



NHÓM 6

trình bày



LÊ TRÂN THUẬN PHÁT

CÂU 3: TRÌNH BÀY HÀM GỌI CALLBACK; KIẾN TRÚC BẤT ĐỒNG BỘ: LIBUV, EVENT LOOP, NON-BLOCKING; XỬ LÝ DỮ LIỆU BUFFER; LÀM VIỆC VỚI FILE TRONG NODEJS. CÓ DEMO VÍ DỤ

CÂU 4: TRÌNH BÀY GIAO THỨC HTTP HOẠT ĐỘNG NHƯ THẾ NÀO TRONG NODEJS? TRÌNH BÀY SỬ DỤNG NODEJS ĐỂ XÂY DỰNG 1 WEB SERVER. CÓ DEMO VÍ DỤ

NGUYỄN HÔNG PHƯỚC

-POWER POINT

TRỊNH PHƯƠNG NAM

CÂU 8: TRÌNH BÀY HỆ QUAN TRỊ CƠ SỞ DỮ LIỆU MONGODB. CÓ DEMO

CÂU 9: XÂY DỰNG API SỬ DỤNG NODE.JS KẾT HỢP MONGODB VÀ EXPRESS FRAMEWORK. CÓ DEMO

VÕ ĐỨC TIÊN

CÂU 5: TRÌNH BÀY CHƯƠNG TRÌNH QUAN LÝ THƯ VIỆN NPM. CÓ DEMO VÍ DỤ

CÂU 7: TRÌNH BÀY EXPRESS FRAMEWORK CỦA NODEJS, REST API. CÓ DEMO

NGUYỄN MINH HIỀU

CÂU 1: TRÌNH BÀY MODULES, EXPORTS, REQUIRE TRONG NODEJS. CÓ DEMO VÍ DỤ

CÂU 2: TRÌNH BÀY EVENTS VÀ EVENT EMITTER TRONG NODEJS. CÓ DEMO VÍ DỤ

CÁC CÂU HỎI

- MODULES, EXPORTS, REQUIRE TRONG NODEJS. CÓ DEMO VÍ DỤ
- EVENTS VÀ EVENT EMITTER TRONG NODEJS. CÓ DEMO VÍ DỤ
- HÀM GỌI CALLBACK; KIẾN TRÚC BẤT ĐỒNG BỘ: Libuv, Event Loop, Non-Blocking; XỬ LÝ DỮ LIỆU BUFFER; LÀM VIỆC VỚI FILE TRONG NODEJS. CÓ DEMO VÍ DỤ
- GIAO THỨC HTTP HOẠT ĐỘNG NHƯ THẾ NÀO TRONG NODEJS? TRÌNH BÀY SỬ DỤNG NODEJS ĐỂ XÂY DỰNG 1 WEB SERVER. CÓ DEMO VÍ DỤ
- CHƯƠNG TRÌNH QUẢN LÝ THƯ VIỆN NPM. CÓ DEMO VÍ DỤ
- EXPRESS FRAMEWORK CỦA NODEJS, REST API. CÓ DEMO
- HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU MONGODB. CÓ DEMO
- Xây dựng API sử dụng Node.js kết hợp MongoDB và Express framework. CÓ DEMO



Nhóm 6

TRÌNH BÀY MODULES, EXPROTS, REQUIRE TRONG NODEJS

[QUAY LẠI TRANG CHÍNH](#)



- Có thể coi module giống như thư viện JavaScript.
- Một tập hợp các chức năng bạn muốn đưa vào ứng dụng của mình.
- Để thêm một module, hãy sử dụng require() hàm có tên module: `var http = require('http');`
- Bây giờ ứng dụng của bạn có quyền truy cập vào module HTTP và có thể tạo máy chủ:

```
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Hello World!');  
}).listen(8080);
```

- Ví dụ: Nhận máy chủ Node.js của riêng bạn
- Tạo một module trả về ngày và giờ hiện tại:

```
exports.myDateTime = function () {  
  return Date();  
};
```
- Sử dụng export từ khóa để cung cấp các thuộc tính và phương thức bên ngoài tệp module.
- Lưu mã ở trên vào tệp có tên "myfirstmodule.js"

CÂU 1



Nhóm 6

- 2. Exports trong Node.js
- Để chia sẻ mã từ một module cho các module khác, chúng ta sử dụng exports hoặc module.exports.
- Các giá trị được gán cho exports sẽ được chia sẻ với các module khác khi chúng sử dụng require.
- Ví dụ:

javascript

// math.js - Đây là module mà ta muốn xuất ra

const add = (a, b) => a + b;

const subtract = (a, b) => a - b;

// Sử dụng exports để chia sẻ các hàm

module.exports = {

add,

subtract,

};

CÂU 1



Nhóm 6

- 3. Require trong Node.js
- Để sử dụng một module trong một module khác, Node.js cung cấp hàm require(). Hàm này sẽ import module cần thiết.
- Ví dụ:

javascript

// app.js - Module chính

const math = require('./math.js'); // Import module math

const sum = math.add(5, 3);

const difference = math.subtract(5, 3);

console.log('Sum:', sum); // Kết quả: Sum: 8

console.log('Difference:', difference); // Kết quả: Difference: 2

CÂU 1



Nhóm 6

- Ví dụ về module

Bước 1: Tạo file bai.js

- Trong thư mục bạn muốn làm việc, tạo một file tên là bai.js. Đây sẽ là module chứa hàm mà bạn muốn sử dụng sau này.
- Mở file bai.js và dán đoạn mã sau:

```
· function sayHello(name) {  
·   return `Hello, ${name}!`;  
· }  
·  
·  
· module.exports = sayHello;
```


CÂU 1



Nhóm 6

- Ví dụ về module

Bước 2: Tạo file app.js

- Tạo một file khác trong cùng thư mục và đặt tên là app.js. File này sẽ sử dụng module bai.js.
- Dán đoạn mã sau vào file app.js:
 - `const sayHello = require('./bai.js');`
 -
 - `const greetingMessage = sayHello('Node.js');`
 - `console.log(greetingMessage); // Kết quả: Hello, Node.js!`

CÂU 1



Nhóm 6

- Ví dụ về module

Bước 3: Chạy chương trình

- Mở terminal hoặc command prompt.
- Điều hướng đến thư mục mà bạn vừa tạo hai file bai.js và app.js.
- Chạy lệnh sau trong terminal để thực thi app.js:

`node app.js`

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  
PS C:\Users\HIEU\Android Studio> node app.js  
Hello, Node.js!  
PS C:\Users\HIEU\Android Studio> 
```

bai.js là một module đơn giản chứa hàm sayHello để chào hỏi.

app.js sử dụng hàm require để import và sử dụng hàm sayHello từ bai.js



Nhóm 6

TRÌNH BÀY EVENTS VÀ EVENT EMITTER TRONG NODEJS

[QUAY LẠI TRANG CHÍNH](#)



- Node.js dựa trên kiến trúc hướng sự kiện không đồng bộ trong đó một số đối tượng nhất định được gọi là emitters định kỳ phát ra (emit) các sự kiện (Events) khiến các Listener Object được gọi.
- Khi đối tượng EventEmitter phát ra một sự kiện, tất cả các hàm được gắn vào sự kiện cụ thể đó được gọi một cách đồng bộ.
- Mỗi hành động trên máy tính đều được coi là 1 sự kiện, ví dụ như : ghi file, đọc file, kết nối đến cơ sở dữ liệu, đọc cơ sở dữ liệu, lấy dữ liệu ra từ cơ sở dữ liệu, ...
- Để sử dụng event trong nodejs, trước tiên chúng ta cần phải sử dụng một module có sẵn trong nodejs đó là events. Tất cả các phương thức hay thuộc tính của event đều là 1 biểu hiện của EventEmitter, do đó để sử dụng các phương thức hay thuộc tính này, chúng ta cần tạo 1 đối tượng EventEmitter.



1. Events và EventEmitter là gì?

- Events (sự kiện): Là các hành động hoặc sự việc có thể diễn ra trong ứng dụng, chẳng hạn như nhấp chuột, kết nối, đọc file thành công, hoặc bất kỳ tác vụ nào mà bạn muốn ứng dụng phản hồi.
- EventEmitter: Là một class trong Node.js mà các đối tượng có thể "phát" (emit) ra các sự kiện, và những đối tượng khác có thể "nghe" (listen) các sự kiện đó để thực hiện các hành động cụ thể.

2. Cách sử dụng EventEmitter

- Bạn có thể sử dụng module events để tạo một EventEmitter. Sau đó, bạn có thể định nghĩa các sự kiện và lắng nghe chúng.

Các bước thực hiện: EventEmitter



Bước 1: Tạo một file JavaScript

1. Mở trình soạn thảo mã nguồn (IDE) như Visual Studio Code (VSCode) hoặc bất kỳ trình soạn thảo văn bản nào bạn đang sử dụng.
2. Tạo một thư mục mới:
 - Bạn có thể tạo một thư mục trên máy tính của mình, ví dụ: nodejs-event-example.
3. Tạo một file mới bên trong thư mục đó, đặt tên là app.js. Đây là file mà bạn sẽ viết code.

Bước 2: Viết mã cho EventEmitter

1. Bây giờ, bạn sẽ viết code trong file app.js để tạo sự kiện và lắng nghe sự kiện đó bằng EventEmitter.
2. Chạy lệnh `node app.js`

Kết quả:

- Sau khi chạy lệnh, bạn sẽ thấy kết quả hiển thị trên terminal như sau:


```
1 // Bước 1: Import module events
2 const EventEmitter = require('events');
3
4 // Bước 2: Tạo một đối tượng EventEmitter mới
5 const eventEmitter = new EventEmitter();
6
7 // Bước 3: Đăng ký một listener cho sự kiện 'greeting'
8 eventEmitter.on('greeting', (name) => {
9   | console.log(`Hello, ${name}! Chào mừng bạn đến với Node.js!`);
0   | });
1
2 // Bước 4: Phát (emit) sự kiện 'greeting'
3 eventEmitter.emit('greeting', 'Hieu');
```

