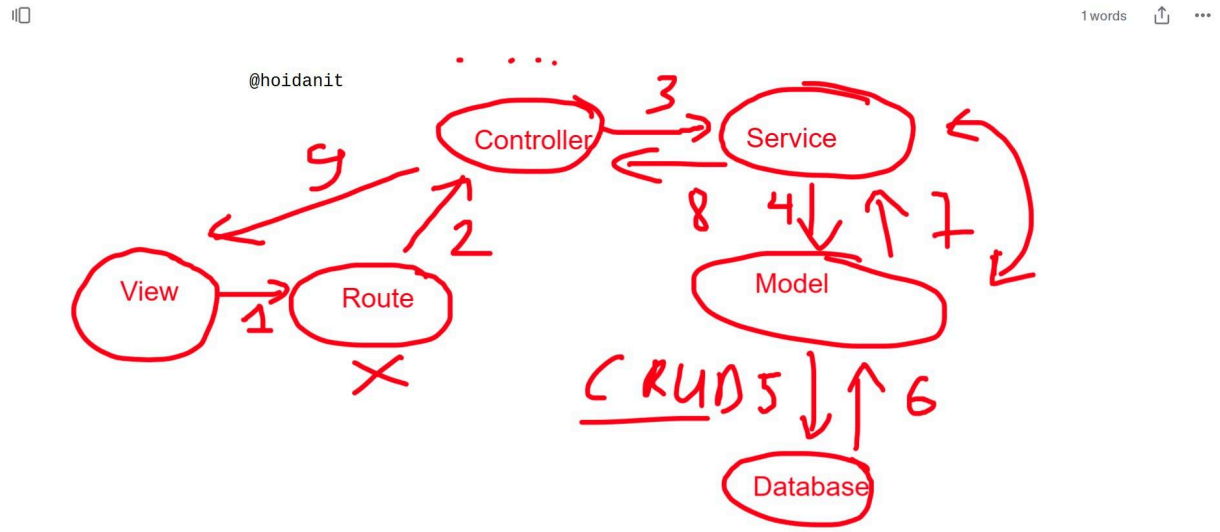
- Sử dụng mô hình MVC:
Model (Prisma)
View (ejs)
Controller

2. Làm sao để phát triển thêm dự án

Định nghĩa thêm model

Viết logic theo mô hình MVC

Thực hiện CRUD (create/read/update/delete)



#66. Phân tích yêu cầu dự án thực hành

Đề bài: tạo một website bán laptop

Lưu ý: đây là website thực hành dùng để "học kiến thức", website thực tế sẽ cần phải cải thiện nhiều hơn nữa, tránh hiện tượng ảo tưởng sức mạnh =))

Ý tưởng: có một trang web để hiển thị sản phẩm cho người dùng lựa chọn, các tính năng chính :

- Hiển thị danh sách sản phẩm (trang chủ /homepage)
- Xem chi tiết 1 sản phẩm
- Tìm kiếm sản phẩm theo tiêu chí (giá cả, số lượng, nhà sản xuất...)
- Giỏ hàng
- Thông báo mua thành công

Tham khảo: https://fptshop.com.vn/may-tinh-xach-tay

#67. Phân Tích Tác Nhân sử dụng hệ thống

Một dự án thực tế, luôn bắt đầu bằng việc xác định rõ, hệ thống có bao nhiêu loại người dùng sử dụng (actor/tác nhân) => biểu đồ usecase

Với dự án thực hành, chia thành 2 loại chính:

1. Người dùng chưa đăng nhập (guest)

Có thể :

- Xem danh sách sản phẩm, xem chi tiết sản phẩm
- Tìm kiếm sản phẩm theo tiêu chí

Không thể:

- Đặt hàng (place order). Vì chúng ta không có thông tin người dùng, hoặc thông tin chưa xác minh. Nếu muốn làm tính năng "lịch sử mua hàng" là không thể

- Thêm/sửa/xóa sản phẩm và thông tin trên website

2. Người dùng đã đăng nhập (user)

Phân chia theo vai trò của người dùng

Vai trò người dùng thông thường: (normal user)

- Kế thừa lại các tính năng của Guest (người dùng chưa đăng nhập)
- Có thể đặt hàng

- Không có quyền hạn thêm/sửa/xóa sản phẩm và thông tin trên website

Vai trò người dùng quản trị: (admin)

- Kế thừa lại các tính năng của User
- Full quyền trong hệ thống

Lưu ý: một website thực tế sẽ có cơ cấu phân quyền phức tạp hơn, ví dụ chia theo phòng ban, chia theo tổ chức: marketing, sale, it...

Tuy nhiên, đấy là 1 team code nên website đấy (và đôi khi là 1 công ty). Chúng ta đang solo và đặt mục tiêu "học hỏi", vì vậy, chỉ chia cấu trúc đơn giản nhất

#68. Phân Tích Thiết Kế Database

Link file Excel phân tích database trong video, download tại đây

Mục tiêu:

- Xác định actor (tác nhân) tham gia vào dự án
- Xác định mối quan hệ giữa các actor

Cách làm:

Bước 1: Xác định người sử dụng hệ thống

Bước 2: Xác định mối quan hệ theo chuẩn 1NF, 2NF, 3NF…
https://techmaster.vn/posts/36270/chuan-hoa-normalization-la-gi-vi-du-ve-1nf-2nf-3nf-bcnf-database

#69. Design Models cho database

Link file Excel phân tích database trong video, download tại đây

Mục tiêu:

- Design chi tiết các thuộc tính cho từng actor
- Xác định khóa chính, khóa ngoại dựa vào mối quan hệ của actor

#70. Prisma Schema

Mục tiêu: có một bức tranh tổng quan về cách code Models khi sử dụng Prisma

1.Prisma Schema
https://www.prisma.io/docs/orm/prisma-schema/overview

Cần chú ý về format rules khi code (nếu không dùng công cụ)

2. Models
https://www.prisma.io/docs/orm/prisma-schema/data-model/models

Models (Prisma) dùng để:

- Đại diện/tượng trưng cho các thực thể (entity) tham gia vào dự án (tư duy theo lập trình hướng đối tượng)

- Ánh xạ (mapping) tới các table trong database. Chúng ta dùng model để tạo ra các tables

- Giúp Prisma Client có thể query được dữ liệu từ database (hỗ trợ code với typescript)

//todo: giải thích các thành phần khi định nghĩa model

3.Cách tạo models và tables trong database

Có 2 trường hợp xảy ra:

- Bạn coding với Prisma, sau đấy bạn tạo database (khuyến khích sử dụng cách làm này)

- Bạn đã có sẵn database với dữ liệu, bây giờ muốn tạo dự án với Prisma, muốn thao tác với database mà không làm mất dữ liệu đã có
(nên backup dữ liệu trước khi làm)

#71. Thực Hành Tạo Models & Tables (Extra)
Tài liệu, tham khảo tại đây
Mục đích: thực hành các câu lệnh CLI mà Prisma hỗ trợ để tạo và sử dụng tables trong database
Lưu ý: trước khi thực hành code, nên sử dụng các công cụ giúp rollback code (ví dụ như version control - Git, hoặc export data, tránh trường hợp thao tác sai dẫn tới mất dữ liệu

Trường hợp 1: Bạn chưa có database. Sử dụng Prisma để tạo tables trong database
Khuyến khích beginners sử dụng cách làm này.
Quy trình thực hiện: tạo database => tạo Prisma (Schema/Model) => tạo tables cho database

Bước 1: Tạo mới database
Mở phần mềm MySQL Workbench, tạo database: **prisma71**

Bước 2: Cập nhật source code để "cắm" backend vào database mới này
//.env
DATABASE_URL="mysql://root:123456@localhost:3306/**prisma71**"

Bước 3: Tạo table cho database từ schema của prisma
https://www.prisma.io/docs/orm/prisma-migrate/getting-started

Chạy câu lệnh sau:
npx prisma migrate dev --name init

Trường hợp 2: Bạn đã có sẵn database, muốn sử dụng Prisma để quản lý tables cho dự án. Bạn nên export (backup) database trước khi thực hiện, tránh trường hợp mất data

Quy trình thực hiện: đồng bộ database và Prisma Models
https://www.prisma.io/docs/orm/prisma-migrate/getting-started#adding-prisma-migrate-to-an-existing-project

Bước 1: Cập nhật source code để "cắm" backend vào database mới này
//.env
DATABASE_URL="mysql://root:123456@localhost:3306/**prisma71**"

Bước 2: Đồng bộ database và source code
//todo

#72. Định nghĩa Models

Lưu ý: bạn không bắt buộc phải học thuộc cú pháp code của Prisma, điều quan trọng là bạn hiểu bạn đang làm gì. Thông thường, phần lớn thời gian khi bạn làm dự án, là bạn code logic của ứng dụng, không phải là làm việc với Model/Schema của Prima

Các câu hỏi trước khi coding:

- Cú pháp code của Model như thế nào ? (hình thù trông ra làm sao)
  Tham khảo cú pháp tổng quan của Model Prisma tai đây

- Các loại dữ liệu nào (datatype - ứng với database bạn sử dụng)
https://www.prisma.io/docs/orm/prisma-schema/data-model/models#defining-fields

- Các thuộc tính mà Prisma hỗ trợ:
  https://www.prisma.io/docs/orm/reference/prisma-schema-reference#attributes

Về cách đặt tên Model/table database:
Nên đặt tên table database số ít (singular) hay số nhiều (plural). Tham khảo tai đây

Việc đặt tên số ít/số nhiều phụ thuộc vào quan điểm cá nhân của từng người, tuy nhiên, mình khuyến khích sử dụng đặt tên số nhiều cho tên table trong database

Lưu ý về tên số ít/số nhiều, và tên order (không có chữ s)
Thống nhất đặt tên tables số nhiều, tên model số it

Ví dụ: Tên model là **user**, mapping tới table **users**

Sử dụng **@@map (mapping theo tên table)**
https://www.prisma.io/docs/orm/reference/prisma-schema-reference#map-1

@map : sử dụng khi muốn mapping theo tên fields

//todo: update source code thực hành mapping tới tables users

=> không cần sửa code, vì thực hiện mapping tại prisma schema

1.Tạo Models cho dự án thực hành

Bước 1: tạo models

users

id	number
username	string
password	string
fullName	string
address	string
phone	string
accountType	string
avatar	string

roles

id
name
description

orders

id
totalPrice

products

id	number
name	string
price	number
image	string
detailDesc	string
shortDesc	string
quantity	number
sold	number
factory	string
target	string

#73. Cập nhật Models

Bước 2: run migrate để tạo table
https://www.prisma.io/docs/orm/reference/prisma-cli-reference#migrate-dev

Sử dụng câu lệnh:
npx prisma migrate dev --name init-project

Update code typescript cho prisma: **npx prisma generate**
https://github.com/prisma/prisma/discussions/4568

https://www.prisma.io/docs/orm/reference/prisma-cli-reference#generate

Bước 3: Tạo Data fake cho tables trong database
https://www.prisma.io/docs/orm/prisma-migrate/workflows/seeding

//todo : tạo data cho table users
Count number of records:
https://www.prisma.io/docs/orm/reference/prisma-client-reference#count

Chapter 8: Module User

Hoàn thiện tính năng CRUD Users kết hợp với hash password và upload file

#74. Tổng quan về chapter

//todo

#75. Design Giao Diện Admin

Các bước test template:

Bước 1: Download source code gốc (original) tại đây
(branch original)

Bước 2: chạy file index.html

Bước 3: Download source code thực hành (starter) tại đây
(branch starter)

chạy file index.html

Việc cần làm:

- Chia cấu trúc views và static resources thành client/admin
- Load template với css/js
- Tạo file **dashboard.ejs**

#76. Chia Layout Admin

Tái sử dụng các thành phần:

- Header
- Sidenav
- Footer

Sử dụng ejs để tái sử dụng code:
https://ejs.co/#features

#77. Hoàn thiện Layout Admin

//todo
Xóa các script không dùng

//update main div

```
<div class="container-fluid px-4">
   <h1 class="mt-4">Manage Users</h1>
   <ol class="breadcrumb mb-4">
      <li class="breadcrumb-item"><a href="/admin">Dashboard</a></li>
      <li class="breadcrumb-item active">Users</li>
   </ol>
   <div>table user</div>
</div>
```

#78. Design Upload File

1. Mục tiêu

Thêm input upload file và select Role, giao diện bao gồm:

- Email (username)
- fullName
- Address
- Phone
- Role
- Avatar (input file)

//tạo seed data cho table Roles
INSERT INTO nodejspro.roles (name, description)
VALUES
("ADMIN", "Admin thì full quyền"),
("USER", "User thông thường");

//truyền role qua view ejs với select
https://stackoverflow.com/a/77742322

https://getbootstrap.com/docs/5.0/forms/form-control/#file-input

```
<div class="mb-3">
  <label for="formFile" class="form-label">Default file input example</label>
  <input class="form-control" type="file" id="formFile">
</div>
```

https://getbootstrap.com/docs/5.0/forms/select/

Design layout: https://getbootstrap.com/docs/5.0/forms/layout/#gutters

```
<div class="col-md-6">
   <label for="inputCity" class="form-label">City</label>
   <input type="text" class="form-control" id="inputCity">
  </div>
  <div class="col-md-4">
   <label for="inputState" class="form-label">State</label>
   <select id="inputState" class="form-select">
     <option selected>Choose...</option>
     <option>...</option>
   </select>
  </div>
```

#79. Image với Preview

Sử dụng jquery, tham khảo tại đây

Lưu ý: cần import Jquery ở đầu file, vì phần code script bên dưới để ở đầu file

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>

        <script>
            $(document).ready(() => {
                const avatarFile = $("#avatarFile");
                avatarFile.change(function (e) {
                    const imgURL = URL.createObjectURL(e.target.files[0]);
                    $("#avatarPreview").attr("src", imgURL);
                    $("#avatarPreview").css({ "display": "block" });
                });
            });
        </script>

<input class="form-control" type="file" id="avatarFile" accept=".png, .jpg, .jpeg">
```

Ý tưởng cách làm:

- Tạo thẻ image (để display none), chỉ hiển thị khi upload file
  <img style="max-height: 250px; display: none;" alt="avatar preview"
  id="avatarPreview" />

- Khi người dùng nhấn nút upload file (sự kiện onchange của JavaScript), phần code JQuery sẽ được chạy
- JQuery sẽ tạo ra url của ảnh, và thay vào thẻ image (display block)

//Lưu ý về lưu cache phía browser

Mở devtool => nhấn chuột phải => empty cache and reload

#80. Nơi nào để lưu trữ file ?

1.Hình thức upload file

Có 2 hình thức upload file : **single** (upload 1 file/lần) và **multiple** (upload nhiều file/lần)

Khi upload multiple files:

- Frontend cần xử lý để truyền lên "multiple" files
- Backend cần xử lý để lưu nhiều file 1 lúc
- Vấn đề sẽ phát sinh nếu quan trọng thứ tự file upload

Khi upload 1 file: frontend gửi 1 file và backend chỉ cần xử lý đúng file đấy (1 file tại 1 thời điểm)

=> **Cách dễ nhất** là upload 1 file. Khi cần upload nhiều file, dùng vòng lặp để upload

2. Nơi nào để lưu trữ file

Hình thức 1: **lưu trữ qua các dịch vụ online,** ví dụ S3 (amazon), cloudflare (R2)...
Ưu điểm: truy cập mọi lúc, mọi nơi. Tự động backup (sao lưu dữ liệu)

Nhược điểm: pay as you go (dùng bao nhiêu trả bấy nhiêu, hoặc mua theo gói cước)

Fun fact: Nếu dịch vụ cloud mà rẻ (khi có nhiều data), thì zalo đã không lưu trữ dữ liệu local (tại máy người dùng)

Hình thức 2: **lưu trữ tại database**
Với database mysql , hỗ trợ định dạng blob, clob (khóa học sern youtube)

Ưu điểm: hoàn toàn miễn phí, có thể lưu tối đa khoảng 4GB/file

Nhược điểm: tốc độ IO chậm (read/write). Phải tự backup

Hình thức 3: **lưu trữ tại máy chủ Server (nơi viết code)**
Ưu điểm: miễn phí, tốc độ IO nhanh hơn database (vì lưu raw file)
Nhược điểm: phải tự backup dữ liệu và tối ưu hóa IO

=> khóa học này sử dụng hình thức 3 (vì đơn giản, tiện lợi & miễn phí)
Bonus: đi làm, sẽ sử dụng ftp server

#81. Upload file với Node.js

1. Nguyên tắc khi upload file

Với hình thức lưu trữ file "tại server" (tức là lưu trữ trong folder chứa source code/hoặc folder trong máy tính)

Bước 1: client gửi file lên server

Bước 2: server sẽ lưu file vào thư mục quy định trước

Bước 3: kiểm tra xem client có thể truy cập được file đã upload hay không

Chúng ta cần kiểm tra, trước khi tiến hành code. Vì nếu mọi thứ chạy bình thường, thì chỉ cần code làm sao để file lưu vào đúng folder đấy là được.

2. Tìm hiểu upload file với EJS

- 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

Cài đặt thư viện:

npm i --save-exact multer@1.4.5-lts.1
npm i --save-exact --save-dev @types/multer@1.4.12

Download file hình ảnh thực hiện trong video tại đây
//test lưu trữ file, và test hiển thị file

#82. Hoàn thiện tính năng Upload file (Part 1)

Mục tiêu:

- Tối ưu hóa code theo mô hình MVC
- Customize upload file: filename, destination, validate (size, extension)
- Test hiển thị file đã upload

Cài đặt thư viện:

npm i --save-exact uuid@11.0.5

Tham khảo: https://stackoverflow.com/a/78999417

```
import multer from 'multer'
import path from 'path'
import { v4 } from 'uuid';

const fileUploadMiddleware = (fieldName: string, dir: string = 'user_image') => {
    return multer({
        storage: multer.diskStorage({
            destination: 'assets/' + dir,
            filename: (req, file, cb) => {
                cb(null, v4() + path.extname(file.originalname));
            }
        }),
        limits: {
            fileSize: 1024 * 1024 * 3
        },
        fileFilter: (req: Express.Request, file: Express.Multer.File, cb: Function) => {
            if (
                file.mimetype === 'image/png' ||
                file.mimetype === 'image/jpg' ||
                file.mimetype === 'image/jpeg'
            ) {
                cb(null, true);
            } else {
                cb(new Error('Only JPEG and PNG images are allowed.'), false);
            }
        }
    }).single(fieldName);
}

export default fileUploadMiddleware;
```

#83. Hoàn thiện tính năng Upload file (Part 2)

Mục tiêu:

- Lưu user vào database
- Hoàn thiện table list user

#84. Hash User Password

Cài đặt thư viện:

npm i --save-exact bcrypt@5.1.1
npm i --save-exact --save-dev @types/bcrypt@5.0.2

#85. Quan Hệ Cho Model - Relationships

Có thể dùng từ Association/ hoặc Relationships

Tài liệu: https://www.youtube.com/watch?v=fpBYj55-zd8

https://www.prisma.io/docs/orm/prisma-schema/data-model/relations

#86. One-to-Many Relationship

Tài liệu:
https://www.prisma.io/docs/orm/prisma-schema/data-model/relations/one-to-many-relations

Sau khi thay prisma schema, cần làm 2 việc:

- Generate phần code typescript: **npx prisma generate**
  (cần tắt dự án trước khi chạy câu lệnh trên)

- Cập nhật database:
  npx prisma migrate dev

  //sử dụng câu lệnh này để tạo migration mà không thực thi câu lệnh sql:
  npx prisma migrate dev --create-only

#87. Hoàn thiện tính năng CRUD User (Part 1)

Các tính năng cần hoàn thiện:

Bước 1: Xóa user

//todo

Bước 2: Xem chi tiết User

- Cần lấy thông tin user, kèm theo role
  <%= options[index] === index ? 'selected' : '' %>

- Hiển thị thông tin chi tiết (role/avatar)

#88. Hoàn thiện tính năng CRUD User (Part 2)

Bước 3: Cập nhật user

//hiển thị avatar

```
<% if (doc.Active) { %>
  <div class="item acitve">
<% } else { %>
  <div class="item">
<% } %>
```

//cập nhật user

```
 ...(avatar !== undefined && { avatar: avatar })
```

Chapter 9: Module Product

Thực hành module CRUD sản phẩm

#89. Tổng quan về chapter

//todo

#90. Design Giao Diện Trang Chủ

Download template client tại đây

//todo
Design giao diện homepage (convert từ html sang ejs)

#91. Chia Layout Client

//todo: chia layout

#92. Bài tập Design View Detail Product

Yêu cầu:

- Tạo url ứng với id
- Tạo view (detail.ejs) ứng với product

#93. Hoàn thiện Layout Client

//todo

#94. Bài tập Design Giao Diện Thêm mới Product

Yêu cầu:

Bước 1: Tạo html form

Tạo form để thêm mới Product, gồm các trường thông tin: (model Product)

- Tên sản phẩm: **name**
- Giá: **price**
- Mô tả chỉ tiết: **detailDesc (dùng textarea cho trường này)**
- Mô tả ngắn: **shortDesc**
- Số lượng: **quantity**
- Nhà sản xuất: **factory**
- Mục đích sử dụng: **target**
- Ảnh sản phẩm: **image**

```
<div class="row">
  <div class="mb-3 col-6">
    <label class="form-label">Factory</label>
    <select name="factory" class="form-select">
      <option value="APPLE">Apple (MacBook)</option>
      <option value="ASUS">Asus</option>
      <option value="LENOVO">Lenovo</option>
      <option value="DELL">Dell</option>
      <option value="LG">LG</option>
      <option value="ACER">Acer</option>
    </select>
  </div>
  <div class="mb-3 col-6">
    <label class="form-label">Target</label>
    <select name="target" class="form-select">
      <option value="GAMING">Gaming</option>
      <option value="SINHVIEN-VANPHONG">Sinh viên - Văn phòng
      </option>
      <option value="THIET-KE-DO-HOA">Thiết kế đồ họa</option>
      <option value="MONG-NHE">Mỏng nhẹ</option>
      <option value="DOANH-NHAN">Doanh nhân</option>
    </select>
  </div>
</div>
```

Bước 2: Test submit data
//todo

#95. Validate Form Input
Trong thực tế đi làm, bạn cần phải xử lý dữ liệu, tại phía frontend (browser) và phía backend (server)

Chỉ validate tại phía frontend là không đủ, vì bạn có thể F12 và chỉnh sửa thông tin HTML

Với khóa học này, mình tập trung validate tại phía backend nodejs, vì nếu làm validate với EJS (view engine) rất vất vả (so với các công cụ frontend chuyên nghiệp khác)

Các công cụ hỗ trợ validate phổ biến:
https://www.npmjs.com/package/express-validator

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

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

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

Cài đặt công cụ:
npm i --save-exact zod@3.24.2

#96. Validate với Zod

Bước 1: Tạo Schema
https://zod.dev/?id=basic-usage

//xử lý với object

Bước 2: handle error
https://zod.dev/?id=error-handling

//todo

#97. Hiển thị thông báo lỗi

Lưu ý: khi sử dụng form của ejs, data nhận được, luôn ở dưới dạng String.
Ví dụ: price = “1”, không phải price = 1 //number

Giải pháp đề ra, khi validate với zod, cần **transform** (biến đổi/chuyển hóa) dữ liệu

Refine: custom logic validate
Transform: biến đổi dữ liệu, ví dụ từ string => number

