1. **Chuyển thư mục trong cmd để chạy JavaScript:**

**Cách 1:**

E:

cd PhatTrienUngDungWeb\BaiTap\JavaScript

node tinhToanCanBan.js

**Giải thích:**

* E: → Chuyển sang ổ **E**
* cd PhatTrienUngDungWeb\BaiTap\JavaScript → Di chuyển đến thư mục **chứa file**
* node tinhToanCanBan.js → Chạy file bằng **Node.js**

**Cách 2:**

node "E:\PhatTrienUngDungWeb\BaiTap\JavaScript\tinhToanCanBan.js"

**Lưu ý:**

* Đường dẫn có khoảng trắng thì đặt trong **dấu ngoặc kép** "..."
* Kiểm tra lại tên file có đúng không (phải là tinhToanCanBan.js, không có lỗi .js.js hoặc .txt)

***\*Xóa lịch sử code vửa chạy: cls***

**Hàm mũi tên:**

Có 3 cách để viết hàm mũi tên:

- [1] Không tham số: () => { ... }

- [2] Một tham số: x => { ... } (không cần dấu ngoặc cho một tham số)

- [3] Nhiều tham số: (x, y) => { ... }

1. **Nginx: Lắng nghe trên cổng 65536 được không? Tại sao**

Không thể lắng nghe trên cổng 65536

Lý do:

* Cổng hợp lệ trong TCP/UDP chỉ nằm trong khoảng 0 - 65535.
* 65536 là số không hợp lệ vì nó vượt quá giới hạn của hệ thống mạng.
* Khi cố gắng sử dụng listen 65536;, Nginx sẽ báo lỗi "invalid port".

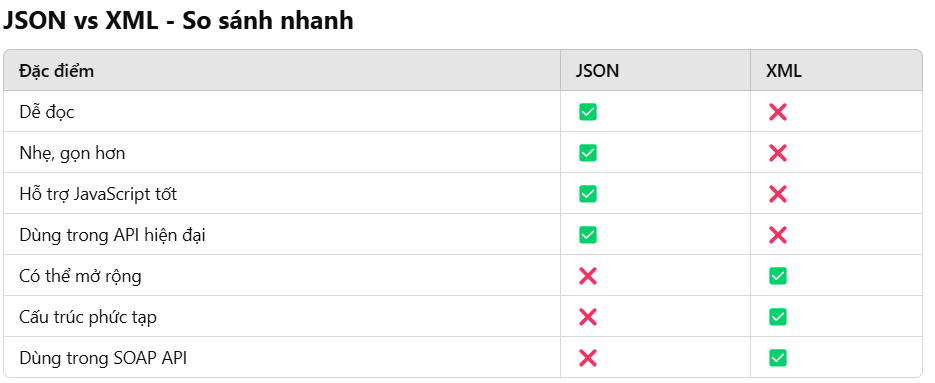
1. **Tổng quan về git:**

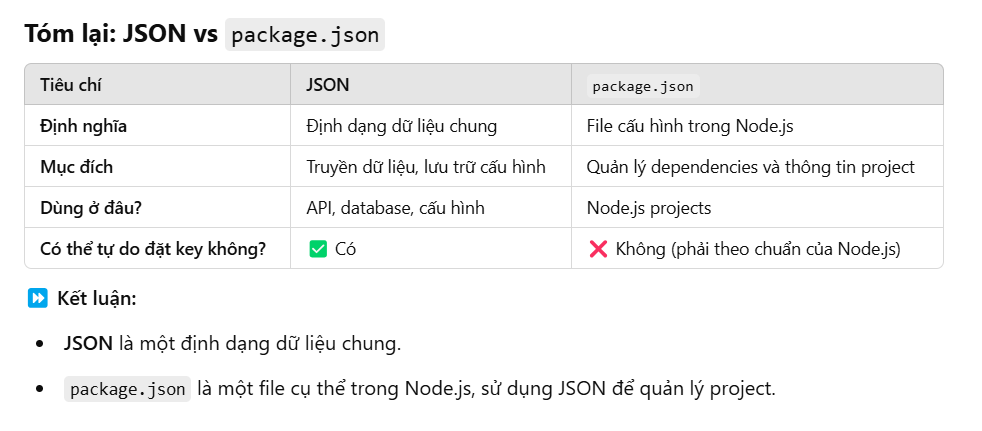
Khởi tạo git: git init (tạo kho chứa)

1. **Nodejs và NPM**

Xem phiên bản của Nodej: node -v

Xem phiên bản của NPM: npm -v







Tắt nodejs: Ctrl + C

Lệnh: ***npm init -y*** được sử dụng để tự động tạo một tập tin package.json trong dự án Node.js mà không cần phải trả lời các câu hỏi từng bước.

{

"name": "ptudw",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"keywords": [],

"author": "",

"license": "ISC",

"dependencies": {

"express": "^5.1.0"

},

"devDependencies": {},

"description": ""

}

Trong đó:

* name: Tên thư mục hiện tại (dự án) sẽ được sử dụng làm tên của dự án.
* version: Phiên bản mặc định là 1.0.0.
* main: Tên file chính (tập tin đầu vào) của ứng dụng là index.js (nếu không có, bạn có thể thay đổi).
* scripts: Chứa một số script mặc định, ví dụ test, có thể tùy chỉnh sau.
* license: Mặc định là ISC.
* description: Trống.