```
PS C:\Users\HIEU\Android Studio> node app.js  
Hello, Hieu! Chào mừng bạn đến với Node.js!  
PS C:\Users\HIEU\Android Studio> 
```



Nhóm 6

TRÌNH BÀY HÀM GỌI CALLBACK; KIẾN TRÚC BẤT ĐỒNG BỘ: Libuv, Event Loop, Non-Blocking; XỬ LÝ DỮ LIỆU BUFFER; LÀM VIỆC VỚI FILE TRONG NODEJS.

[QUAY LẠI TRANG CHÍNH](#)

CÂU 3



Nhóm 6

- Callback: Callback trong Node.js là một khái niệm quan trọng và phổ biến được sử dụng để xử lý các hoạt động bất đồng bộ (asynchronous operations). Nó cho phép mã JavaScript tiếp tục chạy mà không bị chặn bởi các tác vụ mất thời gian như đọc file, truy vấn cơ sở dữ liệu, hoặc gọi API. Callback trong Node.js là một dạng hàm không đồng bộ. Hàm này được gọi sau khi hoàn thành tác vụ cụ thể nào đó. Tất cả các API của Node đều được viết theo các cách của hàm callback.

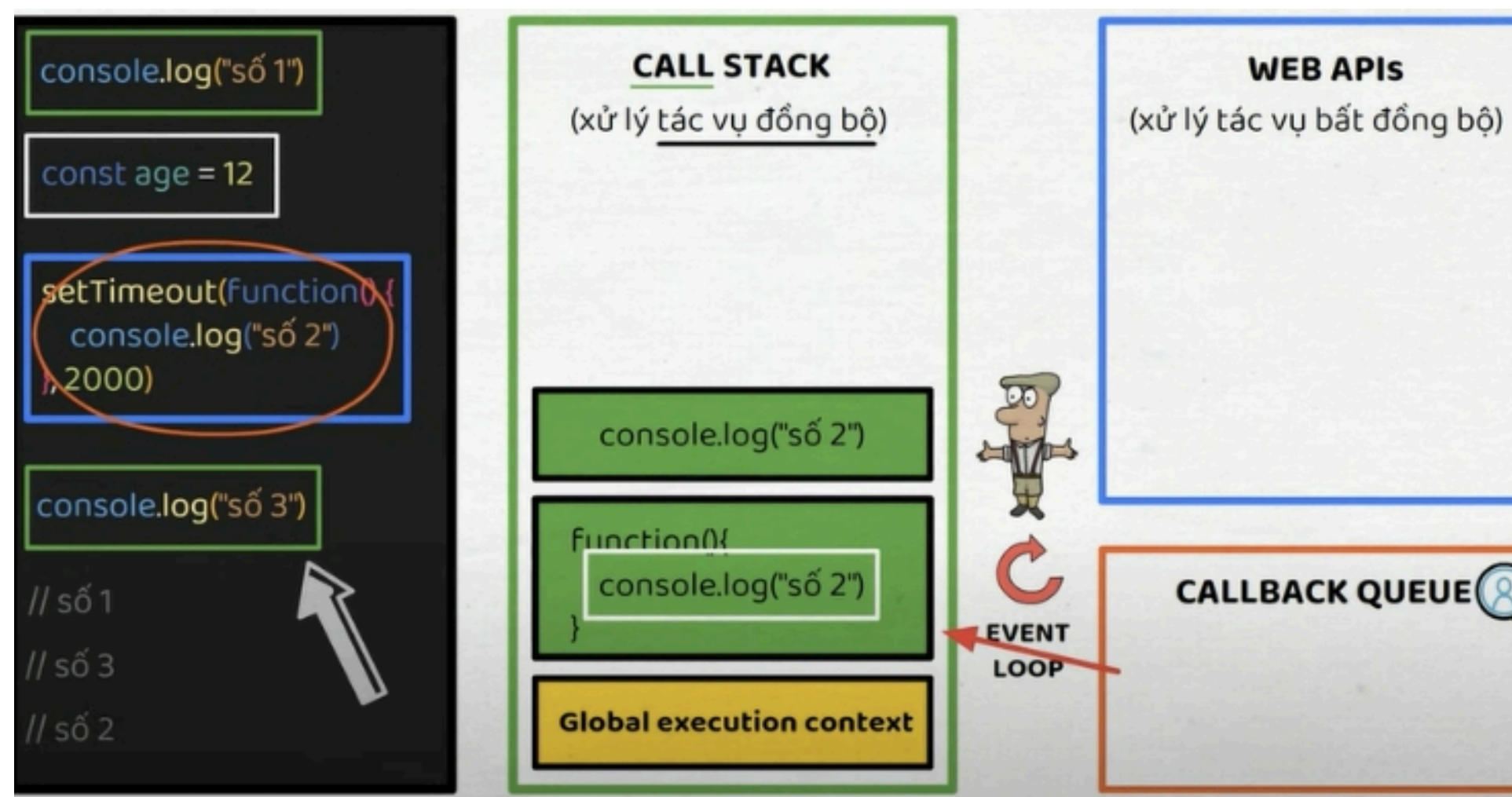
```
const fs = require('fs');

// Hàm đọc file và sử dụng callback
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
    return;
  }
  console.log('File content:', data);
});
```

```
PS D:\UDTMDT\CODE> node "d:\UDTMDT\CODE\app.js"
File content: Hello World TMDT
```

Kiến trúc bất đồng bộ:

- Libuv: Là thư viện C++ hỗ trợ đa nền tảng có trách nhiệm thực hiện thread pool, event loop và các xử lý bất đồng bộ trong Node.js. Trong Node.js, các xử lý có block sẽ ủy quyền qua Libuv, có các thread pool xử lý những hoạt động này.
- Event Loop: Là trái tim của cơ chế bất đồng bộ trong Node.js. Nó liên tục kiểm tra nếu có tác vụ nào trong queue cần thực hiện và xử lý chúng.



+Non-Blocking: Node.js xử lý I/O theo cơ chế non-blocking, nghĩa là các tác vụ I/O sẽ không chặn chương trình, cho phép thực hiện các tác vụ khác trong khi đang chờ I/O hoàn thành.

```
console.log('Start');

setTimeout(() => {
  console.log('Timeout completed');
}, 2000);

console.log('End');
```

```
PS D:\UDTMDT\CODE> node "d:\UDTMDT\CODE\app.js"
Start
End
Timeout completed
```




- Buffer: Buffer trong Node.js được dùng để xử lý dữ liệu nhị phân trực tiếp, như trong trường hợp làm việc với các file hoặc luồng dữ liệu.

```
const buf = Buffer.from('Hello World');  
console.log(buf); // In ra dữ liệu dạng nhị phân  
console.log(buf.toString()); // Chuyển đổi dữ liệu nhị phân thành chuỗi
```

```
PS D:\UDTMDT\CODE> node "d:\UDTMDT\CODE\app.js"  
<Buffer 48 65 6c 6c 6f 20 57 6f 72 6c 64>  
Hello World
```

- Node.js cung cấp các phương thức để đọc, ghi, và xử lý file. Các tác vụ này thường được thực hiện bằng cách sử dụng cơ chế bất đồng bộ (async).

```
const fs = require('fs');

// Đọc file bất đồng bộ
fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log('File content:', data);

  // Ghi vào file mới
  fs.writeFile('newfile.txt', data + '\nNew line added!', (err) => {
    if (err) throw err;
    console.log('File written successfully');
  });
});
```

```
PS D:\UDTMDT\CODE> node "d:\UDTMDT\CODE\app.js"
File content: Hello World TMDT
File written successfully
```



Nhóm 6

TRÌNH BÀY GIAO THỨC HTTP
HOẠT ĐỘNG NHƯ THẾ NÀO
TRONG NODEJS? TRÌNH BÀY
SỬ DỤNG NODEJS ĐỂ XÂY
DỰNG 1 WEB SERVER.

[QUAY LẠI TRANG CHÍNH](#)



HTTP (Hypertext Transfer Protocol) là một giao thức nền tảng cho World Wide Web, được dùng để truyền tải dữ liệu giữa máy khách (client) và máy chủ (server). Trong Node.js, http là một module cốt lõi được tích hợp sẵn, cho phép tạo các máy chủ HTTP để xử lý yêu cầu và phản hồi (request/response).

- HTTP Request: Là một yêu cầu từ client (trình duyệt, ứng dụng) gửi tới server. Một yêu cầu HTTP bao gồm các thành phần chính như method (GET, POST, PUT, DELETE...), headers, và body (dữ liệu được gửi).
- HTTP Response: Là phản hồi mà server gửi lại cho client. Nó bao gồm các thành phần như status code (200, 404, 500...), headers, và body (dữ liệu trả về).

--- Node.js xử lý các yêu cầu HTTP bằng cách sử dụng module http, nơi có thể định nghĩa các tuyến đường (routes) và phản hồi tương ứng cho từng yêu cầu.

--- Có thể dễ dàng tạo ra một web server trong Node.js bằng cách sử dụng module http. Đây là cách cơ bản để tạo ra một máy chủ và xử lý các yêu cầu HTTP.

CÂU 4

Các bước chính để xây dựng một web server:

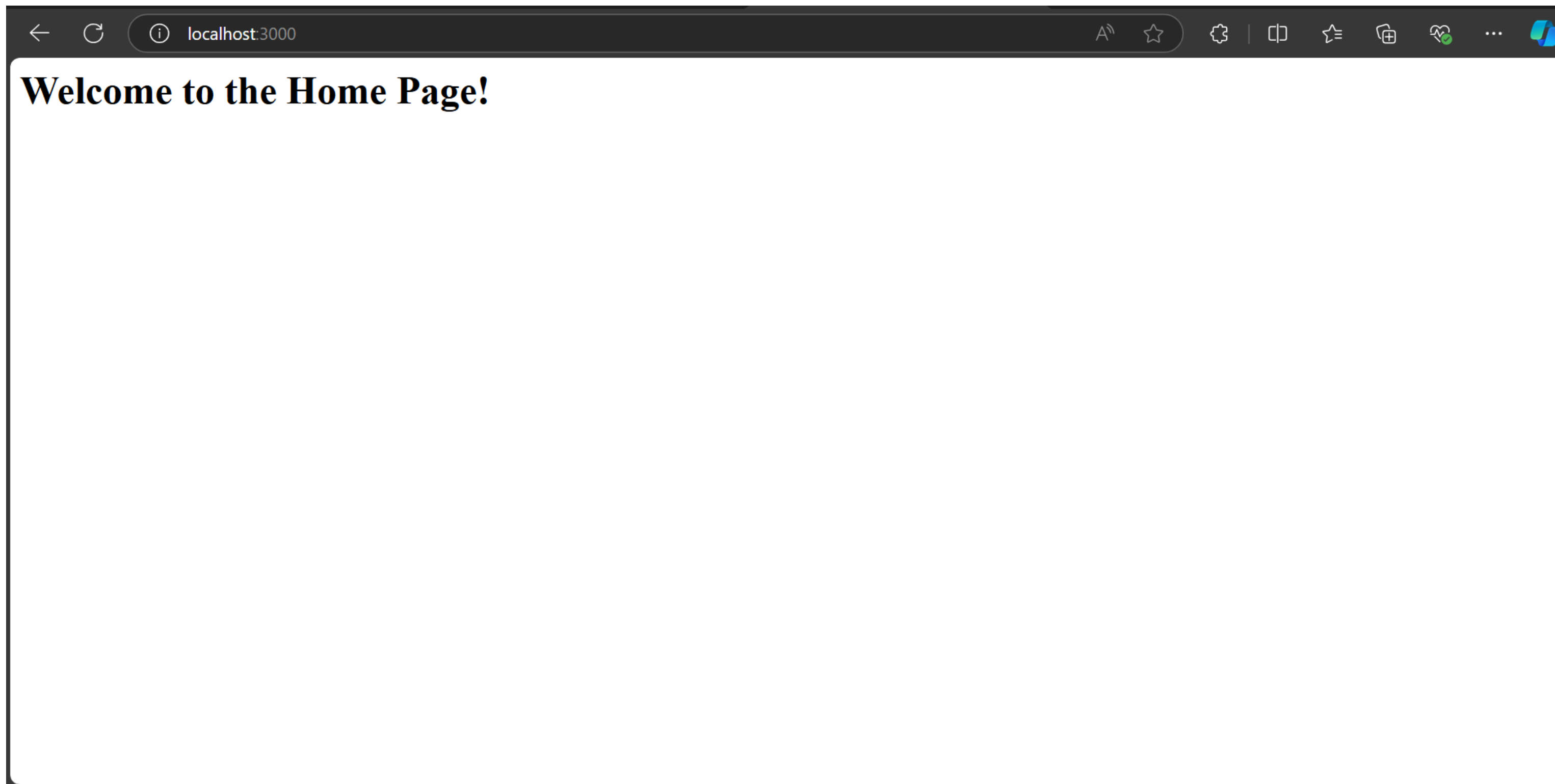
1. Tạo một máy chủ bằng cách sử dụng `http.createServer()`.
2. Xử lý các yêu cầu bằng cách kiểm tra `request.method` và `request.url`.
3. Gửi phản hồi cho client bằng `response.writeHead()` và `response.end()`.