```
  price: z.string()
    .transform((val) => (val === "" ? 0 : Number(val)))
    .refine((num) => num > 0, {
      message: "Số tiền tối thiểu là 1",
    }),
```

//alert của bootstrap
https://getbootstrap.com/docs/5.3/components/alerts/

```
<div class="alert alert-danger mb-3">
  A simple danger alert—check it out!
</div>
```

#98. Bài tập Thêm mới Product
//todo: create a product/ table list products

npx prisma generate

npx prisma migrate dev --name update-sold

#99. Bài Tập Update/Delete Product

```
//todo
const factoryOptions = [
  { name: "Apple (MacBook)", value: "APPLE" },
  { name: "Asus", value: "ASUS" },
  { name: "Lenovo", value: "LENOVO" },
  { name: "Dell", value: "DELL" },
  { name: "LG", value: "LG" },
  { name: "Acer", value: "ACER" },
];

const targetOptions = [
  { name: "Gaming", value: "GAMING" },
  { name: "Sinh viên - Văn phòng", value: "SINHVIEN-VANPHONG" },
  { name: "Thiết kế đồ họa", value: "THIET-KE-DO-HOA" },
  { name: "Mỏng nhẹ", value: "MONG-NHE" },
  { name: "Doanh nhân", value: "DOANH-NHAN" },
];
```

#100. Load Động Data Product cho HomePage

1.Lưu dữ liệu text với MySQL
Với field "**detailDesc**", khi khai báo là kiểu String và không định nghĩa **type**
=> database đang sử dụng varchar 255 (lưu tối đa 255 ký tự)

Về kiểu dữ liệu text: https://stackoverflow.com/a/13932834
https://www.prisma.io/docs/orm/reference/prisma-schema-reference#mysql

npx prisma generate

npx prisma migrate dev --name update-detailDesc

2.Tạo fake data

Link download hình ảnh sản phẩm sử dụng trong video (và file script) tải tại đây

//tạo fake data

3. Hiển thị danh sách products
//todo

#101. Xem Chi Tiết Product

Các bugs cần fix:

- **Format giá tiền**, tham khảo tại đây
  <%= new Intl.NumberFormat('vi-VN', {
  style: 'currency' , currency: 'VND'
  }).format(product.price) %>

- **Fix font chữ**

  font-family: "Raleway", sans-serif;

  font-family: "Open Sans", sans-serif;

- **Fix CSS:**

//todo: add view detail product

Chapter 10: Module Auth

Xây dựng giao diện client, đăng ký, đăng nhập và phân quyền người dùng với Passportjs

#102. Tổng quan về chapter

//todo

#103. Bài Tập Design giao diện Login/Register

//todo

#104. Bài Tập Tính Năng Register

Tạo form, bao gồm:

- fullName
- username
- password
- confirmPassword

//update unique
npx prisma generate
npx prisma migrate dev --name update-unquie

Khi tạo mới user:
userType = SYSTEM
Cần so sánh password và confirmPasword
Hash user password

//validate email
https://github.com/colinhacks/zod#strings

//validate password/confirm password
https://github.com/colinhacks/zod/discussions/3412#discussioncomment-9916377

#105. Middleware là gì ?
Tài liệu:
https://expressjs.com/en/guide/using-middleware.html

1.Khái niệm middleware
Mô hình áp dụng với SSR (server side rendering)

Client, gửi Request => Server, trả về Response

Với express, Middleware là 1 function, có thể can thiệp vào request/response

Đặc điểm của middleware, có keyword **next**

Các middleware đã được sử dụng trong khóa học:
- Config static files, json (req.body)
- Config router
- Config thư viện multer (upload files)

//todo: tạo middleware
404 not found

2. Áp dụng middleware về xác thực người dùng

Authentication: xác thực người dùng (sử dụng username/password). Mục đích là, kiểm tra tài khoản người dùng đăng nhập có hợp lệ hay không ?

Authorization : sau khi người dùng đã đăng nhập thành công, cần xác định quyền hạn người dùng có thể thao tác (CRUD)
Ví dụ: normal user/ admin user

#106. Giới thiệu về Passport.js

1.Passport là gì ?

Passport là một thư viện hỗ trợ quá trình đăng nhập với express

Tại sau dùng passport mà không tự code phần logic này:

- Đảm bảo tính an toàn và tuân thủ flow của login/security

- Hỗ trợ đa dạng hình thức đăng nhập, từ username/password tới oauth2...

2.Cài đặt passport

https://www.npmjs.com/package/passport-local

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

npm i --save-exact passport@0.7.0 passport-local@1.0.0

npm i --save-exact --save-dev @types/passport@1.0.17 @types/passport-local@1.0.38

#107. Tích hợp Passport.js và Express (Part 1)

Tài liệu: https://www.passportjs.org/tutorials/password/prompt/
https://github.com/passport/todos-express-password/blob/bcrypt/routes/auth.js

Bước 1: config passport

- Xác định username/password
- Check user tại database: findByUsername, compare password

```
//todo

passport.use(new LocalStrategy(function verify(username, password, cb) {
  db.get('SELECT * FROM users WHERE username = ?', [ username ], function(err, row) {
    if (err) { return cb(err); }
    if (!row) { return cb(null, false, { message: 'Incorrect username or password.' }); }

    bcrypt.compare(password, row.hashed_password, function(err, result) {
      if (err) { return cb(err); }
      if (!result) {
        return cb(null, false, { message: 'Incorrect username or password.' });
      }
      return cb(null, row);
    });
  });
}));
```

#108. Tích hợp Passport.js và Express (Part 2)

Bước 2: Khai báo route login, sử dụng passport
//todo

Bước 3: cấu hình passport middleware for application
//todo
https://www.npmjs.com/package/passport#middleware
https://github.com/passport/todos-express-password/blob/bcrypt/routes/auth.js#L20

#109. Sử dụng Session (Memory)
Lưu session vào memory
https://www.passportjs.org/tutorials/password/session/

1.Session là gì

Session (phiên đăng nhập) là cách lưu trữ thông tin người dùng đăng nhập (tại server) để tái sử dụng giữa các request, mục đích là xác định, ai là người đang sử dụng hệ thống.

Mô hình hoạt động:
Bước 1: user tiến hành login. Nếu login thành công, user sẽ lưu tại browser session ID (lưu vào cookies)

Bước 2: user truy cập các url trên website, gửi kèm session ID

Server sẽ kiểm tra session id, để biết ai là người đang sử dụng hệ thống

Mặc định, session lưu vào memory (restart lại server sẽ clear data)
https://www.npmjs.com/package/express-session

2. Cài đặt thư viện
https://www.npmjs.com/package/express-session#examples

npm i --save-exact express-session@1.18.1
npm i --save-exact --save-dev @types/express-session@1.18.1

https://www.passportjs.org/concepts/authentication/sessions/

//Data user được lưu vào: **req.user**

#110. Hiển thị Message Lỗi
https://www.npmjs.com/package/express-session

//todo: display error messages
//pass request to callback

req.session
https://github.com/expressjs/session?tab=readme-ov-file#reqsession

#111. Session với Prisma

Tài liệu:
https://www.npmjs.com/package/express-session
https://github.com/expressjs/session?tab=readme-ov-file#compatible-session-stores

https://www.npmjs.com/package/@quixo3/prisma-session-store

//cấu hình prisma với session
npm i --save-exact @quixo3/prisma-session-store@3.1.13

//schema.prisma
```
model Session {
  id       String   @id
  sid      String   @unique
  data     String  @db.MediumText
  expiresAt   DateTime
}
```

Cần update typescript và database:
npx prisma generate
npx prisma migrate dev --name add-session

#112. Giải Thích Mô Hình Hoạt Động của Passport và Session (Extra)

Tài liệu:
https://www.passportjs.org/concepts/authentication/sessions/

1.Giải thích code
cookie.maxAge: Sets session expiration (7 days)
secret: Signs the session cookie for security

resave: Forces session save even if unchanged
saveUninitialized: Saves unmodified sessions

checkPeriod: Clears expired sessions every 2 min
dbRecordIdIsSessionId: Uses session ID as database ID
dbRecordIdFunction: Custom session ID function

serializeUser: lưu thông tin vào session (**mã hóa**)
deserializeUser: giải mã session để lấy thông tin (**giải mã**)

2. Luồng hoạt động của Passport.js và Session:

Bước 1: Thực hiện login

- User nhập username/password, submit form login

- Passport xử lý quá trình đăng nhập này. Nếu thất bại, hiển thị thông báo lỗi

  Đăng nhập thành công thì:

  1. **Lưu data (id/username của user) vào session** và session này được lưu vào database

     serializeUser: chỉ nên lưu thông tin cần thiết (ví dụ id của user), tránh lưu dư thừa (yếu tố security), đồng thời, cookie ở client bị giới hạn size

  2. **Lưu session tại phía browser** (cookies), thông thường, session id

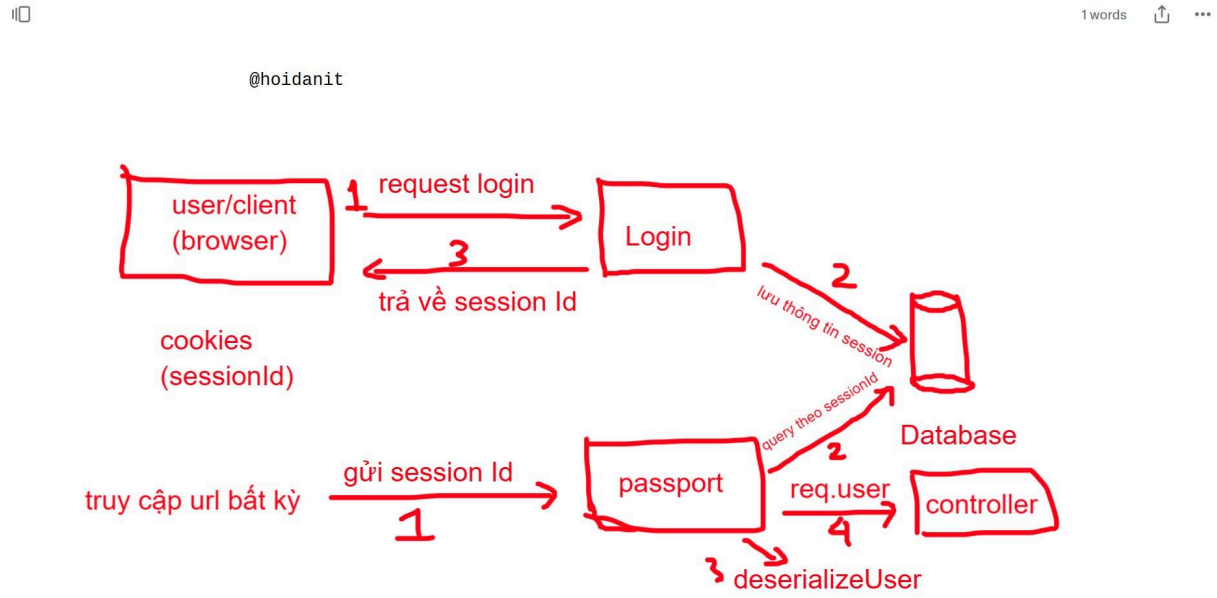
Bước 2: User truy cập vào các đường link url, cần xác định quyền hạn của người dùng, Ví dụ: /admin

Client gửi lên session id

Server sẽ check session Id trong database, nếu tồn tại (chưa bị hết hạn), thực hiện:

deserializeUser
Server sẽ query vào database, theo user id để lấy thông tin user

=> nạp vào req.user



#113. Protected Route (Part 1) - Res.locals

1.hiển thị email user/avatar/address của user đã đăng nhập tại header
//share data between view:
https://expressjs.com/en/5x/api.html#res.locals