Cài đặt lại Express bằng npm: *npm install express*

Kiểm tra lại thư mục node\_modules: *dir node\_modules*

Chạy lại ứng dụng: *node index.js* -> thành công: server dang chay tren cong 9000

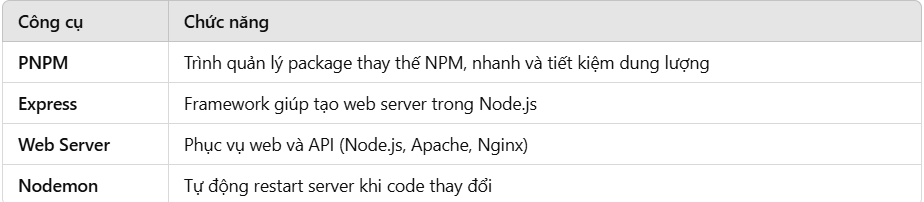
Chạy trên trình duyệt: [*http://localhost:9000/*](http://localhost:9000/)

1. **Nodemon**

Cài đặt Nodemon: *npm install --g nodemon*

Chạy nodemon: *nodemon*

**Kết luận:**



1. **Local và Global**
2. **Cài đặt một gói kiểu cục bộ (local)**

****

**VD:**

Cài đặt gói Express theo kiểu cục bộ: npm install express

Gỡ bỏ một gói cục bộ: npm uninstall express

**\*Chỉ có cài gói theo kiểu cục bộ thì mới được cập nhật vào package.json.**

1. **Cài đặt một gói kiểu toàn cục (global)**

Để cài đặt một gói kiểu toàn cục, sử dụng lệnh npm install có thêm tham số -g

npm install <package\_name> -g

Vd: Cài đặt gói nodemon theo kiểu toàn cục: install nodemon -g

+ Để biết các gói toàn cục được lưu trữ ở đâu: npm root -g

+ Xem các gói được cài đặt theo kiểu toàn cục: npm ls -g --depth 0

Trong đó,

- npm ls: để liệt kê các gói đã được cài đặt trên máy, ls là viết tắt của list

- -g là cờ báo, để chỉ hiển thị các gói kiểu toàn cục (global)

-  --depth 0: chỉ hiển thị các gói toàn cục cấp cao nhất (top-level), không hiển thị các gói là con-cháu (sub-dependencies) của gói cấp cao nhất

***npm init -y -> tạo tập tin tin package.json***

***npm install -g pnpm@latest-10 -> cài đặt phiên bản npm mới nhất***

***pnpm i -s express -> Cài đặt framework Express***

***npm install --g nodemon -> Cài đặt Nodemon***

***npm root -g -> vị trí lưu trữ gói toàn cục***

***npm ls -g --depth 0 -> xem các gói được cài đặt theo kiểu toàn cục***

***npm install <package-name> --save-dev -> cài đặt một gói là devDependencies***

1. **Tổng quan về git**

Khởi tạo git, nhúng vào thư mục của dự án: git init

***Cấu hình định danh người dùng:***

Lệnh git config được sử dụng để đọc, ghi hoặc cập nhật các thiết lập cấu hình của git

*--system: Cấu hình toàn hệ thống (tất cả người dùng, tất cả repo), Áp dụng cho tất cả người dùng, mọi repo Git trên máy.*

VD: git config --system user.name “Quoc Lam”

*--global: cấu hình cho toàn bộ người dùng hiện tại. Áp dụng cho toàn bộ repo Git mà bạn sử dụng với tài khoản hiện tại.*

VD: git config --global user.name “Quoc Lam”

+ git config –global user.email “nguyenlequoclam@gmial.com”

Thư mực:  trong C:\Users\<tên\_người\_dùng> sau đó tìm file .gitconfig

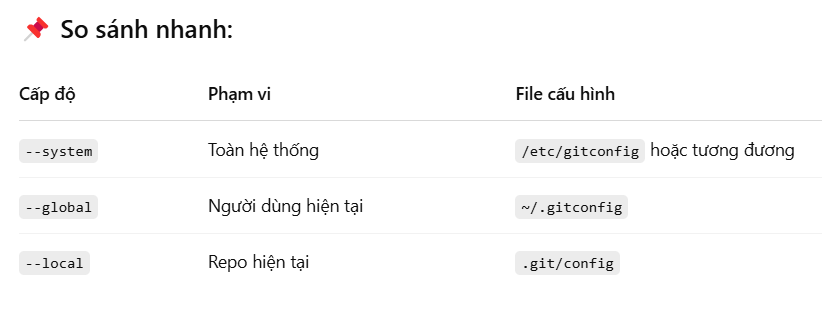
*--local: (mặc định) cấu hình cho repo hiện tại. Áp dụng chỉ trong repo Git hiện tại. Nếu không chỉ rõ cấp độ, Git sẽ mặc định dùng --local.*

VD: git config user.name "Tên riêng cho repo này"

+ git config user.email "email-rieng@example.com"

Thư mục: file .git trong thư mục dự án

***\*Lưu ý****: Trước khi chạy cần phải chạy dưới quyền admin: Windows -> cmd -> Bấm chuột phải vào tiện ích Command Prompt -> Chọn Run as administrator*

**



**Xem thông tin người dùng:**

Lệnh: git config –list

**Xóa định danh người dùng:**

git config --global --unset user.name

git config --global --unset user.email

Tương tự, bạn có thể xóa cấu hình ở phạm vi local hoặc system bằng cách bỏ qua hoặc thay thế --global bằng --local hoặc --system.