```
1  const http = require('http');
2
3  // Tạo server
4  const server = http.createServer((req, res) => {
5    // Đặt header cho response
6    res.setHeader('Content-Type', 'text/html');
7
8    // Xử lý các route khác nhau
9    if (req.url === '/') {
10     res.writeHead(200); // Mã trạng thái 200 OK
11     res.end('<h1>Welcome to the Home Page!</h1>');
12   } else if (req.url === '/about') {
13     res.writeHead(200);
14     res.end('<h1>About Us</h1><p>This is the about page.</p>');
15   } else {
16     res.writeHead(404); // Mã trạng thái 404 Not Found
17     res.end('<h1>404 - Page Not Found</h1>');
18   }
19 });
20
21 // Chạy server tại cổng 3000
22 server.listen(3000, () => {
23   console.log('Server is running on http://localhost:3000');
24 });
25
```



Nhóm 6

CÂU 4





Nhóm 6

TRÌNH BÀY CHƯƠNG TRÌNH QUẢN LÝ THƯ VIỆN NPM.

[QUAY LẠI TRANG CHÍNH](#)

Node Package Manager (NPM) là một trong những công cụ quan trọng nhất trong việc phát triển ứng dụng Node.js. Nó cho phép bạn quản lý các gói (packages) của Node.js, tải về và cài đặt các thư viện, framework, và công cụ mà bạn cần cho dự án của mình. Bài viết này sẽ hướng dẫn bạn cách sử dụng npm trong Node.js để quản lý các gói và thực hiện các tác vụ phát triển phổ biến.

1. Cách cài đặt Node.js và NPM

Trước khi bắt đầu sử dụng npm, bạn cần cài đặt Node.js trên máy tính của mình. Bạn có thể tải phiên bản mới nhất của Node.js từ trang chính của Node.js (<https://nodejs.org/>) và cài đặt nó bằng cách làm theo hướng dẫn.

Sau khi bạn đã cài đặt Node.js thành công, npm sẽ được cài đặt kèm theo. Để kiểm tra phiên bản npm hiện tại của bạn, bạn có thể mở terminal và chạy lệnh sau:

```
PS C:\Users\LENOVO-PC\Desktop\npm> npm -v  
10.5.0
```

2. Cách sử dụng NPM

a. Tạo Một Dự Án Node.js bằng file JSON Để cài đặt các gói npm từ một tệp JSON, bạn cần sử dụng tệp package.json để chỉ định danh sách các gói và phiên bản cần cài đặt. Dưới đây là cách bạn có thể thực hiện điều này:

- Tạo hoặc sửa đổi tệp package.json: Nếu bạn đã có một tệp package.json, bạn có thể sửa đổi nó để thêm các phụ thuộc. Nếu chưa có, bạn có thể tạo một tệp mới bằng cách chạy lệnh sau trong thư mục dự án của bạn:

```
PS C:\Users\LENOVO-PC\Desktop\npm> npm init
```

Lệnh trên sẽ hướng dẫn bạn qua quá trình tạo tệp package.json và bạn có thể nhập thông tin cần thiết cho dự án của bạn.

- Thêm các phụ thuộc vào package.json: Trong tệp package.json, bạn có thể thêm các phụ thuộc vào phần "dependencies" hoặc "devDependencies". Ví dụ:

CÂU 5



Nhóm 6

```
package.json > {} devDependencies
1  {
2    "name": "npm",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "express": "^4.17.1",
13     "axios": "^0.21.4"
14   },
15   "devDependencies": {
16     "nodemon": "^2.0.15"
17   }
18 }
```

Trong ví dụ trên, chúng ta đã thêm các phụ thuộc của dự án vào phần "dependencies" và các phụ thuộc phát triển vào phần "devDependencies". Phiên bản được chỉ định dưới dạng 4.17.1 là phiên bản tối thiểu cần thiết.

Cài đặt các phụ thuộc từ tệp package.json: Sau khi bạn đã chỉ định các phụ thuộc trong tệp package.json, bạn có thể sử dụng lệnh sau để cài đặt chúng.

```
● PS C:\Users\LENOVO-PC\Desktop\npm> npm install
```

Nếu bạn chỉ muốn cài đặt các phụ thuộc phát triển (devDependencies), bạn có thể sử dụng lệnh:

```
● PS C:\Users\LENOVO-PC\Desktop\npm> npm install --only=dev
```

Cài đặt một phiên bản cụ thể từ tệp package.json: Đôi khi, bạn muốn cài đặt một phiên bản cụ thể của một gói từ tệp package.json. Bạn có thể làm điều này bằng cách sử dụng lệnh sau:

```
npm install <package-name>@<version>
```

CÂU 5



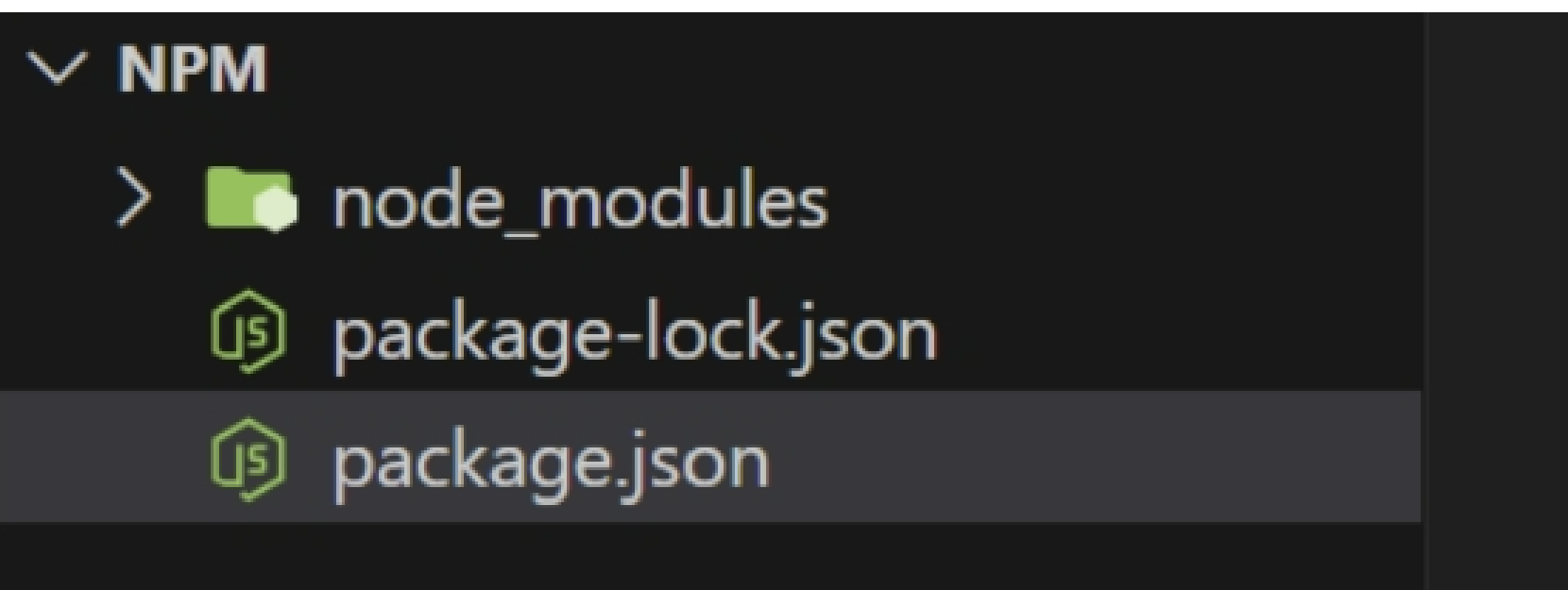
Nhóm 6

VÍ DỤ:

```
PS C:\Users\LENOVO-PC\Desktop\npm> npm install express@4.17.1
```

Điều này sẽ cài đặt gói Express phiên bản 4.17.1 cụ thể.

Sau khi bạn đã thực hiện các bước trên, npm sẽ tự động cài đặt các gói từ tệp package.json của bạn và tạo thư mục node_modules trong thư mục dự án để lưu trữ các gói cài đặt.



b. Cài đặt gói bằng NPM

Để cài đặt một gói npm, bạn có thể sử dụng lệnh sau:

```
npm install <package-name>
```




Nhóm 6

CÂU 5

VÍ DỤ:

```
PS C:\Users\LENOVO-PC\Desktop\npm> npm install express
```

Gói này sẽ được cài đặt trong thư mục node_modules của dự án của bạn và được thêm vào dependencies trong tệp package.json.

c. Cài đặt gói cục bộ:

```
npm install <package-name> --save-dev
```

d. Xóa gói :

```
npm uninstall <package-name>
```

e. Cập nhật gói

```
npm update <package-name>
```

3. Lợi ích của NPM

- Quản lý gói phụ thuộc
- Cộng đồng lớn
- Dễ dàng chia sẻ mã nguồn
- Phiên bản quản lý: npm cho phép quản lý các phiên bản của gói phụ thuộc, có thể chọn sử dụng phiên bản cụ thể của một gói hoặc tự động cập nhật phiên bản khi cần thiết.
- Tự động hóa công việc.
- Hệ sinh thái lớn: npm là một phần của hệ sinh thái Node.js và JavaScript, bao gồm nhiều công cụ và khung làm việc khác nhau như Express.js, React, Angular, Vue.js, và nhiều công cụ khác.
- Hỗ trợ cho công cụ thứ ba: npm có thể được tích hợp với nhiều công cụ quản lý dự án và liên kết với các dịch vụ tích hợp, giúp bạn quản lý dự án một cách hiệu quả.



Nhóm 6

TRÌNH BÀY EXPRESS FRAMEWORK CỦA NODEJS, REST API

[QUAY LẠI TRANG CHÍNH](#)



Định nghĩa về Expressjs là gì?

Express là một framework giành cho nodejs. Nó cung cấp cho chúng ta rất nhiều tính năng mạnh mẽ trên nền tảng web cũng như trên các ứng dụng di động. Express hỗ trợ các phương thức HTTP và middleware tạo ra một API vô cùng mạnh mẽ và dễ sử dụng. Có thể tổng hợp một số chức năng chính của express như sau:

- Thiết lập các lớp trung gian để trả về các HTTP request
- Định nghĩa router cho phép sử dụng với các hành động khác nhau dựa trên phương thức HTTP và URL
- Cho phép trả về các trang HTML dựa vào các tham số.

Cài Đặt :

Để cài đặt express, tạo thư mục express sau đó gõ lệnh :

```
PS C:\Users\LENOVO-PC\Desktop\npm\express> npm install express --save
```

Với câu lệnh trên sẽ lưu phần cài đặt trong thư mục node_modules và tạo thư mục express trong đó. Chúng ta cần cài đặt thêm một số module quan trọng đi cùng express như sau:

- body-parser: Đây là lớp trung gian, xử lý JSON, text và mã hóa URL.
- cookie-parser : Chuyển đổi header của cookie và phân bố đến các req.cookies
- multer - Xử lý phần multipart/form-data

Để cài đặt các module trên, lần lượt chạy các lệnh:

```
● PS C:\Users\LENOVO-PC\Desktop\npm\express> npm install body-parser --save
```

```
● PS C:\Users\LENOVO-PC\Desktop\npm\express> npm install cookie-parser --save
```

```
● PS C:\Users\LENOVO-PC\Desktop\npm\express> npm install multer --save
```

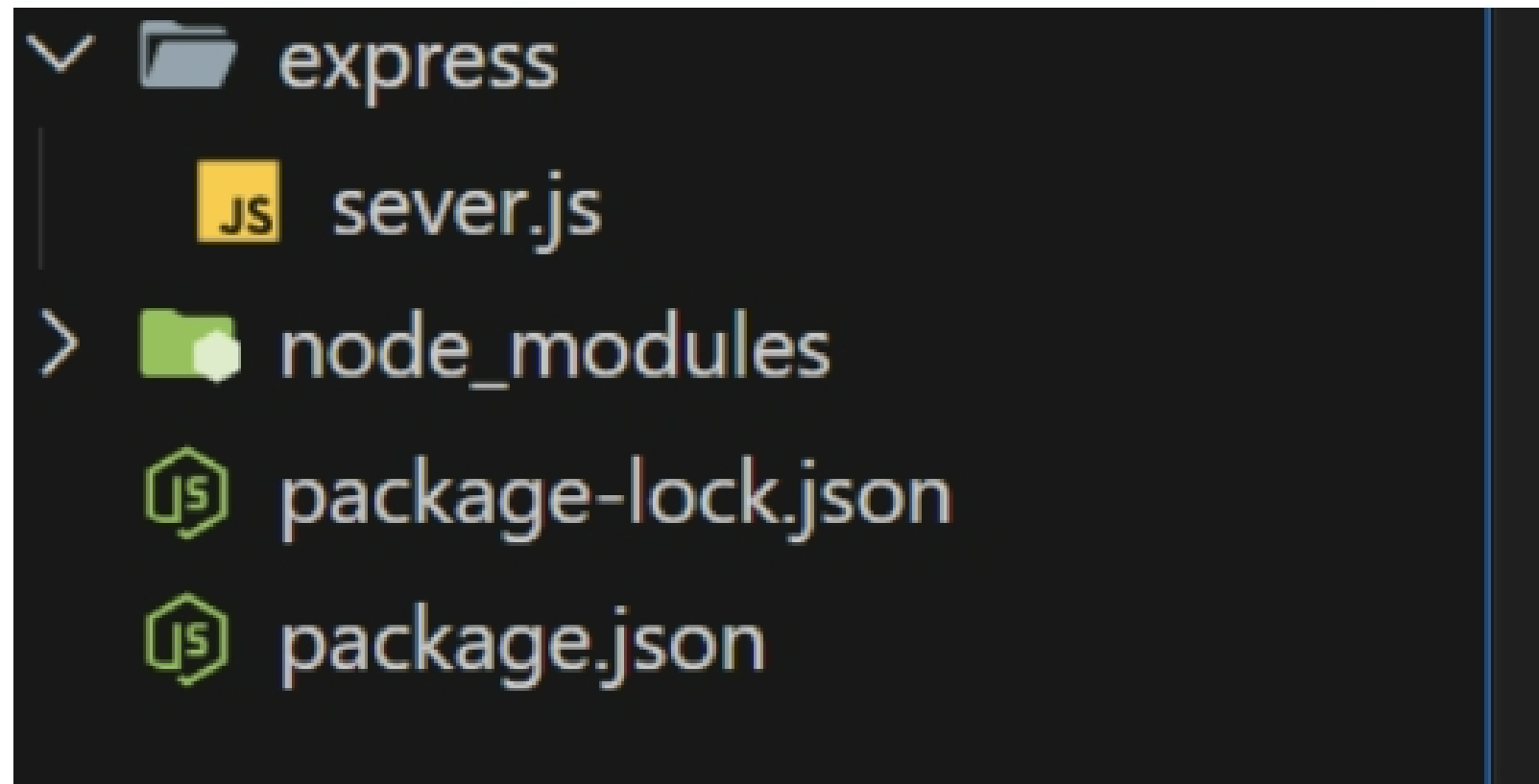
CÂU 7



Nhóm 6

DEMO:

tạo file server.js ở trong thư mục express :



```
express > JS sever.js > ...
1  var express = require("express");
2  var app = express();
3
4  app.get("/", function (req, res) {
5    res.send("Hello World");
6  });
7
8  var server = app.listen(8081, function () {
9    var host = server.address().address;
10   var port = server.address().port;
11
12   console.log(
13     "Ung dung Node.js dang lang nghe tai dia chi: http://%s:%s",
14     host,
15     port
16   );
17 });
18
```


CÂU 7



Nhóm 6

DEMO:

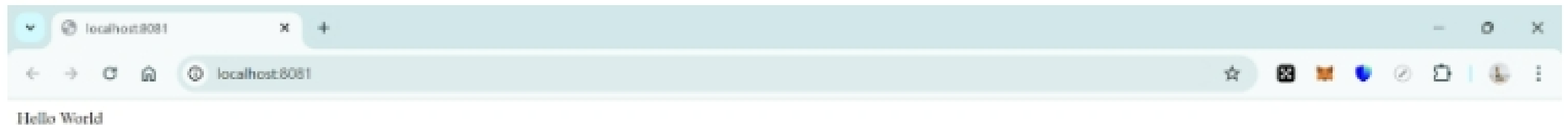
Chạy server để xem kết quả với lệnh:

```
PS C:\Users\LENOVO-PC\Desktop\npm\express> node sever.js
```

KẾT QUẢ:

```
PS C:\Users\LENOVO-PC\Desktop\npm\express> node sever.js  
Ung dung Node.js đang lang nghe tai dia chi: http://:::8081
```

Sau đó, hãy truy cập vào địa chỉ: [Test demo](http://localhost:8081). Kết quả thu được như sau:



Request & Response

Ví dụ Express sử dụng một hàm callback có các tham số là request và response như sau.

```
app.get('/', function (req, res) {  
  // --  
})
```

chi tiết về 2 đối tượng này:

- Request : Biểu diễn một HTTP request, và có các thuộc tính cho các request như các chuỗi truy vấn, tham số, HTML.
- Response : Biểu diễn HTTP response được ứng dụng Express gửi đi khi nó nhận các HTTP request.


CÂU 7



Nhóm 6

Định tuyến cơ bản

Mở file server.js và sửa lại code với nội dung sau:

```
express >  server.js > ...
1  var express = require('express');
2  var app = express();
3
4  // Phương thức get() phản hồi một GET Request về Homepage
5  app.get('/', function (req, res) {
6    console.log("Nhận một GET Request về Homepage");
7    res.send('Hello GET');
8  })
9
10 // Phương thức post() phản hồi một POST Request về Homepage
11 app.post('/', function (req, res) {
12   console.log("Nhận một POST Request về Homepage");
13   res.send('Hello POST');
14 })
15
16 // Phương thức delete() phản hồi một DELETE Request về /del_user page.
17 app.delete('/del_user', function (req, res) {
18   console.log("Nhận một DELETE Request về /del_user");
19   res.send('Hello DELETE');
20 })
21
22 // Phương thức này phản hồi một GET Request về /list_user page.
23 app.get('/list_user', function (req, res) {
24   console.log("Nhận một GET Request về /list_user");
25   res.send('Page Listing');
26 })
27
28 // Phương thức này phản hồi một GET Request về abcd, abxcd, ab123cd, ...
29 app.get('/ab*cd', function (req, res) {
30   console.log("Nhận một GET request về /ab*cd");
31   res.send('Page Pattern Match');
32 })
33
34 var server = app.listen(8081, function () {
35
36   var host = server.address().address
37   var port = server.address().port
38
39   console.log("Ứng dụng Node.js đang lắng nghe tại địa chỉ: http://%s:%s", host, port)
40
41 })
42
```

CÂU 7



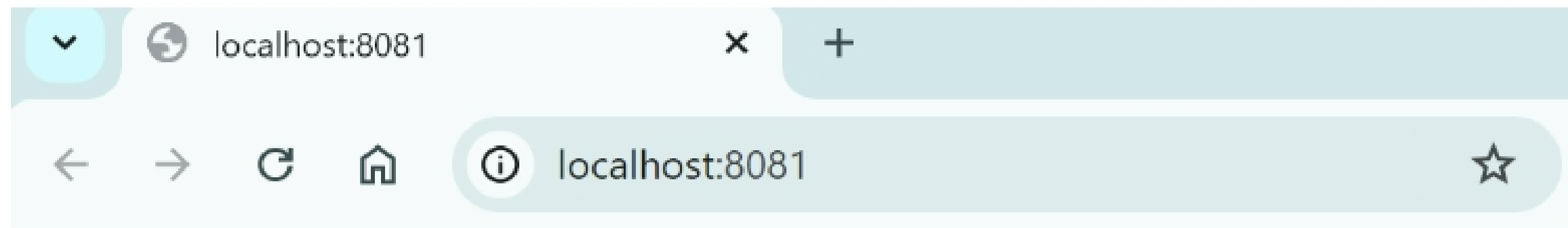
Nhóm 6

Định tuyến cơ bản

chạy file sever.js và kết quả :

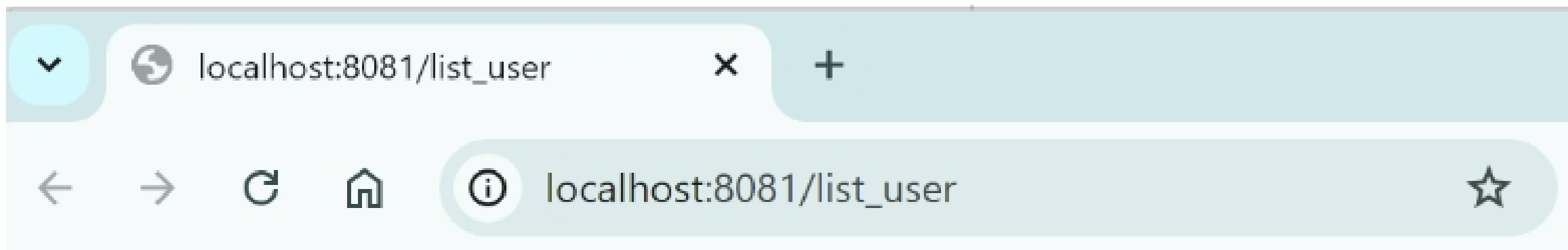
```
PS C:\Users\LENOVO-PC\Desktop\npm\express> node sever.js  
Ung dung Node.js dang lang nghe tai dia chi: http://:::8081
```

Hello Get



Hello GET

List User



Page Listing

CÂU 7



Nhóm 6

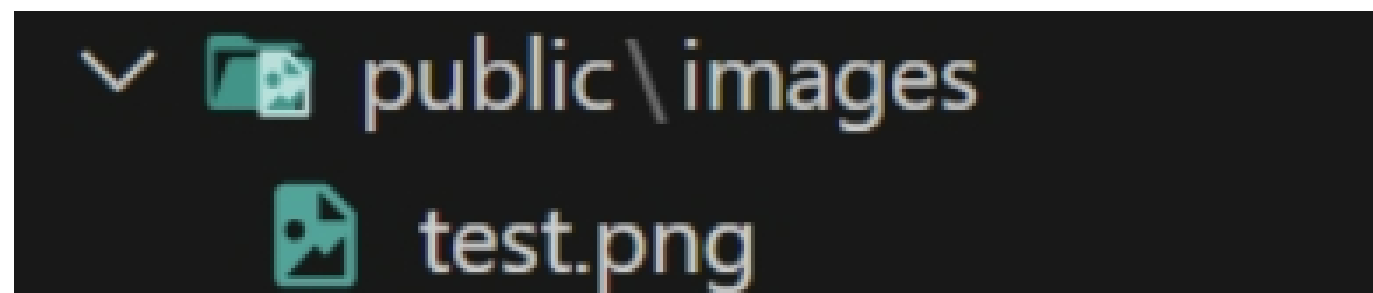
Làm việc với file tĩnh :

Express cung cấp lớp tiện ích trung gian `express.static` để phục vụ cho các file tĩnh như các file hình ảnh, css, js.

Cách hoạt động của nó về cơ bản bạn chỉ cần truyền tên thư mục, nơi bạn chứa các file tĩnh thì `express.static` sẽ xử dụng file đó một cách trực tiếp.

file sever.js :

ví dụ : có một vài hình ảnh trong thư mục `public/images` như sau:

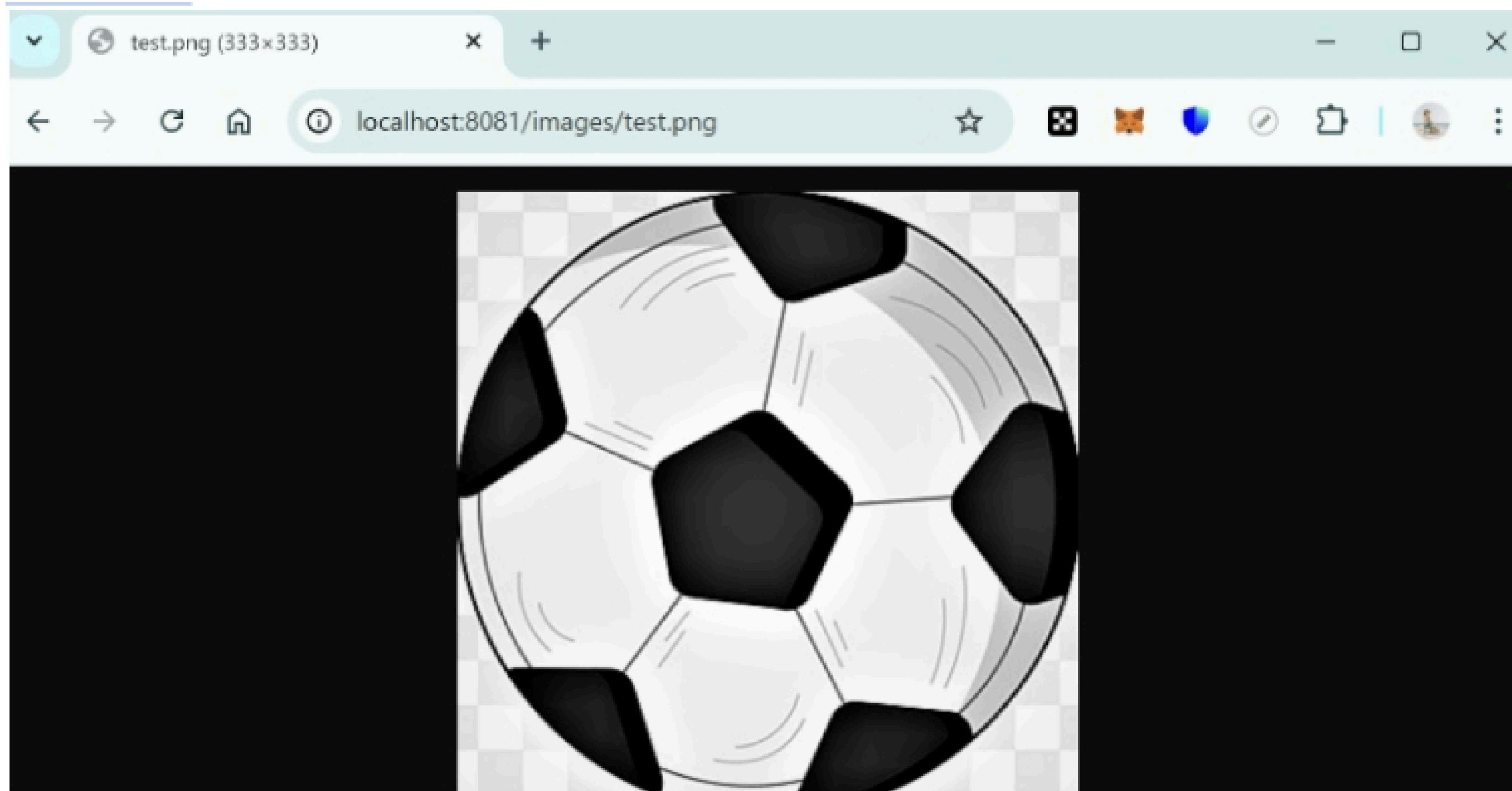