```
//middleware
app.use((req, res, next) => {
   res.locals.user = req.user || null; // Pass user object to all views
   next();
});

//update view
<div class="dropdown my-auto">
   <a href="#" class="dropdown" role="button" id="dropdownMenuLink"
      data-bs-toggle="dropdown" aria-expanded="false" data-bs-toggle="dropdown"
      aria-expanded="false">
      <i class="fas fa-user fa-2x"></i>
   </a>

   <ul class="dropdown-menu dropdown-menu-end p-4" aria-labelledby="dropdownMenuLink">
      <li class="d-flex align-items-center flex-column" style="min-width: 300px;">
         <img style="width: 150px; height: 150px; border-radius: 50%; overflow: hidden;"
            src="" />
         <div class="text-center my-3">
                  @hoidanit
         </div>
      </li>

      <li><a class="dropdown-item" href="#">Quản lý tài khoản</a></li>

      <li><a class="dropdown-item" href="/order-history">Lịch sử mua hàng</a></li>
      <li>
         <hr class="dropdown-divider">
      </li>
      <li>
         <form method="post" action="/logout">
            <button class="dropdown-item">Đăng xuất</button>
         </form>
      </li>
   </ul>
</div>
```

#114. Protected Route (Part 2)

2.đã login, thì không truy cập /login
//todo

3.dựa vào role để redirect sau khi login thành công
https://www.prisma.io/docs/orm/prisma-client/queries/relation-queries

user => homepage,
admin => admin

//sử dụng include để load role, select để chọn thuộc tính (remove password)

4. Chỉ có user có role là ADMIN, mới truy cập /admin
//todo

#115. Logout

//tính năng logout
https://www.passportjs.org/tutorials/password/logout/
https://www.passportjs.org/concepts/authentication/logout/

//fix typescript type **user** khi sử dụng với passport
https://www.prisma.io/docs/orm/prisma-client/type-safety
https://stackoverflow.com/a/66312746

Fix lỗi import type với ts-node:
https://github.com/TypeStrong/ts-node#missing-types

#116. Tối ưu Routes (Extra)

//tạo status code page, download tai đây
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses
Tạo 404, 403, 500

//Check admin role (unauthorized page)
https://expressjs.com/en/guide/migrating-5.html#path-syntax

Chapter 11: Module Cart/Order

Hoàn thiện tính năng giỏ hàng và thanh toán sản phẩm

#117. Tổng quan về chapter

//todo

#118. Phân tích chức năng Giỏ Hàng

Thêm model Cart, CartDetail, thiết lập mối quan hệ:
//model **Cart** (giỏ hàng)
id
user_id //user id
sum //tổng số lượng sản phẩm trong giỏ hàng

//model **CartDetail** (chi tiết giỏ hàng)
id
cart_id
product_id
quantity //số lượng của sản phẩm
price //giá cả của sản phẩm

Lưu ý về mối quan hệ:
User - Cart: 1 -1
https://www.prisma.io/docs/orm/prisma-schema/data-model/relations/one-to-one-relations

Cart - CartDetail (tương tự Order và OrderDetail): 1 - N
https://www.prisma.io/docs/orm/prisma-schema/data-model/relations/one-to-many-relations

//todo: cập nhật prisma schema

npx prisma generate

npx prisma migrate dev --name video-118

#119. Thêm sản phẩm vào Giỏ Hàng (Create)

Logic thực hiện:

- Tại phía frontend, submit form, truyền lên **productId**

```
<form method="post"
   action="/add-product-to-cart/<%= product.id %>">
   <button
      class="mx-auto btn border border-secondary rounded-pill px-3 text-primary">
      <i
         class="fa fa-shopping-bag me-2 text-primary"></i>
      Add to cart
   </button>
</form>
```

- Backend xử lý:
  - + User có giỏ hàng hay chưa (cart), nếu chưa có, tạo mới (create).
    Nếu có rồi, cập nhật giỏ hàng (update)
  - + Tạo chi tiết giỏ hàng (cart_detail)

Tham khảo:

//crud
https://www.prisma.io/docs/orm/prisma-client/queries/crud#create-records-and-connect-or-create-related-records

//with relation
https://www.prisma.io/docs/orm/prisma-client/queries/relation-queries#nested-writes

//giới thiệu upsert
https://www.prisma.io/docs/orm/prisma-client/queries/crud#update-or-create-records

#120. Thêm sản phẩm vào Giỏ Hàng (Update)

//update number field
https://www.prisma.io/docs/orm/prisma-client/queries/crud#update-a-number-field

//giới thiệu upsert
https://www.prisma.io/docs/orm/prisma-client/queries/crud#update-or-create-records

#121. Design giao diện chi tiết Giỏ Hàng

- Design giao diện chi tiết giỏ hàng, tham khảo code tại đây
  //todo

  <nav aria-label="breadcrumb">
  <ol class="breadcrumb">
  <li class="breadcrumb-item">Home
  <li class="breadcrumb-item active" aria-current="page">Chi Tiết Giỏ Hàng

  </nav>

- Hiển thị số lượng giỏ hàng (lưu vào res.locals)
  Khi user đăng nhập thành công (**deserializeUser**), cần lưu thông tin giỏ hàng vào session (req.user)

  //logic xảy ra:
  - Thêm sản phẩm vào giỏ hàng => cập nhật sum tại database. Đồng thời, redirect về trang chủ.

  - Khi user gửi request mới lên server (f5 lại website), hàm **deserializeUser** được chạy, dữ liệu lấy từ database đã cập nhật ở trên

#122. Bài tập Chức năng chi tiết Giỏ hàng
//todo

JS sum array of number:
https://stackoverflow.com/a/43363105

#123. Xử lý tăng/giảm Product trong Cart

1.Xử lý tăng giảm giỏ hàng

Nguyên tắc: tăng 1 đơn vị, giảm 1 đơn vị (và chỉ giảm tới 1)

Bước 1: thêm data attribute cho input (mục đích là cung cấp thông tin cho phần code javascript)

```
<input type="text" class="form-control form-control-sm text-center border-0"
   value="<%=cartDetail.quantity %>"
   data-cart-detail-id="<%=cartDetail.id %>"
   data-cart-detail-price="<%=cartDetail.price %>"
 />
```

data-cart-total-price="<%= totalPrice %>"

Bước 2: Cập nhật logic javascript
Download source code tại đây

Bước 3: Test kết quả

Cần clear cache tại browser. Do cơ chế sử dụng là SSR (server side rendering), mặc định phía browser sẽ lưu cache (css/javascript)

2. Xử lý trường hợp, đã đăng nhập, chưa từng thêm sản phẩm vào giỏ hàng (0)
//todo

#124. Bài Tập Xóa Product từ Cart

Gợi ý cách làm:
Bước 1: Tạo form tại view

<form action="/delete-product-in-cart/<%=cartDetail.id %>" method="post">

Bước 2: xử lý tại controller/service

Lưu ý: do ràng buộc cha/con, cần xóa Cart-detail, rồi mới xóa cart
Xóa cart-detail. Và update sum/hoặc xóa cart

- Tìm cart-detail theo id (id từ view gửi lên)

- Xóa cart-detail

Kiểm tra điều kiện:

- Nếu Cart có sum > 1 => update trừ đi 1 đơn vị (update cart)
- Nếu Cart có sum = 1 => xóa cart

#125. Design Giao Diện Thanh Toán (Checkout)

1.Design giao diện
//todo
Download file design giao diện tại đây

2.Tạo model cho dự án

//orders
totalPrice
receiverAddress
receiverName
receiverPhone
status //PENDING, CANCELED, COMPLETE,

paymentMethod //COD, BANKING...
paymentStatus //PAYMENT_UNPAID, PAYMENT_SUCCED, PAYMENT_FAILED
paymentRef

userId // chỗ này, không cần @unique các bạn nhé, do 1 user, có N orders

//order_detail
price
quantity
orderId
productId

npx prisma generate
npx prisma migrate dev --name video-125

#126. Hoàn thiện tính năng Giỏ Hàng

Mục tiêu: trước khi chuyển qua giao diện thanh toán, cần cập nhật database

Bước 1: Tạo form ejs lấy được thông tin của sản phẩm (id/quantity) trên màn hình
//todo

```
<div style="display: block">
<% cartDetails.forEach( function (cartDetail, index) { %>
   <div class="mb-3">
      <div class="form-group">
         <label>Id:</label>
         <input class="form-control" type="text" value="<%=cartDetail.id %>"
            name="cartDetails[<%= index %>]['id']" />
      </div>
      <div class="form-group">
         <label>Quantity:</label>
         <input class="form-control" type="text"
            value="<%=cartDetail.quantity %>"
            name="cartDetails[<%= index %>]['quantity']"
            id="cartDetails[<%= index %>]" />
      </div>
   </div>
   <% }) %>
</div>
```

Bước 2: Cập nhật code javascript khi tăng/giảm số lượng
//todo (cần clear cache)