**Thay đổi định danh người dùng:**

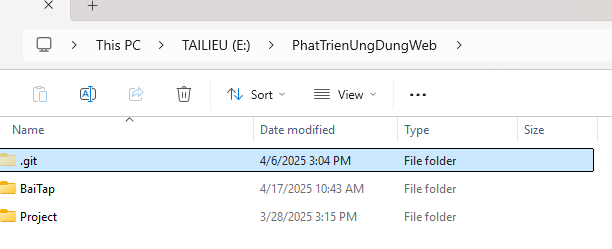
git config --global user.name "Tên mới của bạn"

git config --global user.email "email\_mới@example.com"

Lệnh này sẽ ghi đè lên cấu hình cũ.

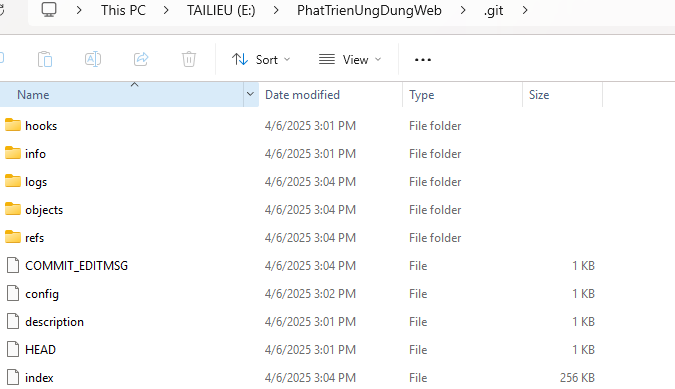
Tương tự, bạn có thể thay đổi cấu hình ở mức local hoặc system bằng cách bỏ qua hoặc thay thế --global bằng --local hoặc --system.

1. **Các khu vực làm việc của Git**

****

Kho chứa (repository): file .git

Thư mục làm việc (working directory): flie BaiTap, Project



Khu vực tổ chức tạm (staging area): file index

***\*Các lệnh của Git***

* Kiểm tra trạng thái: git status
* Thêm thay đổi vào staging area: git add .

- git add <tập\_tin>: thêm một tập tin cụ thể.

- git add .: thêm tất cả thay đổi.

- git add -p: thêm từng phần thay đổi (patch) trong tập tin.

* Commit thay đổi: git commit -m "text"

Để gỡ bỏ tập tin ra khỏi khu tạm:

git rm --cached <tên\_tập\_tin>

Hiển thị lích sử commit: git log

*'use strict'*

*const express = require('express')*

*const app = express();*

*const port = process.env.PORT || 9000*

*// xu ly khi nguoi dung gui request toi web server*

*app.get("/", (req, res) => {*

*const ten = req.query.name;*

*const lop = req.query.lop;*

*res.send(`Tên của tôi là ${ten} và học lớp ${lop}`);*

*});*

*// khoi dong web server*

*app.listen(port, () => {*

*console.log(`server dang chay tren cong ${port}`);*

*});*

http://localhost:9000/?name=NguyenLeQuocLam&lop=CTK46B

protocol (giao thức): http

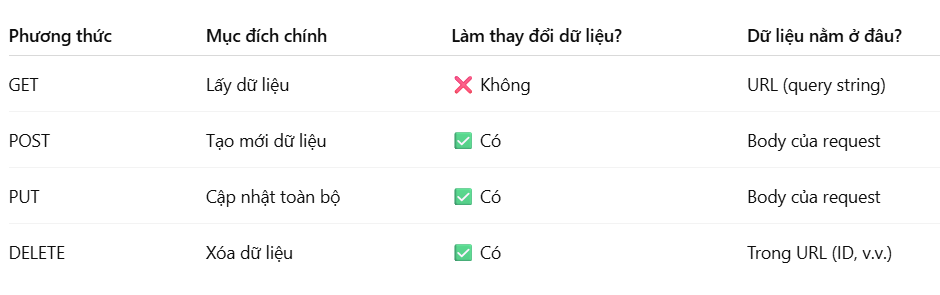
domain (tên miền): localhost

port (cổng): 9000

path (đường dẫn đến 1 vị trí trên server): / (là thư mục gốc).

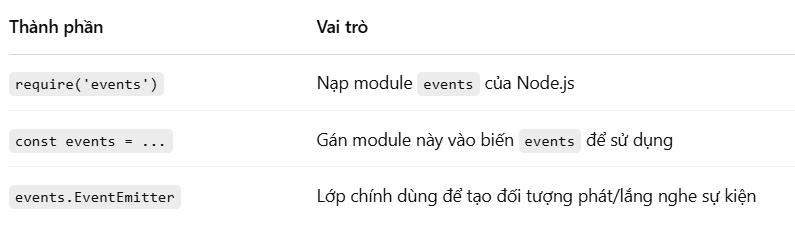
query (truy vấn): ?name=NguyenLeQuocLam&lop=CTK46B (chuỗi các tham số được sử dụng để truyền dữ liệu đến server).