```
express > .\server.js > ...
1  var express = require("express");
2  var app = express();
3
4  app.use(express.static("public"));
5
6  app.get("/", function (req, res) {
7    res.send("Hello World");
8  });
9
10 var server = app.listen(8081, function () {
11   var host = server.address().address;
12   var port = server.address().port;
13
14   console.log(
15     "Ứng dụng Node.js đang lắng nghe tại địa chỉ: http://%s:%s",
16     host,
17     port
18   );
19 });
20
```

CÂU 7

 Nhóm 6

Làm việc với file tĩnh :
kết quả :





Nhóm 6

TRÌNH BÀY HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU MONGODB.

[QUAY LẠI TRANG CHÍNH](#)



MongoDB là gì?

- MongoDB là một phần mềm mã nguồn mở dùng để quản trị cơ sở dữ liệu NoSQL.
- NoSQL (Not only SQL) được sử dụng thay thế cho cơ sở dữ liệu quan hệ (Relational Database – RDB) truyền thống. Cơ sở dữ liệu NoSQL khá hữu ích trong khi làm việc với các tập dữ liệu phân tán lớn. MongoDB là một công cụ có thể quản lý thông tin hướng document cũng như lưu trữ hoặc truy xuất thông tin.
- MongoDB lưu trữ dữ liệu ở định dạng BSON document. Ở đây, BSON là đại diện cho định dạng mã hoá nhị phân của các tài liệu JSON (chữ B trong BSON là viết tắt của Binary). Hay nói cách khác, trong phần backend, máy chủ MongoDB chuyển đổi dữ liệu JSON thành dạng nhị phân, được gọi là BSON, và BSON này có thể được lưu trữ và truy vấn hiệu quả hơn.



Công dụng của MongoDB

- MongoDB giúp các tổ chức lưu trữ lượng lớn dữ liệu trong khi vẫn hoạt động nhanh chóng.
- Thay vì sử dụng các table và row như trong cơ sở dữ liệu quan hệ, vì là cơ sở dữ liệu NoSQL, MongoDB được tạo thành từ collection và document. Document được tạo thành từ các cặp khóa-giá trị (là đơn vị dữ liệu cơ bản của MongoDB). Còn collection, tương đương với table trong SQL, là nơi chứa các bộ document.

Các thuật ngữ MongoDB thường dùng

_id

- _id là một trường bắt buộc trong mọi document của MongoDB. _id được sử dụng để đại diện cho tính duy nhất của một document trong một collection. Trường _id hoạt động giống như khóa chính (primary key) của document.

Document

- Document là đơn vị lưu trữ dữ liệu cơ bản trong cơ sở dữ liệu MongoDB. Document mang vai trò tương tự như row trong các hệ thống cơ sở dữ liệu quan hệ truyền thống.
- Document là một cách để sắp xếp và lưu trữ dữ liệu dưới dạng một tập hợp các cặp field-value. Document trong MongoDB không cần phải có cùng một bộ field hoặc cấu trúc với các document khác trong cùng một collection.