```
const index = input.attr("data-cart-detail-index")
const el = document.getElementById(`cartDetails[${index}]`);
$(el).val(newVal);
```

Bước 3: Cập nhật tại controller
//todo
Sử dụng Req.body

#127. Chức năng Đặt Hàng (Place Order)

Logic xử lý khi đặt hàng:

- Tạo order
- Tạo order-detail

//case đơn giản (thanh toán tất cả sản phẩm)

- Xóa cart tương ứng Xóa Cart_detail

//case phức tạp (chọn sản phẩm để thanh toán)

- Xử lý cart-detail:
    + Nếu số lượng sản phẩm >= sản phẩm trong cart-detail -> xóa
    + Nếu nhỏ hơn -> update

- Xử lý cart:
    + Count cart_detail -> update sum
      Nếu cart_detail = 0 -> xóa cart

#128. Bài tập Quản lý Order tại Admin

1. Design trang thank you page
//todo

2. Show order tại admin
//todo

3. Fix bug create order
//todo

#129. Bài tập Chức năng Lịch Sử Mua Hàng

- Chức năng lịch sử mua hàng (tham khảo file giao diện tại đây)

#130. Xây dựng Dashboard
//todo

//bổ sung thêm
Chức năng thêm sản phẩm vào giỏ hàng (từ trang xem chi tiết sản phẩm)

#131. Prisma Transaction (Extra) - Part 1

Mục đích video này ra đời, giúp bạn tiếp cận với các bài toán mang tính thực tế cao hơn (kiến thức nâng cao)

1.Fix bug

tại trang cart, nếu tăng giảm sản phẩm, cần cập nhật số lượng giỏ hàng trước khi chuyển qua checkout

Số lượng hiển thị tại giỏ hàng, là tổng số lượng của tất cả sản phẩm đã mua, không phải là “loại” sản phẩm mua.

Ví dụ: bạn mua 10 sản phẩm A, 5 sản phẩm B, 1 sản phẩm C

=> giỏ hàng hiển thị 10 + 5 + 1 = 16 (không hiển thị 3, tức là 3 sản phẩm A, B, C)

Khái niệm ACID của database: (tính toàn vẹn và nhất quán của database)

1. **Atomicity** (Tính nguyên tử)
   Một giao dịch (transaction) là một đơn vị không thể chia nhỏ. Hoặc tất cả các bước trong giao dịch được thực hiện, hoặc không có bước nào được thực hiện.

   Nếu một phần của giao dịch thất bại, toàn bộ giao dịch sẽ bị hủy bỏ (rollback), đảm bảo dữ liệu không bị rơi vào trạng thái không nhất quán.

   Ví dụ chuyển tiền A (trừ tiền) => B (cộng tiền)

2. **Consistency** (Tính nhất quán)
   Dữ liệu phải luôn ở trạng thái hợp lệ trước và sau khi một giao dịch xảy ra.
   Một giao dịch không được phép làm cho cơ sở dữ liệu rơi vào trạng thái không hợp lệ.

   Ví dụ:
   Nếu tài khoản A có 500$ và tài khoản B có 300$, sau khi chuyển 100$, tổng số tiền của hệ thống vẫn phải là 500$ + 300$ = 800$ trước và sau giao dịch.

3. **Isolation** (Tính cô lập)
   Nhiều giao dịch chạy đồng thời không được gây ảnh hưởng lẫn nhau.
   Hệ thống cần đảm bảo mỗi giao dịch chạy như thể nó là giao dịch duy nhất.

   Ví dụ:
   Nếu hai người cùng lúc đặt vé máy bay và chỉ còn một ghế trống, chỉ một giao dịch được phép hoàn tất, giao dịch còn lại phải thất bại.

4. Durability (Tính bền vững)
   Khi một giao dịch được xác nhận (commit), dữ liệu sẽ được lưu trữ vĩnh viễn trong hệ thống, ngay cả khi hệ thống gặp sự cố (mất điện, lỗi phần cứng).

   Ví dụ:

   Nếu bạn chuyển tiền thành công và ngân hàng thông báo “Giao dịch đã hoàn tất”, ngay cả khi hệ thống ngân hàng gặp sự cố ngay sau đó, số dư tài khoản vẫn phải được cập nhật chính xác.

2. Transaction

Tài liệu: https://www.prisma.io/docs/orm/prisma-client/queries/transactions

Khái niệm Transaction:

Một "database transaction", ám chỉ một chuỗi các hành động read/write, cần đảm bảo tất cả các hành động đấy phải thành công tất cả/thất bại tất cả

Nếu thành công tất cả, transaction sẽ được **commit** (tức là cập nhật database)

Nếu chỉ cần có 1 hành động failed, transaction sẽ được **rollback** (tức là khôi phục lại dữ liệu trước khi thao tác)

Ví dụ về commit/rollback:
https://sequelize.org/docs/v6/other-topics/transactions/#unmanaged-transactions

Khi nào nên sử dụng transaction của Prisma:

- Khi có nhiều thao tác (read/write) được thực hiện cùng nhau
- Khi có ràng buộc cần đảm bảo
- Khi cần rollback dữ liệu nếu có lỗi

Không nên dùng khi:

- Cập nhật một bản ghi đơn giản (vì nếu failed, data đã không được ghi nhận)

- Các thao tác là độc lập, không ảnh hưởng tới nhau (ví dụ như read data)

#132. Prisma Transaction (Extra) - Part 2

Tài liệu: https://www.prisma.io/docs/orm/prisma-client/queries/transactions

1. Cập nhật trường sold của product

Mục tiêu: Cần check product trước khi tạo order. Nếu product không đủ số lượng, cần cancel việc tạo order

Với bài toán tăng/giảm số lượng product:
https://www.prisma.io/docs/orm/prisma-client/queries/transactions#interactive-transactions

//test bằng cách: tạo product, với quantity = 1

Bước 1: tạo product với quantity = 1;

Bước 2: sử dụng transaction API

Chapter 12: Module Pagination/Query Filter

Tối ưu hóa fetching data với việc phân trang dữ liệu và query theo tiêu chí

#133. Tổng quan về chapter

//todo

#134. Tại sao cần phân trang (Pagination) Data ?

Vấn đề tồn đọng: fetch tất cả data (paginate)

Nếu dữ liệu nhỏ (vài chục, vài trăm rows), không vấn đề gì. Tuy nhiên, vài ngàn, vài trăm ngàn, vài triệu … sẽ là vấn đề.

Why ? lấy càng nhiều data, càng tốn nhiều băng thông (tốc độ truyền tải của mạng internet) và thời gian chờ đợi càng lâu, làm giảm trải nghiệm của người dùng.

Giải pháp đề ra:
Chỉ lấy số lượng data vừa đủ. Câu chuyện này tương tự :
Muốn download file nhanh -> tải file nhẹ thôi (file càng nặng, tải càng lâu, thời gian chờ đợi càng lớn)

Muốn chờ đợi ít -> fetch ít data thôi bạn nhé (và chỉ lấy data cần thiết, không dư thừa)

#135. Khái Niệm Offset/Limit

Tài liệu:
https://www.sqltutorial.org/sql-limit/
https://www.w3schools.com/php/php_mysql_select_limit.asp

Limit: giới hạn số phần tử muốn lấy

Offset: bỏ đi bao nhiêu phần tử , rồi mới lấy data

N1, N2, N3, …. N98, N99, N100 (LIMIT = 10)

PAGE 1: N1, N2, … N10 -> OFFSET = 0
PAGE 2: N11, N12, … N20 -> OFFSET = 10
PAGE 3: N21, N22,... N30 -> OFFSET = 20
…
PAGE 9: N81, N82.. N90
PAGE 10: N91, N92… N100 -> OFFSET = 90

#136. Khái niệm Query String

1.Phân biệt
req.body
req.file
req.params
req.path

req.query
https://expressjs.com/en/5x/api.html#req.query

https://en.wikipedia.org/wiki/Query_string
https://example.com/over/there ?**name=ferret**
https://example.com/path/to/page**?name=ferret&color=purple**

NAME = FERRET

COLOR = PURPLE

https://example.com/path/to/search**?page=1&limit=10**

#137. Design Pagination

Tài liệu: https://getbootstrap.com/docs/5.0/components/pagination/

```
<nav aria-label="Page navigation example">
 <ul class="pagination">
  <li class="page-item">
   <a class="page-link" href="#" aria-label="Previous">
    <span aria-hidden="true">&laquo;</span>
   </a>
  </li>
  <li class="page-item"><a class="page-link" href="#">1</a></li>
  <li class="page-item"><a class="page-link" href="#">2</a></li>
  <li class="page-item"><a class="page-link" href="#">3</a></li>
  <li class="page-item">
   <a class="page-link" href="#" aria-label="Next">
    <span aria-hidden="true">&raquo;</span>
   </a>
  </li>
 </ul>
</nav>
```

#138. Prisma Pagination

https://www.prisma.io/docs/orm/prisma-client/queries/pagination

Prisma sử dụng:
Skip (offset)
Take (limit)

skip: (page - 1) * pageSize

#139. Hoàn Thiện Fetch Users với Pagination

1.Tính tổng số trang

const totalPages = Math.ceil(totalItems / pageSize);

2. Active page, Next/previous page
Loop with ejs, tham khảo tại đây

#140. Bài tập Pagination

Yêu cầu: thêm tính năng phân trang cho phần admin: products, orders

//todo

#141. Bài Tập Chức Năng Product (Client)

```
//add active cho client
const navElement = $("#navbarCollapse");
const currentUrl = window.location.pathname;
navElement.find('a.nav-link').each(function () {
    const link = $(this); // Get the current link in the loop
    const href = link.attr('href'); // Get the href attribute of the link

    if (href === currentUrl) {
        link.addClass('active'); // Add 'active' class if the href matches the current URL
    } else {
        link.removeClass('active'); // Remove 'active' class if the href does not match
    }
});
```