***Lưu ý: cổng mặc định của http là 80 và https là 443 nhưng khi phát triển ứng dụng cục bộ nên chọn cổng 3000, 5000, 8000 để tránh bị xung đột.***

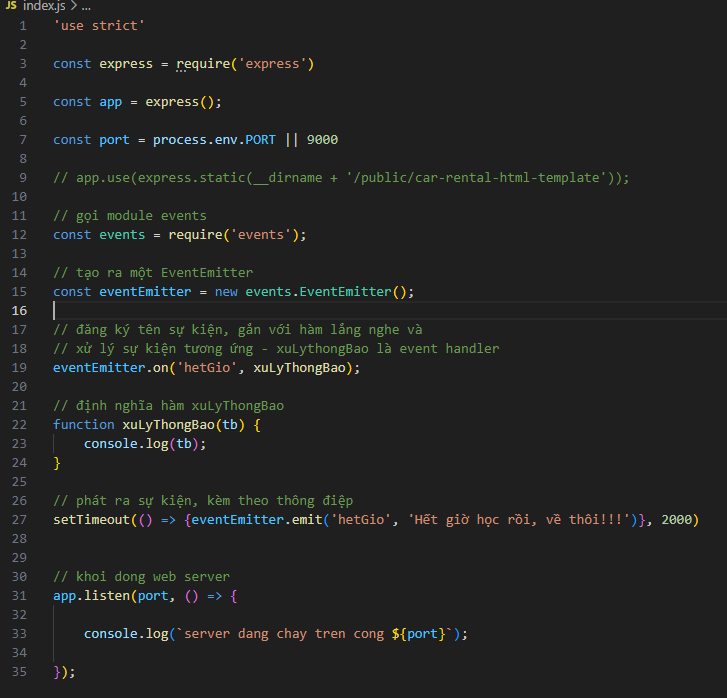


1. **Lập trình hướng sự kiện**

const events = **require**('events');



Giải thích chi tiết



* **EventEmitter** – Bộ phát sự kiện

const events = require('events');

const eventEmitter = new events.EventEmitter();

* **Event Handler** – Bộ xử lý sự kiện

eventEmitter.on('hetGio', xuLyThongBao);

function xuLyThongBao(tb) {

console.log(tb);

}

* **Event** – Sự kiện xảy ra

setTimeout(() => {

eventEmitter.emit('hetGio', 'Hết giờ học rồi, về thôi!!!')

}, 2000)

* **Event Loop** – Vòng lặp sự kiện

Bạn không thấy rõ Event Loop trong code vì nó là **cơ chế nội tại của Node.js**, nhưng nó đang hoạt động ngầm để:

* Chờ sự kiện xảy ra
* Kích hoạt callback tương ứng

1. **Route handler và Middleware trong Express**

**Route handler**

Route handler là một hàm (function) trong Express được sử dụng để xử lý một request khi một route cụ thể được truy cập. Nó là phần thực thi thực tế của ứng dụng khi client gửi yêu cầu đến một URL cụ thể (cùng với phương thức HTTP như GET, POST, PUT, DELETE,...).

**Cấu trúc Route Handler trong Express**

app.METHOD(PATH, callback)

Trong đó:

* **app.METHOD**: Phương thức HTTP (GET, POST, PUT, DELETE, v.v.). Ví dụ: app.get(), app.post(), app.put(), v.v.
* **PATH**: Đường dẫn của route (ví dụ: /home, /login, /users/:id).
* **callback**: **Route handler** (hàm xử lý yêu cầu). Đây là hàm chứa logic xử lý yêu cầu từ người dùng và trả về phản hồi.

Giải thích:

* app.get() – Dùng để trả lời khi người dùng **truy cập trang web**

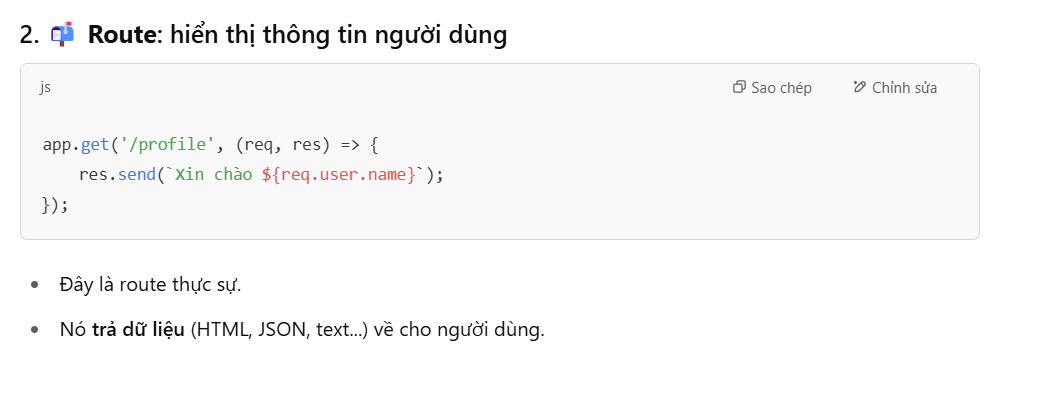
Giống như:  
 Ai đó gõ đường link vào trình duyệt → bạn muốn trả lời gì?