Các thuật ngữ MongoDB thường dùng

Collection

- Collection là một tập hợp các document MongoDB. Collection tương tự như table trong hệ thống cơ sở dữ liệu quan hệ. Các collection có tính chất schema less, do đó các document trong cùng một collection có thể có các trường khác nhau.

Database

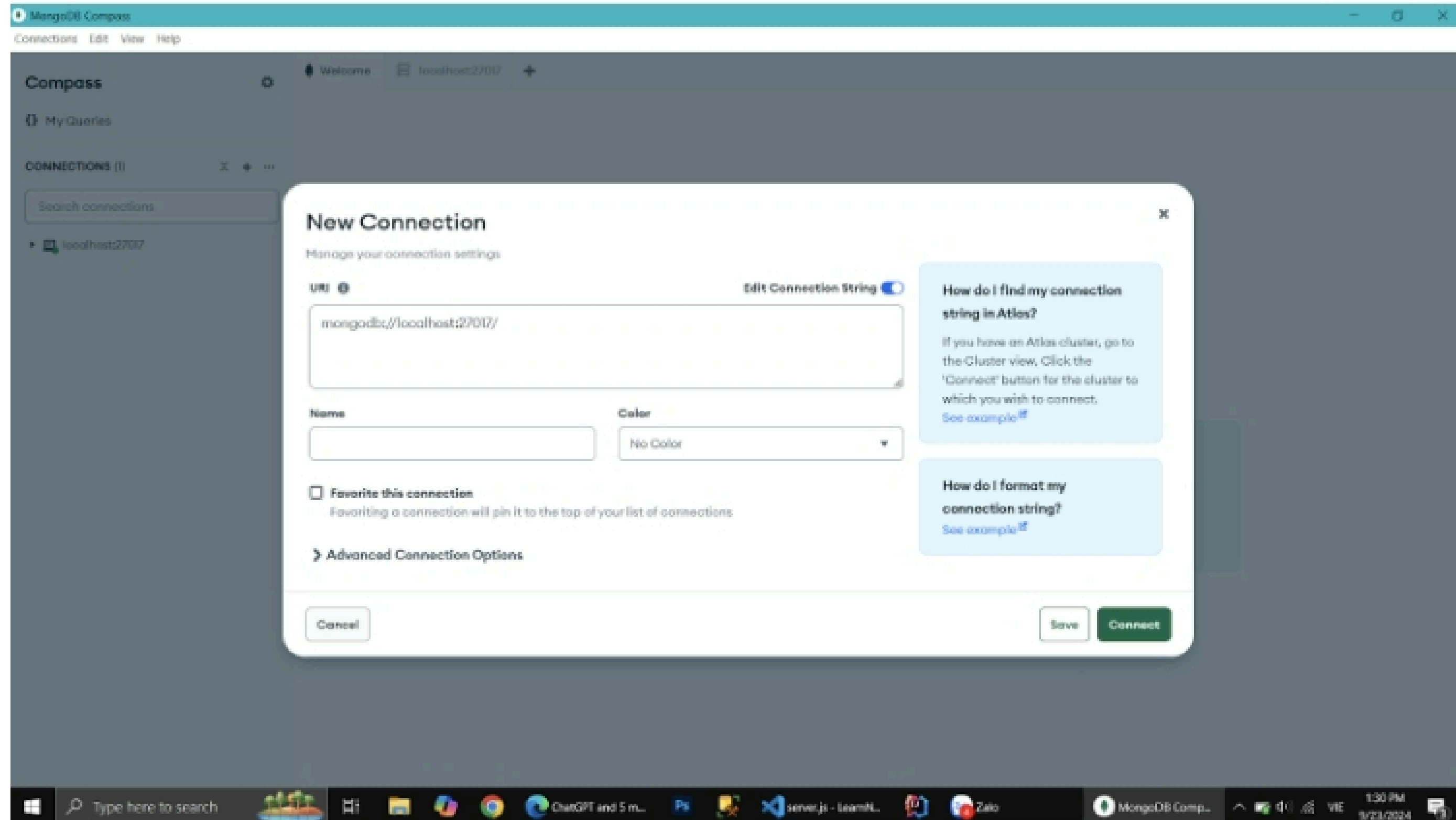
- Trong MongoDB, database là một container vật lý chứa tập hợp các collection. Một database có thể chứa 0 collection hoặc nhiều collection.
- Một phiên bản máy chủ MongoDB có thể lưu trữ nhiều database và không có giới hạn về số lượng database có thể được lưu trữ trên một phiên bản, nhưng giới hạn ở không gian bộ nhớ ảo có thể được phân bổ bởi hệ điều hành.



Nhóm 6

CÂU 8

DEMO:

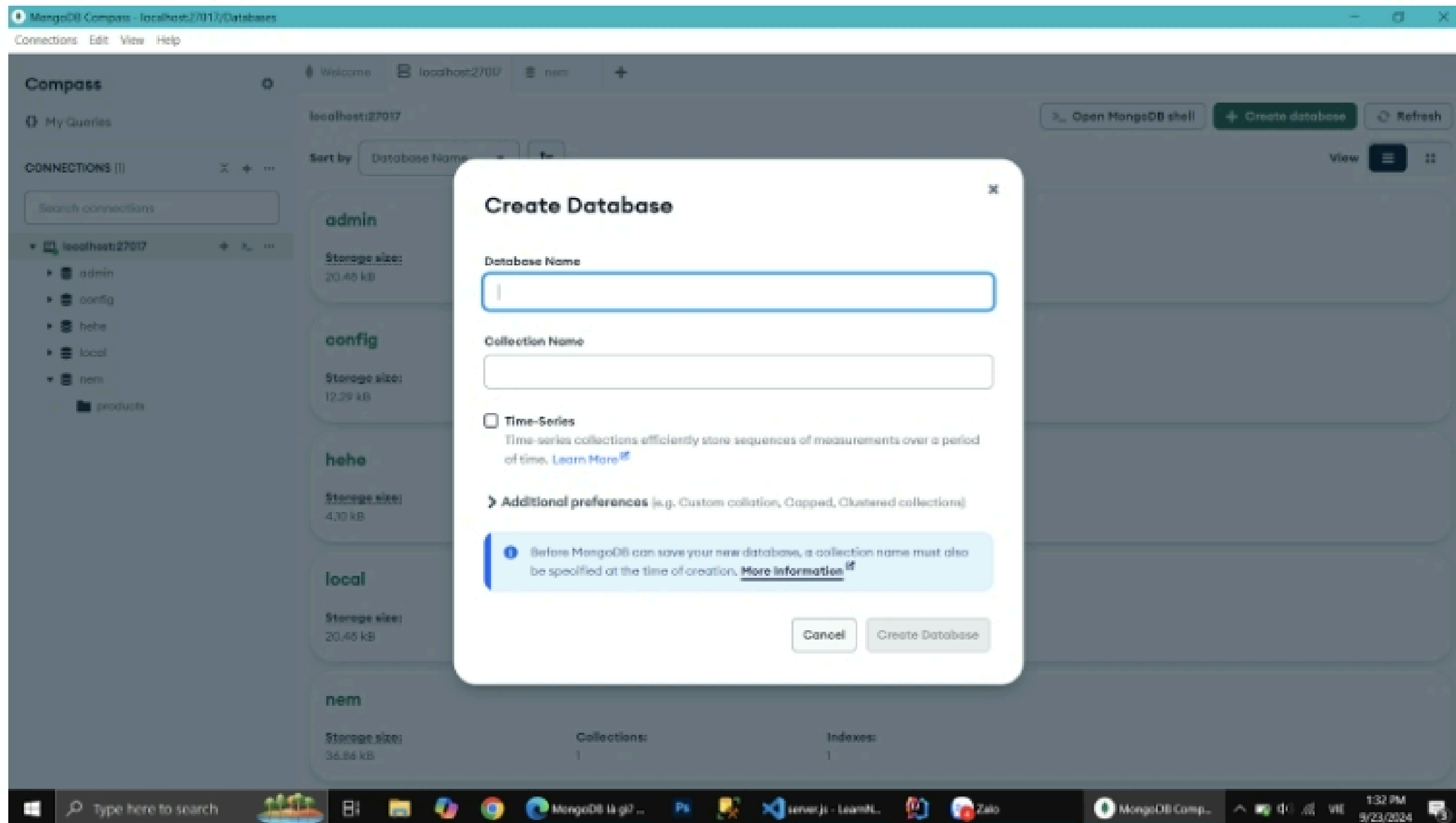




Nhóm 6

CÂU 8

Tạo mới một kết nối trong mongodb

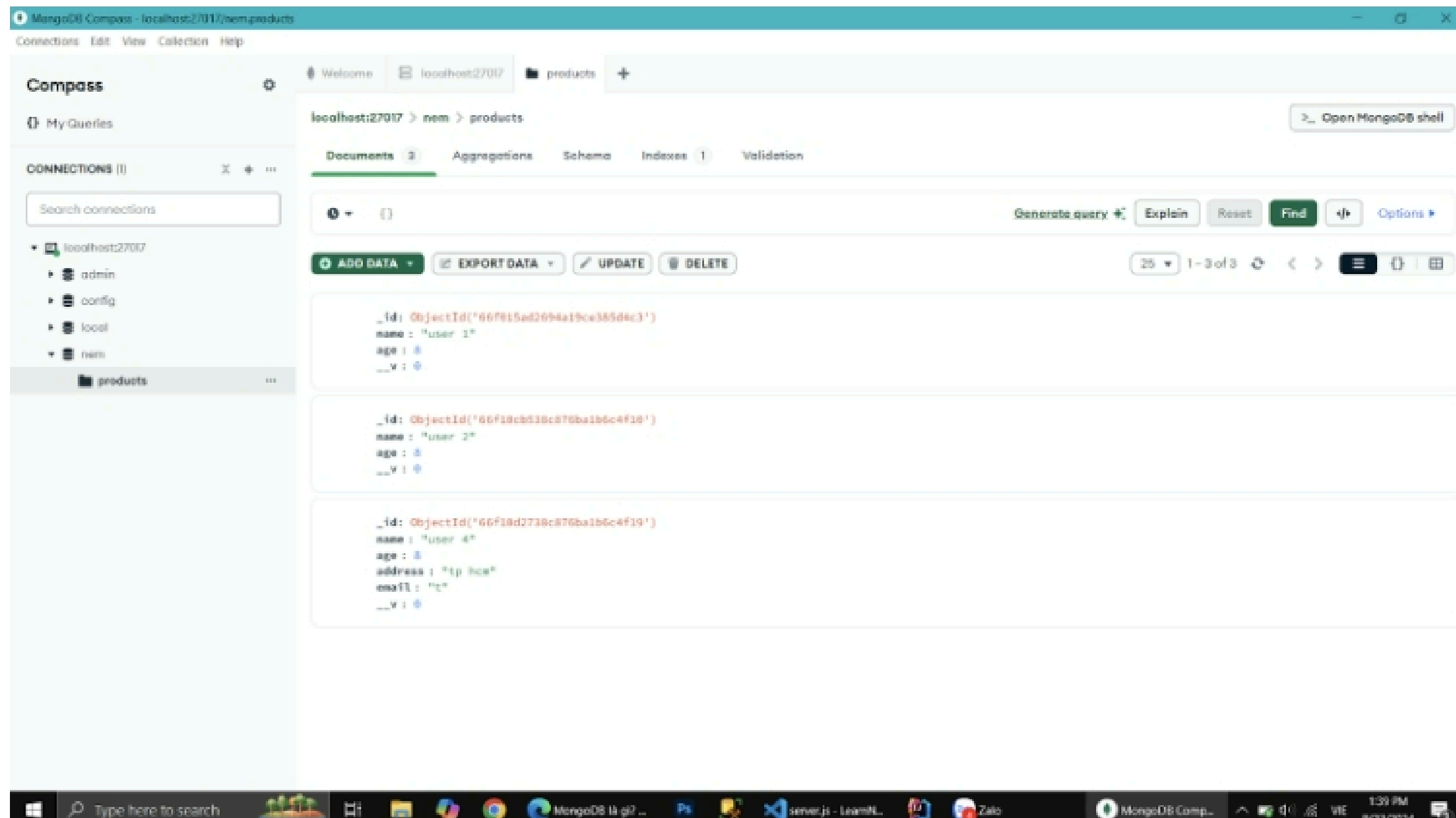




Nhóm 6

CÂU 8

Tạo mới một database và một collections trong database này, collections này là tập hợp các document hay còn tương tự như một bảng table như trong cơ sở dữ liệu quan hệ.



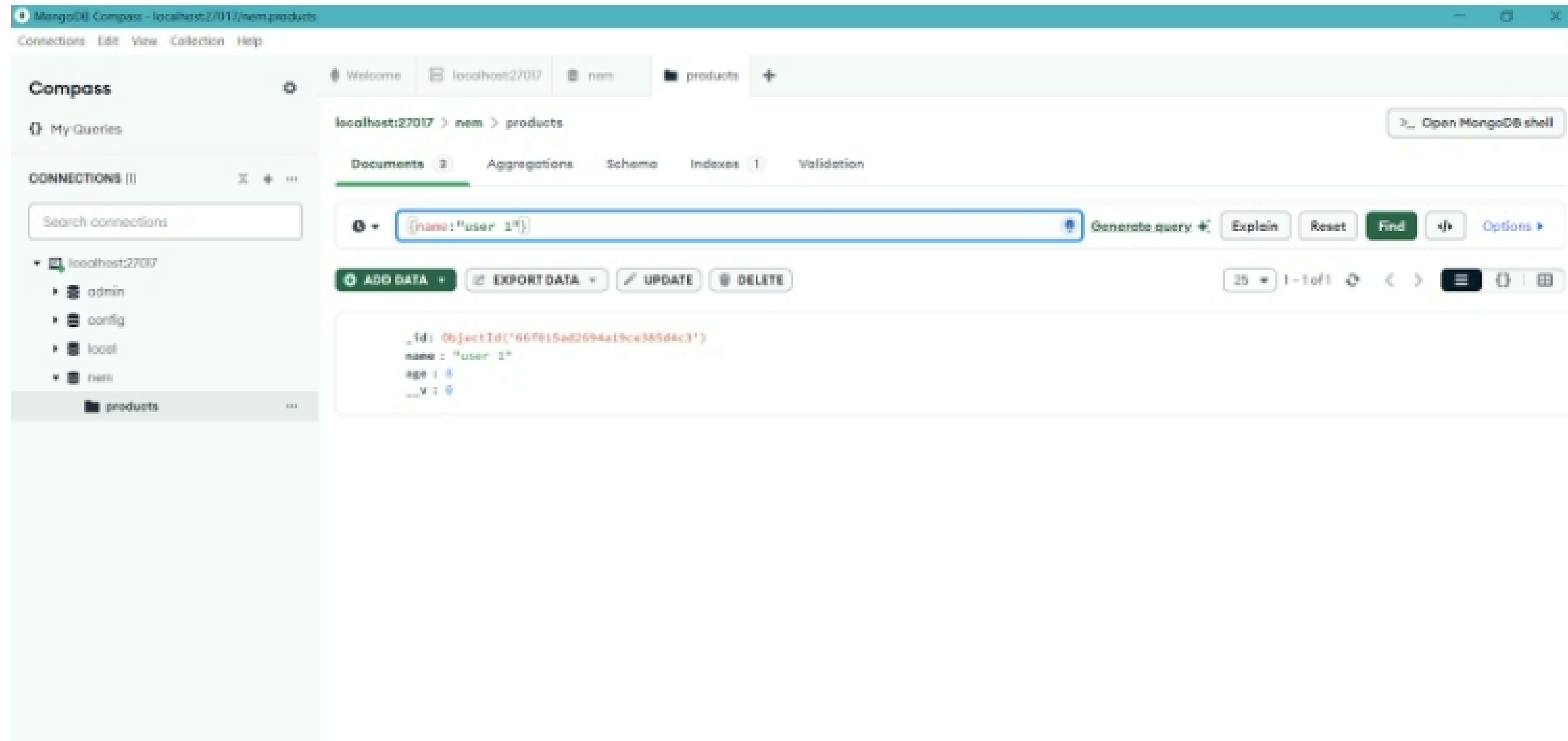


Nhóm 6

CÂU 8

Tạo mới một số bộ dữ liệu trong collections product

Các document trong một collections có thể có các trường dữ liệu khác nhau

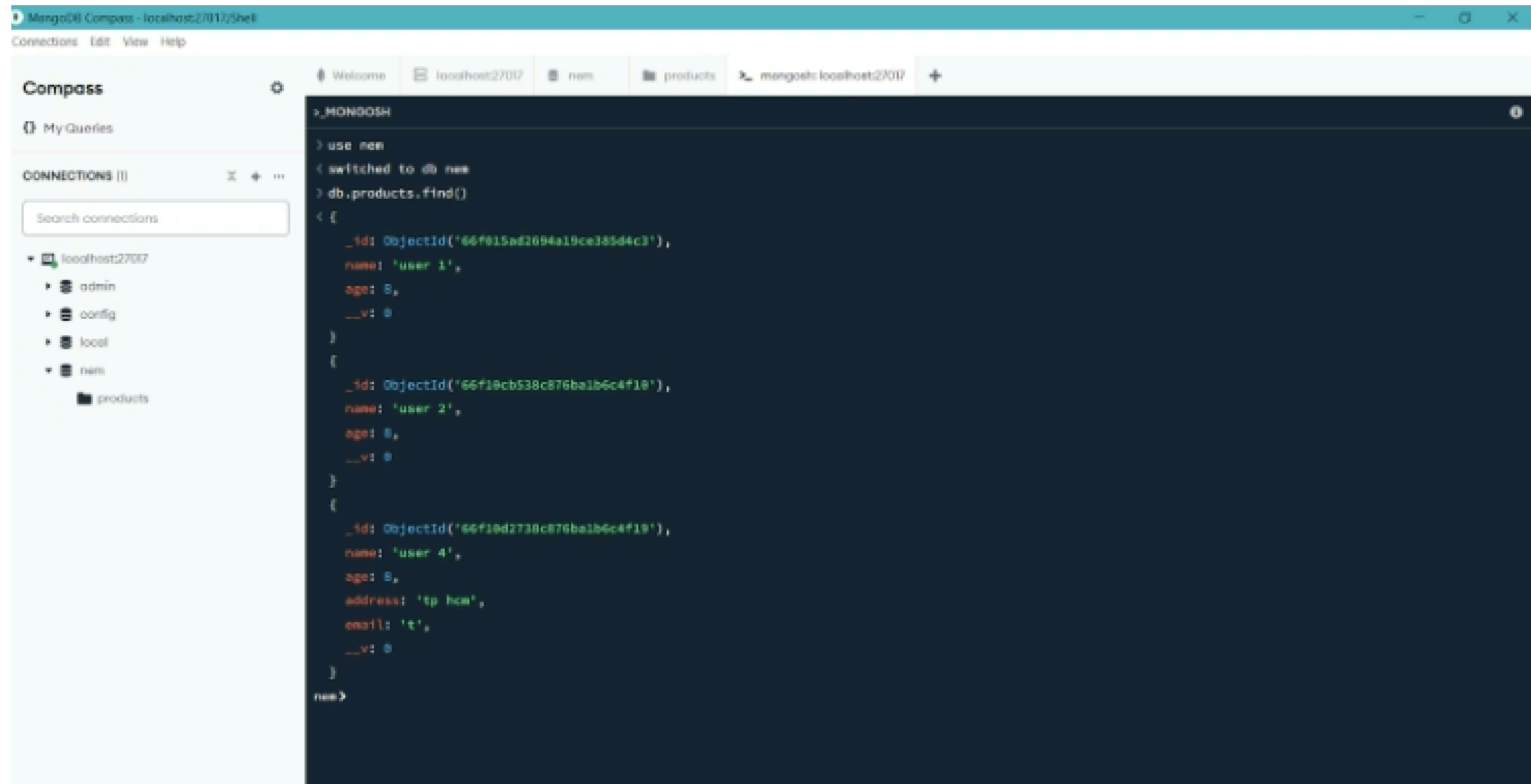




Nhóm 6

CÂU 8

Tìm kiếm các document bằng các key : value



The screenshot shows the MongoDB Compass application. On the left, the 'Connections' panel shows a connection to 'localhost:27017' with a tree view of databases including 'admin', 'config', 'local', 'nem', and 'products'. The 'Products' collection is selected. The main area is the 'MongoDB Shell' with the following commands and output:

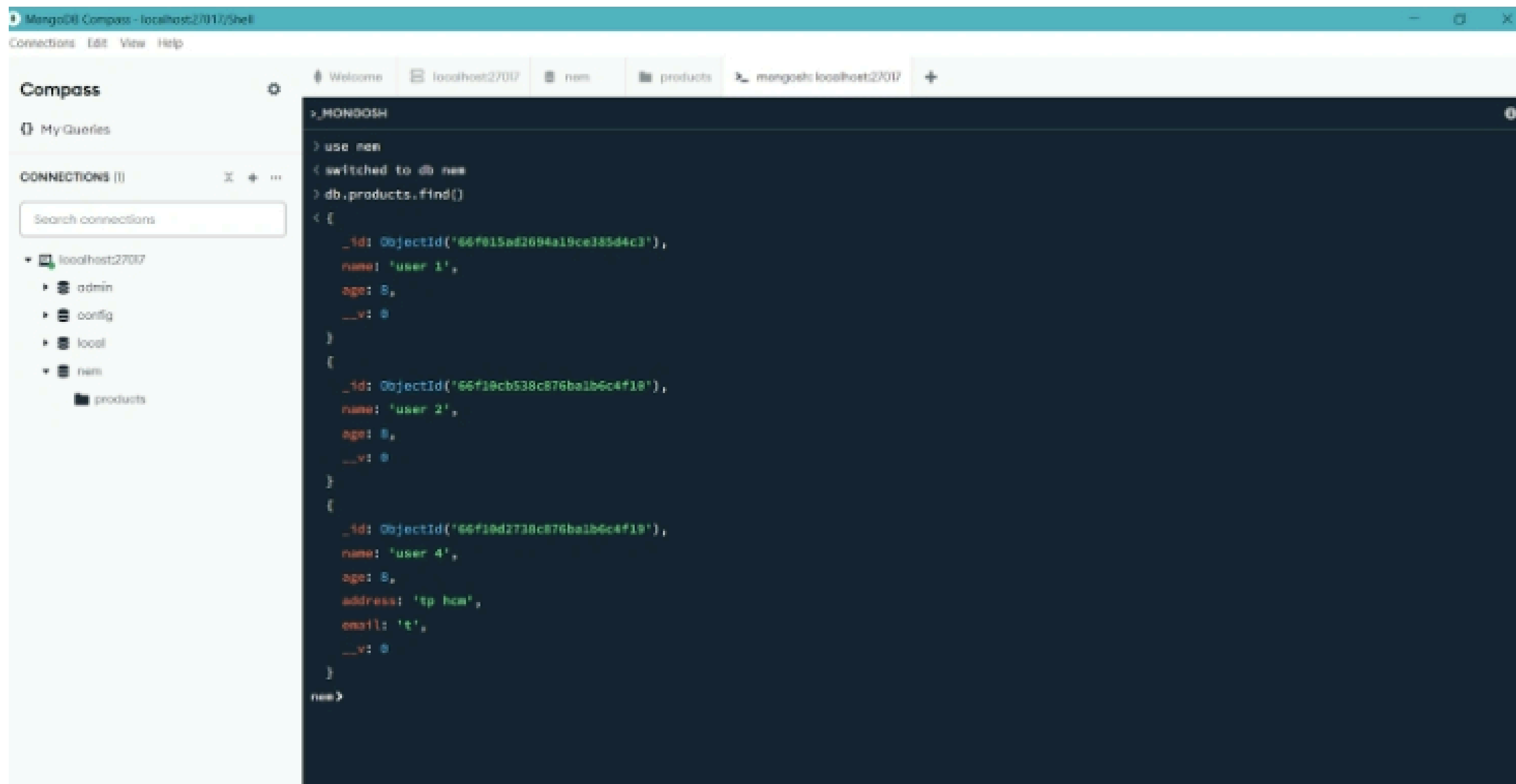
```
> use nem
switched to db nem
> db.products.find()
[ {
  _id: ObjectId('66f615ad2694a19ce385d4c3'),
  name: 'user 1',
  age: 8,
  __v: 0
},
{
  _id: ObjectId('66f19cb538c876ba1b6c4f18'),
  name: 'user 2',
  age: 8,
  __v: 0
},
{
  _id: ObjectId('66f19d2738c876ba1b6c4f19'),
  name: 'user 4',
  age: 8,
  address: 'tp hcm',
  email: 't',
  __v: 0
} ]
```



Nhóm 6

CÂU 8

Tìm kiếm các document bằng các key : value



The screenshot shows the MongoDB Compass application. On the left, the 'Connections' panel lists 'localhost27017' with a tree view showing databases 'admin', 'config', 'local', 'nem', and 'products'. The 'nem' database is selected. The main area displays the 'mongosh' shell with the following commands and output:

```
> use nem
switched to db nem
> db.products.find()
[
  {
    _id: ObjectId('66f615ad2694a19ce385d4c3'),
    name: 'user 1',
    age: 8,
    __v: 0
  },
  {
    _id: ObjectId('66f19cb538c876ba1b6c4f18'),
    name: 'user 2',
    age: 8,
    __v: 0
  },
  {
    _id: ObjectId('66f19d2738c876ba1b6c4f19'),
    name: 'user 4',
    age: 8,
    address: 'tp hcm',
    email: 't',
    __v: 0
  }
]
```



Xây dựng API sử dụng Node.js kết hợp MongoDB và Express framework.

CÂU 9



Nhóm 6

- Express.js là một framework phổ biến được sử dụng để xây dựng ứng dụng web và API thông qua Node.js. Nền tảng được xem là một phương thức xử lý các yêu cầu HTTP, quản lý các tuyến đường, xử lý phần mềm trung gian và nhiều tính năng khác để phát triển hiệu quả ứng dụng web.

kết nối với mongodb

The screenshot shows a VS Code editor with a project structure on the left and a code editor on the right. The project structure includes a 'src' directory with a 'config' subdirectory. The 'config' directory contains a 'db.js' file, which is currently selected and open in the editor. The code in 'db.js' is as follows:

```
src > config > JS db.js > [0] default
1  import mongoose from 'mongoose';
2  import dotenv from 'dotenv';
3
4  dotenv.config()
5
6
7  const connectdb = async (uri) =>{
8      try {
9          await mongoose.connect(uri);
10     }catch (error) {
11         console.log(error);
12     }
13 };
14 export default connectdb;
```

CÂU 9



Nhóm 6

Định nghĩa một schema cho model "Product" trong MongoDB bằng Mongoose. Schema này gồm hai trường:

- name: kiểu String.
- age: kiểu Number.

Cuối cùng, nó tạo và xuất model Product, cho phép tương tác với collection "products" trong MongoDB.

```
src > models > product.js > default
1  import mongoose from "mongoose";
2  import { Schema } from "mongoose";
3
4  const productSchema = new Schema({
5    name: {
6      type: String,
7    },
8    age: {
9      type: Number,
10    },
11  });
12
13
14  export default mongoose.model("Product", productSchema);
```

CÂU 9



Nhóm 6

xử lý yêu cầu HTTP POST nhằm thêm sản phẩm mới vào cơ sở dữ liệu. Khi người dùng gửi một yêu cầu POST tới endpoint này (cùng với dữ liệu sản phẩm trong body), API sẽ xử lý dữ liệu, lưu sản phẩm vào MongoDB, và trả về phản hồi với dữ liệu sản phẩm vừa thêm.

```
src > controller > ProductController.js > addProduct
1  import Product from "../models/product.js";
2
3  export const addProduct = async (req, res) => {
4      const data = await Product(req.body).save();
5      res.status(201).json(data);
6  }
```

CÂU 9



Nhóm 6

định nghĩa một route tại địa chỉ /products với phương thức POST. Mỗi khi có yêu cầu POST đến địa chỉ này, hàm addProduct sẽ được gọi để xử lý yêu cầu và thêm sản phẩm vào cơ sở dữ liệu.

```
File Edit Selection View Go Run Terminal Help
src > routes > api.js > default
1 import express from 'express';
2 import { addProduct } from '../controller/ProductController.js';
3 const router = express.Router();
4
5 router.post('/products', addProduct)
6
7 export default router;
```

The screenshot shows the VS Code interface with the Explorer sidebar on the left displaying the project structure. The 'routes' folder is expanded, and 'api.js' is selected. The main editor displays the code for 'default' in 'api.js', which sets up an Express router with a POST route for '/products' that calls the 'addProduct' function from 'ProductController.js'.

CÂU 9



Nhóm 6

Đoạn code này khởi chạy một ứng dụng web bằng Express.

```
src > server.js > ...
1  import express from 'express';
2  import path from 'path';
3  import configViewEngine from './config/ViewEngine.js';
4  import routerweb from './routes/web.js';
5  import dotenv from 'dotenv';
6  import connectdb from './config/db.js';
7  import routerapi from './routes/api.js';
8
9  dotenv.config();
10 const app = express();
11 app.use(express.json());
12 const port = process.env.PORT || 8081
13
14 connectdb(process.env.DB_URI)
15
16 configViewEngine(app)
17
18 app.use('/v1', routerapi)
19
20 app.listen(port, () => {
21   console.log(`listening on : ${port}`)
22 })
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LING

```
[nodemon] starting 'node ./src/server.js'
listening on : 8081
[nodemon] restarting due to changes...
[nodemon] starting 'node ./src/server.js'
listening on : 8081
```

Find As 🔍 No results ↑ ↓ ✕

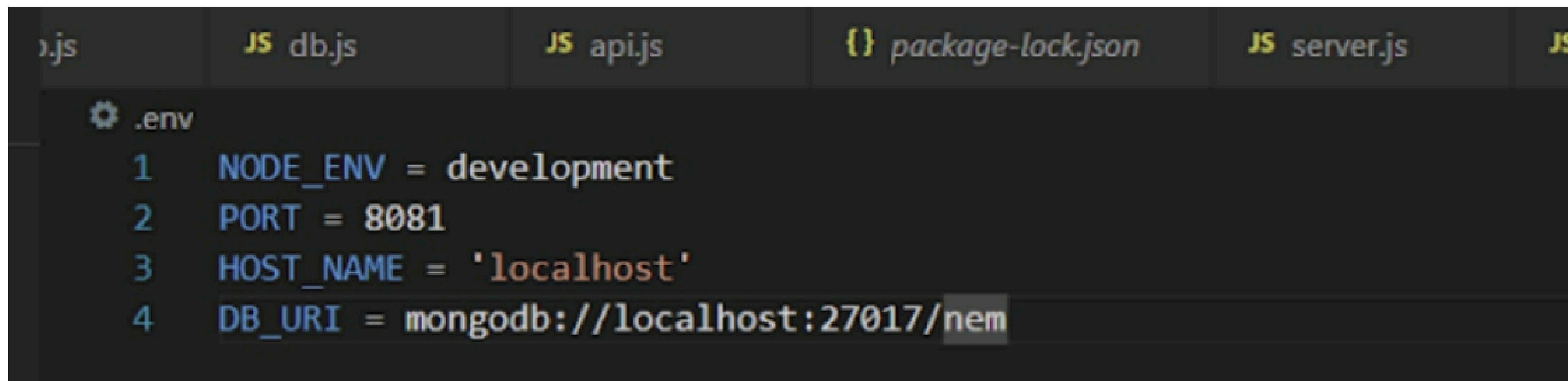
`app.use(express.json());` trong Express có chức năng:

Phân tích dữ liệu JSON: Nó cho phép ứng dụng xử lý các yêu cầu HTTP có định dạng JSON. Khi có một yêu cầu đến server với body là JSON, middleware này sẽ phân tích nó và biến đổi thành một đối tượng JavaScript, cho phép bạn dễ dàng truy cập dữ liệu thông qua `req.body`.

1. Nạp cấu hình từ `.env` và kết nối cơ sở dữ liệu.
2. Định nghĩa route `/cc` cho trang web và `/v1` cho API.
3. Khởi chạy server trên một cổng, lắng nghe các yêu cầu HTTP.

File .env là một file cấu hình được sử dụng để lưu trữ các biến môi trường trong các ứng dụng, thường là các thông tin nhạy cảm hoặc cấu hình mà bạn không muốn đưa trực tiếp vào mã nguồn.

Chúng ta có thể sử dụng thư viện dotenv để truyền các biến môi trường này vào các file ứng dụng.

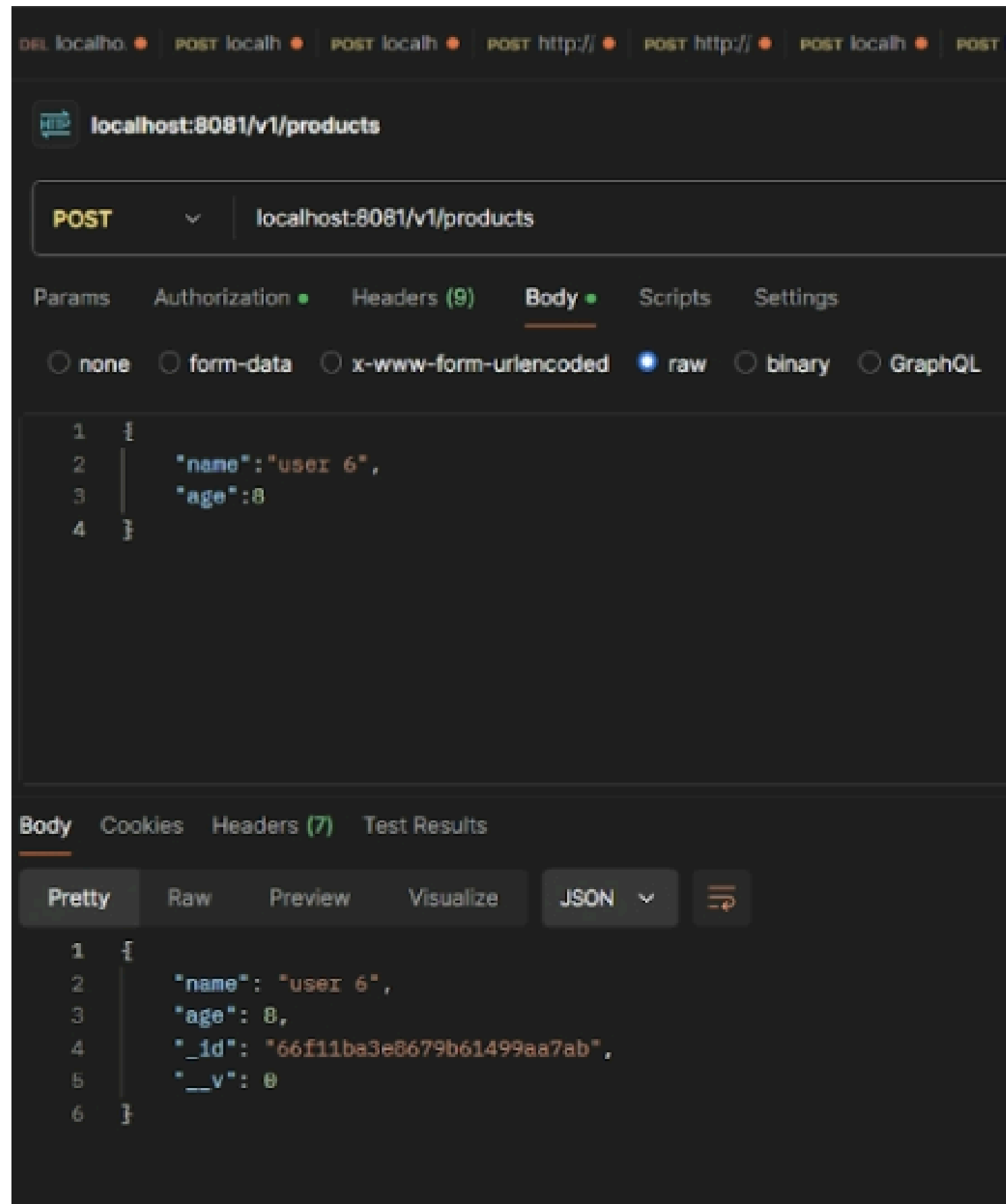


```
.js    JS db.js    JS api.js    {} package-lock.json    JS server.js    JS
. .env
1  NODE_ENV = development
2  PORT = 8081
3  HOST_NAME = 'localhost'
4  DB_URI = mongodb://localhost:27017/nem
```




Nhóm 6

CÂU 9



Tiến hành test bằng posman đối với api addProduct vừa được khởi tạo

_id:

- Đây là mã định danh duy nhất (unique identifier) của tài liệu trong MongoDB. Mỗi tài liệu trong một collection sẽ có một _id riêng biệt, giúp xác định tài liệu đó trong cơ sở dữ liệu. Trong trường hợp này, giá trị "66f11ba3e8679b61499aa7ab" là _id của người dùng.

__v:

- Đây là trường phiên bản (version key) của tài liệu trong Mongoose (ORM cho MongoDB). Nó giúp theo dõi số phiên bản của tài liệu và được tự động tăng lên mỗi khi tài liệu được cập nhật. Trường này có giá trị là 0, có nghĩa là đây là phiên bản đầu tiên của tài liệu.