```
//add pagination
<div class="pagination d-flex justify-content-center mt-5">
<li class="<%= page === 1 ? 'disabled page-item' : 'page-item' %> ">
    <a class="page-link" href="/?page=<%= page - 1  %>"
        aria-label="Previous">
        <span aria-hidden="true">&laquo;</span>
    </a>
</li>
<% for(var i=1; i <=totalPages; i++) {%>
    <li class="page-item">
        <a class="<%= page === i ? 'active page-link' : 'page-link' %>"
            href="/?page=<%= i  %>">
            <%= i %>
        </a>
    </li>
    <% } %>
        <li class="<%= page === totalPages ? 'disabled page-item' : 'page-item' %> ">
            <a class="page-link"
                href="/?page=<%= page + 1  %>"
                aria-label="Next">
                <span aria-hidden="true">&raquo;</span>
            </a>
        </li>
</div>
```

//Design Product page
Tài liệu:
https://getbootstrap.com/docs/5.3/forms/checks-radios/

//todo, tham khảo file giao diện tại đây

#142. Fix Giao Diện Client (Optional)
//todo

#143. Filter và Sorting với Prisma

1. Giới thiệu về các toán tử support
https://www.prisma.io/docs/orm/prisma-client/queries/filtering-and-sorting

Các toán tử support:
https://www.prisma.io/docs/orm/reference/prisma-client-reference#filter-conditions-and-operators

2. Tạo url để giao tiếp giữa frontend và backend
//todo
https://fptshop.com.vn/may-tinh-xach-tay

URL encode:
https://www.w3schools.com/tags/ref_urlencode.ASP

3. Tạo endpoint, test trực tiếp trên url
//todo

#144. Bài Tập Filter và Sorting

Ví dụ:

http://localhost:8080/products?**username**=hoidanit

Lấy ra tất cả user có username bao gồm keyword "hoidanit"

Bước 1: Lấy ra tham số từ query string

Bước 2: Tạo câu điều kiện **where** (filter) /**orderBy** (sort) tương ứng

Bài Tập: (thao tác với model Product)

Yêu cầu 1: http://localhost:8080/products?**minPrice**=1000000

Lấy ra tất cả sản phẩm có giá (price) tối thiểu là 1.000.000 (vnd)

Yêu cầu 2: http://localhost:8080/products?**maxPrice**=15000000

Lấy ra tất cả sản phẩm có giá (price) tối đa là 15.000.000 (vnd)

Yêu cầu 3: http://localhost:8080/products?**factory=APPLE**

Lấy ra tất cả sản phẩm có hãng sản xuất = APPLE

Yêu cầu 4: http://localhost:8080/products?**factory=APPLE,DELL**

Lấy ra tất cả sản phẩm có hãng sản xuất = APPLE hoặc DELL . Truyền nhiều điều kiện, ngăn cách các giá trị bởi dấu phẩy (điều kiện IN)

Yêu cầu 5: http://localhost:8080/products?**price**=10-toi-15-trieu

Lấy ra tất cả sản phẩm theo range (khoảng giá). 10 triệu <= price <= 15 triệu

Yêu cầu 6: http://localhost:8080/products?**price**=10-toi-15-trieu,16-toi-20-trieu

Lấy ra tất cả sản phẩm theo range (khoảng giá). 10 triệu <= price <= 15 triệu và 16 triệu <= price <= 20 triệu

Yêu cầu 7: http://localhost:8080/products?**sort**=price,asc

Lấy ra tất cả sản phẩm và sắp xếp theo giá tăng dần

#145. Chữa Bài Tập Filter và Sorting
//todo

#146. Xử lý JavaScript truyền động URL Filter

Mục tiêu:

- Khi nhấn button filter, cần lấy được các tiêu chí filter, và truyền động lên URL

- Khi reload page, nếu có data trên đường link, cần checkbox filter

Ví dụ:
http://localhost:8080/products?page=1&price=duoi-10-trieu**%2C**10-15-trieu&sort=gia-tang-dan&factory=APPLE**%2C**ACER&target=GAMING%2CTHIET-KE-DO-HOA%2CMONG-NHE

Trên url có những ký tự "%2C", đây là dấu phẩy. Chẳng qua là browser nó encode thành như vậy (đối với ký tự đặc biệt)

Vấn đề trên không ảnh hưởng gì tới hệ thống của chúng ta, vì đa phần các ngôn ngữ server sẽ tự động decode (chuyển cái %2C thành dấu phẩy)

Tham khảo về url encoding: https://en.wikipedia.org/wiki/Percent-encoding

Download file Javascript/View tai đây

#147. Xử Lý Nhiều Điều Kiện Filter
//todo

Ý tưởng: build câu lệnh query

//count + query with transaction
https://stackoverflow.com/a/74334140

Chapter 13: Tổng Kết Mô Hình Server Side Rendering (SSR)

Tổng kết các kiến thức đã học áp dụng mô hình MVC và cơ chế Server side rendering (SSR)

#148. Fix Bug Giao Diện

1.Fix các lỗi tồn đọng

//fix lỗi khi filter không có sản phẩm tồn tại

//fix banner tại trang chủ
Clear cache (nếu cần thiết)

Download file hình ảnh tại đây

#149. Sử Dụng Ajax (Extra)

1.Sử dụng AJAX

Dùng AJAX để không cần reload page
Cứ reload đi. Không sao cả. Học thêm frontend và viết api sẽ giải quyết được vấn đề trên.
Tham khảo: https://github.com/kamranahmedse/jquery-toast-plugin
https://cdnjs.com/libraries/jquery-toast-plugin

Bước 1: Import thư viện

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-toast-plugin/1.3.2/jquery.toast.min.js"
integrity="sha512-zlWWyZq71UMApAjih4WkaRpikgY9Bz1oXIW5G0fED4vk14JjGlQ1UmkGM392jEULP8jb
NMiwLWdM8Z87Hu88Fw==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>

<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/jquery-toast-plugin/1.3.2/jquery.toast.min.css"
integrity="sha512-wJgJNTBBkLit7ymC6vvzM1EcSWeM9mmOu+1USHaRBbHkm6W9EgM0HY27+UtUaprn
taYQJF75rc8gjxllKs5OIQ==" crossorigin="anonymous" referrerpolicy="no-referrer" />
```

Bước 2: Viết **AJAX**
//todo

Bước 3: Tại backend, trả ra dưới dạng API

Áp dụng tại 3 screen: (add product)

- Trang chủ
- Trang xem chi tiết sản phẩm
- Trang filter sản phẩm

Về lỗi return khi viết api:
https://github.com/expressjs/express/issues/5987#issuecomment-2428333462

#150. Nhận xét về dự án thực hành

1.Về nghiệp vụ (requirement)

Về nghiệp vụ:
Điểm nào chưa hợp lý, cứ sửa theo logic các bạn muốn (lưu ý là sau khi xem hết tất cả video của khóa học hãy làm nhé)

2. Về cách code

- Về code frontend đang còn nhiều khó khăn . reload every time.

#151. Cách code dự án của chính bạn

1.Coding dự án từ đầu

Cách nhanh nhất: bạn có thể chỉnh sửa lại source code đã học : xóa bớt controller, services, models… rồi code thêm vào.

Nếu code từ số 0, các công việc cần làm:

- Cài đặt express để có thể chạy website
- Cấu hình view engine (render view)
- Cấu hình typescript, nodemon để hỗ trợ coding
- Sử dụng mô hình MVC
- Code logic của ứng dụng:
    + Thực hiện CRUD users
    + Thực hiện authentication/authorization với Passport.js
    + Thực hiện CRUD với các tính năng còn lại

Bước 1: Phân tích nghiệp vụ bài toán

Từ nghiệp vụ của bài toán, thiết kế database (schema.prisma)

Lưu ý: cập nhật file .env để cắm backend vào database của bạn

Bước 2: Tiến hành coding

- Code view (ejs)
- Định nghĩa route
- Định nghĩa controller
- Định nghĩa service (thao tác với database)

#152. What's Next ?

Cơ chế được sử dụng từ video đầu tiên tới video #152, là SSR (Server Side Rendering)

1.Về backend Node.js

- Tìm hiểu cơ chế JWT (json web token) với passport để viết API
  (todo)

- Sử dụng các loại NoSQL, ví dụ MongoDB (mongoose)
  (todo)

- Sử dụng framework để tiết kiệm thời gian cấu hình, ví dụ như Nest.js

- Phát triển các chủ đề khác, ví dụ streaming, chat...

2. Về frontend

Sử dụng các library/framework chuyên nghiệp, giúp tiết kiệm thời gian coding và tối ưu hơn view engine, ví dụ React, Angular, Vue

//
Tham khảo khóa học Nest.js tại đây

Tham khảo khóa học React Ultimate tại đây

Chapter 14: RESTful APIs với SQL

Thực hành viết Restful API với database SQL

#153. Tổng quan về chapter

//todo

#154. API là gì ?

1.Nhược điểm của mô hình hiện tại SSR

Mô hình dự án hiện tại : SSR - Server side rendering

Client (send request) => Server render HTML (send response)

Do server phụ trách nhiệm vụ render ra view (giao diện), client phụ thuộc 100% vào server

Client: làm nhiệm vụ gửi request (ask)

Server: tính toán logic, kết nối với database, render ra giao diện (làm từ a tới z, sau đấy gửi lại kết quả cho client)

Bài toán đặt ra:
Frontend dùng React, Vue, Angular… làm giao diện website

Backend dùng Spring Boot, Node.js, Laravel… xử lý logic và giao tiếp với database

➡️Làm sao để hai phần này giao tiếp với nhau tạo nên 1 website hoàn chỉnh ?

2. API là gì ?

API viết tắt của **Application Programming Interface,** nghĩa là: Giao diện lập trình ứng dụng.

Nói đơn giản, API là cầu nối giúp các phần mềm, hệ thống, hoặc dịch vụ nói chuyện được với nhau.

Ví dụ thực tế: Gọi món ở quán cà phê = API
Bạn đến quán, ngồi vào bàn **(bạn là client**).

Bạn gọi: “Cho mình 1 ly trà sữa ít đá, thêm trân châu!” (**gửi yêu cầu**).

Nhân viên ghi order → chuyển cho bếp **(nhân viên là API).**

Bếp làm xong → giao đồ uống lại cho bạn (**phản hồi)**.

✅ Bạn không cần biết quầy bếp làm thế nào, chỉ cần nhận đúng đồ uống mình muốn.

➡️ API = Nhân viên order, nhận yêu cầu từ bạn và trả lại kết quả từ "hệ thống bên trong".

💻 Ví dụ lập trình: Gọi API để lấy danh sách người dùng

Client (trình duyệt, mobile app) gửi request:
GET https://myapp.com/api/users

Server phản hồi lại:
```
[
  { "id": 1, "name": "Alice" },
  { "id": 2, "name": "Bob" }
]
```

😎 **Lập trình viên thì thấy:**
GET /api/users là một API endpoint.

Gọi API = giống như hỏi: "Cho tôi danh sách người dùng"

Server trả kết quả về = dữ liệu bạn cần.

=> Về bản chất, **API là một URL được tạo nên bởi backend**.
Url này (endpoint sẽ được frontend sử dụng để lấy dữ liệu (data)

#155. Restful API là gì ?

RESTful API là một kiểu API được thiết kế theo kiến trúc REST (Representational State Transfer), giúp client (web, mobile...) và server giao tiếp với nhau **thông qua giao thức HTTP** một cách rõ ràng, logic và hiệu quả.

Chúng ta học Node.js (backend) viết API (restful api) là viết code tại server (nodejs), phục vụ cho client (web, mobile)...

Ví dụ trong thực tế
https://jsonplaceholder.typicode.com/

Các thành phần cấu thành Restful API:

1. Endpoint (URL)

- Là địa chỉ cụ thể mà client gửi yêu cầu đến.
- Mỗi endpoint tương ứng với một chức năng cụ thể.
- **Công việc tạo ra URL (API) này xảy ra tại backend**

Ví dụ:
GET /users → Lấy danh sách người dùng
POST /users → Thêm người dùng mới
GET /users/1 → Xem thông tin người dùng có ID = 1
DELETE /users/1 → Xoá người dùng có ID = 1

2.HTTP Method (Phương thức)

Method	Ý nghĩa	Ví dụ
GET	Lấy dữ liệu	Lấy danh sách người dùng
POST	Thêm mới dữ liệu	Thêm mới sản phẩm
PUT	Cập nhật toàn bộ	Cập nhật thông tin user
PATCH	Cập nhật một phần	Cập nhật một trường nhỏ
DELETE	Xóa dữ liệu	Xóa user

3. Request (Yêu cầu)

Là dữ liệu client gửi đến server, gồm:

- URL: Địa chỉ API
- Method: Phương thức HTTP (**mục 2 bên trên)**
- Headers: (tùy chọn) – chứa thông tin như loại dữ liệu, token xác thực...
- Body: (chỉ với POST, PUT, PATCH) – chứa dữ liệu gửi lên (**thường là JSON**)

Ví dụ JSON trong request body khi tạo mới user:

```
{
  "name": "Hỏi Dân IT",
  "email": "admin@hoidanit.vn"
}
```

4. Response (Phản hồi)

Khi server xử lý xong, nó trả kết quả lại cho client:

- Status code (200, 201, 400, 404, 500…)
  https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status
- Body (dữ liệu JSON, text…)
- Headers

Ví dụ Response JSON khi tạo mới user

```
{
  "id": 1,
  "name": "Hỏi Dân IT",
  "status": "created"
}
```

#156. GET Method

GET Method
Khi bạn truy cập vào một trang web, trình duyệt của bạn đang gửi một **HTTP GET** request.
===
Sử dụng **router.get** của Express Router để xử lý HTTP GET request.

Dùng khi nào?
- Khi bạn muốn lấy dữ liệu từ server (GET)

- Khi tạo REST API cho frontend hoặc mobile.

- Khi bạn chỉ muốn đọc thông tin, không làm thay đổi dữ liệu. (READ)

Ví dụ về **router.get**
//todo

#157. GET All Users API
//todo

Hướng dẫn sử dụng postman để test API
//todo

#158. Quy tắc đặt tên URL trong RESTful API

Mục tiêu: Việc đặt tên URL trong RESTful API đúng chuẩn giúp code dễ hiểu, dễ bảo trì và phù hợp với quy tắc REST.