* app.use() – Dùng để **làm việc gì đó trước khi xử lý chính (middleware)**

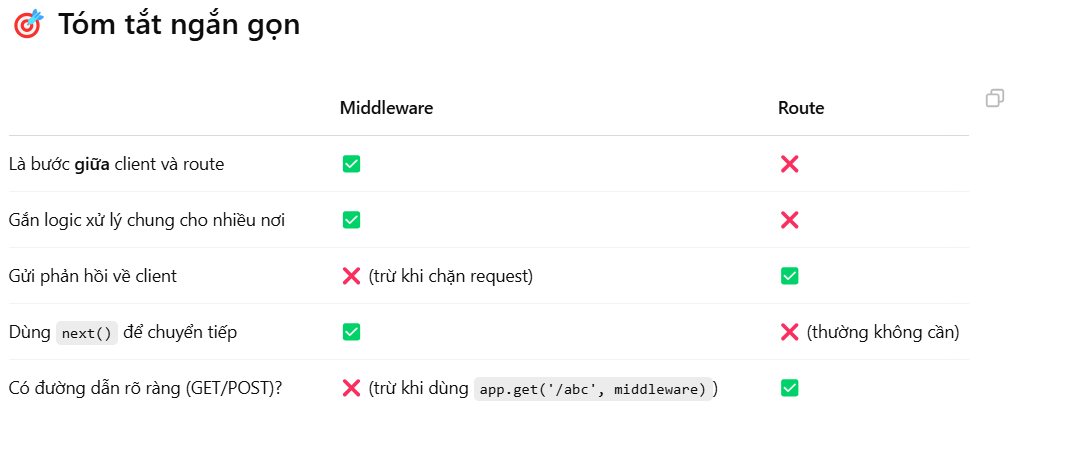
Giống như:  
 Người ta gõ cửa → bạn kiểm tra danh tính trước khi cho họ vào phòng.

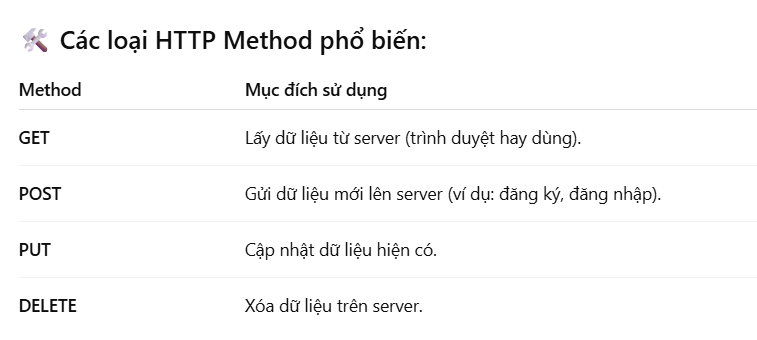
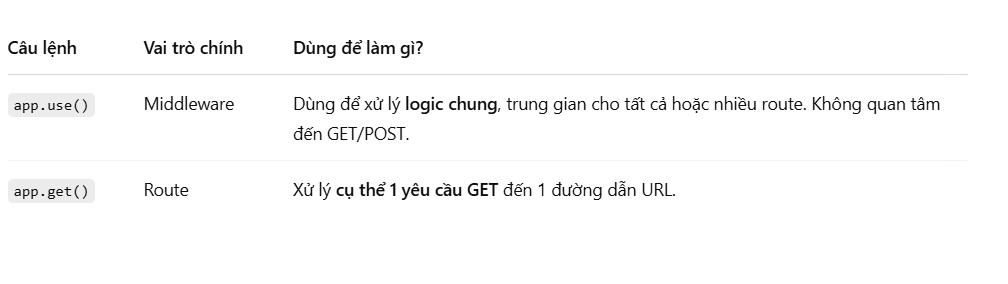




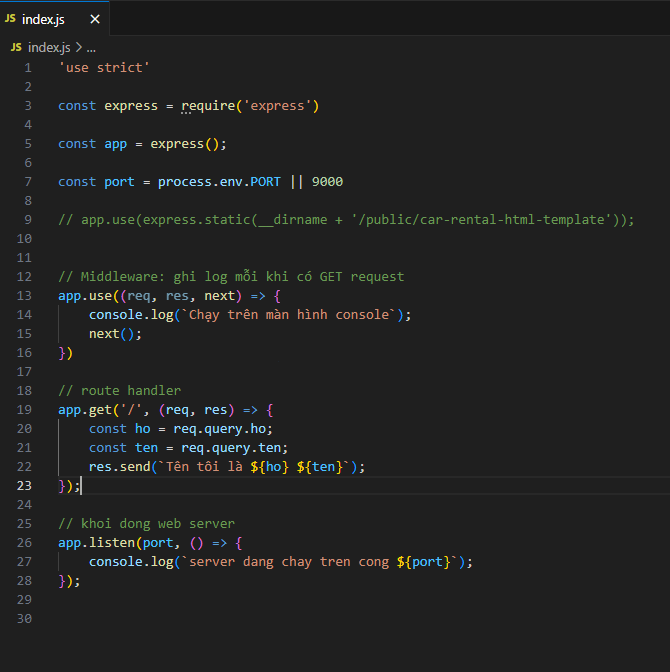








***Lưu ý:*** Khi sử dụng Express, thứ tự khai báo middleware và route handler rất quan trọng. Middleware phải được khai báo trước route handler để có thể thực thi trước khi yêu cầu được xử lý và trả về phản hồi.



Luồng xử lý:

B1: Khi chạy nodemon Express khởi động web server và lắng nghe yêu cầu của cổng 9000. Khi bắt đầu chạy sẽ hiện ra màn hình console: server dang chay tren cong 9000

B2: Từ màn hình trình duyệt (client) gửi yêu cầu GET tới URL: <http://localhost:9000/?name=Lâm>

B3: Khi có yêu cầu GET là URL tời giao diện người dùng (client), middleware được gọi trước tiên để xử lý yêu cầu 0 -> xử lý console.log(…) in ra console -> và gọi next() để sau đó chuyển yêu cầu tiếp tục cho các middleware hoặc route handler.

B4: Sau khi middleware chạy xong, Express sẽ kiểm tra xem có route handler nào phù hợp với yêu cầu không. Ở đây, bạn có một route handler cho đường dẫn / (tức là yêu cầu GET tới /): -> **Phù hợp ->** Sau đó trình duyệt (client) sẽ nhận lại phản hồi

B5: Sau khi gửi phản hồi cho client, server sẽ tiếp tục lắng nghe các yêu cầu tiếp theo và xử lý chúng theo cùng một cách.

1. **Lập trình trang web động**

Có 3 cách để tạo ra giao diện cho trang web:

[1] Tạo giao diện tại phía server

[2] Tạo giao diện tại phía client

[3] Tạo giao diện hỗn hợp, gồm cả ở phía server và phía client

**Tạo giao diện tại phía server (server-side rendering SSR)**

Tạo giao diện tại phía server (server-side rendering - SSR) là kỹ thuật tạo ra nội dung trang web (rendering) bằng cách xử lý logic và tạo ra mã HTML hoàn chỉnh trên máy chủ (server side) trước khi gửi về cho trình duyệt (client).

Cách hoạt động SSR:

1. Người dùng mở website (gửi HTTP request tới server).
2. Máy chủ nhận yêu cầu → thực hiện xử lý logic, lấy dữ liệu từ CSDL (nếu cần).
3. Máy chủ tạo file HTML hoàn chỉnh chứa nội dung.
4. Trình duyệt nhận HTML và hiển thị ngay nội dung cho người dùng.

SSR = tạo HTML sẵn ở server rồi gửi về trình duyệt.

*Ưu:*

* *Nhanh lần đầu: Nội dung hiển thị ngay, không cần đợi JavaScript.*
* *Tốt cho SEO: Google thấy sẵn nội dung trong HTML.*
* *Hỗ trợ thiết bị yếu: Ít JavaScript, giảm tải cho trình duyệt.*

*Nhược:*

* *Tải server cao: Server phải xử lý và render HTML cho mỗi yêu cầu.*
* *Chậm trang sau: Cần gửi request mới cho mỗi trang.*
* *Phát triển phức tạp: Cần hiểu cả backend và frontend.*

**Tạo giao diện phía client (client-side rendering -CSR)**

Tạo giao diện tại phía client (client-side rendering - CSR) là kỹ thuật tạo ra nội dung trang web bằng cách xử lý và tạo ra mã HTML hoàn chỉnh trên trình duyệt (client) bằng Javascript. Khi người dùng truy cập một ứng dụng web sử dụng CSR, máy chủ chỉ gửi về một tập tin HTML rỗng và một tập tin Javascript. Trình duyệt sẽ nhận tập tin Javascript và thực thi mã JavaScript để tạo ra nội dung trang web.

Cách hoạt động CSR:

1. Người dùng truy cập website.
2. Server gửi về 1 file HTML rỗng và 1 file JavaScript lớn (chứa logic render, xử lý…).
3. Trình duyệt chạy JavaScript → JavaScript gọi API → lấy dữ liệu → tạo ra HTML động.
4. Người dùng mới thấy được nội dung website.