RESTful API là gì ?

Là phong cách thiết kế API mà trong đó:

- Mỗi URL đại diện cho một tài nguyên (resource).

- Sử dụng đúng HTTP method (GET, POST, PUT, DELETE...).

1. Sử dụng danh từ số nhiều (plural nouns)

GET /**users** → Lấy danh sách người dùng
GET /**users**/1 → Lấy thông tin người dùng có ID = 1
POST /**users** → Tạo người dùng mới
PUT /**users**/1 → Cập nhật người dùng ID = 1
DELETE /**users**/1 → Xoá người dùng ID = 1

❌ Không dùng động từ trong URL (cách dùng sai):
❌ /getUserById/1
❌ /deleteUser/1

2. Không dùng CamelCase hay dấu gạch dưới

❌ /getAllUsers, /user_list
✅ /users
✅ REST **ưu tiên viết thường, dấu gạch ngang và dấu gạch chéo**
/articles/2024/latest-news

3. Dùng - (dấu gạch ngang) để phân cách từ

GET /user-accounts

4. Sử dụng query parameters cho filter, sort, pagination

GET /products**?category=phone&sort=price&page=2&limit=10**

#159. GET a User API

//todo

#160. POST Method

router.post dùng để xử lý HTTP POST request.

👉 Dùng khi bạn muốn gửi dữ liệu từ client (giao diện, frontend) lên server để:

- Tạo mới dữ liệu (Create)
- Gửi form, đăng ký, đăng nhập
- Gửi JSON từ frontend

req.body: lấy dữ liệu JSON từ request và convert thành javascript object.

2. Tại sao lại dùng POST, mà không truyền data lên url (GET) ?

1. Không an toàn (Security)

- URL có thể bị lưu lại trong lịch sử trình duyệt, file log, cache,...
- Password và dữ liệu nhạy cảm bị lộ dễ dàng!

  => Không nên gửi mật khẩu, số thẻ, token,... qua URL.

2. Giới hạn độ dài URL

- URL bị giới hạn (~2000 ký tự ở một số trình duyệt như Internet Explorer).
- Gửi dữ liệu lớn (nội dung bài viết, JSON, v.v) là không thể qua URL.

3. Không gửi được body (payload)

GET request không có phần body → không gửi JSON hoặc form-data được.

POST thì có thể gửi mọi định dạng: text, JSON, file, form,...

4. Không chuẩn REST khi tạo dữ liệu

#161. Create a User API

//todo

#162. Put Method

req.put được dùng để xử lý các HTTP PUT requests.

👉 Thường được dùng để cập nhật toàn bộ một tài nguyên đã tồn tại — ví dụ như cập nhật thông tin người dùng, sản phẩm, bài viết…

#163. Update a User API

//todo

#164. Phân Biệt PUT và PATCH

Khác biệt	Giải thích
RESTful logic	PUT = yêu cầu gửi toàn bộ object để ghi đè PATCH = chỉ gửi các field cần thay đổi
Code xử lý khác nhau	PUT thường map toàn bộ DTO vào entity PATCH phải kiểm tra từng field (nếu có)
Frontend truyền khác nhau	PUT gửi đầy đủ mọi field PATCH chỉ gửi 1-2 field cần sửa

//todo: code minh họa

#165. Delete Method

router.delete được dùng để xử lý HTTP DELETE request.
👉 Mục đích chính là để xóa một tài nguyên dựa trên ID hoặc một điều kiện nào đó.

#166. Delete User API

//todo

Chapter 15: JSON Web Token (JWT)

Áp dụng mô hình stateless với Restful APIs và JSON Web Token

#167. Tổng quan về chapter

//todo

#168. Authentication là gì?

Định nghĩa:
Authentication là quá trình xác minh danh tính của người dùng hoặc client.
Nó trả lời câu hỏi: "Bạn là ai?" (who are you ?)

Ví dụ thực tế:
Khi bạn đăng nhập vào một trang web (như Gmail), bạn cung cấp email và mật khẩu.
Hệ thống kiểm tra xem thông tin này có đúng không.

Nếu đúng, bạn được xác thực là người dùng hợp lệ.

Trong RESTful API, client (có thể là một ứng dụng frontend hoặc một thiết bị) gửi thông tin (như username/password hoặc token) để chứng minh danh tính.

Mục tiêu:

- Đảm bảo chỉ những người dùng hợp lệ mới có thể truy cập vào hệ thống hoặc các tài nguyên được bảo vệ (protected resources).

- Ngăn chặn truy cập trái phép từ người dùng không được phép.

#169. Authentication vs Authorization

Authentication và Authorization thường bị nhầm lẫn, nhưng chúng là hai khái niệm khác nhau. Hiểu rõ sự khác biệt sẽ giúp bạn thiết kế hệ thống tốt hơn.

1.Authentication (Xác thực)
Câu hỏi: "Bạn là ai?" (who are you ?)

Mục đích: Xác minh danh tính người dùng.

Ví dụ:
Bạn nhập username và password để đăng nhập.
API kiểm tra xem token gửi lên có hợp lệ không.

Kết quả: Nếu xác thực thành công, hệ thống biết bạn là ai (ví dụ: user có ID là 123).

Mã lỗi: 401 (Unauthorized)

2. Authorization (Phân quyền)
Câu hỏi: "Bạn được làm gì?" (what can you do ?)

Mục đích: Xác định quyền truy cập của người dùng sau khi đã xác thực.

Ví dụ:

- Bạn là user thường nên chỉ được xem dữ liệu của mình, không được xóa dữ liệu của người khác.
- Bạn là admin nên có quyền xóa dữ liệu.

Kết quả: Sau khi xác thực, hệ thống kiểm tra quyền (role/permission) của bạn để quyết định bạn có được truy cập tài nguyên hay không.

Mã lỗi: 403 (Forbidden)

3. Mối quan hệ
Authentication luôn diễn ra trước Authorization.

Bạn phải được xác thực (biết bạn là ai) trước khi hệ thống kiểm tra quyền của bạn (Authorization).

#170. Các phương pháp Authentication phổ biến

1. Basic Authentication

Cách hoạt động:

- Client gửi username và password trong header của mỗi request.
- Header được mã hóa dạng Base64 (không phải mã hóa an toàn, chỉ là che giấu).

Ví dụ header: Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=

Ưu điểm: Đơn giản, dễ triển khai.

Nhược điểm:

- Không an toàn (dễ bị tấn công nếu không dùng HTTPS).
- Client phải gửi username/password trong mỗi request, gây tốn tài nguyên.

Khi nào dùng?

- Chỉ dùng trong các ứng dụng nhỏ hoặc môi trường thử nghiệm, và phải dùng HTTPS.

2. Session-based Authentication (Stateful) - Server side rendering

Cách hoạt động:

- Người dùng đăng nhập, server tạo một session ID và lưu nó trên server (thường trong database hoặc memory).
- Session ID được gửi cho client (thường lưu trong cookie).
- Client gửi session ID trong mỗi request để chứng minh danh tính.

Ưu điểm:

- Dễ triển khai trong ứng dụng web truyền thống.
- Có thể thu hồi session (logout) dễ dàng bằng cách xóa session trên server.

Nhược điểm:

- Cần lưu trạng thái trên server (stateful), gây khó khăn khi mở rộng (scaling) hệ thống.
- Không phù hợp cho API dùng trong các ứng dụng phân tán (như mobile app).

3. Token-based Authentication (dùng JWT)

Cách hoạt động:

- Người dùng đăng nhập, server tạo một token (như JWT) và gửi cho client.
- Client lưu token (thường trong localStorage hoặc cookie) và gửi token trong header của mỗi request (thường là Authorization: Bearer <token>).
- Server kiểm tra token để xác thực người dùng.

Ưu điểm:

- Stateless (không cần lưu trạng thái trên server), phù hợp để mở rộng hệ thống.
- Dễ dùng cho API, đặc biệt với ứng dụng mobile hoặc hệ thống phân tán.
- Token có thể chứa thông tin (như user ID, role) trong payload.

Nhược điểm:

- Không thể thu hồi token ngay lập tức (trừ khi dùng blacklist).
- Cần quản lý thời gian hết hạn (expiration).

Khi nào dùng?

Đây là phương pháp phổ biến nhất cho RESTful API, đặc biệt khi dùng JWT.

#171. JSON Web Token (JWT)
https://jwt.io/
Mục tiêu: cần tạo ra token để xác thực người dùng

Giải pháp: sử dụng tiêu chuẩn JWT

1.JSON Web Token là gì ?
JSON Web Token (JWT) là một **chuẩn mở (RFC 7519)** giúp đại diện cho yêu cầu giữa hai bên (client và server) dưới dạng một chuỗi JSON được mã hóa, nhằm xác thực và trao đổi thông tin an toàn.

- **JWT là một token nhỏ, gọn, chứa thông tin đã được mã hóa**, giúp xác thực người dùng và quản lý phiên trong các ứng dụng web và API.

- JWT thường được sử dụng trong các API để xác thực người dùng thay cho cách quản lý session truyền thống.

2. Lý do sử dụng JWT trong việc bảo mật API

JWT giúp xác thực không trạng thái (stateless):
Điều này có nghĩa là server không cần lưu trữ trạng thái của người dùng (**session**), mỗi yêu cầu HTTP đều mang theo thông tin xác thực (token), giúp giảm bớt tài nguyên server và tăng tính mở rộng.

Bảo mật:
JWT sử dụng mã hóa và chữ ký để đảm bảo tính toàn vẹn và xác thực của thông tin.

JWT giúp đảm bảo rằng dữ liệu không bị thay đổi trong quá trình truyền tải và chỉ những người có secret key hoặc public key mới có thể xác minh được token.

Tính linh hoạt:
JWT có thể chứa các claim (thông tin người dùng) mà bạn có thể truy xuất trực tiếp từ token mà không cần phải gọi cơ sở dữ liệu mỗi lần kiểm tra xác thực.

Tính di động:
JWT có thể được sử dụng trong các môi trường phân tán như microservices hoặc single-page applications (SPA) mà không cần phải duy trì session server-side.

3. Cấu trúc của JWT

Một JWT bao gồm 3 phần chính, mỗi phần được ngăn cách bằng dấu chấm (.):

- **Header**: Thông tin về thuật toán mã hóa.
- **Payload**: Dữ liệu (claims) thực tế.
- **Signature**: Đảm bảo tính toàn vẹn của token.

#172. Mô Hình Áp Dụng JWT và RESTful API

Bài toán: phát triển hệ thống bao gồm:

- **Frontend** (React.js)
- **Backend** (Node.js/Express)
- **Sử dụng RESTful APIs** để trao đổi dữ liệu giữa frontend và backend

Do frontend và backend không code chung 1 source code, và thuộc 2 môi trường khác nhau, vì vậy, không thể sử dụng session (như cách đã làm với dự án laptopshop)

=> sử dụng token (mã hóa dưới dạng JWT - JSON Web Token)

Bước 1: Tại giao diện Frontend (React), tạo form login, bao gồm username/password

Nhấn nút submit => gọi API login

Bước 2: Backend xử lý logic login

- Kiểm tra tính hợp lệ của username/password client truyền lên

- **Trả ra token định danh người dùng (đặt tên là access_token)**. Token này được viết theo tiêu chuẩn JWT (json web token)

Bước 3: Frontend nhận được access_token sau khi đã login, sử dụng token này trong các lời gọi API để truy cập nguồn tài nguyên của server.