* CSR = tạo HTML bằng JavaScript ở trình duyệt.*

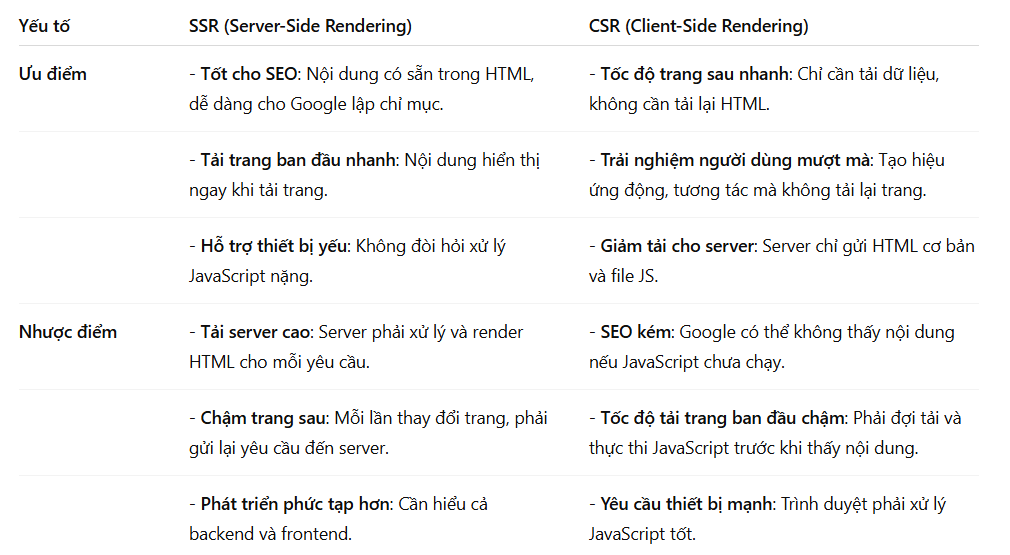
* Ưu: mượt, tương tác tốt, nhẹ server.*

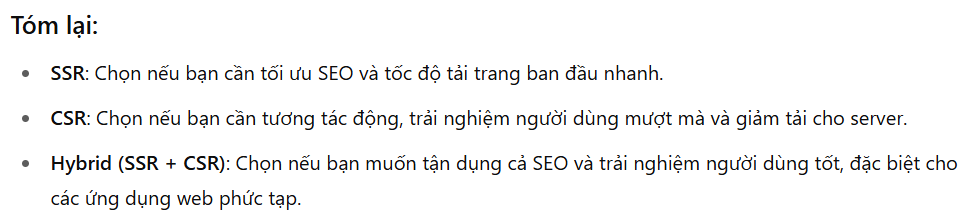
* Nhược: chậm lần đầu, SEO yếu, đòi hỏi client mạnh.*

**Tạo giao diện hỗn hợp, gồm cả ở phía server và phía client**

* SSR cho phép trang web tải nhanh và tối ưu hóa SEO bằng cách gửi HTML hoàn chỉnh từ server lần đầu tiên.
* CSR xử lý các tương tác sau khi trang đã tải xong, cho phép trang web mượt mà hơn với các hiệu ứng động và cập nhật nội dung mà không cần tải lại trang.

Kết quả: Trang web tải nhanh lần đầu, tối ưu SEO, và cung cấp trải nghiệm người dùng tốt với các tính năng tương tác mượt mà.





1. **Express-handlebars**

**Server-side template engine:** công cụ để tạo giao diện (HTML động) phía server bằng cách kết hợp dữ liệu (data) với các mẫu giao diện (templates).

Cách hoạt động:

1. Template: Tập tin HTML chứa placeholder (ví dụ {{name}}) để chèn dữ liệu.
2. Dữ liệu: Lấy từ cơ sở dữ liệu hoặc tính toán ở backend (Node.js, PHP, v.v.).
3. Render: Template engine thay thế placeholder bằng dữ liệu thực tế.
4. Trả kết quả: Gửi HTML hoàn chỉnh về trình duyệt.

Ưu điểm:

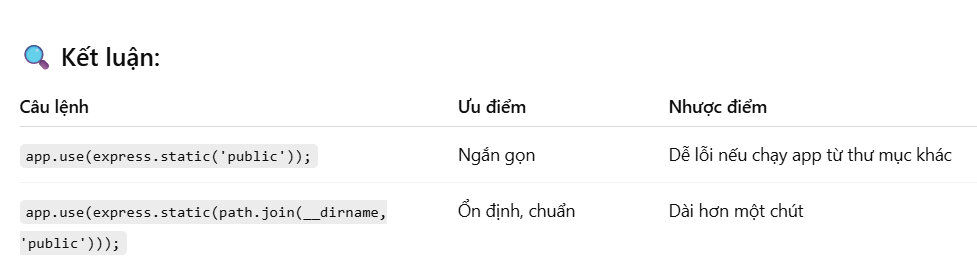
* Hiển thị nội dung động theo dữ liệu.
* Tốt cho SEO: HTML đã có sẵn nội dung.
* Nhanh ở lần tải đầu tiên.
* Tách biệt rõ ràng giữa phần xử lý (backend) và giao diện (frontend).
* Dễ dùng, có nhiều engine như EJS, Handlebars, Pug...