Ví dụ: để truy cập **GET /users**

Frontend sẽ cần đính kèm token ở header (dưới định dạng Bearer Token)

Bước 4: Tại backend, cần có cơ chế xử lý (middleware), để kiểm tra tính hợp lệ của token gửi lên

Nếu hợp lệ, cho phép truy cập nguồn tài nguyên

Không hợp lệ, từ chối truy cập

#173. API Login

1. Cài đặt JWT
https://www.npmjs.com/package/jsonwebtoken

npm i --save-exact jsonwebtoken@9.0.2
npm i --save-exact --save-dev @types/jsonwebtoken@9.0.9

2. Tạo ra access token
//todo

3. Check access token
//todo: kiểm tra data chứa trong token, thời gian hết hạn của token

#174. JWT Middleware

Ý tưởng:

- Sau khi user login thành công, đã có được access_token (JWT)

- Mỗi request truy cập API, cần gửi token này ở header

1. Đưa các tham số vào file môi trường
//todo

2. Payload của token nên chứa data gì
//todo

3.Viết jwt middleware
//kiểm tra token trong mỗi lời gọi request

Bước 1: Lấy token từ header

```
const token = req.headers['authorization']?.split(' ')[1]; // format: Bearer <token>
```

Bước 2: Verify token
//todo

#175. Verify Token

//verify token

//gán thông tin user vào req.user

#176. API Fetch Account

//bảo vệ tất cả endpoint với jwt middleware
//check theo role (sau khi đã có token)
//viết api fetch account

#177. Fix Lỗi CORS
Tài liệu: https://www.npmjs.com/package/cors

Lưu ý: CORS chỉ xảy ra tại phía frontend (browser), không xảy ra đối với server

Cài đặt:
npm i --save-exact cors@2.8.5
npm i --save-exact --save-dev @types/cors@2.8.17

#178. Test Full Dự Án (Frontend+Backend)

1.Trước khi thực hành

- Bạn đã xem và cài đặt Node.js như hướng dẫn tại video #10, như vậy máy tính bạn sẽ có môi trường Node.js để chạy code Frontend

- Bạn không bắt buộc phải biết code Frontend, mình sẽ cung cấp sẵn full source code frontend để bạn test full dự án

  Các video tiếp theo sẽ hướng dẫn code frontend theo từng tính năng cụ thể

2.Thực hành

Bước 1: Download source code frontend (final) tại đây

Bước 2: Build dự án frontend

- Update file .env (hoặc file api.ts) nếu backend chạy trên port khác 8080
- Cài đặt thư viện: **npm i**
- Build dự án: **npm run build**
- Chạy dự án frontend tại chế độ build: **npm run preview**

Tại bước 2 này, cần kiểm tra xem giao diện có bị lỗi CORS hay không ?

Bước 4: Test full Frontend/Backend
//todo

Bước 5: Rollback code

Nếu bạn code theo cách video tiếp theo, hãy xóa đi phần code thay đổi (nếu có), như vậy nó sẽ đảm bảo là code của bạn và code trong video hướng dẫn tại các video tiếp theo, là giống nhau

#179. Setup dự án thực hành Frontend

1.Môi trường thực hiện
Đảm bảo rằng máy tính bạn đã cài đặt Node.js v**22.13.0**

Nếu bạn chưa cài đặt Node.js, tham khảo video #10

2.Yêu cầu trước khi thực hành

Cần phải có hiểu biết cơ bản về React và TypeScript.

Nếu bạn là beginners, chưa từng học React, có thể tham khảo:

- Khóa học React Ultimate tại đây
- Khóa học React Portfolio tại đây
- Lộ trình học React từ a tới z tại đây

3.Dự án thực hành

Download/Clone dự án thực hành tại đây

4.Giải thích về cấu trúc dự án thực hành
//todo

Lưu ý về bản quyền source code (license) khi sử dụng ?

#180. Frontend: Chia Layout

1.Cài đặt thư viện

Sử dụng chính xác:
npm i --save-exact antd@5.24.7 @ant-design/icons@6.0.0 react-router@7.5.1

- Làm UI với antd: https://ant.design/
- Điều hướng trang (router) với React router: https://reactrouter.com/home

2.Chia base layout

Lưu ý: sử dụng antd v5 và react-router v7

Mục tiêu: /trang chủ và /users

Bước 1: chia base component theo routes
https://reactrouter.com/start/data/routing

//todo

Bước 2: tạo UI
//todo
https://ant.design/components/menu

#181. Frontend: Design Login

Tài liệu:
https://ant.design/components/form

https://axios-http.com/docs/intro
https://www.npmjs.com/package/axios

Cài đặt
npm i --save-exact axios@1.8.4

//todo

#182. CORS là gì ?

1.Nguồn gốc (Origin) là gì?

Một origin được định nghĩa bởi sự kết hợp của **protocol** (giao thức như http, https), **domain** (tên miền như example.com), và **port** (cổng như 80, 3000).
Ví dụ:
http://example.com và **https://example.com** là hai origin khác nhau (khác protocol).

http://example.com:3000 và **http://example.com:4000** là hai origin khác nhau (khác port).

http://example.com và **http://sub.example.com** cũng là hai origin khác nhau (khác domain).

Khi một trang web ở một origin (ví dụ: **http://frontend.com**) gửi yêu cầu tới một origin khác (ví dụ: **http://api.com**), trình duyệt sẽ áp dụng chính sách **Same-Origin Policy** (chỉ cho phép yêu cầu trong cùng origin).

CORS là cách để nới lỏng chính sách này một cách an toàn.

2.CORS là gì ?

CORS (Cross-Origin Resource Sharing) là **một cơ chế bảo mật được tích hợp trong các trình duyệt web,** cho phép hoặc hạn chế các yêu cầu HTTP từ một nguồn gốc (origin) này tới một nguồn gốc khác.

Nó được thiết kế để bảo vệ người dùng khỏi các cuộc tấn công nguy hiểm như truy cập trái phép dữ liệu từ các trang web không đáng tin cậy, đồng thời vẫn cho phép các ứng dụng web hiện đại hoạt động linh hoạt.

Ví dụ về tác dụng của CORS:

Ví dụ : Bảo vệ người dùng khỏi tấn công CSRF (Cross-Site Request Forgery)

Bối cảnh:
Bạn đăng nhập vào trang ngân hàng **https://bank.com** và có một cookie phiên (session cookie) để xác thực.

Một trang web độc hại **http://evil.com** cố gắng gửi yêu cầu đến **https://bank.com/transfer?amount=1000&to=attacker** để chuyển tiền từ tài khoản của bạn.

Trường hợp 1: Không có CORS:
Trình duyệt tự động gửi cookie của **bank.com** cùng với yêu cầu từ **evil.com** vì cookie được lưu trong trình duyệt.

Nếu **bank.com** không kiểm soát origin, yêu cầu này có thể thành công, dẫn đến việc kẻ tấn công đánh cắp tiền của bạn mà bạn không hề hay biết.

Trường hợp 2: Với CORS:
Trình duyệt gửi **header Origin: http://evil.com** trong yêu cầu tới **bank.com.**

Server của **bank.com** kiểm tra và thấy **http://evil.com** không nằm trong danh sách Access-Control-Allow-Origin (chỉ cho phép **https://bank.com**).

Server từ chối yêu cầu bằng cách không trả về header Access-Control-Allow-Origin phù hợp hoặc trả về lỗi.

Kết quả: Yêu cầu bị chặn, dữ liệu của bạn được bảo vệ.

=> Muốn fix lỗi CORS, cần thực hiện tại backend (frontend chỉ dùng và không cần quan tâm tới lỗi này)

#183. Backend: Fix Lỗi CORS

//todo

#184. Frontend: Tính Năng Login

- Login thành công, lưu token vào localStorage, redirect về trang chủ

- Thêm phần hiển thị thông tin ở header
  https://ant.design/components/menu

  //todo

#185. Frontend: React Context

Mục tiêu: chia sẻ data giữa các component bằng cách sử dụng React Context

1.Sử dụng React Context

https://react.dev/learn/passing-data-deeply-with-context

https://react.dev/reference/react/useContext

https://codemod.com/registry/react-19-remove-context-provider

//sử dụng với typescript
https://react.dev/learn/typescript#typing-usecontext

//áp dụng chia sẻ access_token, user data...

#186. Frontend: API Fetch Account
//todo

#187. Frontend: Hiển Thị Loading

//loading : minh họa với delay của network
https://ant.design/components/spin

Center div: https://stackoverflow.com/a/22278778

//phần header động cho đăng nhập và chưa đăng nhập

//tính năng logout

#188. Frontend: Hoàn Thiện CRUD Users

- Viết crud users
- Hoàn thiện homepage
- Add footer

Download phần source code thực hành ứng video này tại đây

#189. Nhận Xét về Ưu/Nhược Điểm của JWT

Ưu điểm:

1. **Không cần lưu trạng thái (Stateless)**

- Server không cần lưu phiên làm việc (session) – mọi thông tin xác thực nằm trong token.
- Phù hợp cho kiến trúc microservices hoặc serverless.

2. **Dễ dàng mở rộng hệ thống (Scalability)**

- Vì không lưu session, các request có thể đến bất kỳ server nào mà không cần chia sẻ session storage.

- JWT có thể chứa thông tin người dùng và các quyền trong payload => giúp giảm số lần truy vấn cơ sở dữ liệu.

- Hỗ trợ tốt cho Single Sign-On (SSO): JWT được thiết kế để làm việc tốt trong hệ thống nhiều dịch vụ cần xác thực chung.

Nhược điểm:

1. **Không thể thu hồi token dễ dàng:**
   Nếu không có cơ chế blacklist, JWT sẽ hợp lệ cho đến khi hết hạn – nguy hiểm nếu bị rò rỉ.

2. **Không thể cập nhật thông tin người dùng dễ dàng:**
   Ví dụ: Nếu người dùng thay đổi role, JWT cũ vẫn hợp lệ trừ khi ép buộc đăng xuất.

3. **Phải bảo mật cẩn thận:**

- **Nếu lưu JWT ở localStorage** => dễ bị tấn công **XSS**.
- **Nếu lưu ở cookie** => dễ bị tấn công **CSRF** nếu không cấu hình đúng.
- Không thích hợp cho thông tin nhạy cảm: dù payload có thể mã hóa (nếu dùng JWE), phần lớn JWT chỉ được ký (JWS), chứ không mã hóa => có thể đọc được nếu không mã hóa.

Kết luận:

JWT rất phù hợp với các ứng dụng RESTful, microservices, và hệ thống cần SSO.

Tuy nhiên, với các hệ thống cần khả năng thu hồi token nhanh hoặc chứa thông tin nhạy cảm, JWT cần được sử dụng cẩn trọng hoặc kết hợp với các biện pháp bổ sung như token blacklist, refresh token,...

#190. Các Bước Phát Triển Tiếp Theo

Gợi ý hướng tự thực hành:

- Tự làm dự án thực hành, convert phần admin của web laptopshop sang thành giao diện React

- Protected route theo role

- Viết cơ chế refresh token

- Tìm hiểu thêm về các cơ chế bảo mật của jwt: lưu ở localStorage/ cookies