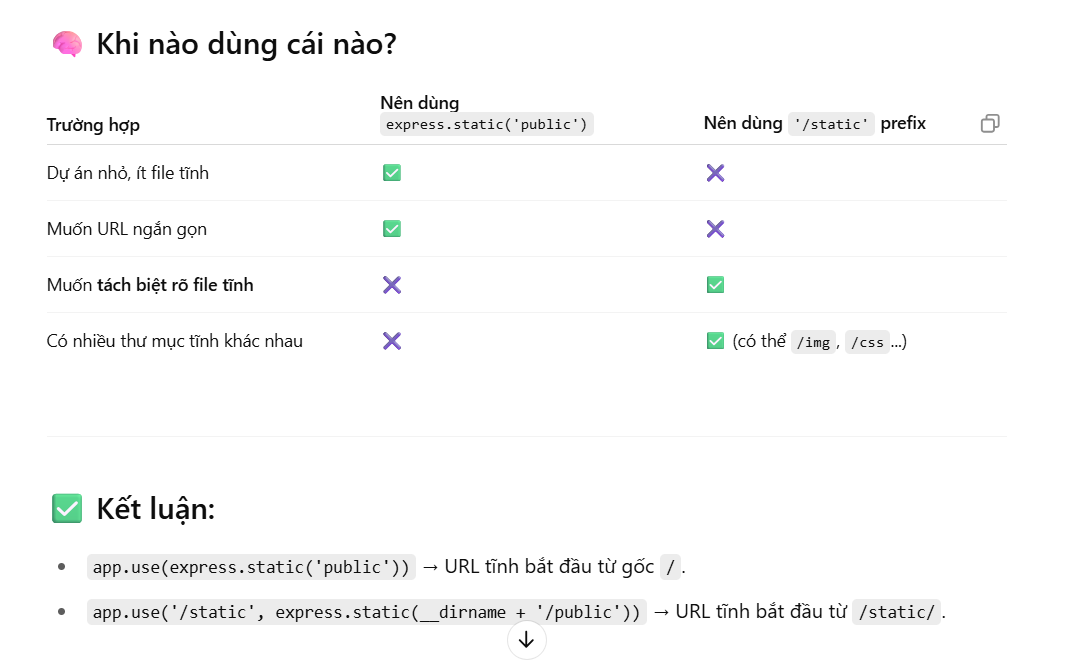
Nhược điểm:

* Không phù hợp cho ứng dụng tương tác cao (SPA).
* Cần tải lại trang mỗi khi dữ liệu thay đổi, không mượt như ứng dụng dùng JavaScript phía client.
* Khó tái sử dụng lại giao diện khi cần thay đổi nhỏ theo hành vi người dùng.

**dependency (thư viện)**

**Cách truy cập đường dẫn CSS”**

****

****

****

****

**Đây là mô hình hoạt động của Web Server theo kiến trúc Client-Server**

Nó chia làm 2 phía: Phía Client và phía Sever, phía Client dùng 3 ngôn ngữ để làm: HTML, CSS, JavaScript, phía Sever dùng 1 trong 6 ngôn ngữ như: PHP, Node.js, Python, Java, C#, Ruby

Bên sever: dịch và thực thi mã

B1: Người dùng dùng trình duyệt(Browser) gửi yêu cầu(Request) đến Web Server dựa trên giao thức HTTP và phương thức GET để gửi yêu cầu.

B2: Phân tích Request data: URL encoding, GET/POST Data, Cookies

* URL Encoding: Mã hóa ký tự đặc biệt trong URL để truyền dữ liệu an toàn.
* GET/POST Data: Gửi dữ liệu từ client lên server. GET hiển thị trên URL, POST ẩn trong body request.
* Cookies: Lưu thông tin trên trình duyệt để duy trì đăng nhập, giỏ hàng, hoặc theo dõi người dùng.

Sau khi lấy thông tin của gói Request data thì đẩy xuống Web Application (mã nguồn Backend) để xử lý

B3: Web Application biết được yêu cầu của Web Server và xử lý yêu cầu, truy xuất dữ liệu từ cơ sở dữ liệu gửi trở về cho Web Application

B4: Đổ dữ liệu sang định dạng HTML (vì trình duyệt không hiểu ngôn ngữ)

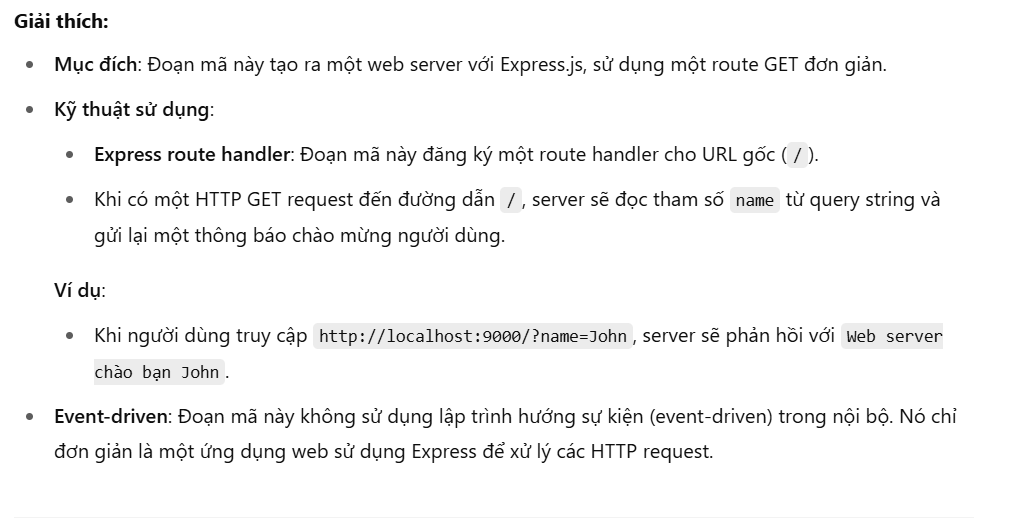
B5: Web Application gửi HTML đã xử lý sau đó trả về Web Server

B6: Trình duyệt người dùng hiển thị nội dung response

B7: Sau khi nhận và hiển thị HTML, trình duyệt tiếp tục gửi thêm request khác để lấy các nội dung tĩnh của ứng dụng web (HTML, CSS, JavaScript, hình ảnh, và các tài nguyên khác).

Đoạn mã 1:





Đoạn mã 2:



